HYDROLOGICAL CYCLE ALGORITHM FOR SOLVING OPTIMISATION PROBLEMS

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Supervisors

Professor Ajit Narayanan

Associate Professor Jacqueline Whalley

2018

By

Ahmad Wedyan

School of Engineering, Computer and Mathematical Sciences

Abstract

This research proposes a new nature-inspired algorithm called the *Hydrological Cycle Algorithm* (HCA), which simulates the movement of water drops in the hydrological water cycle. In the HCA, a collection of artificial water drops pass through various hydrological water cycle stages, such as flow, evaporation, condensation and precipitation in order to generate solutions. Each stage plays an important role in generating the solution and helps to avoid premature convergence. The HCA differs from other particle-based algorithms by using direct and indirect communication among the water drops, which helps to improve the overall performance and solution quality. The similarities and differences between HCA and other water-based algorithms are identified, and the implications of these differences on overall performance are discussed. In proof-of-concept experiments, the effectiveness and efficiency of HCA are evaluated on well-known discrete, continuous, static, and dynamic benchmarked optimisation problems. The experimental results were found to be competitive and validate the effectiveness of the proposed algorithm and its ability to escape from local optima solutions to converge on the global solution. In conclusion, the HCA provides a new particle-based conceptual framework within which existing and future work in water-based algorithms can be positioned.

Table of Contents

List of Tables

List of Figures

Copyright

Attestation of Authorship

"I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (expect where explicitly defined in the acknowledgements), nor material which to a substantial extension has been submitted for the award of any other degree or diploma of a university or other institute of higher learning."

[illegible]

Signature of candidate

Ahmad Wedyan

Auckland, New Zealand

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisors, Prof. Ajit Narayanan and Assoc. Prof. Jacqueline Whalley for their immense knowledge, motivations, constructive suggestions, comments, inspiration, and guidance. I could not have imagined having a better advisor and mentor for my PhD study.

My sincere thanks also go to School of Engineering, Computer & Mathematical Sciences administrative staff, who facilitated our study. I would especially like to thank Terry Brydon, Saide Lo, Karishma Bhat, and Quan Bai.

I would like to thank my parents, brothers, and sisters for their financial and spiritual support throughout my study, and my life in general. Their love and guidance are the main motivation for everything I do and give me an optimistic view of life.

Last but not the least, I would like to thank my colleagues and friends for their support and cooperation. In particular, I am grateful to Vijay Naidu for the stimulating discussions, support, and encouragement.

“*There is no better designer than nature*.”

Alexander McQueen

Chapter 1 Introduction

This chapter introduces the work accomplished in this thesis. It provides a general overview of various types of optimisation problems and the nature-inspired innovations. It also explains the motivation, approaches and underlying research questions, and clarifies the main contributions of this work. Finally, the contents of the remaining chapters are summarised.

1.1 Optimisation Problems

The term *optimise* means to improve the performance of something as far as possible. In computer science and operations research, an *optimisation problem* finds the best solution among a set of feasible solutions by trying different variations of the inputs (Rothlauf, 2011). In simple terms, optimisation is the process of finding the best possible solution. Complex optimisation problems are ubiquitous in the real world, especially those requiring decision-making among a set of alternatives. To solve many real-life problems, we mathematically formulate them as optimisation problems. A typical optimisation problem contains an objective function, variables, and constraints. The objective function is the function to be minimised or maximised. The variables are input to the objective function, and the constraints impose restrictions and limitations on the variables' values.

Optimisation problems can be classified into continuous or discrete based on the value domain of the variables, and into static or dynamic based on whether they are time-independent or time-dependent (Biegler, 2010; Rothlauf, 2011). Moreover, some problems can be categorised into single or multiple objective problems. In multiple objective problems, several objectives must be optimised simultaneously, but these objectives sometimes conflict. In dynamic problems, the variable values may be changed by unexpected events. The block diagram in Figure 1-1 illustrates the classification of optimisation problems. An optimisation problem may also be classified as a combination of these types (NEOS, 2016).

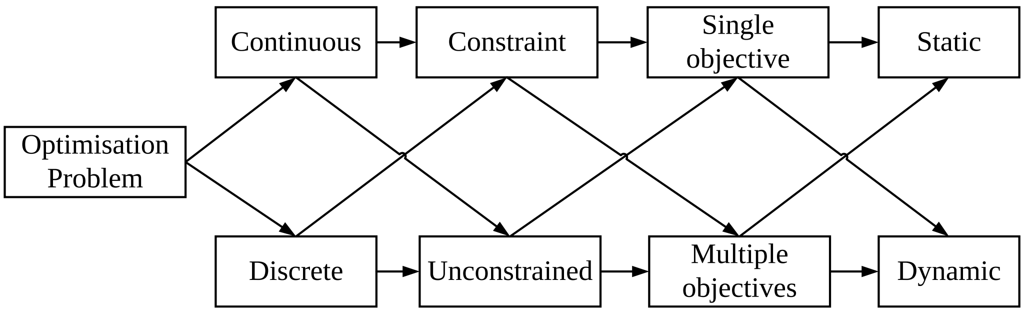


Figure 1-1. Classification of optimisation problems

The most intractable optimisation problems are *NP-hard* (i.e. non-deterministic polynomial-time hard) problems (Paz & Moran, 1981). NP-hard problems include many real-life problems. A solution is the act of solving a problem, however, there is no guarantee that this solution will be the optimal especially for NP-hard problems. Thus, a solution may be approximate or the best currently known. A classical NP-hard problem is the travelling salesman problem (TSP), in which a travelling salesperson must visit each of *n* cities exactly once and return to the start city. The TSP can be considered as a static discrete problem with a single objective function; to minimise the total cost of visiting all cities. Another discrete optimisation problem is the vehicle routing problem (VRP), which must serve a given number of customers at the anticipation lowest total cost with the smallest number of vehicles. Note that the VRP is a multiple-objectives problem that depends on the problem constraints. Moreover, because the problem inputs may change over time, the VRP also constitutes a dynamic optimisation problem.

Finding the best solution to an NP-hard problem by exhaustive search (i.e. brute force) is infeasible because the solution space grows exponentially with the problem size. Moreover, an optimal solution to an NP-hard problem is not guaranteed. For this reason, such problems are solved by metaheuristics algorithms rather than exact or heuristic methods. Exact methods such as dynamic programming and branch-and-bound are simple techniques suitable for small problems with simple constraints. These methods will always find a global solution. Heuristic methods are problem-dependent techniques which solve medium-sized problems by heuristics, with no guarantee to find the global solution.

Metaheuristic methods are problem-independent, and may use different heuristics simultaneously to find approximate solutions of large problems with complex constraints (Talbi, 2009). Furthermore, metaheuristic methods expand the search space in anticipation of bettering the solution within a reasonable execution time. When seeking an optimal solution, metaheuristic algorithms depend on two major processes (exploration and exploitation). Exploration aims to acquire as much information as possible about promising solutions by performing a random search in the solution space, thus diversifying the generated solutions and avoiding local optima. Exploitation returns the search to promising solutions that have been previously explored, increasing the chance of reaching the global optimum. However, if the exploitation is too deep, the search becomes trapped in a local optimum, whereas intense exploration may miss the global solution. Controlling the balance between exploration and exploitation is usually considered critical when searching for more refined solutions as well as more diverse solutions. Therefore, the algorithm performance depends on the proper balance between these two processes.

Metaheuristic algorithms can be evaluated using the criteria of robustness, efficiency, effectiveness, and applicability (Talbi, 2009). For instance, if an algorithm is executed many times to solve a specific problem, the results after each run will differ because the selected search path and the starting conditions vary among the runs. The number of evaluations required to reach the optimal or best-known result may also vary from run to run. An algorithm is defined as *robust* (i.e., is performance-stable) if its performance remains steady over different executions. Robustness can be measured by the success rate of the algorithm, which defines the number of successful executions (ending with acceptable solutions) relative to the total number of executions. A solution is deemed acceptable if the absolute difference between the obtained and best-known results is below a specified threshold. Therefore, to compare the robustness performances of different algorithms on the same problem, we must compute the average performance (success rate) and its variation in each algorithm. The efficiencies of algorithms can be compared by averaging the number of evaluations in each algorithm (the fewer the number of evaluations, the higher the efficiency). The effectiveness of an algorithm measures the algorithm's ability to find an best-known solution within a reasonable execution time, and the quality of the results when the algorithm is executed on different problems (Chase, Redemacher, Goodman, Averill, & Sidhu, 2010). The applicability refers to the simplicity of adapting the algorithm to different types of problems. An applicable algorithm can be adapted to a new problem without major structural changes and without affecting the relationship and consistency between the variables. Figure 1-2 presents a simple taxonomy of optimisation algorithms.

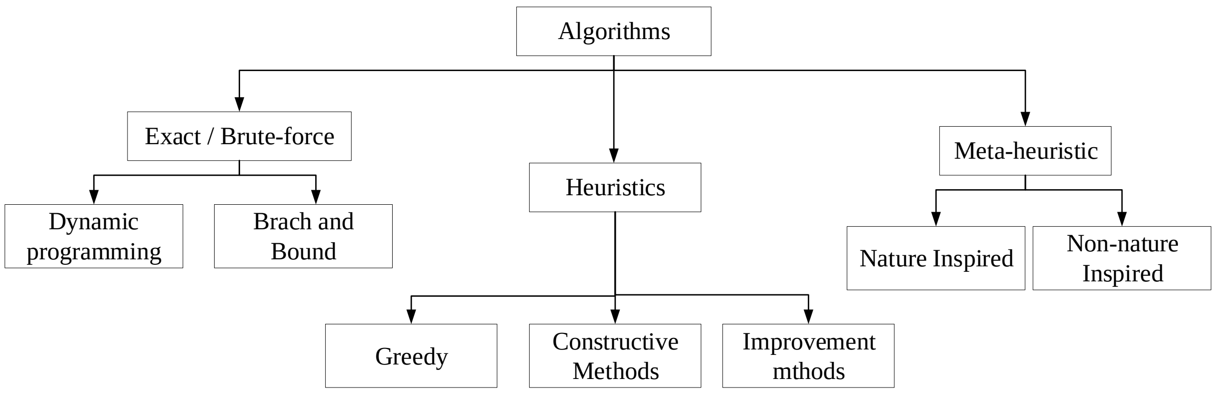


Figure 1-2. Taxonomy of optimisation algorithms

In order to create new algorithms, computer scientists have explored the nature looking for inspiration sources that can be used as computational approaches to solve problems.

1.2 Inspiration from Nature

By exploring nature, we reveal its many astonishing mysteries. Throughout life, humans are continually learning from nature and emulating natural phenomena. Therefore, nature is considered as an inspirational source for many human innovations. The process by which humans copy natural behaviours is called *Biomimicry*, a compendium of the Greek words *bios* (life) and *mimesis* (imitate) (Benyus, 1997). As a science, biomimicry refers to the human simulation activities of natural phenomena, processes, strategies, elements, or systems to solve various real-life problems (Benyus, 1997; Passino, 2005). Generally, nature solves complex problems in unique ways that maximise the efficiency and effectiveness of the solution. Therefore, natural phenomena have been widely observed by scientists and philosophers, who seek to analyse and understand them and explain their mechanisms then replicate them as artificial systems. Countless existing systems and inventions have been created by mimicking nature. For instance, after studying the self-cooling system in termite dens, architect Mick Pearce and his colleagues designed the Eastgate Centre building in Zimbabwe to be ventilated and cooled based on this termite den system. They were able to reduce the energy consumption needed for cooling the building (Nel Yomtov, 2014). Among the most famous examples of biomimicry is the construction and design of early aeroplanes based on bird flight.

In recent decades, a new field called *Nature-Inspired Computing* (NIC) has emerged within the computer sciences. This evolving field has been embraced by many computer scientists, due to its relevance to the design of computing models, computational materials, and novel techniques (de Castro, 2006; Rozenberg, Bck, & Kok, 2012). Indeed, the mechanisms of some natural phenomena are sufficiently well-understood to be adopted as problem-solving algorithms. An active branch of natural computing called *Nature-Inspired Algorithms* (NIAs) has received increasing attention and been massively developed in recent years (Brabazon, O'Neill, & McGarraghy, 2015). NIAs include models, methods, and algorithms inspired by different natural sources or mechanisms. These algorithms follow a systematic procedure with rigid rules and some degree of randomness. Based on their inspiration sources, NIAs are broadly categorised as evolutionary, biological, social swarm, physical, or chemical systems (Jr., Yang, Fister, Brest, & Fister, 2013). Owing to the diversity of these inspiration sources, NIAs adopt different metaphors and processes. Typically, the metaphorical name of an algorithm reflects the source of inspiration behind its design.

Many observable natural phenomena are complex because their overall behaviour is influenced by multiple elements. This complexity is further increased by the existing interactions and relationships among the elements and the environment. Examples of such phenomena are weather, water flow, earthquakes, and animal foraging processes. In addition, the behaviours of some natural systems are unpredictable and vague, with a large degree of randomness. These systems are best modelled by chaos theory (Lampton, 2000). In some biological systems, the entities perform tasks and respond to external events through dynamic interactions with other entities in the system. Therefore, by studying and understanding the workings of these systems and how the entities share information and interact during tasking, we can design powerful computer algorithms (Allen & Peissel, 1993; Chan, 2001).

This research develops a novel nature-inspired metaheuristic algorithm called the Hydrological Cycle Algorithm (HCA) for solving optimisation problems. The algorithm is based on the hydrological water cycle, a well-known natural phenomenon that describes the continuous movement of water from land to sky and vice versa (Davie, 2008). The water flows by four processes; evaporation, condensation, precipitation and runoff. The cycling changes the state of the moving water, exerting a purifying effect. Circulation begins with the heating of surface water by the sun, which changes the water drops to vapours which evaporate into the sky (atmosphere). Once in the sky, the vapour condenses into drops. As the condensed drops combine with other drops, they densify and become more influenced by gravity force, causing precipitation. When the ground becomes saturated, the water flows over the ground surface, and the cycle repeats.

The mechanisms of the separate stages and the full cycle can be readily exploited in an optimisation algorithm. First, rivers, without obstacles in their way, can be seen as seeking to follow the shortest paths towards the oceans. Second, each stage plays an important role in continuing the cycle, and each influences the other stages. Therefore, the HCA algorithm is divided into four stages that mimic the natural redistribution of freshwater flow, which enters the sea through rivers and returns to the land and rivers as rain. The rivers carve out a ‘solution path’ in the search landscape by soil removal and deposition. In the HCA, a collection of water droplets finds a solution by passing through the four stages of the hydrological cycle. Each stage plays an important role in generating the solution. The water cycle and algorithm procedures are fully explained in Chapter 3.

1.3 Motivations of the Research

This research was partly motivated by the author’s personal passion and admiration of nature. The ambiguity and mechanisms of natural phenomena never fail to excite and surprise.

Moreover, some natural systems, such as those that adapt to dynamic changes in the environment, will always challenge the capability of humans.

The design of efficient optimisation algorithms for solving hard problems is a much-researched topic in Computer Science. This research area has resulted in many different optimisation methods, that have been inspired by nature, and applied to solving a diverse set of real-world problems. Scholars have successfully solved many problems by efficient state-of-the-art optimisation algorithms. These algorithms often work well for one type of problem but are unable to solve a different problem. In other cases, they require major changes in their structure or working mechanism in order to be applied to different types of problems. Additionally, these algorithms are not necessarily future proof, since different types of problems impose different levels of diversity and complexity. Thus, existing algorithms may not be able to effectively solve new complex problems that continue to emerge. Therefore, the request for new computer algorithms that solve new types of problems is constantly expanding. Natural phenomena and systems inspire the designs of multipurpose algorithms that can handle such problems. Furthermore, the success and capability of NIAs encouraged us to expand and develop this field. This research is, in part, motivated by the need for optimisation algorithms that can be applied to complex real-life problems.

The *No-Free-Lunch* (NFL) theorems posit that no particular algorithm performs better than all other algorithms for all problems and that what an algorithm gains in performance on some problems can be lost on other problems. For example, suppose that algorithm *x* outperforms algorithm *y* in some problems. By the NFL theorem, algorithm *y* will outperform algorithm *x* in other problems (Wolpert & Macready, 1997). Consequently, we can infer that some algorithms are more applicable and compatible with particular classes of optimisation problems than others. Therefore, choosing the appropriate optimisation algorithm for a given problem is a nontrivial process. Especially given that there are no clear guidelines on choosing the best algorithm for a given problem in the literature. Conversely, the use of an algorithm and the types of problems it can solve is well known. In practice, an algorithm must be chosen based on a researcher's experience, previous studies of similar problems, or by trial-and-error (X.-S. Yang, Cui, Xiao, Gandomi, & Karamanoglu, 2013). The question 'why is this algorithm better than another for the given problem?' defies a clear answer, instead an answer requires a deep analysis of each algorithm and the inter-relationships among the algorithm parameters. In addition, a prior full understanding of the problem search space is required. Nonetheless, simple factors may dictate the final choice of an algorithm, such as the algorithm's simplicity, flexibility and ability to escape from local optima. In order to be able to develop algorithms that are more readily extended and used on a variety of different optimisation problems (a more generic solution) we first need to understand what aspects of an algorithm make it better at solving one type of problem than another. Thus, this research also will investigate the nature of the problems and existing algorithms and their applications in order to be able to achieve a better nature inspired algorithm. Thus, the structure of some algorithms makes it easier to apply them on specific problem domain. For instance, the ant colony optimisation is readily applicable on routing-based problems because of its natural structure.

In spite of the popularity of NIAs, many challenges remain that require further research and enhancements. For example, established water-based algorithms do not take into account the broader concepts of the hydrological water cycle, such as how water drops and paths are formed, or the fact that water is a renewable and recyclable resource. That is, previous water-based algorithms partially utilised some stages of the natural water cycle and demonstrated only some of the important aspects of a full hydrological water cycle for solving optimisation problems. Therefore, a further aim of this thesis is to delve deeper into hydrological theory and the fundamental concepts surrounding water droplets to inform the design of a new optimisation algorithm. This is aim is based on the conjecture that an algorithm that is closer to the "real" system is likely to demonstrate improved performance and be able to solve a broader range of optimisation problems with minimal modification. It is intended that from this new algorithm a new hydrological-based conceptual framework can be established which can be used to inform and position existing and future work in water-based algorithms.

One problem that many common algorithms face is that they cannot escape from local optima (stagnation). Another is that they often converge too quickly on a suboptimal solution. These weaknesses are a result of the design of the algorithm and improper implementation of the exploration and exploitation processes. The correct balance between exploration and exploitation is essential in optimisation algorithms, as both processes largely influence the convergence speed and the diversity of the generated solutions (Agarwal & Mehta, 2014). A further issue for many current algorithms, that can influence the degree of exploration and exploitation among other things, is the lack of communication between the algorithm entities. This lack of communication often degrades an algorithms performance.

Motivated these issues that are inherent in many current optimisation algorithms, a new metaheuristic algorithm was proposed, designed and evaluated in this research. The algorithm was based a natural phenomenon, the water cycle. From the onset, the algorithm was designed in order to be robust, efficient, effective, flexible, and applicable to different kinds of problems.

The design was informed by weakness identified in similar existing algorithms sourced from the literature. The main premise for this work that an algorithm that was closer to the natural system, which was the inspiration for the algorithm, should provide solutions to the weaknesses of other existing algorithms and thus give a better-quality solution.

1.4 Research Questions

This research aims to answer the following research questions:

Q1. *How should we design an optimisation algorithm based on the hydrological water cycle in nature?*

Q2. *How do the additional stages of the hydrological cycle algorithm influence the quality of the solutions?*

Q3. *To what extent can the hydrological cycle algorithm solve the travelling salesman problem and the capacitated vehicle routing problem?*

Q4. *Can the performance of this algorithm compete with those of other similar algorithms?*

Q5. *What factors affect the performance of the hydrological cycle algorithm when solving continuous optimisation problems?*

Q6. *Can the hydrological cycle algorithm deal with other NP-hard problems?*

Q7. *To what extent can the hydrological cycle algorithm cope with the dynamic vehicle routing problem?*

1.5 Contributions of the Thesis

The main contribution of this thesis is the design and implementation of a novel nature-inspired algorithm called the HCA. The developed algorithm was trialled on several kinds of NP-hard problems as a proof-of-concept. Initially, the algorithm was evaluated in simulation experiments that revealed how the parameters cooperate to avoid local optimal solutions. These experiments also elucidated the overall behaviour of the algorithm and its behavioural differences from the original Intelligent Water Drops (IWDs) algorithm.

The HCA functionalities are built into a mathematical model, which is implemented in MATLAB V.8.4 (R2014b) software. The algorithm is evaluated on optimisation problems in different categories (discrete, continuous, static, and dynamic). Each remaining chapter of this thesis explores a specific optimisation problem in one of these categories and explains the solution method of the algorithm. The performance of HCA is analysed in comparative studies of the results obtained by HCA and other algorithms. It is worth mentioning that extending the problem scope from static to dynamic is an interesting achievement by itself, as techniques for handling dynamic environments are scarce in the literature. Moreover, through this extension, we show how the cyclic water operation can be exploited for dynamically updating the inputs.

To validate the full hydrological inspired framework, the HCA was evaluated on a variety of operational research and logistics benchmarked problems, including the travelling salesman problem (HCA-TSP), continuous optimisation problems represents by mathematical benchmark functions (HCA-COP), and the capacitated vehicle routing problem (HCA-CVRP). The algorithm was then used to solve the dynamic CVRP (HCA-DCVRP). In each problem, the algorithm was tested on synthetic and benchmark datasets sourced from the literature. These benchmarked problems are useful for evaluating performance characteristics of any new approach, especially the effectiveness of exploration and exploitation processes. Solving these problems is challenging for most algorithms. The HCA’s performance is evaluated in terms of finding high-quality solutions and computational effort (number of iterations). As the third contribution, a unique representation was designed for the search space of the continuous problem.

Finally, the effectiveness of the algorithm is measured by comparing its results with those of other metaheuristics algorithms (state-of-the-art algorithms). Our minimal criterion for success is that the HCA algorithm performance is comparable to other well-known algorithms including those that only partially adopt the full HCA framework. Therefore, the main aim is not to prove the superiority of HCA over other algorithms. Instead, the aim is to provide researchers in science, engineering, operations research and machine learning with a new water-based algorithm for dealing with optimisation problems. Overall, the HCA provides high-quality solutions to various kinds of problems. Therefore, the HCA presents as a promising alternative optimisation algorithm.

A further contribution is a comprehensive literature review presenting an analysis of the strengths, weaknesses and application areas of some well-known nature-inspired algorithms.

1.6 Research Approach

This research designs a new nature-inspired algorithm. To this end, it adopts the "*Design Science*" methodology, the approach used in similar research (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007). The research proceeded through four main phases. First, a strong knowledge of the research background was acquired and distilled into a comprehensive literature review, leading to the initial research questions. Based on these questions, the HCA was designed and implemented. Next, the algorithm was tested in a series of experiments.

Before moving to complicated problems, the algorithm was run on a simple problem that verified its concept and enabled tuning of the parameter values. During the test stage, benchmark datasets were collected from the literature, and the accuracy of the solution search was evaluated in a statistical analysis. The obtained results were also compared with those of other metaheuristics algorithms, and the performance and total execution time of the algorithm were evaluated. These key steps are illustrated in Figure 1-3.

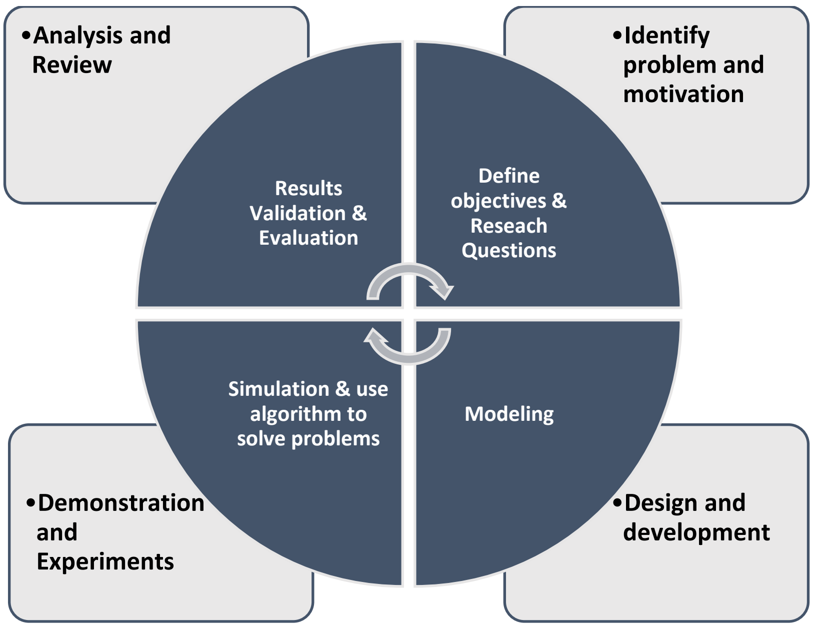

•Analysis and Review
•Identify problem and motivation
Results Validation & Evaluation
Define objectives & Reseach Questions
Simulation & use algorithm to solve problems
Modeling
•Demonstration and Experiments
•Design and development

Figure 1-3. Design Science methodology

When basing a new algorithm on a natural phenomenon, specific steps must be followed. First, we must sharpen our perception by observing and studying the mechanisms of the natural phenomenon. Second, we must mimic the structure of the natural phenomenon (and hence identify the main stages of the algorithm). This step provides an abstract view of a conceptual model that explains some of the assumptions being considered. Third, we develop mathematical formulations that describes the overall behaviour of the algorithm and the relationships among the variables. The mathematical model provides an abstract view and improved understanding of a wider class of systems. Commonly, a mathematical model represents or describes real-life situations by numbers, variables and equations. The numbers are constants, and variables symbolise factors that change their values under certain conditions. The equations explain the relationships among the variables and the effects of the variables on other variables. In the test stage, the algorithm should be implemented on various problems. Finally, guided by the test results, the mathematical model is refined and improved. These steps are summarised in Figure 1-4.

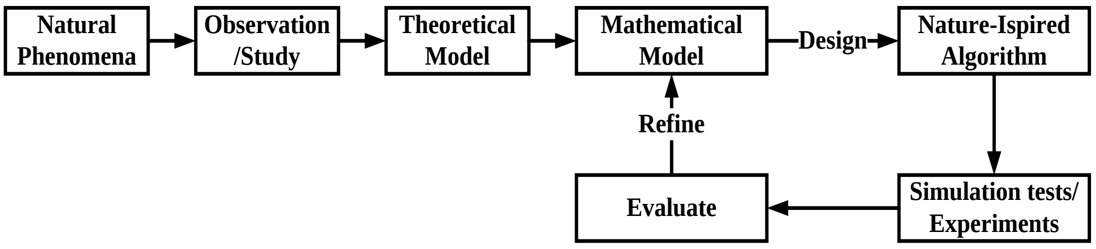


Figure 1-4. Main steps in developing a new natural-inspired algorithm

These steps map to the main steps in the design science methodology. The study and analysis of the natural phenomenon leads to the definition of the objectives and motivations for the research. Building the mathematical model and designing the algorithm form steps in the development stage. Through the testing and the evaluation steps the results obtained are used to help improve the algorithm.

1.7 Outline of the Thesis

This thesis is structured into eight chapters as summarised in Figure 1-5.

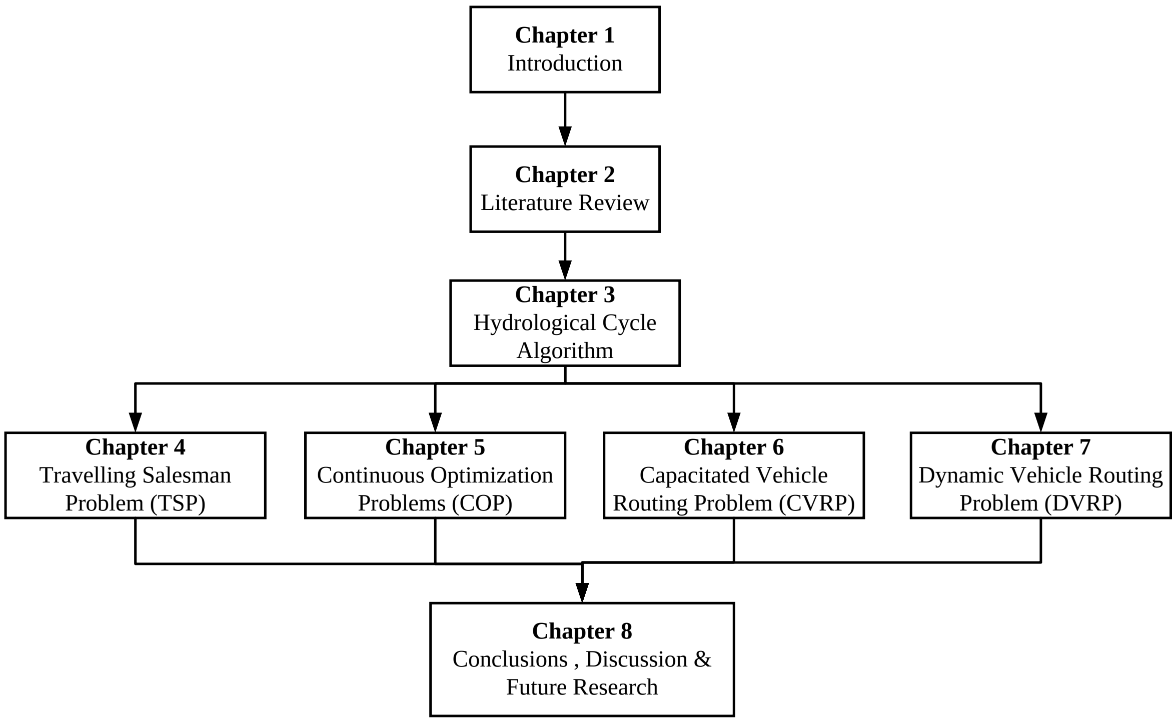

Chapter 1
Introduction
Chapter 2
Literature Review
Chapter 3
Hydrological Cycle Algorithm
Chapter 4
Travelling Salesman Problem (TSP)
Chapter 5
Continuous Optimization Problems (COP)
Chapter 6
Capacitated Vehicle Routing Problem (CVRP)
Chapter 7
Dynamic Vehicle Routing Problem (DVRP)
Chapter 8
Conclusions , Discussion & Future Research

Figure 1-5. Organisation of the thesis

The remainder of the thesis is organised as follows:

Chapter 2: This chapter analytically reviews several well-known nature-inspired algorithms. Of particular interest are algorithms inspired by the movement and activities of natural water. Weaknesses in the designs of existing approaches, along with successful applications of these approaches, are also highlighted. Finally, the chapter compares the general procedures of similar algorithms to identify their differences and similarities.

Chapter 3: This chapter begins with a detailed explanation of the natural hydrological water cycle, on which the algorithm is based. It then details the HCA, including the main procedure, its working mechanism, and its mathematical model. The HCA is competed against IWD algorithm in specific systematic experiments. Finally, the structure of HCA is compared with those of other water-based algorithms.

Chapter 4: This chapter solves the TSP by HCA. The algorithm is tested on generated synthetic instances and benchmark datasets from the literature. The performance of the algorithm is observed and analysed. The results of HCA and other algorithms are compared.

Chapter 5: This chapter solves COPs by HCA. The algorithm is tested on a benchmark of continuous mathematical test functions obtained from the literature. The efficiencies of HCA and other algorithms are statistically compared. A new representation of continuous problems is also proposed.

Chapter 6: This chapter solves a combinatorial optimisation problem, namely, the CVRP, by HCA. A mathematical formulation of the CVRP is presented, and local improvement methods for solving this problem are outlined. The algorithm is tested on generated synthetic instances and benchmark datasets from the literature. The obtained results are analysed and discussed.

Chapter 7: This chapter solves a dynamical environment problem, namely, the DCVRP, by HCA. The problem is overviewed, and some possible events in this problem are explained. The algorithm is tested on adapted and synthetic instances with various criteria. The computational results analysed and discussed.

Chapter 8: This chapter discusses the work undertaken in this thesis and concludes the study. It also suggests ideas and directives for future work.

1.8 Publications

The following research papers have been composed and published throughout the doctoral study of this research (Wedyan & Narayanan, 2014; Wedyan, Whalley, & Narayanan, 2017, 2018). In these papers, I am the main author, and have contributed more than 80% in each paper. These published papers have been used in different sections of this thesis.

- Wedyan, A. F., and Narayanan, A. (2014). Solving capacitated vehicle routing problem using intelligent water drops algorithm. In *10th International Conference on Natural Computation, ICNC 2014*. https://doi.org/10.1109/ICNC.2014.6975880.
    - This paper was used in Chapter 6 to compare the results of IWD algorithm with HCA results.
- Wedyan A. F., Whalley J., and Narayanan A. (2017). "Hydrological Cycle Algorithm for Continuous Optimization Problems," Journal of Optimisation, vol. 2017, Article ID 3828420, 25 pages. https://doi.org/10.1155/2017/3828420.
    - This paper has been used in Sections 2 and 3 of Chapter 3, and in Sections 2, 3, and 4 of Chapter 5.
- Wedyan A. F., Whalley J., and Narayanan A. (2018). "Solving the Traveling Salesman Problem using Hydrological Cycle Algorithm," American Journal of Operations Research, vol. 8, No. 3, Article ID 1040612, (in press).
    - This paper has been used in Sections 1, 2 and 3 of Chapter 4.

Chapter 2 Literature Review

"Biology and computer science – life and computation – are related. I am confident that at their interface great discoveries await those who seek them"

[Leonard Adleman, Scientific American, August 1998]

This chapter provides an extensive analytical review of the most well-established algorithms, particularly, the nature-inspired algorithms. It also examines various common aspects of the design process of nature-inspired algorithms. A simple taxonomy of these algorithms is presented based on their natural sources. The natural source, features, working mechanism, strength, and weakness of each algorithm is described. The pseudocode of each algorithm is also provided. Similar algorithms are compared in a comprehensive conceptual analysis that attempts to identify the best application and success areas, and the most suitable problem domains, of each algorithm.

This chapter is organised as follows. Section 2.1 discusses the common aspects of designing nature-inspired algorithms. Sections 2.2 review twenty-two algorithms and their applications to different types of problems. Section 2.3 provides a comparative analysis of these algorithms by comparing the frameworks of similar types of algorithms. Conclusions are presented in Section 2.4.

2.1 Design of Nature-Inspired Algorithms

Nature-inspired algorithms are named after the sources that inspired them. In an NIA, natural phenomena are viewed as processes that can be computationally modelled. The success of an NIA depends on whether the algorithm properly balances its exploration and exploitation behaviours. Many NIAs solve problems by following similar steps (i.e., by sharing a similar paradigm) (X.-S. Yang, Deb, Fong, He, & Zhao, 2016). The common underlying process of NIAs is presented in Figure 2-1.

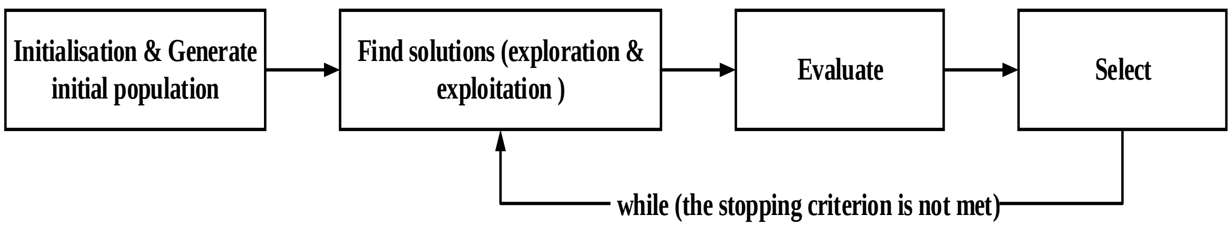

Initialisation & Generate initial population
Find solutions (exploration & exploitation)
Evaluate
Select
while (the stopping criterion is not met)

Figure 2-1: General paradigm of a nature-inspired algorithm

Each step implements a particular sub-process towards generating the final solution. Although many NIAs share similar steps and certain characteristics, the designs and mathematical models of each step differ among the algorithms. Points of difference include:

- Algorithm entities.
- Initial values of the entities.
- Distribution of the entities.
- Solution generation mechanism.
- Moving the entities.
- Choosing the next point.
- Problem representation.
- Evaluation and selection.
- Termination condition.
- Information sharing.
- Auxiliary actions.

The entities[1] in an NIA usually perform specific operations and generate potential solutions to a problem, by interacting with the environment or with each other (de Castro, 2006). The representation (i.e. metaphorical name), capability and properties of an entity depends on the algorithm. For instance, the entities in an ant colony optimisation (ACO) algorithm are artificial ants that generate solutions by moving from one place to another. In a genetic algorithm (GA), the entities are artificial chromosomes that generate solutions through chromosomal operations such as mutations and crossovers. Despite these differences, the entities in any NIA algorithm implement three common processes—self-adaptation, cooperation and competition—in the search for a global solution.

Self-adaptation is the process by which an entity improves itself in each iteration. Cooperation is the ability of the entities to collaborate and share information to improve the solution quality. Competition represents the competitive survival ability of an entity (Rashedi, Nezamabadi-pour, & Saryazdi, 2009). In some algorithms, the number of entities affects the quality of the final solution. A large number of entities can discover a large area of the search space, but consume much computational time. Furthermore, the number of entities is not fixed in some algorithms, where it may change (increase or decrease) based on some conditions. This technique may restrict the exploration of the search space with the decreasing of number of entities or increasing the computational efforts when increasing the number of entities.

The configurations and strategies of the initial values also differ among the algorithms. Some algorithms initialise the entities with randomly generated values; others assign no initial values to the entities. Like the number of entities, the initial entity values are known to influence the solution quality (Maaranen, Miettinen, & Penttinen, 2007; L. Zhang, Yang, & Elsherbeni,

[1] The terms entities, agents, and individuals are interchangeably used in different algorithms.

2009). To enhance the performance, some algorithms initialise the entities using a systematic function such as a Gaussian distribution kernel rather than randomly generated values.

When distributing the entities over the search space, some algorithms distribute the entities randomly or uniformly over a portion of the search space, while others start all entities from the same position. For example, the particle swarm optimisation (PSO) algorithm, in some of its variants, customises the topology of the particle distribution (Medina, Pulido, & Ramirez-Torres, 2009).

When generating solutions, each algorithm discovers the search space in different ways under different rules. The quality of the obtained solution depends on the exploration (i.e. diversification) and exploitation (i.e. intensification) processes employed by the algorithm. The most effective NIAs provide a suitable trade-off between exploration and exploitation. Moderation is required because extreme exploration generates diverse solutions and may find a high-quality solution, but lowers the convergence speed, whereas an extreme (intense) exploitation risks trapping in local optima or premature convergence. Some algorithms rely on the number of entities to balance the exploration and exploitation. A large number of entities widens the exploration area of the solution space but increases the execution time, whereas a few entities search only a small area of the solution space. Other algorithms, such as the gravitational search algorithm (GSA), start with an extreme exploration of the search space and gradually shift to an intense exploitation of the promising areas.

In most NIAs, the entities are continually moved from one point to another as they explore the search space. Therefore, the selection of the next best point critically affects the fitness of the generated solution. Some algorithms, such as the ACO algorithm, control the entities' movements by pseudo-random transition probability rules. Other algorithms impose an attractive force between the entities that affects their movement, or shift the previous entities' positions in random ways, like the PSO algorithm. Furthermore, a solution can be constructed in parallel or sequentially. In parallel solution construction, N solutions are incrementally built by N entities. In sequential construction, each entity builds one solution in turn. The performances of these constructions depend on the problem being solved.

The evaluation step, which evaluates the fitness of the solution obtained by each entity, is similar for most NIAs. The solution quality measure depends on the objective function of the problem being solved, and whether the optimisation is a minimisation or a maximisation.

At the end of each iteration, the evaluation is followed by a selection process, which comprehensively compares the solution qualities of all entities. Usually, the selection process identifies the best entity (i.e. the elite entity) and uses its solution to update the best previous solution. This leads to competition among the entities, as fitter entities will survive or be rewarded. Selection is a crucial process that affects the search direction in subsequent iterations by favouring or avoiding regions of the search space. In some problems, selection dramatically improves the algorithm performance by reducing the effort needed to rediscover areas of previous solutions. Algorithms select the best solution (i.e., entity) by different mechanisms and techniques. The easiest and commonest method is best-entity (i.e., elitism) selection based on the fitness value, which incarnates the *survival of the fittest* principle (natural selection) (Spencer, 1864). More complicated and probabilistic selection methods include roulette wheel, tournament, and rank selection. Other algorithms, such as the simulated annealing (SA) algorithm, sometimes accept a non-better solution under some probabilistic criteria. This prevents the algorithm from trapping in local optimal solutions.

Some algorithms implement optional auxiliary processes, such as local improvement methods, at the end of each iteration. Some of these methods cannot be completed by a single entity, and require cooperation among all entities or some selected entities only. The ACO algorithm conditionally implements *daemon actions* after each iteration to handle the problem specifications (Dorigo, Stützle, & Stutzle, 2004).

The problem representation (i.e., the search space mapping) is the input to the algorithm. Some problems are easily represented while others (especially those involving permutations) are difficult or complex. Also, some problems can be represented in multiple ways with varying degrees of difficulty. In these cases, the problem description determines whether the problem will be easy or difficult to solve (Fink, 2002). Therefore, the problem in each algorithm should be represented in a suitable form. Successful problem-solving by an algorithm requires compatibility between the problem representation and the algorithm specifications. Many algorithms are designed for a specific problem representation, which complicates or restricts their applicability to other types of problems. For instance, the problem may be represented as a graph, or constructed in multidimensional space. For this reason, some problems are popularly solved by certain algorithms that deliver outstanding performance in those instances.

Algorithms differ not only in their designs, but also in their number of parameters and mathematical calculations in each step. These differences vary the overall performances and complexities of different algorithms. Meanwhile, the relationships and harmony among the variables influence whether the algorithm can escape local optima, and partly control the exploration and exploitation processes. As the results of one step are input to the next step, the performance of each step depends on the success of its prior steps. Therefore, a defect in the design of one step will degrade the performance of the successive steps.

A successfully designed algorithm must also ensure cooperation and interaction between the entities in the system. Typically, this can be achieved by efficiently utilising the information exchange and sharing among all or some of the entities. Such collective problem solving significantly enhances the overall performance of the algorithm and produce high-quality solutions (al-Rifaie, Bishop, & Blackwell, 2012; Hogg & Williams, 1993). Some algorithms facilitate interaction and communications among the entities; others treat the entities as autonomous agents with no interactions. In most algorithms, the entities communicate either directly or indirectly. Most commonly, the global-best solution is directly communicated to all entities, guiding them towards that location. Meanwhile, the entities communicate indirectly through the environment.

In general, many NIAs emulate natural phenomena through rules and randomness. A *stochastic* algorithm embeds randomness within the various components of the algorithm (Brownlee, 2011). This randomness helps to produce diverse solutions, especially in multimodal problems, and promotes escape from undesired local solutions when seeking the global solution. Obviously, if the randomness is insufficient, the algorithm becomes excessively deterministic and generates the same solutions in multiple executions because of limiting the exploration process.

Finally, algorithms differ in their behaviours. For example, some algorithms track the current global best solution, whereas others save several latest-best solutions. Furthermore, the termination strategy of the algorithm depends on the user’s specifications and preferences. Common stopping criteria are the maximum iteration number, maximum CPU time, non-improvement of the solution quality after a certain number of iterations, or improvement below a specified threshold from the previously obtained solutions.

It is worth distinguishing algorithms by their *level of inspiration*, which measures the similarity between the algorithm framework and the natural phenomenon mimicked by the algorithm. Algorithms with a strong degree (correspondence or appeal) of inspiration completely simulate all processes/activities observed in the natural phenomenon, whereas those with a weak degree of inspiration simulate selected processes from the overall system. In some algorithms, the extent of the natural inspiration is unclear. That is, when declaring that an algorithm is inspired by nature, we cannot always clarify the extent of the natural inspiration (Steer, Wirth, & Halgamuge, 2009). On the other hand, some algorithms incorporated or introduced other non-natural components to be part of the algorithm, raising the doubts about its authenticity. In addition, some natural processes are more suitable for solving computational problems and are more easily transferred into computational processes, than others. On the other hand, some natural systems spontaneously behave in certain ways. Therefore, before exploiting a natural phenomenon in an optimisation algorithm, we must understand what this phenomenon accomplishes and optimises in nature. More details about the rationale behind nature-inspired algorithms can be found in Steer et al. (2009).

In conclusion, the conceptual designs of algorithms differ in many respects. Moreover, the various ways of implementing an algorithm (i.e. design strategies) crucially affect the performance, convergence, success rate, and solution quality of the algorithm. Choice of the best implementation will depend on the problem being solved or can be decided from previously reported preferences.

2.2 Nature-Inspired Algorithms

Throughout the past decades, the number of nature-inspired optimisation algorithms has increased remarkably. These algorithms have been extensively evaluated and enhanced to improve their computational performance and mitigate their weaknesses. Within this new era, researchers are increasingly applying NIAs to real-life problems. As the plethora of existing NIAs cannot be covered in a single review, this section reviews only some of the commonly used algorithms. The algorithms are estimated to be of particular importance to the present thesis because they share certain aspects. Each algorithm will be described as mentioned in its original version.

NIAs can be categorised in different ways. A simple categorisation is the source of inspiration behind the algorithm (Siddique & Adeli, 2015), which classifies NIAs into the following hierarchy:

- Nature-inspired algorithms
    - Physics-based algorithms (PBA)
    - Chemistry-based algorithms (CBA)
    - Biology-based algorithms (BBA)
        - Evolutionary algorithms (EA)
        - Swarm intelligence-based algorithms (SIA)
- Non-nature-inspired algorithms

These categories, and a selection of algorithms in each category, are elaborated in Figure 2-2.

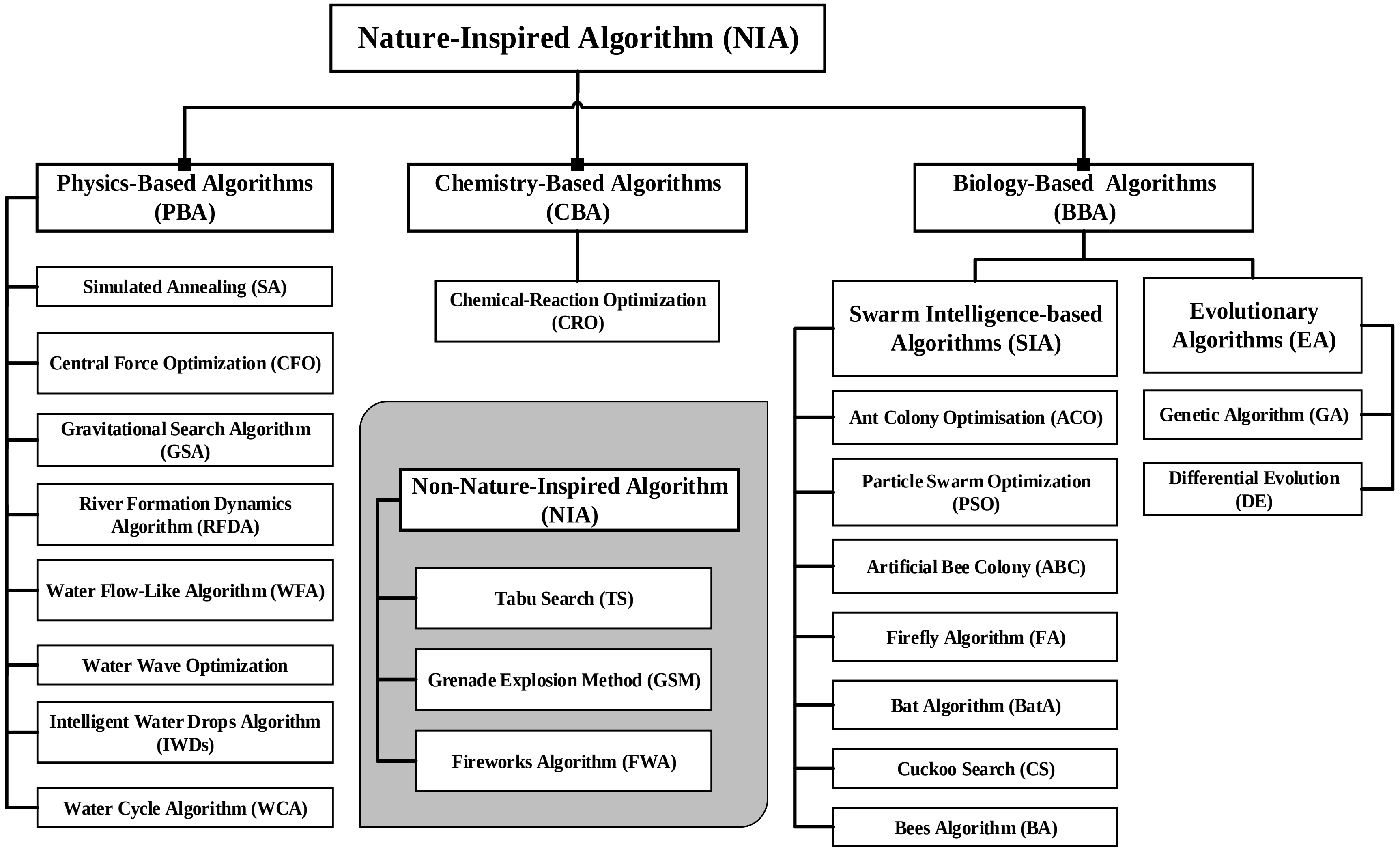

Nature-Inspired Algorithm (NIA)
Physics-Based Algorithms (PBA)
Chemistry-Based Algorithms (CBA)
Biology-Based Algorithms (BBA)
Simulated Annealing (SA)
Central Force Optimization (CFO)
Gravitational Search Algorithm (GSA)
River Formation Dynamics Algorithm (RFDA)
Water Flow-Like Algorithm (WFA)
Water Wave Optimization
Intelligent Water Drops Algorithm (IWDs)
Water Cycle Algorithm (WCA)
Chemical-Reaction Optimization (CRO)
Non-Nature-Inspired Algorithm (NIA)
Tabu Search (TS)
Grenade Explosion Method (GSM)
Fireworks Algorithm (FWA)
Swarm Intelligence-based Algorithms (SIA)
Evolutionary Algorithms (EA)
Ant Colony Optimisation (ACO)
Particle Swarm Optimization (PSO)
Artificial Bee Colony (ABC)
Firefly Algorithm (FA)
Bat Algorithm (BatA)
Cuckoo Search (CS)
Bees Algorithm (BA)
Genetic Algorithm (GA)
Differential Evolution (DE)

Figure 2-2: Classification of nature-inspired algorithms (adapted from Siddique & Adeli, 2015, P.710)

Although this classification can help to distinguish algorithms, some algorithms can be classified into more than one of these classes. Among the many algorithms proposed in the literature, some algorithms provide higher-quality solutions than others for the same problem. These high-performance algorithms have gained popularity and their use has extended to other problems. Why an algorithm outperforms another on the same problem is difficult to discern, but certain factors contribute to high performance, including accurate design of the algorithm (especially the mathematical model), powerful exploration and exploitation processes, ability to escape from local optima, a good convergence rate, and few parameters to be tuned. The most popular algorithms are characterised by high performance and efficiency, an easily understood procedure, easy implementation, and easy extendibility to different types of problems.

Two algorithms can be partially or fully combined into *hybrid algorithms*, which deliver high performance and high-quality solutions to optimisation problems. Generally, one algorithm explores the solution space while the other exploits promising regions. Alternatively, the strengths or properties of one algorithm substitute the weaknesses of another algorithm to enhance the overall performance (Rao Venkata & Savsani, 2012). Algorithms can also be modified (i.e., altered in structure) to solve a particular kind of problem. Some algorithms can be easily implemented and executed in a parallel technology, which divides tasks across multiples processers and combines the results output by all processors.

Responding to the increasing interest in NIAs, many experts have compiled books describing NIA procedures (Brownlee, 2011; R Chiong, 2009; Engelbrecht, 2007; Rao Venkata & Savsani, 2012; Xing & Gao, 2013; X.-S. Yang, 2014a). Many reviews of NIAs have also been published (Bhuvaneswari, Hariraman, Anantharaj, & Balaji, 2014; Jr. et al., 2013; Kari & Rozenberg, 2008; Siddique & Adeli, 2015; Zang, Zhang, & Hapeshi, 2010).

The next sections examine some algorithms as presented in the literature in more detail focusing on the critical aspects for this thesis namely, exploitation and exploration techniques, similar aspects, and application problem type. For full details of the mathematical models and equations of these algorithms the reader should refer to the original literature on the topic.

2.2.1 Physics-based Algorithms (PBAs)

Physics is rich in theories, mechanisms, and phenomena related to matter, objects and energy. Physical theories include thermodynamics, quantum mechanics, fluids, and electromagnetism. Physicists use mathematical models to describe the possible relationships between objects, energy conservation, and the general properties of matter. Physics is governed by laws such as conservation of mass, momentum, and energy, the rules of fluid motion, Newton's laws of gravitation, and thermodynamic principles.

Computer scientists have designed many PBA-based optimisation tools that exploit physical principles, processes, concepts, motion mechanisms, or states of matter. Some PBA algorithms are quite similar in their theoretical frameworks: reflecting the conceptual interconnection and commonality between certain laws of physics or similar fundamentals of these laws. Some of these algorithms are described in the following subsections. Water-based algorithms (on which this thesis is based) form a subcategory of PBAs. A review of PBAs can be found in Biswas et al. (2013), and Can and Alatas (2015).

2.2.1.1 Simulated Annealing Algorithm

The earliest PBA the simulated annealing (SA) algorithm was formulated by Khachaturyan et al. (1979), and then extended and used for solving TSP by Kirkpatrick et al. (1983). Annealing is the process of heating a metal to its melting point, then allowing it to slowly cool. During the cooling, the particles become arranged in a steady state (i.e., a low energy state). Metal particles move easily at high temperatures and slowly at low temperatures (van Laarhoven & Aarts, 1987). To exploit this process in problem-solving, SA has a temperature-representing variable that starts at an initial value and gradually decreases to a specified value.

The SA is considered as a probabilistic iterative algorithm; that is, a point is moved in the search space at every iteration. The position of the point represents a candidate solution of the problem. After evaluating the new position of the point, the SA updates the current best-solution if the new position is better than any previous solution. However, the SA accepts worse solutions with various acceptable probabilities. The probability depends on the temperature, which reduces as more of the solution space is explored. Therefore, the SA allows wide exploration at high temperatures and restricts or narrows the search at lower temperatures. These processes continue until the algorithm converges to the optimal/near optimal solution. Figure 2-3 shows the main pseudocode of the SA based on our interpretation of the algorithm (Eglese, 1990).

```
PROCEDURE: Simulated annealing algorithm
$T$ := initial temperature
current point := generate a point in a random position
REPEAT UNTIL: stopping criterion is met
      WHILE iteration < maximum iteration number
          Point := move to a new position
          //compare the two points
          // If the new point is better
          IF ($f$ (point) < $f$ (Initial point))
               current point := point
               // If the new point is worse
          ELSE IF rand [0,1]  < $\exp\{-(f(\text{point}) - f(\text{current point})) / T\}$
               current point = point
          END IF
      END WHILE // until system reaches equilibrium state
      $T := \alpha \times T$ // decrease the temperature
END REPEAT // reaches frozen state
RETURN: current point
```
Figure 2-3. Pseudocode of simulated annealing

Owing to its simplicity, SA has been widely applied to various optimisation problems, both discrete and continuous. Table 2-1 summarises some recent applications of the SA.

Table 2-1. Simulated annealing applications

Application	**References**
TSP	(Kirkpatrick et al., 1983); (P. Tian & Yang, 1993); (Bookstaber, 1997); (Z. Wang, Geng, & Shao, 2009); (Geng, Chen, Yang, Shi, & Zhao, 2011); (Yang Li, Zhou, & Zhang, 2011); (C. Wang, Lin, Zhong, & Zhang, 2015); (Zhan, Lin, Zhang, & Zhong, 2016); (X. Wu & Gao, 2017)
COP	(Corana, Marchesi, Martini, & Ridella, 1987); (Cardoso, Salcedo, & Feyo de Azevedo, 1996); (Bertsimas & Nohadani, 2010); (Avila & Valdez, 2015); (Kadry & El Hami, 2015); (Askarzadeh, Coelho, Klein, & Mariani, 2016)
VRP	(Breedam, 1995); (Y. Xiao, Zhao, Kaku, & Mladenovic, 2014)

The SA incorporates no entities, but finds a solution simply by evaluating a point generated at a varying position (single-point search). Therefore, the SA generates one solution in each iteration. The SA can escape from local optima by accepting bad solutions. The main drawbacks of the SA are the lack of communication or interactivity for improving the solution, the lack of guidance by a heuristic function, and the large dependence of the performance on the temperature (control parameter) and update technique. Therefore, fine-tuning this single variable is a delicate process. Another drawback is that when applied to some problems, the SA converges slowly to the optimal solution. However, the SA is advantaged by its low memory demands during execution and solution generation (i.e., memoryless algorithm).

Zhan et al. (2016) solved the TSP using SA and a list-based technique. The main objective of the list base technique was to simplify the tuning of the temperature value. This technique involves keeping a priority queue of values that control the decrease in temperature. Each iteration, the list is adapted based on the solution search space. The maximum value in the list is assigned the highest probability of becoming a candidate temperature. The SA employs local-neighbour search operators such as 2-Opt (a.k.a. 2-Optimal), 3-Opt (a.k.a. 3-Optimal), insert, inverse, and swap. The 2-Opt and 3-Opt operations work by deleting two or three edges, respectively, and reconnecting the edges in different orders to reduce the total cost. The effectiveness of this algorithm has been measured in variously sized benchmark instances. The results were competitive with those of other algorithms.

Geng et al. (Geng et al., 2011) solved the TSP using adaptive SA combined with a greedy search. The greedy search was intended to improve the convergence rate. The SA implemented three types of mutations with different probabilities: vertex insertion, block insertion, and block reversion. The algorithm was tested on sixty benchmark instances. The results showed that the SA algorithm was more effective in some cases (in terms of CPU time and accuracy) than other algorithms (like: ACO, GA, PSO, and neural network).

Alfa et al. (1991) combined SA with the 3-Opt operation to solve VRPs. Their algorithm is based on a Route First and Cluster Second approach. The algorithm starts by building a large tour that visits all the nodes. At the end of each iteration, it then generates three random numbers that indicate the locations of customers in the tour to be swapped, and generates eight new solutions by a 3-Opt enumeration. The best solution is selected with a probability that depends on the temperature value. In three examples with different numbers of customers (30, 50, and 75), the algorithm achieved the best-known results, but only after an extremely long execution time.

Lin et al. (2006) applied the SA to the CVRP. After building an initial solution using the savings algorithm, the SA improves the solution using 2-Opt and insertion. When tested on fourteen benchmark instances, the algorithm found the best-known solution in six instances, and solutions with small percentage deviations from the best-known result in the remaining instances.

In Harmanani et al. (2011), the CVRP was solved by SA combined with problem-knowledge information operators. The algorithm starts with a feasible initial solution that is deterministically generated by a greedy algorithm. The SA produces new solutions using two types of transformations: one transformation moves the five closest nodes from one route to another based on the vehicle capacity, the other selects the five most distant nodes and removes them from their routes. On Euclidean instances with integer edge costs, the algorithm found the best-known solution in small instances, and near-optimal solutions in large instances.

2.2.1.2 Central Force Optimisation

Formato (2007) proposed a new PBA called central force optimisation (CFO). The CFO algorithm is based on the motions of objects under gravitational forces, which is known as gravitational kinematics. In CFO, the entities are a set of probes that fly in the search space under the gravitational force. The probe has two properties (position and acceleration) and is updated following the laws of motion of objects in a gravitational field. The probe position is determined in 3-dimensional Cartesian coordinates. Analogous to Newton's law of gravity, the CFO is a deterministic algorithm (that is, its computation includes no randomness). Therefore, the initial positions of the probes are deterministically selected from a uniform distribution.

The probes generate solutions through continuous movements, and are attracted and accelerate towards each other according to the difference between their masses; the bigger the mass, the greater the attraction. The mass of a probe reflects the fitness of its solution, corresponding to the value of the objective function to be optimised. In CFO, Formato (2007) used the differences between the masses instead of the actual masses to avoid extreme pull to closest probes or premature convergence. If the probe flies outside the boundaries, it is relocated to the midpoint between its initial position and the minimum or maximum value of the search space. Figure 2-4 shows the pseudocode of the CFO based on our interpretation of the algorithm (Formato, 2007).

```
PROCEDURE: Central Force Optimisation
Generate a set of probes
Compute initial probe positions (uniformly distribution)
Calculate the initial accelerations for each probe
Evaluate the fitness of each probe
REPEAT UNTIL: stopping criterion is met
      Update the position of each probe
      IF (the new position of the probe outside the range)
            Relocate the probe to a new position
      END IF
      Evaluate the fitness of each probe, and update its mass
      Update the acceleration of each probe
END REPEAT
RETURN: the position of the best probe
```

Figure 2-4. Pseudocode of the central force optimisation algorithm

The CFO algorithm has been mostly applied to continuous domain problems. Some of these applications are listed in Table 2-2.

Table 2-2. Central force optimisation applications

Application	References
COP	(Formato, 2008); (Formato, 2009); (Asi & Dib, 2010); (Formato, 2011); (J. Liu & Wang, 2014); (Yong Liu & Tian, 2015); (J.-S. Wang & Song, 2015); (J.-S. Wang & Song, 2015)

The main drawback of the CFO is its deterministic nature. In some problems, this algorithm generates the same solution at every iteration and risks becoming trapped in local optimal solutions. Furthermore, being deterministic, its performance depends on the quality of the initial value of the probes. Also, as the movements depend only on the masses and not on the distances between the probes, the algorithm may converge prematurely.

2.2.1.3 Gravitational Search Algorithm

The gravitational search algorithm (GSA), proposed by Rashedi et al. (2009), is based on the principle of moving objects and Newton's law of gravity. According to Newton's law, particles in the universe are attracted by a gravitational force whose magnitude is directly proportional to the masses of the particles and inversely proportional to the squared distance between the particles. Therefore, the entities in the GSA are represented as objects with position and mass properties. The position and mass of an object represent a possible solution to the problem and the quality of that solution, respectively, and are updated according to the objective function. Objects move continuously, and their new positions are influenced by the positions of other objects. The magnitude of the gravitational force is biased toward objects with high mass. The position of the most massive object is the position of the global best solution. Therefore, the objects collaborate by direct communication through their gravitational forces.

To control the exploration, the massive objects move more slowly through the search space than lighter objects. Moreover, to balance the exploration and exploitation processes, GSA controls the number of objects that can influence other objects by specifying a number *K*, which restricts the number of large influencers (the *K-best* objects). *K* is a parameter initialised based on the user preference. This number is gradually reduced as the iterations proceed. Only the *K-best* objects can attract the other objects. This technique is designed to gradually decrease the exploration while increasing the exploitation, thereby enhancing the performance of the algorithm. However, if not well controlled, it may affect the exploration and exploitation ability of the algorithm. Figure 2-5 shows a simple pseudocode of the GSA based on our interpretation of the algorithm (Rashedi et al., 2009).

```
PROCEDURE: Gravitational Search Algorithm
Initialise parameters: K-best, gravitational-constant, inertial
masses of agents (M), velocity(V), position (X)
Generate an initial population, with random positions
REPEAT UNTIL: stopping criterion is met
      FOR each object do:
            Evaluate the fitness of each object
            Calculate mass and acceleration for each object
            Update position, acceleration, and velocity
            Update worst(t) and best(t) fitness of the objects
            Update global-solution
      END FOR
END REPEAT
RETURN a solution
```

Figure 2-5. Pseudocode of the gravitational search algorithm

Originally, the GSA was intended for continuous-domain problems. Later, a binary gravitational search algorithm (Rashedi, Nezamabadi-pour, & Saryazdi, 2010) was adapted for problems with discrete variables. One drawback of GSA is the complex operations, which increase the computational time of solving a problem (Beheshti & Shamsuddin, 2013). In addition, the GSA does not depend directly on the global-best location found so far; instead, the entities are affected by the current top-best solution for movement. Therefore, the location of the global solution might be lost through the execution, limiting the exploitation capability of the algorithm (Yin, Guo, Liang, & Yue, 2017). Some problems solved by GSA are summarised in Table 2-3.

Table 2-3. Applications of the gravitational search algorithm

Application	**References**
COP	(Yazdani, Nezamabadi-pour, & Kamyab, 2014); (Mirjalili & Hashim, 2010); (Gao, Vairappan, Wang, Cao, & Tang, 2014); (Yin et al., 2017); (A. Zhang et al., 2018)
TSP	(Dowlatshahi, Nezamabadi-pour, & Mashinchi, 2014); (Ibrahim, Ibrahim, Ahmad, & Yusof, 2015b)

Wang and Song (2015) analysed the effect of the parameters on the GSA performance in some numeric test functions. They found that the accuracy and convergence rate of the algorithm are extremely sensitive to the fine-tuning of the parameters.

Dowlatshahi et al. (2014) solved the Euclidean TSP (i.e. the nodes correspond to points in a two-dimensional space and the cost is the Euclidean distance between the nodes) using a discrete GSA (DGSA) that replaces the classical attractive movement of entities with a technique called *path relinking*. The initial values of the entities are generated by a randomised greedy method, with each entity generating a solution to the problem. The solution and other information are stored in two arrays called *tour* and *inverse*. This representation is considered to facilitate the manipulation of the tours by two types of movement operations (dependent and independent). Dependent movement is achieved by a swap operator, while independent movement is based on a modified 2-Opt operation. The DGSA was tested on a number of differently sized benchmark instances, and generated higher-quality solutions than other algorithms in most of these instances.

To rectify the poor exploitation ability of GSA (Yin et al., 2017) combined GSA with crossover. The crossover was expected to improve the quality of the current solution and enhance the exploitation capability of the algorithm. Initially, the entities are randomly distributed in the search space to diversify the solutions. The algorithm was tested on a number of high-dimensional benchmark test functions. The results confirmed the higher performance of the modified GSA than the original GSA.

Ibrahim et al. (2015a) proposed a discreet multi-state GSA and tested it on the TSP. In this algorithm, each entity maintains a state vector that stores its possible solution to the problem. Transitions between two states are made by a new mechanism, and one state is selected from all candidate states in each iteration. This process repeats until the algorithm converges towards the best-known solution. When tested on a number of benchmarks instances, the solution quality was improved over that of the original algorithm.

2.2.1.4 Water-based Algorithms (WBAs)

A number of optimisation algorithms are based on certain factors or processes related to the activities and natural movements of water. The WBAs have demonstrated their ability in solving different types of optimisation problems. Although these algorithms share a common principle—that water always flows towards lowlands—they differ in their mathematical models and stages. The following subsections review all of the known WBAs in order of their published date.

2.2.1.4.1 River Formation Dynamics

By eroding and depositing sediments, flowing water plays a crucial role in river formation. These processes have indirectly changed the landscape of the ground surface and helped the formation of new riverbeds. Based on the mechanism of river formation, Rabanal, Rodríguez, and Rubio in (2007) proposed an optimisation algorithm called river formation dynamics (RFD). RFD models entities as droplets that flow on the ground, preferentially towards low altitudes. The problem is represented as a graph with an altitude assigned to each node. The node altitude decreases when eroded by the drops or increases when sediment is deposited. The probability of choosing a node is influenced by the altitude values of the surrounding nodes and the distances between the nodes.

The algorithm starts with a number of drops and a flat graph representing the solution space. All nodes have the same initial altitude. From their start nodes, the drops traverse the graph towards a destination node. In each iteration, the nodes undergo various erosion processes, followed by a deposition process. The deposition is applied on all nodes to increase the altitude range among the nodes and to allow exploration of different paths.

Analogously to SA, the drops in an RFD algorithm can choose a high-altitude node with a certain probability. This feature enables escape from local optima by the exploration of various paths and improves the search quality. Another feature is the changing number of drops. The drops can gather when meeting at the same node. A single drop with higher size is generated as result of gathering multiple drops. The drops start with size *N*, and may vary as they move. When a drop splits, it generates multiple drops with smaller size. The advantage of gathering drops is to reduce the computational effort and improves the efficiency. Figure 2-6 shows a simple pseudocode of the RFD based on our interpretation of the algorithm (Rabanal et al., 2007).

```
PROCEDURE: River Formation Dynamics
Generate a set of drops
Initialise the drops and nodes altitude
REPEAT UNTIL: stopping criterion is met
      Move drops
      Erode paths
      Deposit sediments
      Evaluate the drops
END REPEAT
RETURN the best solution
```

Figure 2-6. Pseudocode of river formation dynamics

The RFD algorithm has been tested on a number of *NP*-complete problems, such as TSP and dynamic TSP (Rabanal, Rodríguez, & Rubio, 2008b), Minimum Spanning Tree (Rabanal, Rodríguez, & Rubio, 2008a), robot navigation (Dąbkowski, Redlarski, & Pałkowski, 2013), and wireless routing (Sharma & Gupta, 2016).

In solving the TSP, Rabanal et al. (2009) represented the problem as a landscape with all cities initially at the same altitude. They slightly adjusted the representation by cloning the start-point city, allowing water to return to that city. Water movement is affected by the altitude differences among the cities and the path distances. The solutions (tours) are represented as sequences of cities sorted by decreasing altitude. To prevent the water drops from immediately eroding the landscape after each movement, the algorithm is modified to erode all cities when the drop reaches the destination city. This modification prevents quick reinforcement and avoids premature convergence. When tested on a number of TSP instances, the algorithm obtained a better solution than ant colony optimisation, but required a longer computational time. The authors concluded that the RFD algorithm is a good choice if the solution quality is more important than the computational time.

2.2.1.4.2 Water Flow-like Algorithm

The water flow-like algorithm (WFA), developed by Yang and Wang in (2007), mimics the movement of water from high to lower altitudes. In WFA, the entities (water flows) drift from high-altitude locations to lower-altitude locations under gravitational force. The water-flow direction is affected by the ground topology, which is mapped as a two-dimensional array approximating the solution space of the objective function. While travelling downhill, the water may split into many sub-flows at intersections or on rugged terrain. The number of new sub-flows depends on the momentum and kinetic energy of the water flow; high-momentum flows generate more sub-flows than low-momentum flows. The locations of the new sub-flows are determined by locating the neighbours of the original flow. Moreover, the velocity and mass of the original flow are divided among the new sub-flows. In contrast, many sub-flows will merge when flowing through the same location. Therefore, WFA generates a solution by splitting, moving and merging of the water flows.

Two additional processes (evaporation and precipitation) guide the algorithm towards better solutions and the discovery of new areas. Evaporation removes part of the stagnated water flows, and precipitation generates new water flows at different locations. A flow with sufficient momentum can move from a lower to a higher altitude, widening the exploration area and assisting escape from local optima. Unlike other metaheuristic algorithms, the WFA is self-adaptive as the number of water flows dynamically changes through the execution, mediated by the merging, splitting, evaporation and rainfall operations.

Initially, the algorithm generates one water-flow at a random location, and assigns it a velocity and a mass. As a water-flow flows in different directions, it can split or merge later with other flows after the split. The velocity determines whether the water can move; a stagnant water-flow remains at its current location until it merges with another flow or evaporates. Each water-flow location (solution) at each altitude is evaluated using the objective function. Figure 2-7 shows the pseudocode of the WFA based on our interpretation of the algorithm (F.-C. Yang & Wang, 2007).

```
PROCEDURE: Water Flow-Like Algorithm
Initialise parameters (velocity and mass, water-flow momentum, number
of sub-flows split from a flow)
Generate an initial water-flow in a random position
REPEAT UNTIL: stopping criterion is met
      FOR each flow, if velocity > 0
            Move water-flow
            Check the splitting condition
            Check the merging condition
            Check the water evaporation condition
            IF (more water-flow is required)
                  Precipitation
                  Check the merging condition
            END IF
            Update best-solution
      END FOR
END REPEAT
RETURN: best-solution
```

Figure 2-7. Pseudocode of the water flow-like algorithm

Although merging the water flows reduces the number of redundant searches and minimises the computational cost, the mergers can limit the exploration process (by reducing the number of water-flows) and the number of generated solutions. Another drawback is that the water flows are guided by only one heuristic (altitude), which may lead to premature convergence.

The WFA has been applied to various problems, such as bin packing problems (F.-C. Yang & Wang, 2007), and the multiobjective continuous problem (F.-C. Yang & Ni, 2014). The WFA is also used to solve the TSP (Srour, Othman, & Hamdan, 2014). Initially, a solution to the water-flow is generated using a nearest-neighbour heuristic. In successive iterations, they are moved by insertions and 2-Opt procedures. The evaporation and precipitation operations are unchanged from the original WFA. These processes repeat until the stopping criteria are met. The solution is represented as a sequence of nodes to be visited and stored in a one-dimensional array with length equalling the number of cities.

Zainudin et al. (2015) solved fourteen benchmark CVRP instances by the WFA. Each water flow in the WFA–CVRP generates a candidate solution to the problem. The initial solution is generated by randomly grouping customers into routes without violating the problem constraints. Zainudin et al. augmented the WFA algorithm with methods such as swap, move, and 2-Opt, which improve the solution quality. The splitting and moving operations shift the locations of the water flows, and the swap operation mimics precipitation, enhancing the locations of the weak water flows. The problem is represented as a one-dimensional array whose length equals the number of customers plus one for the depot. The CVRP–WFA found the best-known solutions in 9 instances, and well competed with other algorithms in the remaining instances. The authors also found a relationship between the iteration number and the solution quality.

2.2.1.4.3 Intelligent Water Drops algorithm

The intelligent water drops (IWD) algorithm is a population-based optimisation algorithm proposed by Hosseini in (Shah-Hosseini, 2007). The IWD was inspired by the natural flow behaviour of water in a river, and by what happens in the journey from water drops to the riverbed (Shah-Hosseini, 2009). The IWD algorithm will be described in careful detail because part of this thesis involves modifying and extending this algorithm.

In nature, the movement of water drops is governed by Earth's gravity, the ground topography, and obstacles in the water flow (Vinogradov, 2009). Scattered water chooses the path of least resistance (i.e., the path with the least soil deposition and fewest obstacles). In the absence of obstacles, water drops, under gravitational force, take the shortest path (a straight path) towards the oceans. During this movement, the water carries and deposits an amount of soil along its path. Figure 2-8 depicts a water drop carrying soil from one point to another along the riverbed. The amount of carried soil is reflected in the size of the water drop. As seen in the figure, the water drop enlarges as it travels downstream, while the amount of soil on the riverbed decreases. The amount of soil carried by a water drop equals the amount of soil taken from the riverbed. Therefore, the water drops accumulate soil from the riverbed as they traverse the waterway.

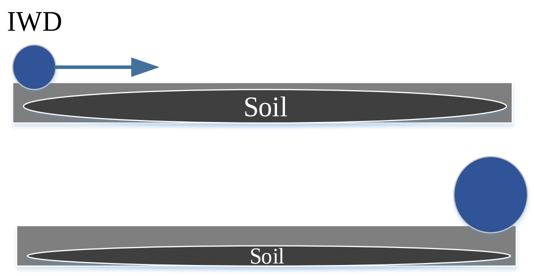


Figure 2-8. A water drop carrying soil while moving downstream (Shah- Hosseini, 2008, P.196)

The maximum amount of soil that a water drop can carry depends on the droplet's velocity; the higher the velocity, the more powerful the drop, and the greater the amount of soil carried. Figure 2-9 depicts two water drops moving from one point to another. The faster water drop (indicated by the large arrow) can carry more soil than the slower one.

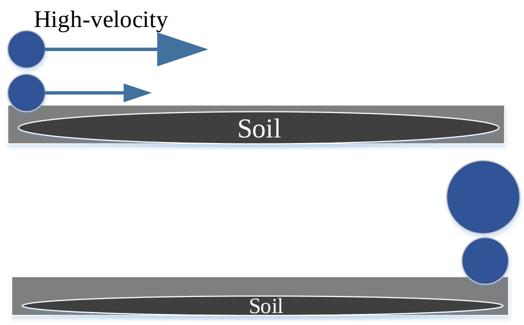


Figure 2-9. Relationship between water drop velocity and amount of soil carried by the water drop (Shah-Hosseini, 2008, P.196)

Conversely, the velocity of a water drop depends on the amount of soil between two points on the path. In regions of less and more soil the velocity of the water drop increases and decreases respectively. The amount of soil added to the water drops also depends on the time in which the water drop moves from its current location to the next one; droplets spending more time between the two locations carry less soil. The time between locations is calculated by the linear motion law (time = distance/velocity). Therefore, the time increases with decreasing velocity or increasing distance between the two locations.

To simulate this natural flowing process, the generated IWDs are assigned a movement velocity (*IWD_Velocity*) and an amount of carried soil (*IWD_Soil*) (Shah-Hosseini, 2007). The values of these two properties are changed and updated as each water drop moves from one point to another. Initially, each IWD travels at a specific velocity and carries no soil. During its movement, it removes some soil from the paths and increases its velocity. Additionally, the IWD algorithm has two categories of parameters; static and dynamic parameters. The values of the static parameters remain constant throughout the algorithm execution, whereas those of the dynamic parameters continually change. Some of these dynamic parameters are re-initialised after each iteration.

When applying the IWD algorithm, the problem to be solved should be presented as a graph with a set of nodes and a set of edges. The amount of soil along each edge is initialised to a specific value. The algorithm then generates a number of water drops (usually equal to the number of nodes), and randomly distributes them on the nodes (imitating a precipitation process). Each IWD continually moves from one node to another without violating the problem constraints (imitating a runoff process). This process continues until each IWD has visited all nodes and completed its solutions, thus completing the current iteration. The algorithm stops when the maximum number of iterations is reached, or when some other termination criteria are met. The IWD equations stated later in this section are taken from Shah-Hosseini (Shah-Hosseini, 2007) without modification.

Initially, a water drop moving to another node must choose the node with the highest probability among the unvisited nodes. The probability of visiting a node depends on the existing soil along the path leading to that node, and is given by Eq. (2.1).

$$P^{IWD}{}_{i}(j) = \frac{f\big(soil(i,j)\big)}{\sum_{k \notin vc(iwd)} f\,\big(soil\;(i,k)\big)} \tag{2.1}$$

where (*i*) is the current node and (*j*) is the next candidate node to be visited by the current IWD, and

$$f\,\big(soil(i,j)\big) = \frac{1}{\varepsilon s + g\big(soil(i,j)\big)} \tag{2.2}$$

Here εs is a constant equal to a very small value (i.e. 0.001), to prevent possible division by zero. The g (soil(i, j) is a function used in the computation, as follows:

$$g\big(soil(i,j)\big) = \begin{cases} Soil(i,j) & if \min\limits_{l \notin vc(iwd)} (soil(i,l)) \geq 0 \\ Soil\;(i,j) - \min\limits_{l \notin vc(iwd)} \big(soil(i,l)\big) & otherwise \end{cases} \tag{2.3}$$

The *min* function returns the minimum value among its arguments. Once visited by an IWD, a node is stored in the *visited list* "*vc*" of that IWD, which prevents future visits by the same IWD. While moving from node *i* to node *j*, an IWD updates its velocity $vel^{IWD}(t+1)$ according to the amount of soil existing on that path. The velocity increase is larger on paths with less soil. Mathematically, this behaviour is represented as follows:

$$Vel^{IWD}(t+1) = Vel^{IWD}(t) + \frac{av}{bv + cv * \big(soil(i,j)\big)}, \tag{2.4}$$

where the parameters *av*, *bv*, and *cv* control the velocity update process. The value of these parameters can be adjusted experimentally. In addition, each IWD updates the amount of soil along the edge between nodes *i* and *j* by the following equation:

$$\triangle\, soil(i,j) = \frac{as}{bs + cs * time\big(i,j;Vel^{IWD}(t+1)\big)}, \tag{2.5}$$

where the parameters *as*, *bs*, and *cs* influence the rate of change in the amount of soil. The travel time of an IWD between two nodes is calculated as follows:

$$time\big(i,j;Vel^{IWD}(t+1)\big) = \frac{HUD\ (i,j)}{max(\varepsilon v, Vel^{IWD}(t+1))} \tag{2.6}$$

The parameter εv = 0.0001 reverses the sign of a negative velocity. HUD represents a problem-dependent heuristic function. After calculating the changes in the soil, the amount of soil existing on an edge decreases as follows:

$$Soil(i,j) = (1-p)\times soil(i,j) - \ p\times \Delta soil(i,j) \tag{2.7}$$

The coefficient p = 0.9 controls the amount of soil decrease. The amount of removed soil is added to the amount of soil *Soil*^{IWD} carried by the water drop, as follows:

$$Soil^{IWD} = Soil^{IWD} + \ \Delta soil(i,j) \tag{2.8}$$

At the end of each iteration, the solution generated by each IWD is evaluated, and the best local solution is selected and compared with the best global solution found so far. If the local solution (*L_Sol*) is better than any previous solution, the global solution (*G_Sol*) is updated with the new value:

$$G\_Sol = \begin{cases} L\_Sol & if(L\_Sol < \ G\_Sol) \\ G\_Sol & \qquad Otherwise \end{cases} \tag{2.9}$$

Moreover, the best IWD is rewarded by updating the global soil along the edges that belongs to its solution, as follows.

$$Soil(i,j) = (1-p)\times soil(i,j) + \left(p\times \ \frac{2\ \times Soil^{IWD}}{Nc\ (\ Nc-1)}\right) \tag{2.10}$$

where *Nc* is a variable representing the number of nodes. By applying the above steps, the IWD algorithm incrementally constructs the solutions until the maximum number of iterations is reached. Therefore, the IWD algorithm can be considered as a population-based constructive algorithm. Figure 2-10 shows the pseudocode of the IWD algorithm based on our interpretation of the algorithm (Shah-Hosseini, 2007).

```
PROCEDURE: Intelligent Water Drops
Initialise static and dynamic parameters (initial soil, initial soil,
other control parameters)
Create and distribute a set of IWDs
REPEAT UNTIL: stopping criterion is met
      WHILE all IWD not completed their solution
            FOR each IWD
                  Choose the next node
                  Update IWD velocity
                  Update soil amount on the used edge
                  Update the soil carried by the IWD
            END FOR
            Evaluate the solution of each IWD
            IF (a new solution is better than the current best-solution)
                  Update the best-solution with the new solution
            END IF
      END WHILE
END REPEAT
RETURN: best-solution
```

Figure 2-10. Pseudocode of the intelligent water drops algorithm

The IWD algorithm proposed by Shah-Hosseini (Shah-Hosseini, 2007) was tested on the TSP. Experiments confirmed that the IWD algorithm can solve this problem, and obtains good results in some TSP instances. In a later publication, (Shah-Hosseini, 2008) modified their IWD algorithm and incorporated a new heuristic for solving the multiple knapsack problem (MKP). In the knapsack problem, a set of items must be fitted into a limited knapsack capacity while maximising the profit. The modified IWD algorithm found the best-known solutions in some of the tested MKP instances.

This algorithm attracted the attention of other researchers, who improved it and applied it to other problems. For example, Duan et al. (2008) solved the air-robot path planning problem by an IWD algorithm. This problem seeks the path between two points in the airspace under constraints and the requirements of reasonable performance. Experiments confirmed the feasibility of the proposed IWD algorithm in solving the target problem.

Msallam & Hamdan (2011) presented an improved adaptive IWD algorithm. The adaptive part changes the initial value of the soil and the velocity of the water drops during the execution. The change is made when the quality of the results no longer improves, or after a certain number of iterations. Moreover, the initial-value change was based on the obtained fitness value of each water drop. Msallam and Hamdan used some of the modifications proposed in Shah-Hosseini (2009); that is, the amount of soil along each edge is reinitialised to a common value after a specified number of iteration, except for the edges that belong to the best solution give less soil. These modifications diversify the exploration of the solution space and help the algorithm to escape from local optima. When tested on the TSP, the new adaptive IWD algorithm outperformed the original IWD.

The IWD has also been applied to continuous optimisation problems. Shah-Hosseini (2012) utilised binary variables in the IWD and created a directed graph of (*M*×*P*) nodes, where *M* denotes the number of variables, and *P* identifies the precision (number of bits) of each variable, assumed as 32. The nodes in the graph were connected by 2× (*M*×*P*) directed edges, each taking a value of zero or one (see Figure 2-11). One advantage of this representation is its intuitive consistency with natural water flow in a specific direction.

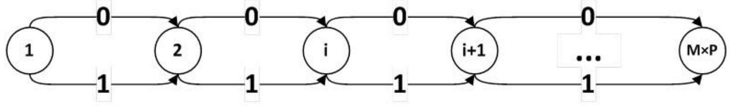


Figure 2-11. Representation of a continuous problem in the IWD algorithm

The IWD algorithm was also augmented with a mutation-based local search to enhance its solution quality. In this process, an edge is selected at random from a solution and replaced by another edge, which is kept if it improves the fitness value. This process is repeated 100 times for each solution. After mutating all of the generated solutions, the soil is updated along the edges belonging to the best solution. However, the variables' values must be converted from binary to real numbers to be evaluated. Furthermore, once the mutation (but not the crossover) is introduced into IWD, whether IWD offers advantages over standard evolutionary algorithms using simpler notations for continuous numbers is doubtful.

Booyavi et al. (2014) solved the CVRP by an improved IWD algorithm that embeds features such as the saving value as another heuristic, and generates a candidate list. The convergence of the algorithm is improved by imposing a min–max rule (lower and upper bound) on the soil updating formula. All IWDs start from the depot and select their next customers from the candidate list. The solutions are sequentially built; that is, one IWD starts when the prior IWD has finished. Furthermore, the obtained solutions are enhanced using a 2-Opt operation. At the end of each iteration, the elite IWDs are favoured by reducing the amount of soil on the edges belonging to their solutions. When tested on fourteen benchmark instances, the improved algorithm well controlled the exploration and exploitation processes and effectively solved the instances.

These studies demonstrate the ability of the IWD algorithm to solve different optimisation problems. In addition, the results of the IWD algorithm are very promising compared with those of other metaheuristics algorithms. However, the IWD algorithm does not exploit the properties of full water cycling in nature, so its performance was limited for some problems. Analysing the IWD reveals it has some parameters that are not fully exploited. This limits the performance of the algorithm. For instances, the carrying soil by a water drop is not fully exploited, and the velocity of a water drop is increased within small ratios. More limitation will be discussed in Section 4 of Chapter 3.

2.2.1.4.4 Water Cycle Algorithm

The water cycle defines the continuous movement of the water through different stages between the sky and the earth. On the ground, water flows downhill in different directions, forming streams and rivers that eventually reach the sea or ocean. Streams are generally smaller than rivers, and can form rivers by joining other small streams. However, streams differ from rivers in many aspects. Streams are shallower water bodies and transfer less water than rivers. Rivers are typically deeper than streams, with wider banks, meaning that rivers can transfer more sediments. In nature, water from different sources evaporates, and later descends to the earth as precipitation.

Part of the water cycle, and the drifting of river and stream waters towards the sea, inspired the water cycle algorithm (WCA) by Eskandar et al. in (2012). The WCA generates a set of raindrops and randomly distributes them in the search space. The locations of these raindrops evolve to minimise the objective function. The best raindrop is selected to represent a sea. A certain number (determined by the user) of the top water-drops are then marked as rivers, and the remainder are marked as streams. The streams flow towards the rivers or directly to the sea. Eventually, all streams and rivers reach the sea location. In this classification hierarchy, the sea, rivers and streams represent the global-best, local-best and other potential solutions, respectively. The evaporation process buffers the algorithm from premature convergence. Evaporation is applied when a river and the sea are separated by less than a specified threshold. The chosen threshold value determines the search capability; a large value encourages exploration and a small value facilitates exploitation near the optimal solution.

Consequent to water evaporation, the precipitation process revives the search with new streams created in different locations. After re-evaluating the new stream locations, the rivers and sea are relocated at the updated best locations. Figure 2-12 presents a simple pseudocode of the WCA based on our interpretation of the algorithm (Eskandar et al., 2012).

```
PROCEDURE: Water Cycle Algorithm
Initialise parameters (number of raindrops, number of rivers, number
of streams, evaporation threshold [dmax])
Generate a set of initial streams in random locations
Evaluate the locations of the streams
Determine the rivers and sea
REPEAT UNTIL: stopping criterion is met
      Flow streams toward rivers
      Flow rivers toward sea
      Evaluate the new location of the streams and the rivers
      IF the new location of the stream is better than one of the rivers
         Exchange positions of river with the stream
      END IF
      IF the new location of the river is better than sea's locations
         Exchange positions of sea with the river
      END IF
      IF (Evaporation condition is satisfied)
         Generate new streams
      END IF
      Reduce the value of dmax
END REPEAT
RETURN: Location of the sea (best-solution)
```
Figure 2-12. Pseudocode of the water cycle algorithm

WCA constructs a solution using a population of streams, where each stream is a candidate solution. It initially generates random solutions by distributing the streams at different positions throughout the problem's dimensions. The best stream is labelled as the sea and a specific number of streams are considered as rivers. The streams are guided towards the rivers and sea. The sea position is the global-best solution, whereas the river locations are local-best solutions.

In each iteration, if a stream is sufficiently fit, the streams position can be swapped with any of the river positions or (counter-intuitively) the sea position. Specifically, if a stream solution is better than that of a river, that stream becomes a river, and the corresponding river becomes a stream. The same process occurs between rivers and sea. The rivers/streams partially evaporate when they near the sea (i.e., when their separation from the sea is close to zero). Following evaporation, rainfall generates new streams at different positions. These evaporation–precipitation processes help the algorithm to escape from local optima.

The computational effort, accuracy and efficiency of the WCA has been evaluated on constrained and unconstrained continuous problems (Eskandar et al., 2012; Sadollah, Eskandar, Bahreininejad, & Kim, 2015b). The WCA has also been applied to multi-objective continuous problems (Sadollah, Eskandar, Bahreininejad, & Kim, 2015a).

An improved version of WCA called dual-system (DSWCA) has been presented for solving constrained engineering optimization problems (Luo, Wen, Qiao, & Zhou, 2016). The DSWCA improvements involved adding two cycles (outer and inner). The outer cycle, in accordance with the WCA, is designed to perform exploration. The inner cycle is based on the ocean cycle and designed as an exploitation process. A process mimicking the confluence of rivers was included to improve convergence speed. The improvements aim to help the original WCA avoid falling into local optimal solutions. The DSWCA provided better results than WCA. In a further extension of WCA, Heidari et al. (2017) proposed a chaotic WCA (CWCA) that incorporates thirteen chaotic patterns with WCA movement process to improve WCA's performance and deal with the premature convergence problem. The experimental results showed improvement in controlling premature convergence. However, WCA and its extensions introduce many additional features many of which deviate from the essence of the inspiration from nature. These "unnatural" additions aim to improve performance and make WCA competitive with non-water based algorithms.

2.2.1.4.5 Water Flow Algorithm

In his PhD thesis, Tran (2011) proposed the water flow algorithm (WFA) inspired by the natural flow of water from high areas to lower areas. Tran's algorithm simulates the meteorological part of the hydrological cycle, alongside the soil erosion process (Tran, 2011). The water cycle and erosion processes are intended to balance the exploration and exploitation capabilities of the WFA.

In nature, the erosion rate depends on the amount of precipitation, type of soil, falling force of the water at that location, the velocity of the flowing water, and other factors. At each iteration, the WFA creates a cloud that generates a random number of drops of water (DOWs). The DOWs are randomly distributed at different locations on the ground, and flow through the erosion process. The longitudes and latitudes (i.e. location coordinates) of the DOWs on the ground constitute the possible solutions. The altitude of a DOW represents its fitness, which is calculated based on the corresponding objective function of the problem. For example, in a minimisation problem, a DOW with a smaller altitude has a higher-quality solution than a DOW at higher altitude. The gravitational force that moves the DOWs to different locations is represented by a heuristic function, and the movement occurs by a hill-sliding search algorithm. A DOW can move to a different location only when it meets the erosion condition; that is, when its solution quality has not improved after a specified number of iterations. In essence, erosion prevents the DOW from trapping in local optimal solutions, and facilitates the finding of the global solution. Precipitation generates a number of new drops of water at the locations of the clouds.

The WFA utilises three types of memory, namely, a position list (*P*-list), an un-eroded position list (*UE*-list), and an eroded position list (*E*-list). The *P*-list keeps track of the best-solution locations found so far. The *UE*-list stores the locations of DOWs that have not yet met the erosion condition. The *E*-list (tabu list) stores the previous locations of the eroded DOWs that have been eroded, preventing the algorithm from generating new DOWs at these locations. This is intended to reduce the computational time.

In WFA, the erosion capability depends only on the amount of precipitation and its falling force. The amount of precipitation equals the number of new DOWs that have been generated. Erosion can occur when the amount of precipitation reaches a fixed threshold (*MinEro*). The falling force is associated with the objective function at that location, and is also affected by the distance between the cloud and a DOW on the ground. For instance, the two clouds *C*1 and *C*2 in Figure 2-13 have the same altitude, but the falling force is larger at the lower location *Y* than at the higher location *X*. Therefore, the chance of erosion is higher at lower locations than at higher locations. This process helps the algorithm to intensify the search near the local-best solutions.

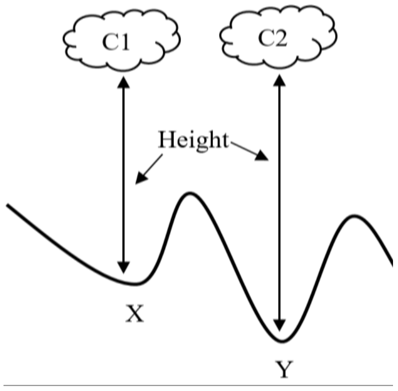


Figure 2-13. Relationship between altitude and erosion capability in the WFA

Figure 2-14 shows a simple pseudocode of the WFA based on our interpretation of the algorithm (Tran, 2011).

```
PROCEDURE: Water Flow Algorithm
Initialise parameters (maximum number of clouds (MaxCloud), maximum
number of DOWs for each cloud (MaxPop), minimum number of DOWs to start
erosion process (MinEro), maximum iterations number without improvement
((MaxUIE)
REPEAT UNTIL: stopping criterion is met, or MaxCloud is reached
      Create a new cloud
      Randomly generate a set of DOWs less than MaxPop, with locations
      not in E-list
      FOR each DOW
            Apply a local search algorithm to move the DOW to a new
            location
      END FOR
      Evaluate the location of each DOW
      Evaluate the soil hardness
      Update P-list and UE-list
      FOR each location in UE-list
            IF (erosion condition is met)
                  //Apply erosion process based on erosion capability
                  WHILE new location is found, or reach MaxUIE
                        Apply steepest descent hill-sliding algorithm
                        IF (the new better solution is found)
                              Update UE-list, P-list, and E-list;
                        END IF
                  END WHILE
            END IF
      END FOR
END REPEAT
RETURN: best-solution (best location from P-list)
```
Figure 2-14. Pseudocode of the water flow algorithm

The WFA algorithm was tested on three problems: the permutation flow shop scheduling problem, the quadratic assignment problem, and the CVRP (Tran, 2011). The algorithm delivered promising results. However, Tran's WFA focuses on erosion as a metaphor and other phases of the natural cycle, such as evaporation and condensation, were neglected. These omitted stages are considered important in the real water cycle and significantly influence the generation of water drops.

To solve the CVRP, Tran modified his original algorithm into the two-level WFA (2LWFA) algorithm, and linked it with LINGO[2] software. In the first level, LINGO constructs a number of initial solutions for the problem. The solutions are generated by solving the mathematical CVRP model based on the integrity constraints and the decision variables. Next, the obtained solutions are adjusted to feasible values by two proposed procedures, *check-and-fit* and *perturbation scheme*. The perturbation scheme corrects infeasible solutions by a cross-exchange operation. In the second level, the modified WFA finds the global optimal solution among all feasible solutions. The erosion process is implemented by a variable *K*-Opt neighbourhood search (*K* = 3 or 4), while the 2LWFA exploration phase is performed by the 2-

[2] LINGO is a comprehensive tool designed to help users to build and solve linear optimisation models quickly, easily, and efficiently (see: http://www.lindo.com/)

Opt procedure. The performance of the modified algorithm was tested on a small benchmark CVRP dataset (with 10 instances). The experimental results showed that relative to the original WFA, the 2LWFA improved the quality of the initial solutions generated during the first level by 20.9% on average (Tran, 2011).

2.2.1.4.6 Water Wave Optimisation

In seas and oceans, water movement creates waves that propagate towards the shore. These waves are powered by geophysical mass flow energy and wind, and undergo refraction and breaking based on the water depth and presence of obstacles. Furthermore, waves have different wavelengths and travel at various speeds. The wavelengths depend mainly on the water depth and the current speed of the waves; the longer the wavelength, the higher the speed. When a wave moves from deep to shallow water, its height increases and its wavelength decreases. In addition, deep-water waves will more likely disperse than waves in shallow water. When a wave reaches the shore, it breaks and loses its energy.

The water wave optimisation (WWO) algorithm, recently proposed by Yu-Jun Zheng in (2015), is based on the motions of particles in water waves. In WWO, wave sets are considered as entities subjected to three operations: propagation, refraction, and breaking. The solution space of the problem is represented as a multi-dimensional space analogous to the seabed topography. The quality of the wave solution (its location) is evaluated using the inverse-depth of the seabed, being worst (lowest fitness) at the deepest part of the seabed, and high in shallow waters. Each wave has two properties, a height and a wavelength. A good wave is characterised by a large height (high energy) and short wavelength, whereas a weak wave has low-amplitude with a long wavelength.

During the execution process, each wave changes its location or creates a new wave through the propagation operation. If a wave moves outside of the spatial dimensions, it is reset at a random location within the range. If the new location improves the fitness of the wave, the wave height is set to the maximum height. Otherwise, the original height wave is decreased by one. This technique allows the high-fitness waves to extend the exploitation, while the low-fitness waves explore the local regions. Furthermore, the wave height determines whether the wave will survive in successive iterations; a wave with zero height needs to be replaced. The height of a wave increases when the wave propagates from a low-fitness location (deep water) into a high-fitness location (shallow water). Otherwise, the wave height will decrease. The wavelength of each wave is updated at the end of each iteration.

In WWO, waves with zero height are refracted, so the directions of the waves are deflected significantly in deep water. Wave breaking is applied on a wave that must improve its solution by performing more local searches around a promising location. Figure 2-15 presents a simple pseudocode of the WWO based on our interpretation of the algorithm (Zheng, 2015).

```
PROCEDURE: Water Wave Optimisation
Initialise parameters (velocity and mass, water-flow momentum, number
of sub-flows split from a flow)
Generate a set of water waves in a random position
REPEAT UNTIL: stopping criterion is met
      FOR each wave
            Propagate a wave to a new location
            Evaluate the new location
            IF the new location is better than the old location
                  IF (the new location better than the best-solution
                  found so far)
                        Apply breaking operation on the wave
                        Update the best-solution with the new value
                  END IF
            ELSE
                  Decrease the wave length by 1
                  IF (wave length equal to zero)
                        Apply refraction on the wave
                  END IF
            END IF
            Update the wavelength of the wave
      END FOR
END REPEAT
RETURN: best-solution
```

Figure 2-15. Pseudocode of water wave optimisation

The WWO algorithm has been tested on some continuous benchmark problems (X. Wu, Zhou, & Lu, 2017; Zheng, 2015), and on a real train scheduling problem in China (Zheng, 2015).

Wu, Liao, and Wang (2015) tested the WWO algorithm on the TSP. In WWO, each wave generates a solution, and its fitness is measured by the total cost of the tour. The wavelength of the wave is inversely proportional to the fitness; the better the solution, the shorter the wavelength. For the TSP, the WWO operators were adapted to handle problems with a discrete domain. The propagation operator mutated the tours with a probability equal to the wavelength. Therefore, a bad solution (i.e., a long-wavelength solution) was more likely to be mutated. The refraction operator enhanced the tour by choosing a random subsequence of cities from the best solution found so far, and replacing it with a subsequence of the original tour. The breaking operation generated a number of new waves by performing swap operations between two previous waves. The algorithm was tested on seven benchmark instances of different sizes. In comparison studies, the WWO algorithm competed well against the genetic algorithm and other optimisation algorithms, and solved the TSP with good results, but with slightly longer computational time than the genetic algorithm.

Wu et al. (2017) applied the WWO for function optimisation and engineering problems. An elite opposition-based strategy is used to enhance the convergence rate and accuracy of the calculation, mainly by improving the diversity of the generated solution. The breaking operation was combined with a local neighbourhood search that further searched the promising areas. The propagation operation was redesigned to better balance the exploration and exploitation processes. When evaluated on benchmark test functions, the algorithm achieved a higher convergence rate with more precise solutions than other algorithms.

2.2.2 Chemistry-based Algorithms

Chemistry-based algorithms simulate chemical laws or the behaviours of atoms or molecules in chemical reactions. However, this category has received little attention from computer science scholars. One possible reason is the difficulty of detecting a relationship or correspondence between a chemical reaction and an implementation of an optimisation algorithm.

2.2.2.1 Chemical Reaction Optimisation Algorithm

The chemical-reaction optimisation (CRO) algorithm was first articulated by Lam and Li (2010). In any chemical reaction, the substance molecules interact to reach the most stable (lowest energy) state or the equilibrium state. Therefore, the CRO simulates the movements of the molecules induced by chemical reactions. The solution to a problem is generated by a set of molecules. Each molecule consists of a number of atoms with defined properties, such as atom type, bond length, bond torsion, and bond angle. The authors of the CRO algorithm grouped these properties and described them as the *molecular structure*. The molecules can be distinguished by their numbers of atoms or their properties. Each molecule in the CRO has potential and kinetic energies, whose values depend on the objective function being solved. The representation of the molecular structure depends on the problem. The molecular structure changes (i.e., a new solution is found) when the new potential energy exceeds the previous one, or when the sum of the new potential and kinetic energy exceeds the old potential energy. Higher-energy molecules have more possibilities to change their structures and generate different solutions than low-energy molecules. The kinetic energy variable helps the algorithm to escape from local optima solutions.

During any chemical reaction, molecules collide with each other or with the container walls. These collisions give rise to four elementary reactions that occur under different conditions, namely, synthesis, on-wall ineffective collisions, intermolecular ineffective collisions, and decomposition (Lam & Li, 2010). Which reaction occurs is determined by a random value between [0, 1] and a user-specified threshold. These elementary reactions are used to manipulate the solutions and help the redistribution of energies among the reacting molecules. Each elementary reaction exerts a different impact on the molecules' energies. Furthermore, the number of molecules can change in any iteration. The number of molecules is unchanged during on-wall or inter-molecular collisions, increased in decomposition, and decreased in a synthesis collision. Thereby, the collision reactions control the exploration-to-exploitation ratio. Specifically, decomposition and synthesis control the exploration, while the on-wall or inter-molecular collisions control the exploitation. Figure 2-16 shows the main pseudocode of the CRO based on our interpretation of the algorithm (Lam & Li, 2010).

```
PROCEDURE: Chemical Reaction Optimisation
Initialise parameters (potential and kinetic energies, PopSize,
Minimum potential energy MinPE, MoleColl, buffer)
Create a number of molecules
Randomly distribute molecules in the solution space
REPEAT UNTIL: stopping criterion is met
     B := Generate random number from [0, 1]
     IF B > MoleColl
        // Uni-molecular collision
         Select one molecule at random
         Apply (On-wall ineffective collision Or Decomposition)
     ELSE
        // Inter-molecular collision
        Select two molecules at random
        Apply (Intermolecular ineffective collision Or Synthesis)
      END IF
      Evaluate the fitness of each molecule
      Update global solution
END REPEAT
RETURN: global-solution
```
Figure 2-16. Pseudocode of the chemical reaction optimisation algorithm

The CRO maintains the energy conservation law, which states that energy cannot be created or destroyed, but only transformed from one form to another (Incropera, DeWitt, Bergman, & Lavine, 2007). In CRO, the energies can transfer between the molecules and the container walls (the energies which are stored in a buffer). Energy conservation in CRO is similar to that in simulating annealing, and the decomposition and synthesis operations are analogous to the GA crossover and mutation operations, respectively (Lam & Li, 2012). The main drawback of the CRO is the high number of required parameters. The CRO algorithm has been evaluated on discrete and continuous domains in various optimisation engineering problems (see Table 2-4).

Table 2-4. Chemical Reaction Optimisation applications

Application	References
COP	(Lam & Li, 2010); (Bechikh, Chaabani, & Said, 2015); (J. J. Q. Yu, Lam, & Li, 2015); (Z. Li, Li, Nguyen, & Chen, 2015); (Dandu, 2013); (Nouioua & Li, 2017)
TSP	(Sun, Wang, Li, & Gao, 2011)
VRP	(Dam, Li, & Fournier-Viger, 2016)

Sun et al. (2011) solved the TSP by combining CRO with Lin–Kernighan local searching. Each molecule represented one possible solution. The functionalities of the four elementary reactions were replaced with local search methods. The on-wall ineffective collision was simulated by 2-Opt and 3-Opt operations. New solutions were generated by the decomposition reaction, which mixed an existing solution with a randomly generated solution using a modified crossover operation. The inter-molecular ineffective collision generated two new solutions by crossover of two previous solutions. The synthesis reaction generated a new solution from two previous solutions by an operation called *distance-preserving crossover*. The proposed algorithm and two existing algorithms were evaluated on various benchmark instances, and the results were reported as relative deviations from the best-known results.

To solve the CVRP, Dam et al. (2016) combined CRO with a unified tabu search (UTS), which searches the local neighbourhood under problem constraints. The solution of each molecule was stored in a vector, and a delimiter separated each sequence of customers according to the vehicle capacity. The inter-molecular ineffective collision operator is a specialised genetic crossover used to minimise the number of vehicles required. Other reaction operators performed different types of crossover, swap, or mutation operations adapted from the genetic algorithm. The UTS corrected any generated solution that violated the problem constraints, and guided the algorithm towards convergence and the best-known solution. The algorithm was evaluated on a number of benchmark instances. The algorithm proved efficient and yielded higher quality solutions than other metaheuristic algorithms.

2.2.3 Biology-based Algorithms (BBA)

This category refers to computational optimisation algorithms inspired by biological systems, biological activities, organism phenomena, and natural evolution (X.-S. Yang, 2005). Recently, there has been an immense and rapid development of algorithms in this category, as computer scientists have relied on biological systems for inspiration. The BBAs can be classified into two major subsets: evolutionary algorithms (EAs) and swarm intelligence-based algorithms (SIAs). Conceptually, however, the two subsets overlap. The following subsections briefly discuss some of these algorithms, and their applications to NP-hard problems.

2.2.3.1 Evolutionary Algorithms

In nature, diverse species emerge and develop through natural selection and inheritance, improving their ability to survive, compete and reproduce. Evolution by natural selection of the fittest individuals is known as Darwinian theory (Darwin, 1968). EA algorithms are based on the processes and mechanisms of natural biological evolution (Eiben & Smith, 2003). EAs solve optimisation problems by a few common mechanisms, typically, by simplified versions of reproduction, mutation, recombination, and selection. These operations play important roles in the modification and propagation of genetic features (i.e., DNA and chromosomes) from parents to offspring in the population.

EAs assign a genotype representation, encoded as fixed-length strings or vectors, to individuals in the population. Later, the genotype must be converted to its actual value (its phenotype) for evaluation (Fogel, 1995). The fitness of an individual is determined by the quality of its solution to the objective function. Each individual in the population strives to survive in successive generations (survival-of- the-fittest principle), and the individuals with higher fitness will survive and reproduce with higher probability than those with lower fitness.

2.2.3.1.1 Genetic Algorithm

The genetic algorithm is a well-known evolutionary algorithm introduced by Holland in 1975. This population-based algorithm simulates the genetics underlying the natural selection process. The entities in GA are represented as chromosomes (i.e., a set of genes). Traditionally, each chromosome is encoded as a binary string or array (0s and 1s), but in certain problem domains, chromosomes can also be encoded as integers, real numbers, or characters (Mitchell, 1998).

Each chromosome represents a candidate solution and evolves toward the best-known solution. The chromosome can be altered by operations such as selection, mutation, crossover, and recombination. The chromosomes in the initial population contain randomly generated values. By repeatedly applying the genetic operations, the GA guides the chromosomes towards the best-known solution. At the end of each iteration, the obtained solutions are evaluated using an objective function. In every iteration, the population is regenerated or modified to form a *generation* for the next iteration. The algorithm usually stops when a termination condition, such as reaching the maximum number of generations, is met. The GA operations play specific roles in finding the best-known solution, and can be adapted to suit the representation. The selection operation determines which chromosomes among the population will reproduce.

Selection can be based on rank, elitism (fitness), tournament outcome, or roulette wheel selection.

The selected chromosomes are subjected to crossover (exploitation) and mutation (exploration) operations. Crossover simulates the reproduction process, by which two selected parents produce offspring. In this operation, the parents' chromosomes are split at different points, then reconnected after genetic exchange to form new chromosomes. The crossover operation generates one or two children that inherit some features from both parents. It thus behaves as an exploitation mechanism that converges the algorithm towards the best-known solution. Common implementations of crossover are one-point, two-point, or blended (for real-number cases) crossovers (Wong, 2015).

The GA mutation operation resembles the natural random changes in organisms' genes. The mutation operation selects one chromosome and flips or alters some of its genes. This operation helps the algorithm to escape from local optima, and can be viewed as a randomization or exploration mechanism. Figure 2-17 shows the pseudocode of the GA (Holland, 1992).

```
PROCEDURE: Genetic Algorithm
Initialise parameters (population size, chromosome size, max generation
crossover type, crossover rate, mutation type, mutation rate, selection
strategy)
Create a population
Randomly generate the values for each chromosome
Evaluate the population
generation := 1
REPEAT UNTIL: stopping criterion is met
      Select individuals for reproduction
      Apply crossover and mutation
      Evaluate the fitness of each individual
      Replace the old population with the new generation
      generation := generation +1
END REPEAT
RETURN: global-solution
```
Figure 2-17. Pseudocode of genetic algorithm

The GA has solved countless complex and high-dimensional problems of various types in academia and industry, and provides high-quality solutions with excellent performance. Like other algorithms, GA has some weaknesses. First, its performance is sensitive to the initial parameter values. This sensitivity was clarified by Maaranen et al. (2007), who investigated the importance and effect of the initial population values on the quality of the final solution. Additionally, parameter tuning significantly influences the search capability, and proper tuning is time-consuming. Some problems are not easily represented by the GA. Finally, GA is computationally expensive (Hassan, Cohanim, de Weck, & Venter, 2005).

Chelouah and Siarry (2000) also considered the choice of the initial population values, and modified the algorithm to intensify the search in the most promising area of the solution space. The main problem with standard GAs for optimisation continues to be the lack of “long-termism”; that is, the fitness for exploitation may need to be secondary to ensure adequate exploration of alternative solutions in the solution space.

Since its inception, GA has been extensively applied to problems in continuous or discrete domains. GA applications to continuous multimodal functions are reviewed in Casas (2015) and Digalakis and Margaritis (2002).

GA has also been applied to TSPs in various configurations (Braun, 1991; Grefenstette, Gopal, Rosmaita, & Van Gucht, 1985). Larranaga, Kuijpers, Murga, Inza, and Dizdarevic (Larranaga, Kuijpers, Murga, Inza, & Dizdarevic, 1999) reviewed the different representations and operators of GAs in TSP applications. Other papers have surveyed the application of different GA versions to the TSP (Moorthy, 2012; Potvin, 1996; Rao & Hedge, 2015; Vaishnav, Choudhary, & Jain, 2017).

Takahashi (2005) solved the TSP using a modified GA with adaptable crossover operators. In this algorithm, the current crossover operator can be changed by another type of crossover operator at any time of the execution. The change is applied when the fitness of the individuals stabilises. The crossover operation aims to minimise the total cost of the tours. In the early stages, the algorithm selects the city closest to the current city. The change of crossover operators is best applied between the first and 83rd iteration. On data of 200 cities, the algorithm improved the quality of the obtained solutions over other tested algorithms.

Other researchers have used the GA algorithm to solve the VRP and its variants. Baker and Ayechew (2003) tried different GA techniques in the VRP, and obtained high-quality solutions while accelerating the convergence of the algorithm. The techniques included varying the initial population (structuring it using the sweep algorithm or randomising it), and applying different neighbourhood search methods (2-Opt, swapping, λ-interchange). They selected the parents by tournament selection and generated offspring by crossover of two (randomly chosen) points. Finally, the algorithm was tested on 14 instances of the VRP. In a comparison evaluation, the solutions of their algorithm were approximately 0.5% above the best-known results of tabu searching.

Bräysy and Gendreau (Bräysy, 2001) reviewed the application of GA in VRPs with time windows. They then compared the results of these investigations with those of recent metaheuristics algorithms on the same problem. Pure GA was outperformed by other algorithms, but not to any significant extent.

Prins (Prins, 2004) presented a hybrid GA for large VRP instances. He resolved the weakness in the standard crossover operation using a special splitter procedure. The GA was hybridised with a local search procedure as a mutation operator. On benchmark instances with a wide range of customer numbers (50–483), his algorithm outperformed the tabu search in 14 Christofides instances.

Jeon, Leep, and Shim (2007) applied a hybrid GA to a VRP with heterogeneous vehicles and more than one depot. They generated the initial solution by improved heuristic methods. The proposed algorithm enhanced by a gene exchange operation and a flexible mutation rate. They also applied *GENI*-exchange operations and route exchanges. This algorithm outperformed the existing GA on some VRP instances.

Masum, Shahjalal, Faruque, and Sarker (2011) mixed the GA with some heuristics during the crossover process. However, this algorithm required many generations to optimise the results. To alleviate this problem, they added an insertion heuristic to enhance the crossover result, but this process consumed more runtime than the original algorithm.

William et al. (2008) hybridised the GA with the savings method and the nearest heuristic for a multi-depot VRP. They developed two versions of the hybrid genetic algorithm. In the first and second versions, the initial solutions are generated randomly and by applying the savings and nearest- neighbour algorithms, respectively. Both versions employ a swap procedure to improve the solution quality. The two versions were tested and compared on instances of different sizes. The second version outperformed the first version.

Hanshar and Ombuki-Berman (2007) solved the DVRP using a GA, that represents a candidate solution by a chromosome consisting of positive and negative integer nodes. A positive node denotes that the customer is not assigned to a vehicle, whereas a negative node represents a set of clustered customers that have been committed to a vehicle. The authors assumed only a pickup scenario, which must handle the arrival of new customers over time. An event scheduler is used to generate a sequence of static VRPs and control GA. A new and effective representation and crossover operation were introduced for this problem. A specific number of the population was randomly selected using the tournament technique. The evolution adopted an elitist strategy that passed the top 1% of solutions onto the next generation. Mutation was performed by an inversion operator that reverses the order of the customers between two points.

The performance of the modified GA and two other algorithms (tabu search and ACO, for comparison) were evaluated on a set of benchmarks derived from the static VRP. The proposed GA minimised the travel costs (19 out of 21) more effectively than tabu and ACO.

Elhassania, Jaouad, and Ahmed (2014) applied the GA to a DVRP that must minimise the total travel cost while dealing with the arrival of new customers. Their approach divides the day into several equal periods. In each period, the event manager creates a static VRP and inputs it to the GA, which generates a solution. In terms of minimising the travel cost, the proposed GA competed well with other approaches in the literature.

Using a similar approach to (Montemanni, Gambardella, Rizzoli, & Donati, 2002), AbdAllah, Essam, and Sarker (2017) modified the GA for DVRPs. Their modification assists the algorithm to diversify the generated solutions and escape from local optima. The DVRP considers the changing customers as the system progresses. The algorithm improves the initial-solution generation of the population in each time slice, the selection process, the swap mutation, and the detection of the local optimal condition. Initial solutions for the population are generated by heuristic and random procedures. Selection is achieved by a modified tournament technique that increases the population diversity by sometimes selecting less fit individuals. The solutions are enhanced by applying reverse and swap processes on the chromosomes. An evaluation approach called *weighted fitness* was also presented. In each time slice, the GA solves a sequence of static VRPs produced by the event manager. The improved GA outperformed other algorithms in both time-based and weighted fitness evaluations.

2.2.3.2 Swarm Intelligence-based Algorithms

Swarm intelligence-based algorithms simulate the foraging, migration, reproduction and prey evasion behaviours of organisms. Species such as ants, bees, wasps, spiders, fish, birds, and bacteria achieve their intended roles (complete their work tasks) by forming groups of moving individuals and the coordination of their activities. Groups of these species are considered as decentralised, distributed, adaptive, and self-organized systems (X.-S. Yang, 2014b). The maximum performance of a biological system requires a full collaboration, interaction, and communication among all individuals in the group, and the overall success depends on the collective behaviour and contribution of all individuals. Individuals communicate in different ways, either directly among themselves or indirectly through the environment.

In SIAs, the solution for a problem emerges through interactions among the entities (collective behaviour). These interactions are consequent to task division and conflict avoidance.

2.2.3.2.1 Ant Colony Optimisation Algorithm

Ants are considered to be social insects. Millions of ants live together in a colony, performing different roles either inside or outside the nest. Worker ants play a very important role in foraging for food. Single ants generally accomplish very little, but collaborative groups can complete massive tasks and solve complex problems. Obviously, the success and efficiency of foraging ants depend on their ability to coordinate and organise their activities. Such coordination requires a high degree of communication among the ants. Ants communicate by depositing a chemical material called a *pheromone*, which signals the promising paths to other ants. This process is known as *stigmergy* (Jackson & Ratnieks, 2006). By intensifying their pheromone deposits, ants provide an indirect communication channel that establishes a short path between their nest and the food source. Initially, the ants will random-walk until they find a food source, which is essential for locating different food sources. When the forager ants find a good food source, they lay a pheromone trail on the soil surface as they return to the nest. The quantity of pheromone depends on the continued ability of the food source to attract the other ants, thereby reinforcing the good paths. Following the pheromone trail, other ants find the food source, setting up a positive feedback mechanism that increases the probability of more ants choosing the same path. Ants scent the pheromone using their antennae, and are more attracted towards intense pheromone deposits. The pheromone evaporates slowly, allowing the discovery of new food sources in new regions when the current source is exhausted.

Another fascinating feature of ants is their ability to deal with environmental changes. For example, when an obstacle blocks the current path to the food source, the ants seek an alternative short path to the food (Goss, Aron, Deneubourg, & Pasteels, 1989); see Figure 2-18.

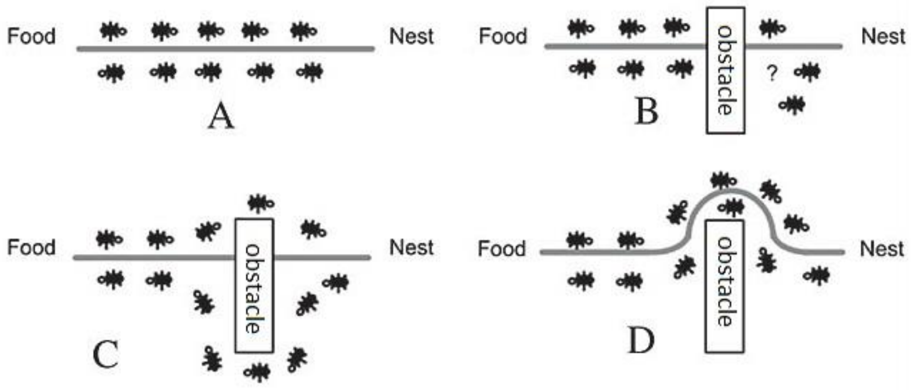


Figure 2-18. Ant behaviour around an obstacle (Perretto & Lopes, 2005, P. 583)

Ant systems provide an excellent inspiration source for optimisation algorithms. The field began in 1991, when Marco Dorigo and his colleagues proposed variants of an ant system (AS) algorithm that simulates the behaviour of real ants, and applied them to the TSP (Dorigo, Maniezzo, & Colorni, 1991b).

In his PhD thesis (1992), Marco Dorigo modified and improved the AS algorithm as the ACO algorithm. The main improvements were changing the transitions rules into complex pseudorandom-proportional rules and allowing the pheromones to evaporate from the ants' trails over time. The ACO algorithm generates a set of artificial ants that explore the solution space by the same foraging principle used by real ants. The ants follow the shortest path between the nest and a food source. As a foraging ant moves from one place to another, it deposits an artificial pheromone on its path. The amount of pheromone deposit depends on the quality and quantity of the solution, and the path distance. Other ants will prefer and follow the path with the highest pheromone ratio. Table 2-5 summarises the chronological development of the ACO algorithm and its variants (Dorigo et al., 2004).

Table 2-5. Chronological order of ACO algorithms

Variant name	Year	Reference
Ant system	1991	(Dorigo, Maniezzo, & Colorni, 1991a), (Dorigo et al., 1991b), (Colorni, Dorigo, & Maniezzo, 1991), (Dorigo, Maniezzo, & Colorni, 1996)
Elitist AS	1992	(Dorigo et al., 1991a), (Dorigo et al., 1991b), (Dorigo, 1992)
Ant-Q	1995	(Dorigo & Gambardella, 1995) (Dorigo & Gambardella, 1996)
Ant colony system	1996	(Gambardella & Dorigo, 1996), (Dorigo & Gambardella, 1997b)
Max-Min AS	1996	(Stützle & Hoos, 1996), (Stutzle & Hoos, 1997), (Stützle & Hoos, 2000)
Ranked-based AS	1997	(Bullnheimer, Hartl, & Strauss, 1997)
ANTS	1999	(Maniezzo, 1999)

Typically, an ACO problem is represented as a graph with a number of nodes and edges. A number of artificial ants are placed either on random nodes or the same node, depending on the problem specification. The ants traverse between nodes in different directions (random walk), depositing an amount of artificial pheromone on the edges. As time elapses, the repeatedly used paths accumulate more pheromone than the less travelled paths. Therefore, each edge is assigned a variable representing the amount of pheromone. The amount of deposited pheromone depends on the length of the edge or the fitness of the solution generated by the ants. In general, the pheromone concentration is higher on shorter paths than on longer paths. When visiting a new node, an ant calculates the probability of all possible nodes and chooses the most probable node. Therefore, the edges with more pheromone have a higher chance of being selected. At the end of each iteration, the pheromones are partially evaporated to prevent the algorithm from falling into local optima. The pheromone amount is decreased more on the long paths than on short paths. Over time, some edges will be repeatedly favoured, marking them as possible best-known solutions. The ants can build the solution in parallel (each ant moves to one node in each iteration) or in series (one ant finds one solution in each iteration). Furthermore, the quality of the generated solutions can be improved by *demon actions*. Figure 2-19 shows the basic pseudocode of the ACO algorithm, adapted from (Dorigo, Birattari, & St, 2006).

```
PROCEDURE: Ant Colony Optimisation
Initialise parameters (Swarm size, initial pheromone, other control
parameters)
Generate a set of ants
FOR each edge
      edge := initial pheromone
END FOR
REPEAT UNTIL: stopping criterion is met
      Place ants at the starting points
      FOR each ant
            Choose the next node based on the pseudo-random-rule
            Update pheromone trail
      END FOR
      Apply Demon actions
      Evaluate the constructed solutions
      Apply pheromone evaporation from all edges
      Apply global pheromone update on edges belonging to the best local
      solution
      Update the global best solution
END REPEAT
RETURN: Global best-solution
END PROCEDURE
```

Figure 2-19. Pseudocode of ant colony optimisation

The foraging behaviour and exploratory patterns of Argentine ants have been tested in a double-bridge experiment (Deneubourg, Aron, Goss, & Pasteels, 1990; Goss et al., 1989). In the experimental setup, real ants were placed in a container, and the nest was separated from the food source by a bridge with two branches. The branches were set equal (i.e., a diamond-shaped bridge) or unequal (i.e., one branch being longer than the other); see Figure 2-20. The ant behaviours were monitored to measure their exploratory ability. In both experiments, the ants were able to find the shortest path. Inspired by these experiments, researchers investigated the convergence of the artificial ants in the ACO algorithm in simulation experiments called *double-bridge*. The results proved that the ACO converges and that different percentages of the ants pass over the two bridges.

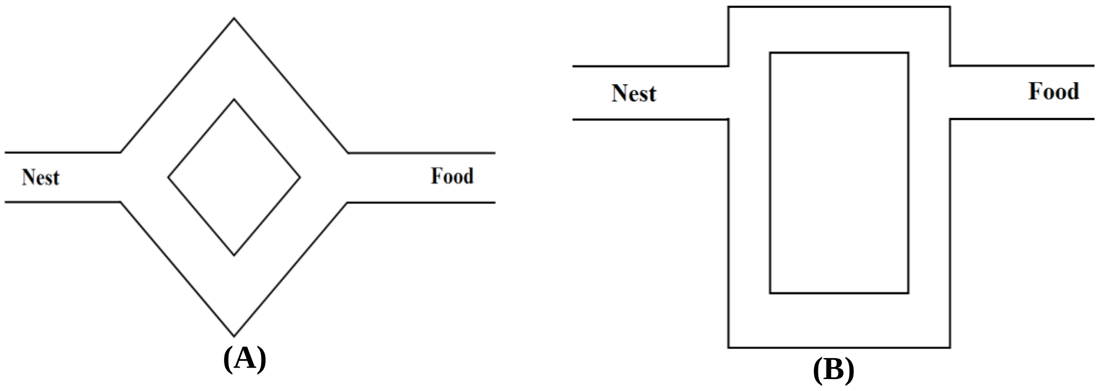


Figure 2-20. Double-bridge experiments

The main feature of the ACO is the emergence of the best solution through the ants' interaction and collaboration. The ACO algorithm is also advantaged by the ease of parallel implementation. Moreover, ACO can be simply combined with heuristics that influence the ants' choice of the next path, thereby improving the overall performance. However, in the simple ACO version, the ants tend to stagnate and fall into local optima, especially in large-size problems. Moreover, the slow convergence rate of ACO increases the time to reach the global optimum solution. The ants in ACO must be guided by a problem-dependent heuristic, especially in the early stages of the execution, because all edges are initially deposited with the same amount of pheromone. Some of these weaknesses have been resolved in different variants or extensions of ACO.

Different optimisation problems have been solved by ACOs with various enhancement techniques. The ACO algorithm has been successfully applied, tested and evaluated on countless problems on discrete or continuous domains. Many COPs have also been solved by ACO. For example, Mathur et al. (2000) divided the function domain into a number of regions representing trial solution spaces for the ants to explore. They then distributed two types of ants (global and local) within these regions. During their exploration, the global and local ants updated the global and local fitness of the regions, respectively. The solution quality was improved by adding mutation and crossover operations.

Socha and Dorigo (2008) extended the ACO algorithm to continuous domain problems. In this version, the discrete probability is replaced by a continuous probability and the next point to visit is selected by a Gaussian probability density function. The pheromone evaporation procedure is modified, and old solutions are removed from the candidate-solution pool and replaced with new and better solutions. Each ant solution is weighted by its own fitness. Various COPs have been solved by ACO (Z. Chen, Zhou, & Luo, 2017; Garai, Debbarman, & Biswas, 2013; Guo & Zhu, 2012; Liao, de Oca, Aydin, Stützle, & Dorigo, 2011; Seçkiner, Eroğlu, Emrullah, & Dereli, 2013; J. Xiao & Li, 2011; X. Zhang, Zhang, & Gao, 2010), or by ACO-inspired approaches that depart from the original ACO structure (Bilchev and Parmee 1995; Dréo and Siarry, 2004; Liu et al. 2014; Monmarché et al. 2000).

Ant colonies have been applied to the TSP (Dorigo & Gambardella, 1997a; Stutzle & Hoos, 1997) on symmetric and asymmetric graphs (Gambardella & Dorigo, 1996). For solving TSPs, Hlaing and Khine (2011) initialised the ant locations by a distribution approach that avoids search stagnation, and places each ant at one city. The ACO is improved by a local optimisation heuristic that chooses the next-closest city, and by an information entropy that adjusts the parameters. When tested on a number of benchmark instances, the improved ACO delivered promising results; especially, the improvements increased the convergence rate over the original ACO.

Among the many problems solved by ACO are the VRP and its variants. The first ant system for VRPs was proposed by (Bullnheimer, Hartl, & Strauss, 1999). They improved the solution quality using some heuristic values such as savings and 2-Opt. They measured the influence of the number of elitist ants on the solution quality, and the initial placement of the ants on the results. The algorithm was tested for different parameter sets, and achieved promising results when competed with other algorithms.

For solving the CVRP, Tan et al. (2012) combined ACO with improved heuristic methods such as 2-Opt, 3-Opt, and swap. These improvement strategies enhanced the performance of ACO, but also increased the runtime of the algorithm.

Khoshbakht and Sedighpour (2011) modified the ACO algorithm by adjusting the evaporation rate equation. This amendment aims to avoid premature convergence while depositing more pheromones on strong solutions. The evaporation rate is based on the problem size and the current iteration number. In a CVRP evaluation, the 3-Opt operation improved the solution quality over the original ACO.

Ezzatneshan (2010) solved the VRP by mixing the ACO algorithm with an exact algorithm. First, he applied the minimum spanning tree using a Kruskal or Prim algorithm. He then constructed the solution by applying an ant colony on each branch. The result was enhanced by local improvement searches such as 2-Opt and 3-Opt. His algorithm yielded the best-known solutions for some VRP instances.

Mazzeo and Loiseau (Mazzeo & Loiseau, 2004) tried various techniques and combinations of the ACO components to enhance the solution quality of the VRP. A tabu list prevented the algorithm from selecting previously selected customers. The solution was built in both sequential and parallel implementations. In the sequential construct, an ant generated a solution before the next ant could begin. In parallel, the ants generated their solutions at the same time. The ants were randomly distributed on different customer locations. The authors also tested two types of transition rules: random- and pseudo-random- proportional rules. At the end of each iteration, a global pheromone update and an improvement operation (2-Opt) were applied to the best solutions (selected by the elite-ant strategy). Furthermore, the number of neighbours in the selection pool was reduced by a candidate list. The algorithm stops when the maximum number of iterations is reached or when there is no improvement over a certain number of iterations. The authors found that parallel is superior to sequential building, and the improvement operation enhanced the quality of the solutions. The best candidate-list size was 25% of the total number of customers. When tested on three benchmark instances, the algorithms obtained very good results for problem sizes up to 50 customers, and promising results in larger instances. Comparisons with the results of other algorithms proved the competitiveness of their ACO algorithm.

Bell and McMullen (Bell & McMullen, 2004) solved the VRP by modifying an ACO designed for TSPs. Similarly to Mazzeo and Loiseau (Mazzeo & Loiseau, 2004), they improved the solution quality by techniques such as 2-Opt and a candidate list. Starting from the depot, all ants choose the next node from a candidate list. The ants build a solution sequentially, using the inverse of the distance between the customers as a second heuristic for deciding the next node. In this algorithm, large VRP instances are solved by multiple ant colonies. The authors tested both single and multiple ant colonies with different candidate-list sizes. The candidate-list size was found to affect the solution quality and the computational time. The solution lay within 1% of the best-known solution in small instances, and was improved by the multiple ant colonies in large instances.

For solving DVRPs by ACO, Montemanni et al. (2002) borrowed an idea proposed by Kilby et al. (1998), which divides the DVRP into a sequence of static VRPs. Therefore, the working day *T* is divided into several time slices (*nts*), each of length *T/nts*. The new orders arriving during a time slice are postponed to the next time slice. In each slice, the dispatcher (or events manager) creates a static VRP by considering all newly arrived orders in that time, except the served and committed orders. After tracing the vehicle locations and their capacities, the VRP is solved by ACO and the current routes are updated using the obtained solution. The main objective of each static VRP is minimising the travelling time. The cut-off time (*Tco*) and advanced commitment time (*Tac*) are user-defined parameters. The *Tco* and *Tac* control the number of orders for processing and the orders committed to the drivers, respectively. The ACO was tested on artificial instances adapted from benchmark static VRPs, varying the number of available service vehicles, length of the working day, and the service time of each order. Furthermore, the ACO was tested on a real DVRP case. The authors introduced an important feature called the *pheromone conservation procedure* that propagates information about the good solutions from one time slice to the next. The solution is facilitated by several dummy depots (one for each vehicle), marking the location of the last customer committed to the vehicle. In experimental tests, the ACO yielded good results provided that the slice length was properly tuned (between 10 and 50).

Tian, Song, Yao, and Hu (2003) solved the DCVRP by a hybrid ant system. Old information from the previous search is preserved by depositing pheromone. They proposed a technique initialises the pheromone on new nodes (customers), and applies the 2-Opt method to improve the vehicle routes. They tested their algorithm on three instances with different parameter settings. The results demonstrated that the hybrid ant system outperformed the standard ant system.

Rizzoli, Montemanni, Lucibello, and Gambardella (2007) solved a set of real-life DVRPs by ACO. The first problem, based on a supermarket chain in Switzerland, was associated with time windows. The second problem involved a distribution company in Italy requiring both pickup and delivery. The third problem was a time-dependent VRP of freight distribution in Padua City, Italy. The final problem was a DVRP in Lugano City, Switzerland, which receives requests from new customers during the delivery.

Ahmmed, Rana, Haque, and Mamun (2008) designed a multiple-ACO solution for the DVRP with time window (DVRPTW). One ACO minimises the number of required vehicles, while the other minimises the travelling distance. The two colonies collaborate by exchanging information, represented by pheromone update. The proposed algorithm was tested on a set of 56 benchmark problems.

Jun, Wang, and Zheng (2008) solved the DVRPTW by a hybrid multi-objective ACO algorithm. The pheromone update is controlled by an evolutionary algorithm, which also speeds the convergence of the algorithm. The vehicle number and cost are treated as independent objectives. The algorithm performed well in simulation experiments.

2.2.3.2.2 Particle Swarm Optimisation

The particle swarm optimisation (PSO) algorithm was proposed by Kennedy and Eberhart in 1995. This algorithm simulates the social behaviour of animal groups, such as flocks of birds and schools of fish. PSO is a stochastic optimisation algorithm. Each particle in the PSO algorithm represents a single possible solution, and is defined by its velocity and position in a swarm of like particles. These particles continuously move within a multi-dimensional search space, and accordingly update their velocities and positions. The velocity update of each particle is based on its previous velocity and previous best position (personal influence) of the particle, and on the local or global best position (swarm influence). The position is then updated according to the newly obtained velocity. This update process is affected by a random number that helps the algorithm to escape from local optima (Bonyadi, Michalewicz, & Li, 2014).

PSO starts with a swarm of particles distributed randomly in the search space. In each iteration, the particles generate solutions that are evaluated according to the objective function to be optimised. Each particle updates its previous best solution (*pbest*) to obtain a new solution. At the end of each iteration, the best solution among all particles becomes the local-best solution (*lbest*). If *lbest* is better than any previous solution, it replaces the current global-best solution (*gbest*). The *gbest* is shared among all particles during all iterations. As the iterations proceed, the particles gradually converge towards the position of *gbest*. Local optimal solutions are overcome by different techniques, such as increasing the number of particles and inserting a random value into the velocity update. Figure 2-21 is the basic pseudocode of the PSO algorithm, adapted from (Kennedy, 2011).

```
PROCEDURE: Particle Swarm Optimisation Algorithm
Initialise parameters (particle sizes, velocities, positions)
Create a number of particles
Randomly distribute particles in the solution space
REPEAT UNTIL: stopping criterion is met
      FOR each particle
            Update the velocity
            Update the position
            Evaluate the fitness of each particle
            IF the new value is better than the previous best value
            (pbest)
                  Update the current pbest with the new value
            END IF
            IF pbest is better than lbest
                  Update lbest
            END IF
      END FOR
      IF lbest better than gbest
            Update gbest
      END IF
END REPEAT
RETURN: global best solution
```

Figure 2-21. Pseudocode of the particle swarm optimisation algorithm

PSO has been applied widely to various types of optimisation problems on continuous domains. Typically, the PSO algorithm guides the whole swarm towards the locations of the *pbest* and *gbest* particles. The main problem with applying PSO on discrete domains is finding the proper encoding and mapping onto the solution space. Y. Zhang, Wang, and Ji's article provides a comprehensive survey on PSO and its applications (2015).

James and Russell (1995) examined the performance of the PSO algorithm on continuous optimisation functions. Later, researchers such as Chen et al. (2013) modified the PSO algorithm to cope with premature convergence.

The performance of PSO has also been improved by applying crossover and mutation operators (e.g., Esquivel and Coello, 2003). Once such GA operators are introduced, the advantages of PSO over standard evolutionary approaches to the same problem become questionable. Further surveys and reviews of using PSO for COPs were conducted by Bonyadi and Michalewicz (2017), Kaur and Kaur (2015), and Yu Liu et al. (2011).

Ai and Kachitvichyanukul (Ai & Kachitvichyanukul, 2009) solved the CVRPTW with capacity time windows with as heuristic PSO algorithm. This work extended their previous paper, which solved the CVRP using the standard PSO algorithm. When tested on some CVRPTW benchmark datasets, the heuristic variant obtained best-known solutions in small instances (with 25–50 customers) within a reasonable time.

In the TSP solution of Shi, Liang, Lee, Lu, and Wang (2007), an uncertain searching technique is associated with the particle movements in PSO. The convergence speed is increased by a crossover operation that eliminates intersections in the tours. The update equations of the original PSO are modified to suit the TSP problem. The proposed algorithm was extended to generalised TSPs by employing a generalised chromosome. On various benchmark instances, the proposed algorithm proved more efficient than other algorithms.

Zhong, Zhang, and Chen (2007) developed a modified discrete PSO, called C3DPSO, for TSPs. *C*3 refers to a mutation factor that balances the exploitation and exploration in the update equation, buffers the algorithm against trapping in local optima, and avoids premature convergence. The solution of each particle is represented as a set of consecutive edges, requiring modifications in the update equations. The C3DPSO was tested on six benchmark instances with fewer than 100 cities. The proposed algorithm yielded more precise solutions within less computational time than the original PSO algorithm.

Wang, Huang, Zhou, and Pang (2003) solved the TSP by a PSO with various types of swap operations, which assist the algorithm in finding the best solutions. The swap operation exchanges the positions of two cities, or the sequence of cities between two routes. When tested on a 14-node problem, the algorithm searched only a small part of the search space due to its high convergence rate.

The CVRP has been solved by a hybrid discrete PSO algorithm (A. Chen, Yang, & Wu, 2006). This algorithm combines PSO with an SA that controls the acceptance solutions and assists escape from local optima. Each particle generates its own solution to the CVRP, which is saved in a 2D array with length equalling the number of customers multiplied by the number of potential vehicles. The point-to-point movement sequence of a particle represents a tour of the vehicle. The second row of the array stores a binary value (1 if the customer of interest is served by a relevant vehicle, and 0 otherwise). To improve the solution quality, the positions of two customers are swapped by a pair-exchange operation. The hybrid algorithm was tested on a number of benchmark instances with different sizes. Its higher performance than other algorithms was somewhat compromised by the longer computational time.

Ai and Kachitvichyanukul (Ai & Kachitvichyanukul, 2009) configured a PSO with two solution representations and tested it on the CVRP. The two representations (*SR*-1 and *SR*-2) arrange the solutions and facilitate the decoding process. The quality of the constructed solutions is enhanced by three local improvement methods: 2-Opt, 1-0 exchange, and 1-1 exchange. The distance between a customer and a vehicle dictates the priority of servicing that customer by the vehicle. When tested on several benchmark instances, the *SR*-2 improvement enabled near-optimal solutions but increased the computational effort. Overall, the proposed algorithm solves the CVRP with higher proficiency than other PSO algorithms.

Khouadjia et al. (2012) compared the DVRP solutions of adapted PSO and variable neighbourhood search (VNS). The event scheduler creates static instances in divided time slices and controls the customers committed to vehicles. Local searching is performed by a 2-Opt operator. The PSO and VNS are implemented in a simple discrete representation that allows the insertion of new customers into planned routes. Both algorithms were tested on a newly designed set of benchmarks and on existing benchmarks in the literature. The VNS provided shorter routes than existing approaches, whereas PSO produced new best solutions. On the new benchmarks, PSO and VNS provided superior solutions in small and large instances, respectively. Additionally, PSO outperforms VNS in highly dynamic instances as VNS serves fewer customers than PSO and maintains only one solution.

Okulewicz and Mańdziuk (2013) proposed a two-stage PSO for DVRPs. In the first stage, customers are assigned to specific vehicles by PSO, which constitutes a bin packing problem. Next, the best customer sequence for each vehicle is found by PSO (one PSO instance per vehicle), which constitutes solving a TSP. As implemented in Montemanni et al. (2002), the working day is divided into a series of time slices. In evaluation tests, the proposed algorithm outperformed other approaches in 6 out of 21 benchmark instances.

Okulewicz and Mańdziuk (2017) proposed a two-phase multi-swarm PSO for solving DVRPs. The initial solutions are generated by a clustering heuristic. The PSOs in the first and second phases assign requests to vehicles and optimise the obtained routes, respectively. Information is directly passed between subsequent states. When tested and assessed on benchmark instances, the algorithm proved more effective than other algorithms published in the literature.

2.2.3.2.3 Artificial Bee Colony Algorithm

Bees are social flying insects that (like ants) live in colonies. Each group of bees is responsible for a specific task. Worker bees are required to collect nectar and pollen from flowers, which feed the colony and enable honey production. When scouting bees discover a food source, they return to the hive and communicate the location, distance and type of the food source to other bees. Depending on the distance of the food source, bees perform a dance with a specific pattern (the so-called round or waggle dance) that accurately depicts the information.

In 2005, Karaboga proposed an artificial bee colony (ABC) optimisation algorithm that simulates the behaviour of real bees when collecting food and sharing information. The ABC algorithm generates a set of artificial bees playing different roles in the solution construction. The number of bees in each category (employed, onlooker, and scout) can vary. Typically, the swarm is divided into ~50% employed bees, ~50% onlookers, and one scout bee. The employed bees mark the locations of the best solutions; each bee visits one of the solutions and returns to the hive, where it dances in front of the onlookers. The dance informs the location and quality of the solution to the onlookers, who have remained in the hive waiting for information from the employed bees. Having obtained the necessary information, the onlookers must choose among the locations found by the employed bees. As the onlookers prefer locations with a high percentage of nectar, the quantity of nectar determines the solution quality. Here, a solution is the location of a food source, and its quality is evaluated by the objective function. The scout bees fly continuously and randomly in different directions to find new food sources (new better solutions). When the food source of an employed bee is exhausted (i.e., a better solution has been found), that bee becomes a scout and begins searching for new solutions. The problem is represented in a multidimensional space, where the locations of the food sources represent the coordinates of a potential solution.

The ABC algorithm performs three steps: moving the employed and onlooker bees to the food sources, calculating the nectar amounts, and controlling the number of scout bees and directing them towards new possible areas. The algorithm starts with a random distribution of the employed bees in the search space, and calculates the nectar amount in each location (the fitness of each location). The onlooker bees calculate the probability of selecting each location based on the nectar amount. The chance of being visited by an onlooker bee increases at locations with high nectar amount. Figure 2-22 gives the pseudocode of the ABC algorithm, based on our interpretation of the algorithm (Karaboga, 2005).

```
PROCEDURE: Artificial Bee Colony
Initialise parameters (population size, number of employed bees,
number of onlookers, number of scouts)
Generate an initial population at random positions
Send the scout bees
REPEAT UNTIL: stopping criterion is met
      Send employed bees to find food sources
      Calculate nectar amounts
      Onlooker bees calculate the probability of visiting each source
      Send onlooker bees to good food sources, and calculate nectar
         amounts at nearby areas
      Scout bees perform random search to find new food sources
      Memorise the best food source (best-solution location) found so
         far
END REPEAT
RETURN: best-solution
```

Figure 2-22. Pseudocode of the artificial bee colony

The employed and onlooker bees in the ABC algorithm control the exploitation, while the scout bees control the exploration process. The ABC algorithm has successfully solved different problems, but is mostly applied to unconstrained and constrained continuous problems. Some of these researches are listed in Table 2-6.

Table 2-6. Applications of the artificial bee colony algorithm

Application	References
TSP	(Karaboga & Gorkemli, 2011); (Kiran, Iscan, & Gündüz, 2013); (Kocer & Akca, 2014); (Hu et al., 2016); (Choong, Wong, & Lim, 2017)
COP	(Karaboga & Basturk, 2007); (Kiran & Babalik, 2014); (Huo, Zhang, & Zhao, 2015); (Yavuz, Aydin, & Stutzle, 2016); (B. Liu, Li, & Pan, 2016); (Gan Yu, 2016); (Yaghoobi & Esmaeili Toudeshki, 2016); (B. Zhang, Liu, Zhang, & Wang, 2017); (Liang, Wan, & Fang, 2017); (Cui, Li, Zhu, et al., 2017); (Cui, Li, Wang, et al., 2017); (Q. Lin et al., 2018)
CVRP	(Szeto, Wu, & Ho, 2011), (Brajevic, 2011), (S. Z. Zhang & Lee, 2015)

Szeto et al. (2011) applied the ABC to the CVRP. The ABC–CVRP first builds an initial solution by sequentially assigning one random customer to one vehicle route, and evaluates the cost of the initial solutions. The onlooker bees select the best solutions by the roulette-wheel technique, and find other possible solutions by seven neighbourhood operations (involving swaps, reversals and insertions with different variations and setups). They then calculate the nectar amounts (qualities) of the solutions. After ranking the solutions, the employed bees are assigned to the best ones, and the poor solutions are abandoned. The scout bees continue looking for new solutions until they find a new best solution, when their role changes to employed. These steps repeat until the termination condition is met. The CVRP solution is stored in a vector with length equalling the number of customers plus the possible number of vehicles. In this vector, the order of customers to be visited by a vehicle is separated by 0's starting from the depot. When tested on two benchmark datasets, the ABC–CVRP outperformed the original ABC implementation, but required longer computational time.

Brajevic (2011) solved the CVRP by an adjusted ABC that stores the solution in a two-dimensional array. The length of the first row is the product of the customer and vehicle numbers. Each cell of the second-row stores either 0 or 1. The algorithm starts with randomly distributed initial solutions. The routes in each solution must be checked to ensure that they satisfy the problem constraints. The overall solution quality is improved by a swap-mutation procedure (i.e. a 2-Opt) that changes the value in a specific cell of each route. Moreover, an exchange operation randomly selects two customers (one from each of two routes) and exchanges their positions. The algorithm was tested on twelve small-scale instances from the literature, varying the maximum number of iterations to suit the problem size. In some instances, the algorithm tended to become trapped in local optima.

Zhang and Lee (2015) solved the CVRP by an adjusted ABC that improves the exploration ability over the original ABC algorithm. The solution of each bee is stored in a vector, in which the customer sequences up to the vehicle's capacity are separated by 0's. Each customer sequence represents a vehicle tour. The algorithm starts with feasible permutation solutions that are randomly generated to improve the algorithm performance. The employed-bee phase searches the neighbourhood by performing a customised swap operation called the *best-route mapping crossover*, which finds superior new solutions. When tested on benchmark instances, the adapted algorithm significantly outperformed the original ABC.

Kiran, Iscan, and Gündüz (2013) proposed a discrete ABC algorithm for TSPs. They replaced the original ABC update equations by well-known TSP neighbourhood search operations reported in the literature. These operations are applied between two points or between two node sequences. The point-to-point operations are random swap, 3-Opt and random insertion. The sequence operations are random swap, random insertion, random reversing, random reverse swap, and random reverse insertion. Initially, each employed bee is assigned a random permutation solution. The modified ABC algorithm was tested along with other algorithms on nine benchmark instances, and yielded promising results.

To solve the TSP, Kocer and Akca (2014) improved the ABC algorithm with local search techniques. The tasks of the bees are determined by a decision-making mechanism based on a loyalty function and the fitness function. In comparison with other algorithms, the 2-Opt operation was found to enhance the solution quality. The experimental results demonstrated the ability of the algorithm to solve small-size problems.

2.2.3.2.4 Bees Algorithm

Another algorithm based on the foraging behaviour of honeybees is the bees algorithm (BA), developed by Pham et al. (2005). The BA generates solutions by combining neighbourhood and random searches. Initially, the scout bees are randomly distributed at different locations in the search space. The fitnesses of the locations visited by the scout bees are evaluated using the objective function. The bees at the fittest locations (considered as selected or elite bees) perform an extended neighbourhood search to find new promising solutions. These new locations are re-evaluated, and the best of them recruit a number of new bees, which perform additional nearby searches. To control the number of explored locations, only a proportion of the highest-fitness bees will survive to the next iteration. At the start of the next iteration, the survivors are again randomly distributed in different locations. The BA repeats the above operations until the termination condition is met. Figure 2-23 shows the pseudocode of the simplest BA based on our interpretation of the algorithm (Pham et al., 2005).

```
PROCEDURE: Bees Algorithm
Initialise parameters (number of scout bees, number of selected sub-
locations for neighbourhood search, number of best locations, number
of bees recruited to the best locations, number of bees recruited to
other locations)
Generate a number of scout bees and distribute them at random locations
Evaluate the solution quality of the bees
REPEAT UNTIL: stopping criterion is met
      Select elite bees.
      Choose top locations for neighbourhood searching
      Recruit new bees for the best new locations, and evaluate the
         fitness
      Choose the best bees
      Allow the other bees to randomly search and evaluate the fitness
END REPEAT
RETURN: best-solution
```

Figure 2-23. Pseudocode of the bees algorithm

The BA has mostly been applied to continous benchmark functions (Nasrinpour, Bavani, & Teshnehlab, 2017; Pham et al., 2006; Pham & Castellani, 2009, 2014, 2015). Fenton (2016) enhanced the BA to solve the VRP. The initial solutions are generated by an insertion heuristic, and later solutions are obtained by a large-neighbourhood search operation. Finally, the obtained solutions are enhanced by a repair heuristic using the λ-interchange operation (with λ = 2). In an experimental evaluation, the algorithm outperformed the original BA, delivering high-quality solutions within a short computational time.

2.2.3.2.5 Firefly Algorithm

A firefly is a winged insect that produces visible light by a chemical reaction, a phenomenon known as *bioluminescence*. Fireflies are very active at night, using their light to attract small prey insects. Male fireflies also blink to attract female mates (i.e., make mating calls), alerting them to the males' locations. Naturally, the light signals of males are understood only by females of the same species. The specific patterns of the flashing lights enable communication and location detection among the fireflies. The light varies in intensity and flashing rate. The light intensity at a location is inversely proportional to distance from the source (i.e., follows the inverse square law) (Warren & Warren, 1958). Therefore, the fireflies' lights are visible only within a limited distance.

The firefly algorithm (FA) is an optimisation algorithm inspired by the behaviour and activity of fireflies (X.-S. Yang, 2008). The artificial fireflies are assumed to be unisex, enabling their mutual attraction. A firefly attracts other fireflies with a strength that depends on the brightness of its light. Less bright fireflies are attracted towards brighter fireflies. The attractiveness is also proportional to the Cartesian distance (specific radius) between the fireflies. As mentioned above, the perceived intensity reduces with increasing distance from the light source. As the attractiveness is affected by the varying light brightness and distances between the fireflies, the fireflies perform diverse movements in the search space and explore many regions. When a firefly is not attracted to any other fireflies, it moves randomly through the space. The light brightness represents the firefly's fitness (i.e., its solution quality), and is measured using the objective function. The light of any firefly is assumed to be visible everywhere in the problem domain. Therefore, the brightest firefly will attract all other fireflies and represents the global solution.

The algorithm begins with multiple fireflies distributed randomly through the search space. Once its solution has been evaluated, the firefly is assigned a light brightness based on its solution quality. The brighter the fireflies, the greater their attractiveness. Figure 2-24 gives the pseudocode of the FA based on our interpretation of the algorithm (X.-S. Yang, 2008).

```
PROCEDURE: Firefly Algorithm
Initialise parameters (light absorption coefficient)
Generate a set of fireflies
Distribute the fireflies randomly in the search space
Evaluate the fitness of each firefly
Calculate the light intensity according to the objective function
REPEAT UNTIL: stopping criterion is met
      FOR each firefly i
            FOR each firefly j
                  IF light of firefly i < light of firefly j
                        Move firefly i toward firefly j
                  END IF
                  Update the attractiveness based on the distance
                  Evaluate the fitness of the new firefly’s location
                  Update the light intensity
            END FOR
      END FOR
      Sort the fireflies by their fitnesses
      Identify the global best firefly
END REPEAT
RETURN: best-solution
```
Figure 2-24. Pseudocode of the firefly algorithm

Originally, the FA was applied to continuous problems (e.g., X.-S. Yang, 2009). However, only the brightness guides the fireflies to the global solution, so the FA converges very quickly.

To solve the TSP, Kumbharana and Pandey (2013) adjusted the FA for use on discrete domains by re-customising the fireflies’ movements. The initial solutions are generated as random permutations. The similarity between two solutions is measured by the Hamming distance, or by calculating the number of swap-steps required to reach similarity between two solutions. The firefly movements are represented by a proposed *inversion mutation* operation. The FA provided better results than other algorithms in some of the instances.

2.2.3.2.6 Cuckoo Search

The cuckoo has a fascinating unexplained breeding behaviour. The cuckoo parasitically impels other birds to mind its own eggs until they hatch. Reproduction begins when the female cuckoo lays a single egg in another bird's nest, leaving the nest owner with the tasks of nurturing and breeding. The cuckoo selects a host bird whose eggs resemble its own eggs. Alternatively, by a mysterious mechanism, it can mimic the pattern and colour of hosts’ eggs that are unlike its own. To maximize the chance of its own egg hatching, the cuckoo displaces one egg from the host nest. However, some of the host birds recognise the new egg as an imposter, and will throw it out or abandon the nest. Furthermore, cuckoos must synchronise their egg-laying operations with those of their host species to guarantee that all eggs will hatch together. Characteristically, cuckoo eggs have a shorter incubation time than the eggs of other birds. Therefore, when hatched, the cuckoo chick throws the remaining unhatched eggs, or even the host hatchlings, out of the nest. The cuckoo chick can then remain alone and receive all the food, thereby increasing its survival opportunity.

Inspired by the aggressive reproduction strategy of the cuckoo, Yang and Deb (2009) designed an optimisation algorithm known as cuckoo search (CS). In CS, each of the generated cuckoos lays one egg in a randomly selected nest. The number of available host nests is limited. The host bird discovers the cuckoo egg with a certain probability. If the cuckoo egg is discovered, the host either abandons the nest and builds a new one in a different location, or ejects the egg from the nest. Each egg represents a solution, and a cuckoo egg represents a new potential better solution. The quality of the egg is evaluated using the objective function of the problem. If the cuckoo egg is better than an existing egg, the weak egg is removed and replaced with the cuckoo's egg. Only the nest containing high-quality eggs remains in the succeeding iterations.

Levy flights improve the performance of CS by enabling better exploration of the search space. Levy flights describe the random-walk behaviour of some foraging animals, namely, linear movements separated by sudden changes in direction (Bartumeus Ferré, 2005). Figure 2-25 shows the main pseudocode of the CS based on our interpretation of the algorithm (X.-S. Yang & Deb, 2009).

```
PROCEDURE: Cuckoo Search
Initialise parameters (abandon rate, nest number, Levy step size,
discovery probability)
Generate cuckoos
Generate a set of host nests
REPEAT UNTIL: stopping criterion is met
      Produce a new cuckoo egg using Levy flights
      Evaluate the fitness of the egg
      Select one of the nests at random
      IF (cuckoo egg is better than the existing egg)
            Replace the weak egg with the cuckoo egg
      END IF
      Evaluate the fitness of all nests and rank them
      Abandon the low-quality nests and build new ones.
      Update the best-solution found so far
END REPEAT
RETURN: best-solution
```

Figure 2-25. Pseudocode of cuckoo search

For performance enhancement, CS uses two kinds of random walk (local and global). The CS has solved a number of optimisation problems, some of which are summarised in Table 2-7.

Table 2-7. Cuckoo search applications

Application	References
TSP	(Jati, Manurung, & Suyanto, 2012); (Ouaarab, Ahiod, & Yang, 2014)
COP	(Tuba, Subotic, & Stanarevic, 2011); (X.-S. Yang & Deb, 2013); (Singh & Singh, 2015); (Kordestani, Firouzjaee, & Reza Meybodi, 2018)

Jati et al. (2012) solved the TSP by a discrete CS. The solution is represented as a permutation sequence of nodes and stored in an array. Each cuckoo starts with a random solution and generates one solution per iteration. The difference between a solution and the best solution found so far is measured using a discrete step size, and is based on the number of different edges between the two solutions. The solution space is exploited by performing a local random walk around the best solutions. The solutions are updated using the inversion mutation operation. To guarantee diversity of the generated solutions, a selection process checks whether any cuckoos yield the same solution in each iteration. The algorithm obtained good results on simple benchmark instances, but tended to fall into local optima in complex instances.

Ouaarab et al. (2014) presented another discrete CS for TSPs. In their algorithm, a cuckoo egg represents a solution, and the nests' locations correspond to city locations. The solution (tour) is saved as a cyclic permutation. The neighbourhood is searched by two operations: 2-Opt and double-bridge, which elicit a small and a large change, respectively. The step length relates to the Levy flight size. When tested on forty-one benchmark instances, this CS outperformed other algorithms in terms of solution quality.

2.2.3.2.7 Bat Algorithm

Bats are the world's only flying mammals. Although bats have eyes and can see, they rely mainly on their ears to discover and locate surrounding objects at night. Bats possess a superior sonar system called *echolocation,* by which they detect prey, search for food sources, and avoid obstacles. Bats emit ultrasonic sounds of species-specific frequencies, which disperse and bounce off nearby objects (Surlykke, Boel Pedersen, & Jakobsen, 2009). When bats receive the echoes, they extract the information from them (obtaining detailed images of the environment) and behave accordingly. The information contains the sizes and shapes of objects, distance to the objects, and current speed of the bat. Other features of the produced sound are loudness, pulse rate, frequency, and wavelength.

The echolocation mechanism of bats inspired the development of a metaheuristic algorithm called the bat algorithm (BA) (X.-S. Yang, 2010). In the BA, a number of artificial bats fly in a multidimensional search space. Each bat has two properties (velocity and location) that are updated at every iteration. The sound generated by each bat is characterised by its loudness, pulse rate, and frequency. The values of these variables depend on the quality of a bat's solution. For simplicity, the frequency is restricted between a specified maximum and minimum. Also, the pulse rate is limited to the range [0, 1], where 1 is the maximum. One bat marks the location of the best solution and guides the other bats to this location. Figure 2-26 shows the pseudocode of the BA based on our interpretation of the algorithm (X.-S. Yang, 2010).

```
PROCEDURE: Bat Algorithm
Initialise parameters
Generate a set of bats
Randomly initialise velocity and location of all bats
Evaluate the fitness of each bat
Calculate pulse rates and loudness
REPEAT UNTIL: stopping criterion is met
      Move the bat to a new location by adjusting the frequency
      Update the velocity and location of the bat
      IF (rand > pulse rates)
            Select a solution from the candidate solutions at random
            Perform local search (random walk) around the selected solution
      END IF
      Generate a new solution by moving bats randomly
      IF (rand < loudness, and the new solution is better than the previous solution)
            Accept the new solution
            Increase the pulse rate
            Reduce the loudness
      END IF
      Rank the bats and identify the best-bat (best-solution)
END REPEAT
RETURN: Best-solution
```

Figure 2-26. Pseudocode of the bat algorithm

If the frequency variations in the BA are replaced with a random variable, and the loudness and pulse rate are set to 0 and 1, respectively, the BA becomes identical to PSO (X.-S. Yang, 2010). The BA also shares some similarity with the harmony search algorithm.

The BA has solved several engineering optimisation problems (X.-S. Yang & Hossein Gandomi, 2012), numerical optimisation problems (Chakri, Khelif, Benouaret, & Yang, 2017; P. W. Tsai, Pan, Liao, Tsai, & Istanda, 2011; Yilmaz & Küçüksille, 2015), and TSPs (Osaba, Yang, Diaz, López-Garcia, & Carballedo, 2016; Saji, Riffi, & Ahiod, 2014). Taha, Hachimi and Moudden (2015) adapted the BA for solving the CVRP. They represented the solution as a set of vectors (one vector for each required vehicle). They also enhanced the solution quality by introducing insert, invert, two single-point swap, and single-point swap operations. When tested on nine benchmark instances, the adapted BA well competed with the GA.

2.2.4 Non-Nature-Inspired Algorithms

This section reviews three human-made algorithms that were not inspired by nature. While these are non-nature-inspired algorithms, they share similar steps with nature-inspired algorithms. Therefore, it worth comparing their performance with nature-inspired algorithms.

2.2.4.1 Tabu Search algorithm

Tabu search (TS) is a metaheuristic algorithm introduced by Glover in 1986. In simple terms, the TS combines a local search or heuristics with memory. The local search techniques can be suited to the requirements and constraints of the problem. The memory tracks the most recent solutions and saves them in a data structure called a tabu list. The number of solutions held by the memory depends on the implementation technique of the TS, and can be many (long-term memory) or few (short-term memory). The tabu list helps the algorithm to avoid revisiting (also referred to as cycling) the stored solutions, thus reducing the computational time. In some problems, when the search space comprises both feasible and infeasible solutions, a solution is added to the tabu list if it violates the problem constraints. The basic pseudocode of the TS is given in Figure 2-27 (Glover, 1986).

```
PROCEDURE: Tabu Search
Initialise parameters (Tabu-list max-size)
best-solution := {}
Tabu-list := {}
REPEAT UNTIL: stopping criterion is met
      Perform local search
      Evaluate the fitness of all solutions and rank them
      New solution := Obtain a solution from the candidate list
      IF (New solution not in Tabu-list and better than best-solution)
            best-solution := New solution
      ELSE
            Add the New solution to the Tabu-list
      END IF
      IF (the size of Tabu-list > max-size)
            Remove the oldest solution from the list
      END IF
END REPEAT
RETURN: best-solution
```

Figure 2-27. Pseudocode of tabu search

The TS has successfully solved difficult optimisation problems such as symmetric TSPs (Knox, 1994). TS implementations on the TSP have been recently reviewed in Basu (2012). The review analyses the effect of different implementations of TS for solving TSPs. It found that many studies tested TS on small TSPs (100 nodes or less), and it identified the effect of the initial solution on the final solution. Furthermore, the short-term memory is commonly used with TSP.

Gendreau et al. (1994) used TS to solve a VRP with capacity and route length constraints. Their algorithm includes an insertion procedure that moves a customer from one route to the next adjacent route. The algorithm was tested on 14 benchmark instances with different numbers of customers (50–199). Their TS outperformed some of the existing algorithms and achieved the best-known solution in some instances. Barbarosoglu and Ozgur (1999) also solved the VRP by TS.

Bräysy et al. (Bräysy & Gendreau, 2002) surveyed the use of the TS algorithms in VRPs with time windows. They compared and analysed these algorithms on well-known benchmark instances. The survey revealed that the algorithms are very competitive, and TS algorithm is a good approach to tackle VRPTW.

Yang, Li, and Li (2007) proposed an algorithm called angle-based crossover TS for CVRPs. In this algorithm, the exploration and exploitation processes are implemented by a GA crossover operator and an elite selection strategy, respectively. The angle-based operation, which is based on the sweep heuristic, improves the performance of the algorithm. When tested on twelve CVRP benchmark instances, the proposed algorithm proved more efficient than the sweep algorithm and the original TS.

2.2.4.2 Firework Algorithm

Fireworks are combustible chemicals that create spectacular effects and patterns during explosions. Ignited fireworks produce light, noise, and smoke, and generate floating materials that burn and create coloured sparks or flames. Depending on the manufacturing quality, firework explosions can be spectacular events or fizzlers. The quality of a firework is determined by the numbers and locations (distributions) of the generated sparks. A good explosion generates many sparks concentrated within a small range around the main location of the firework. In contrast, a bad firework generates a few, randomly scattered sparks within a large region of the space.

Tan and Zhu (2010) designed a firework algorithm (FWA) that imitates the explosion mechanism and firing process of fireworks. A set of fireworks is fired in different directions of the search space. When a firework explodes, the generated sparks fill the space around the original location of the firework, enabling further local searching. Therefore, the FWA searches two neighbourhoods: one near the origin of the firework, the other around its sparks. The spark locations are then evaluated using the objective function. These processes are repeated in each iteration until the maximum number of iterations is reached, or the global optimal solution is found. In the FWA, a good firework (a strong firework with a high explosion amplitude) represents a good solution or a solution that is very close to a good solution. Consequently, the search is extended to the immediate neighbourhood of the good firework. Alternately, the search radius can be extended far from the main firework location to find better locations. Figure 2-28 shows a simple pseudocode of the FWA based on our interpretation of the algorithm (Y. Tan & Zhu, 2010).

```
PROCEDURE: Firework Algorithm
Initialise parameters (number of sparks generated by each firework)
Select a number of random locations for fireworks
REPEAT UNTIL stopping criterion is met
    Fire n fireworks into the selected locations
    FOR each firework do
        Evaluate the solution quality of the firework
        Generate a number of sparks based on the firework's fitness
        Scatter the sparks of the firework according to amplitude of explosion
        Evaluate the solution quality of the sparks
    END FOR
    Rank all the solution, and keep the n locations in the next iteration
    Update the best solution
END REPEAT
RETURN: best-solution
```

Figure 2-28. Pseudocode of the firework algorithm

The FWA has been enhanced by adding a mutation operator and a selection strategy. The mutation is based on a Gaussian distribution, and the selection strategy determines which fireworks will continue in successive iterations. Selection is determined by the Euclidean distance between the fireworks. The chance of choosing a firework increases with increasing distance from other fireworks, which diversifies the solutions in the following iterations.

In FWA, the explosion operation is mainly responsible for both local and global searching. Because the algorithm lacks other heuristics that decide the locations of new fireworks (apart from sparks that create diversity), the FWA is prone to falling in local optima. The fireworks do not interact and do not affect each other, but are affected by the location of the global solution. Moreover, the FWA procedure focuses on the promising areas only, without moving the search into other areas. The algorithm may also require a long time to reach the global solution.

The FWA has been validated on different types of problems, including numerical function optimisation (Y. Tan & Zhu, 2010), the TSP (Y. Tan, 2015), and the CVRP (Abdulmajeed & Ayob, 2014).

Tan (2015) solved the TSP by the FWA. The original explosion technique in this algorithm is adapted for discrete domains, which are not effectively handled by random local search methods. There are two explosion operations –a two-edge swap operation (2-Opt) and a three-edge swap operation (3-Opt) – and a 2H-Opt mutation operation for discrete problems. The selection strategy chooses the highest-fitness fireworks and sparks. In experimental evaluations, the algorithm effectively solved small instances, but was outperformed by other algorithms in large instances. This weak performance is attributable to lack of interaction among the fireworks.

Abdulmajeed and Ayob (2014) solved the CVRP by the FWA. The CVRP solution is represented as a vector of customer sets (vehicle loads) separated by 0s. An initial solution is randomly generated within the bounds of the problem constraint. New solutions are generated by a neighbourhood search operation implemented by random-swap and random-move actions. This algorithm obtained the best-known solution in six out of fourteen benchmark instances, and was competitive with other algorithms in the remaining instances.

2.2.4.3 Grenade Explosion Method

A grenade is a small deadly explosive device (i.e., a bomb), usually thrown by hand. Different types of grenades have different explosion capabilities. Once the safety pin is removed, the grenade will detonate after a certain amount of time or when it hits the ground. The destructiveness of a grenade explosion depends on the number of generated shrapnel pieces and their spread distance. To maximise the destruction, the targeted objects must lie within a specific radius of the explosion, which can be achieved by trying to throw the grenade close to the objects.

The grenade explosion method (GEM) is an evolutionary algorithm that optimises real-valued problems (Ahrari and Atai 2010). The explosion mechanism of the GEM avoids the crowding of candidate solutions around a local minimum. The GEM explodes a number of grenades in a two-dimensional space, generating a number of shrapnel pieces. The shrapnel pieces are thrown in random directions, destroying objects in their vicinity. The radius of the thrown shrapnel pieces is controlled by a specific parameter, and any objects within this length are destroyed. A large length favours exploration, whereas a small length allows exploitation of the near regions. Therefore, the search capability depends on selecting the proper spatial extent of the explosion. The GEM calculates the losses and effect of each piece of shrapnel. High losses and effects indicate valuable objects within the searched area. Another grenade is thrown in the valuable area, incurring greater loss, and the algorithm eventually converges.

A GEM feature called the *territory radius* prevents two grenades from searching in a close vicinity. This feature disperses the grenades, encouraging exploration of the entire search space.

The territory radius is controlled by a variable whose value depends on the problem domain or the user preferences. The numbers of grenades and produced shrapnel pieces also need to suit the problem domain. Figure 2-29 shows a simple pseudocode of the GEM based on our interpretation of the algorithm (Ahrari & Atai, 2010).

```
PROCEDURE: Grenade Explosion Method
Initialise parameters (Number of grenades, number of shrapnel pieces
Nq, shrapnel length)
Generate n grenades and randomly distribute them at separate locations
REPEAT UNTIL stopping criterion is met
      Evaluate the fitnesses of the grenades and rank the grenades
      Produce a number of shrapnel pieces for each grenade
      IF (the new location of the shrapnel is outside the feasible
      region)
            Relocate it inside the feasible region
      END IF
      Evaluate the fitness of the new shrapnel
      IF (a new solution is better than the best current solution)
            Check the territory radius
            Move the grenade to that location
      END IF
END REPEAT
RETURN: best-solution
```
Figure 2-29. Pseudocode of the grenade explosion method

The GEM has many crucial parameters that require careful tuning. Improper parameter settings will slow the convergence rate of the algorithm or trap it in local optima. The grenade and its shrapnel play similar roles to the employed and onlooker bees, respectively, in the ABC algorithm, and selecting the number of grenades in the GEM has the same effect as choosing the number of bees in the ABC algorithm.

The GEM has mainly been tested on multimodal benchmark functions (Ahrari & Atai, 2010). In terms of runtime, the GEM outperformed other algorithms in seven out of eight benchmark tests. However, the algorithm must maintain the non-intuitive territory radius that prevents shrapnel pieces from approaching each other, forcing them to spread uniformly over the search space. An explosion range for the shrapnel is also required. These control parameters work well in problem spaces containing many local optima, as the global optimum is found after suitable exploration. However, the relationship between these control parameters and problem spaces with unknown topology is unclear.

2.3 Comparison between Nature-Inspired Algorithms

The working mechanisms of different algorithms share certain similarities. Despite being inspired by different sources and employing different mathematical models and computational techniques, all nature-inspired algorithms guide the entities towards the best solution. Convergence to the global optimum is achieved by enhancing paths with more pheromone, less soil, attraction to the global-best, heaviest or brightest entity, attraction to sea locations, or some similar mechanism. Regardless of technique, the way of controlling the technique and the effect of one entity on other entities must be carefully considered, as these factors affect the convergence rate of the algorithm and the likelihood of trapping in local optimal solution. Two algorithms can be distinguished by their control parameters and mathematical formulations, or by comparing the steps of the algorithms. Many studies compare the performances of multiple algorithms on a specific problem.

The abovementioned algorithms can be classified as trail-based or point-based algorithms. Trail-based (or edge-based) algorithms assign values to the trail lines that connect points. The entities of the algorithm choose the edge with the highest visitation probability among the edges. Therefore, an entity moves from one place to another along a connection (i.e., an edge) between the occupied and soon-to-be occupied site. This restricts the movements to nodes with connections to other nodes. The proper input for trail-based algorithms is usually a graph representation with no globally best edge. The ACO and IWD algorithms are examples of trail-based algorithms. In contrast, point-based algorithms assign values to the positions of points. In each iteration, the entities move to a high-fitness point or to one with higher visitation probability than the other points. In some point-based algorithms, choice of the next node is affected by the position of the global-best point. Such algorithms are characterised by free movement of the entities in a multidimensional space; the entities are not required to move along connections between the points. Algorithms in this category include PSO, GSA, CFO, and FA.

Based on these differences, we can understand that trail-based algorithms are suitable for discrete (e.g. combinatorial) problems in which the node sequence is important. In contrast, point-based algorithms are suitable for continuous domain problems, in which the values of the points are more important than their ordering.

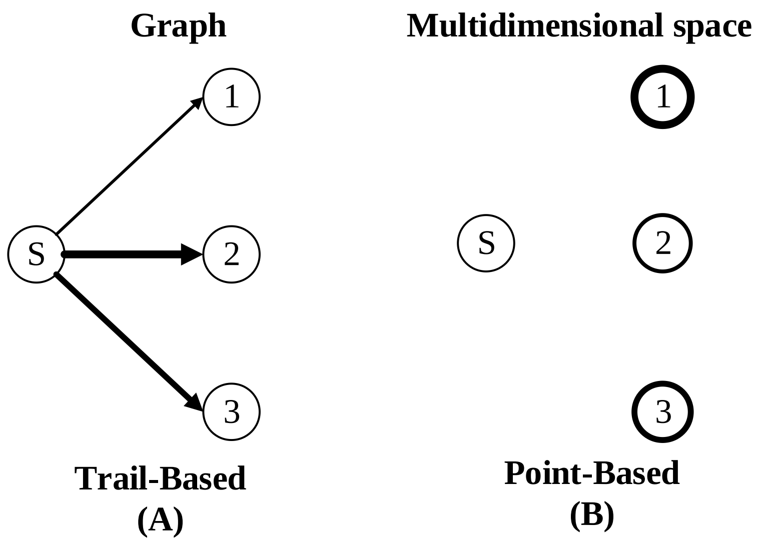


Figure 2-30. Structures of trail-based point-based algorithms

In the trail-based structure of Figure 2-30 (A), an entity at position *S* must choose between nodes 1, 2 and 3. Thicker connection lines (edges) will be selected with higher probability than thinner lines. In this case, the entity will preferentially visit node 2. In the point-based structure of Figure 2-30 (B), the thickness of the circle represents the fitness of the point. In this case, entity S will preferentially select node 1.

Various algorithms will be compared in the following paragraphs. These algorithms are inspired from different sources, but share similar working mechanisms. For example, the entities in the GSA and PSO algorithms continuously move through a multidimensional space, but their motions are governed by different techniques. For instance, the particles in PSO move towards the *lbest* and *gbest* positions without considering the distances between the particles. In the GSA, the object dynamics depend on the overall forces exerted by other entities and the inter-particle distances. Therefore, GSA does not use memory to track the *gbest* or *lbest* positions.

The GSA and CFO algorithms are both based on Newton's laws of motion, but differ in their formulations and expressions. More specifically, the two algorithms differ in their mass calculations and usage of the gravitational constant. Moreover, CFO is a deterministic algorithm, whereas GSA is a stochastic algorithm. Finally, the initial values of the entities are assigned by a deterministic rule in CFO, and by random selection in GSA. One advantage of CFO is that the probe movement depends on the mass difference between two attractive entities (which avoids extreme pulling), whereas in GSA, all objects move towards the heaviest object. Th CFO technique helps to avoid tapping in local optimal solutions.

Both PSO and GA algorithms assign random values to the initial population. The PSO requires no crossover or mutation operations, as the particles move towards the locations of the global and local best solutions while updating their locations and velocities. The GA solves a problem through the survival-of-the-fittest principle. PSO is generally more easily implemented than GA, as fewer parameters need to be tuned.

The GSA, CFO, PSO, FA, and BA algorithms share similar concepts, but generate their solutions using slightly different approaches. In these algorithms, the entities are initially randomly distributed in an *N*-dimensional search space, then rearrange as the weak entities move towards the strongest one. Typically, the location of the global-best solution (*g**) is shared among the entities. However, the ways of representing the attraction and controlling the attraction differ among the algorithms (Green, Aleti, & Garcia, 2017). The main weakness of these algorithms is their tendency for premature convergence, and consequent trapping in local optima.

The WCA and PSO algorithms share similar structures but use different nomenclatures for their components. For instance, the PSO algorithm specifies the *gbest* and *lbest* points, where the *gbest* point guides (or effects) the *lbest* points towards its own location. Similarly, the WCA represents the local-best and global-best solutions by a number of rivers and the sea, respectively. The streams and rivers are guided towards the sea. In PSO, a particle at a *pbest* point updates its position towards the *gbest* point. Analogously, the WCA exchanges the positions of the sea and a river with higher fitness than the sea. Differently from PSO, the WCA applies evaporation and raining operations that help the algorithm to escape from local optima.

The BA and ABC algorithms are inspired by the behaviours of foraging bees, but slightly differ in their implementations. First, the ABC algorithm uses fewer parameters than the BA algorithm. Second, the bee types in ABC are called employed, onlooker, and scout, whereas in BA, they are called elite, recruit, and scout. Third, the BA directs the search for top locations by an elitist strategy and neighbourhood searching, which increase the convergence speed. The ABC combines a local search method with a global search strategy, and the bees memorise the best previous locations. Finally, the bees in ABC share information through their wiggle dance, whereas information sharing is absent in BA. The scout bees in both algorithms perform random searches in the solution space. Further experimentally based comparisons are reported elsewhere (Karaboga & Akay, 2009; H. Li, Liu, & Li, 2010).

Table 2-8 summarises and highlights the main similarities/differences among the above algorithms.

Table 2-8. Conceptual comparisons among algorithms inspired by living and non-living natural phenomena

Algorithm, Year	**Entity type**	**Entity attributes or properties**	**Problem representation**	**Source of inspiration**	**Initial values of entities**	**Direct communication**	**Depend on global best solution**	**Accept bad solution**	**movement**	**Changing number of entities**
SA 1981	single point	fitness	one-dimensional	metal annealing process	random	no	no	yes	random	no
CFO 2007	probes	position, acceleration, and mass	multidimensional search space	motion of objects	deterministic values using uniform distribution	no	yes	no	difference between masses (towards bigger mass)	no
GSA 2009	objects	position, acceleration, velocity, and mass	multidimensional search space	motion of objects	random positions	no	*N* top-best solutions	no	towards bigger mass	no
RFD August, 2007	drops	velocity, momentum energy	graph	river formation mechanism	same altitudes	no	no	yes	regions with low altitudes	yes
WF-LA May 2007	water flow	velocity, mass, momentum, and kinetic energies	multidimensional search space	movement of water	random locations	no	no	yes	lower attitudes	yes
IWD September, 2007	intelligent water drop	velocity, soil, and carried soil	graph	water flows into rivers	based on user preferences	no	no	no	regions of less soil	no
WFA 2011	drops of water	longitude, latitude, three different types of memory, and force of gravity	multidimensional search space	hydrological cycle in meteorology+ erosion process	random locations	no	no	no	lower altitudes	no
WCA November 2012	streams	location	multidimensional search space	water cycle + streams/rivers drift toward the sea	random positions for streams	no	yes	no	rivers and sea positions	no
WWO 2015	waves	height and wavelength	multidimensional search space	water waves motion phenomena	random locations	no	no	no	shallow water	yes

Algorithm, Year	**Entity type**	**Entity attributes or properties**	**Problem representation**	**Source of inspiration**	**Initial values of entities**	**Direct communication**	**Depend on global best solution**	**Accept bad solution**	**movement**	**Changing number of entities**
CRO June 2010	molecules	potential and kinetic energies, molecular structures	multidimensional search space	interactions of molecules in a chemical reaction	random locations	no	no	no	point of minimum energy	yes
GA 1975	chromosome	fitness	a vector or string	natural selection process and evolution	random values	no	no	no	better fitness	no
ACO 1991	ants	tracking of ant movements	graph	ants' foraging mechanism	based on user preferences	indirect	no	no	paths with more pheromone	no
PSO 1995	particle	velocity and position	multidimensional search space	social behaviour of bird flocking	random values	no	yes	no	movement towards *gbest*	no
ABC 2005	bees	position	multidimensional search space	bees' foraging mechanism	random locations	yes	yes	no	paths with more nectar	no
BA 2005	bees	position	multidimensional search space	bees' foraging mechanism	random locations	no	yes	no	top-elite positions	no
FA 2008	firefly	position and light	multidimensional search space	flashing behaviour of fireflies	random locations	no	yes	no	brightest entities	no
CS 2009	Cuckoo bird	number of eggs, nest locations	multidimensional search space	breeding behaviour	random locations	no	yes	no	high probability	no
BA 2010	bats	velocity and position of pulse rates (frequency or wavelength) and loudness	multidimensional search space	bats' echolocation system - find prey/food	random locations	no	yes	no	random walk + loudness and emission rate	no
TS 1986	list	tabu list	set of lists	local search combined with memory usage	random solutions	no	-	no	finding new better solution	no
FWA 2010	fireworks	position, explosion amplitude, and number of sparks	multidimensional search space	explosion mechanism and sparks generation	random locations	no	yes	no	Strong explosions	no
GEM 2010	grenades	position, radius, and number of shrapnel pieces	multidimensional search space	explosion mechanism and shrapnel generation	random locations	no	yes	no	highest-loss areas	no

As can be seen in Table 2-8, algorithms based on natural phenomena share many similar features. Some of these features have been improved or extended to deal with different types for problems. However, the performance of each algorithm must be experimentally evaluated on a specific problem in order to conduct comparisons.

The review of these algorithms shows that nature-inspired swarm approaches tend to introduce evolutionary concepts of mutation and crossover for sharing information or escape local optima, to adopt non-plausible global data structures to prevent convergence to local optima or add more centralised control parameters to preserve the balance between exploration and exploitation in increasingly difficult problem domains, thereby compromising self-organization. When such approaches are modified to deal with COPs, complex and unnatural representations of numbers may also be required that add to the overheads of the algorithm as well as leading to what may be appear to be ‘forced’ and unnatural ways of dealing with the complexities of a continuous problem.

2.4 Conclusions

Nature is a rich source of inspiration for designers of optimisation algorithms. These robust natural sources dynamically adapt to environmental changes in the optimal manner. By observing and studying these systems, computer scientists and engineers have designed new and powerful optimisation algorithms. Essentially, the features of a natural process or phenomenon (conceptual framework) are transformed into a computational process or information processing for problem-solving. However, some natural sources are better viewed or described as an optimisation problem than other sources.

The NIAs field has advanced under the continuous demand for new algorithms with different structures that can solve new problems with new features. The usability of novel algorithms, and their applicability to large-size problems, has been facilitated by massive improvements in computational power. Among the huge number of developed algorithms, choosing the most suitable algorithm for a specific problem is a difficult task. There are no guidelines are available though it is generally accepted that certain algorithms are more applicable to specific types of problems than others.

The literature review also revealed the similarities among various algorithms inspired wholly or partially by the same inspiration source. The success of an algorithm, which largely determines its popularity, depends on proper algorithm design, ease of use of the algorithm, and appropriate exploration–exploitation balance. Algorithm design must also consider the trade-off between the computational time and solution quality. Solution quality can be improved by modifying the algorithm procedure, adapting the computational equations, and/or introducing other artificial factors. There is always room for improvement in any algorithm. Researchers continually seek better ways of adjusting the parameter values or updating them dynamically. Proper representation of the problem also crucially affects the performance of the algorithm.

Finally, although researchers dispute the need for further algorithms, fruitful ideas and systems that will inspire new developments are waiting to be discovered. A good example is the natural hydrological water cycle, in which flowing water seeks the shortest path towards oceans or seas while avoiding obstacles. More details are provided in the next chapter.

Chapter 3 Hydrological Cycle Algorithm

"Look deep into nature, and then you will understand everything better"

[Albert Einstein]

3.1 Introduction

This chapter presents the developing steps in the development of the HCA for solving optimisation problems. The HCA simulates the movement of water through the hydrological cycle.

Section 3.2 provides an overview of the water cycle in nature, explaining in detail the main stages of this cycle. Section 3.3 addresses the HCA's main structure and its mathematical model. Section 3.4 illustrates the differences between the IWD and the HCA and is supported by comparing experiments. Finally, section 3.5 summarises a conceptual comparison between the HCA, water-like algorithms, and other established nature-inspired algorithms. This comparison aims to highlight the major differences and similarities between HCA and other water-based algorithms in terms of their structures.

3.2 Hydrological Water Cycle in Nature

Water is one of the world's most valuable resources, and forms around 71% of the earth. In nature, water is always moving. This movement is called the water cycle, and is one of the most complicated and important systems on earth. The water cycle, also known as the hydrologic cycle, describes the continuous movement of water in the environment where every single drop of water on earth takes part in this cycle (Davie, 2008). The term hydrology means the study of water, in which the term "hydro" is originated in Greek and means water, and the term "logy" means study. It is renewable because water particles are constantly circulating from the land to the sky and vice versa. Therefore, the amount of water remains constant. Water droplets move from one place to another by natural physical processes, such as evaporation, condensation, precipitation, and runoff. In these processes, the water passes through different states as shown in Figure 3-1.

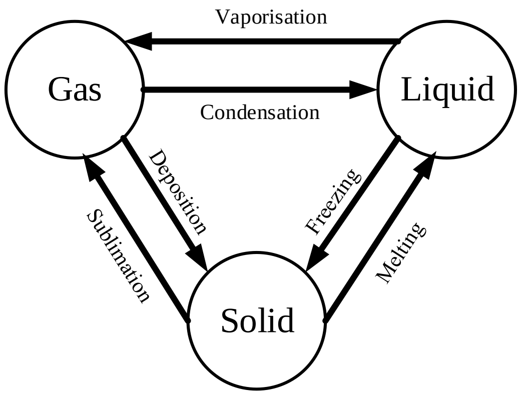


Figure 3-1. The states of the water (Splung, 2015)

The water cycle supports us with fresh and pure water, also it is important to both the weather and the climate and redistributes water around the earth. This cycle is powered by solar radiation. Therefore, the water cycle starts when the surface water gets heated by the sun, causing the water from different sources to change into vapour and evaporate into the air (atmosphere). The water vapour then cools and condenses as it rises to become drops. These drops combine with each other and eventually become so saturated and dense, then they fall back because of the force of gravity in a process called precipitation. The resulting water flows on the surface of the earth into ponds, lakes, and oceans where it again evaporates back into the atmosphere. Then, the entire cycle is repeated. This cycle is described in Figure 3-2 (Perlman, 2015).

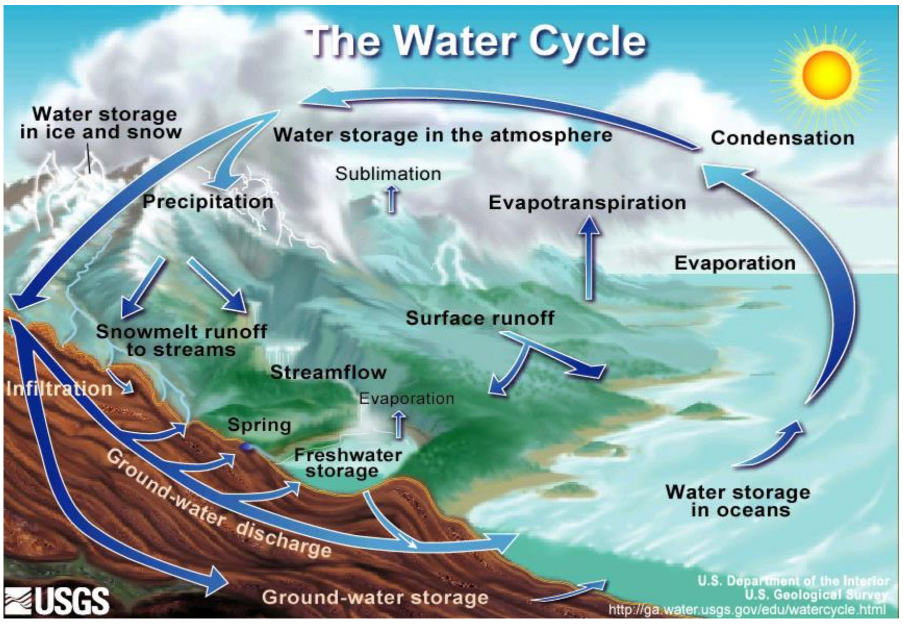


Figure 3-2. The hydrological water cycle (Perlman, 2015).

The water balance can be expressed by a mathematical equation that describes the hydrologic processes within a time period, as follows (Davie, 2008):

$$P = R + E + T + \Delta S \tag{3.1}$$

Or,
$$P \pm R \pm G \pm E \pm \Delta S = 0 \tag{3.2}$$

where, *P*: Precipitation, *R*: Surface Runoff, *G*: Groundwater flow, *E*: Evaporation, *T*: Transpiration, *ΔS*: change in storage in a specific time period.

Here, the research focuses on the main four stages of the water cycle, which are:

- Flow (Runoff)
- Evaporation
- Condensation
- Precipitation

3.2.1 Flow (runoff)

The general process of water flow and its actions are explained previously in Chapter 2 Section 2.2.1.4.3. Water movement plays an important part in reshaping the ground's surface. This is because flowing water drops are full of energy and are able to carve the ground during their movement, forming streams and rivers.

As the water flows down rivers and streams, it collects sediment and transports it away from the original location. This is known as erosion and deposition. These processes change the topography of the ground by forming new paths with more soil removal, or the fading of other paths as they become less likely to be chosen as a result of too much soil deposition. These processes are highly dependent on the velocity of water. For instance, a river is continually picking up and dropping off sediment from one point to another. When the river flow is fast, soil particles are picked-up, and the opposite happens when the river flow is slow due to the insufficient of its velocity. Moreover, there is a strong relationship between the water's velocity and the suspension load (i.e., the amount of dissolved soil in the water) that it currently carries. The water velocity is higher when only a small amount of soil is carried as opposed to when a large amount of soil is carried.

Water flow occurs when rain falls on soil that is fully saturated or when a large amount of rain falls over a short period. Typically, the flow is estimated as the difference between the precipitation and the evaporation that does not infiltrate the soil and therefore moves over the surface. There are several factors that affect the amount of water flow on the ground, such as rainfall amount, rainfall duration, soil type, land topography, temperature, wind, and humidity (Maidment, 1993).

In addition, the term “flow rate” or “discharge” can be defined as the volume of water in a river passing a defined point over a specific period (Southard, 2006). Therefore, the flow rate equals flow velocity multiplied by the cross-sectional area, see Figure 3-3 (Perlman, 2015).

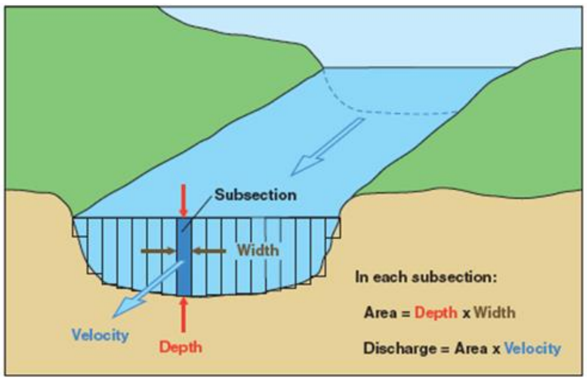


Figure 3-3. The relationship between discharge, velocity, and area (Perlman, 2015).

The cross-sectional area is calculated by multiplying the depth by the width of the stream. Therefore, the velocity of the flowing water is defined as the quantity of discharge divided by the cross-sectional area (Eq. (3.4)). In nature, rivers have a non-uniform depth and width along their length.

$$Q = V \times A \tag{3.3}$$

$$V = \frac{Q}{A} \Rightarrow V = \frac{Q}{W \times D} \tag{3.4}$$

$$D = \frac{Q/V}{W} \Rightarrow D = \frac{Q}{V \times W} \tag{3.5}$$

where Q is flow rate, A is cross-sectional area, V is velocity, W is width, and D is depth. Based on these formulas, it can be noted that there is an inverse relationship between the velocity and the cross-sectional area (Colby, 1961). The velocity increases when the cross-sectional area is small, and vice versa. If we assume that the river has a constant flow rate (velocity × cross-sectional area is constant) and that the width of the river is fixed, then we can conclude that the velocity is inversely proportional to the depth of the river, in which case the velocity decreases in deep water and vice versa. Therefore, the amount of soil deposited increases as the velocity decreases. This explains why less soil is taken from deep water, and more soil is taken from shallow water. A deep river will have water moving more slowly than a shallow river. Even with this explanation, the velocity may be varied within the different layers of the river. The velocity will be faster near the water surface compared with the velocity near the riverbed due to the friction with the riverbed material. A plot of flow velocity versus depth of flow is called the velocity profile, which is illustrated in Figure 3-4 (Southard, 2006).

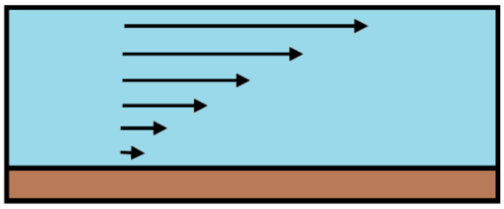


Figure 3-4. Flow velocity versus depth

As identified earlier, the water velocity keeps fluctuating within the different positions in the river over time. In addition to river depth, there are other factors affecting the velocity of water drops and causes them to accelerate or decelerate. These factors include obstructions, amount of soil existing in the path, topography of the land, force of gravity, variations in the channel cross-sectional area, and the amount of dissolved soil being carried (the suspension load).

3.2.2 Evaporation

Evaporation occurs when solar radiation hits the surface of water or land leading to an increase in temperature. When water reaches a sufficient temperature (i.e. maximum evaporation point of 100°C), it changes from a liquid state to a gaseous state. The vapour (gas) rises into the atmosphere, leaving behind impurities (salt and soils). When a direct change occurs from the solid to the gaseous state without passing through the liquid state, it is called sublimation. Water can also evaporate from other sources, such as from plants through transpiration.

According to Bishop and Locket (2002), water particles are in continuous motion on the surface of the water. The source of this motion is the kinetic energy of these particles. This energy varies from one particle to another. Kinetic energy increases with an increase in temperature (kinetic energy of a particle α particle temperature) thus, a particle that has a high kinetic energy has a greater chance of breaking its connection with other particles and escaping into the atmosphere. Therefore, the number of particles that evaporate increases as the temperature increases. This kinetic energy can be transferred from one particle to another if they collide, and the amount of energy transmitted depends on how they collide. The water temperature decreases when the particles with a high temperature leave the water in a process called evaporative cooling (Bishop & Locket, 2002).

Evaporation is very important in the water cycle as it ensures the system remains in balance. In addition, the evaporation process helps to decrease the temperature and keep the air moist and has an important influence on weather and climate. There are some important factors which affect the evaporation rate (i.e., the amount of liquid changing to gas per second) in addition to temperature. The factors affecting evaporation rate are: temperature, surface area, wind speed, air pressure, humidity, and density. Generally, in nature, it is difficult to measure the precise evaporation rate due to the existence of different sources of evaporation; hence, estimation techniques are required (Davie, 2008).

3.2.3 Condensation

In this process, the water vapour changes from a gaseous state to a liquid state. Condensation increases when the water vapour cools to 0°C, because of the low temperature in the atmosphere. This process plays an important role in the formation of clouds. Additionally, the condensation process affects air circulation in which hot air rises and cold air descends.

Condensation is an essential process as it completes the water cycle. When water vapour condenses, it releases the same amount of thermal energy initially required to convert it vapour. This energy is then released into the environment again. The condensation process is the opposite of the evaporation process. Particles with high kinetic energy and high temperature have a greater chance to evaporate (Bishop & Locket, 2002). The amount of heat that is removed at the evaporation stage is strongly related to the amount of heat that is added at the condensation stage.

The condensation process is important in the formation of new and pure water particles. The process starts when water vapour rises into the sky; then, as the temperature decreases in the higher layer of the atmosphere the particles with a lower temperature have a greater chance to condense (Bishop & Locket, 2002). Figure 3-5 (a) shows that there are no attractions (or connections) between the particles at high temperature. As the temperature lowers, the particles collide and combine with each other and start to form small clusters as shown in Figure 3-5 (b).

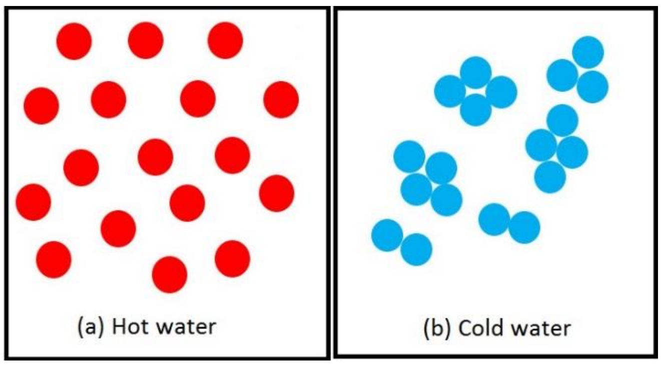


Figure 3-5. Illustration of the effect of temperature on water droplets (Bishop & Locket, 2002)

Evaporation and condensation are competing processes and occur sequentially. The condensation rate (the number of gas particles that return to liquid state per second) depends largely on the vapour pressure and increases if the temperature of the gas is decreased.

An interesting phenomenon in which some of the stronger (larger) water drops may eliminate other weaker (smaller) water drops occurs during the condensation process as some of the water drops—known as collectors—fall from a higher layer to a lower layer in the atmosphere (the troposphere). Figure 3-6 depicts the action of a water drop falling and colliding with smaller water drops resulting in water drop growth as a result of coalescence with other water drops.

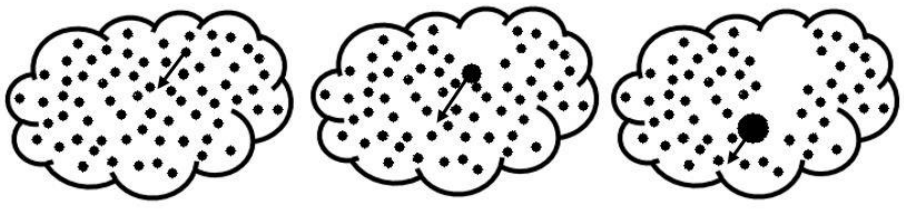


Figure 3-6. A collector water drop in action (Wallace & Hobbs, 2006).

Only water drops that are sufficiently large will survive the trip to the ground without vanishing in the lower layers of the atmosphere. Of the numerous water droplets in the cloud, only a portion will make it to the ground.

3.2.4 Precipitation

Precipitation is the process of water drops falling from the atmosphere to the ground. Different forms of precipitation such as drizzle, rain, hail, snow, sleet, and freezing rain may occur based on the surrounding temperature. Precipitation occurs as a result of the continuous condensation of water vapour to form water drops. These drops grow by colliding and coalescence with each other, until they become heavier and then fall as a result of the force of gravity.

Precipitation is important to complete the water cycle. It supports the earth with fresh and pure water and helps to maintain the balance in the atmosphere. This planet would be a massive desert without precipitation. The chance of precipitation increases when the atmosphere is saturated with water vapour. The precipitation rate can be determined by the amount and the duration of rainfall.

The process of precipitation may occur in different places and in different amounts. In addition, raindrops vary in velocity and size. After the precipitation process, water drops can spread in several ways. Some water drops may flow on the ground; others may penetrate the soil to form groundwater or may be intercepted by plants.

3.3 The Proposed Algorithm

Given the brief description of the hydrological cycle above, the IWD algorithm can be interpreted as covering only one stage of the overall water cycle—the flow stage. Although the IWD algorithm takes into account some of the characteristics and factors that affect the flowing water drops through rivers, and the actions and reactions along the way, the algorithm neglects other features of the full hydrogeological process that could be useful in solving optimisation problems.

Studying the hydrological water cycle along with the IWD algorithm gave us the inspiration to design a new and more powerful algorithm within which the original IWD algorithm (with modifications) can be considered a subpart, see Figure 3-7.

The proposed algorithm called Hydrological Cycle Algorithm (HCA) is based on a simulation of the water movement through the hydrological cycle. Therefore, it is a water-based algorithm. To emulate the movement of the water drops in this natural hydrologic water cycle, the algorithm has been divided into four main stages: Flow (Runoff), Evaporation, Condensation, and Precipitation. Each stage will have the same functionality as the corresponding stage in the natural hydrological cycle. Each stage also has a role in constructing the solution. A mathematical model was designed for the main activities that occur at each stage. These stages work to complement each other and occur sequentially. The output of the current stage is considered as input to the next stage.

In our first development of HCA, we considered temperature to be the main factor driving the water cycle. The whole cycle starts when there is a change in the temperature. We assumed that the algorithm starts with the flow stage.

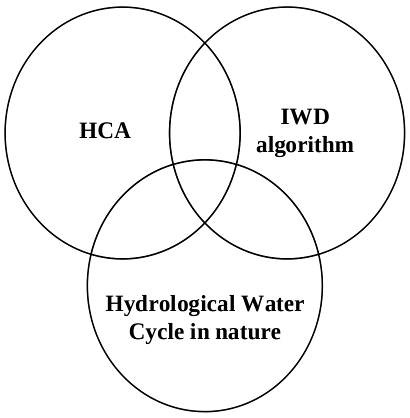


Figure 3-7. Relationship between water cycle, IWD algorithm and HCA

The HCA can be also considered to be a swarm-intelligence optimisation algorithm, as it simulates the behaviour of a collection of water drops moving through the water cycle. In swarm algorithms, a number of cooperative homogenous entities explore and interact with the environment to achieve a goal, which is usually to find the global-optimal solution to a problem through exploration and exploitation. Cooperation can be accomplished by some form of communication that allows the swarm members to share exploration and exploitation information to produce high-quality solutions (al-Rifaie et al. 2012; Hogg and Williams 1993). The aim of exploration is to acquire as much information as possible about promising solutions, while the exploitation aims to improve such promising solutions. Controlling the balance between exploration and exploitation is usually considered critical when searching for more refined solutions as well as more diverse solutions. Also important is the amount of exploration and exploitation information to be shared, and when.

Information sharing in nature-inspired algorithms usually includes metrics for evaluating the usefulness of information and the mechanisms for how it should be shared. In particular, these metrics and mechanisms should not contain more knowledge or require more intelligence than can be plausibly assigned to the swarm entities, where the emphasis is on emergence of complex behaviour from relatively simple entities interacting in non-intelligent ways with each other. Information sharing can be done either randomly or non-randomly among entities. Different approaches are used to share information such as direct or indirect interaction. Two sharing models are: one-to-one, in which explicit interaction occurs between entities; and many-to-many (broadcasting), in which interaction occurs indirectly between the entities through the environment (Ding et al. 2011).

Usually, either direct or indirect interaction between entities is employed in an algorithm for information sharing, and the use of both types in the same algorithm is often neglected. For instance, the ACO algorithm uses indirect communication for information sharing, where ants lay amounts of pheromone on paths depending on the usefulness of what they have explored. The more promising the path, the more pheromone is added, leading other ants to exploit this path further. Pheromone is not directed at any other ant in particular. In the artificial bee colony (ABC) algorithm, bees share information directly (specifically, the direction and distance to flowers) in a kind of waggle dancing (Goel & Panchal, 2013). The information received by other bees depends on which bee they observe and not the information gathered from all bees. In a GA, the information exchange occurs directly in the form of crossover operations between selected members (chromosomes and genes) of the population. The quality of information shared depends on the members chosen and there is no overall store of information available to all members. In PSO, on the other hand, the particles have their own information as well as access to global information to guide their exploitation. This can lead to competitive and cooperative optimisation techniques (Yuhua Li, Zhan, Lin, Zhang, & Luo, 2015). However, no attempt is made in standard PSO to share information possessed by individual particles. This lack of individual-to-individual communication can lead to early convergence of the entire population of particles to a non-optimal solution. Selective information sharing between individual particles will not always avoid this narrow exploitation problem but, if undertaken properly, could allow the particles to find alternative and more diverse solution spaces where the global optimum lies. As can be seen from the above discussion, the distinctions between communication (direct and indirect) and information sharing (random or non-random) can become blurred.

In HCA, we utilise both direct and indirect communication to share information among selected members of the whole population. Direct communication occurs in the condensation stage of the water cycle, where some water drops collide with each other when they evaporate into the cloud. On the other hand, indirect communication occurs between the water drops and the environment via the soil and path depth. Utilising information sharing helps to diversify the search space and improve the overall performance through better exploitation.

Furthermore, in some algorithms, indirect communication is used not just to find a possible solution but also to reinforce on previous promising solutions. In some cases, such reinforcement may lead the algorithm to getting stuck in the same solution, which is known as *stagnation behaviour* (Mavrovouniotis & Yang, 2010). Such reinforcement can also cause a premature convergence in some problems. Therefore, an important feature used in HCA to avoid this problem, which is the path depth. The depths of paths are constantly changing, and deep paths will have a lower chance of being chosen compared with shallow paths. More details will be provided about this feature later in this chapter.

The HCA can be described formally as consisting of a set of artificial *water drops* (WDs), such that

$$HCA = \{WD_1, WD_2, WD_3, \ldots, WD_n\}, \text{ where } n \geq 1$$

The characteristics associated with each water drop are as follows:

- V^{WD}: The velocity of the water drop.
- $Soil^{WD}$: The amount of soil the water drop carries.
- ψ^{WD}: The solution quality of the water drop.

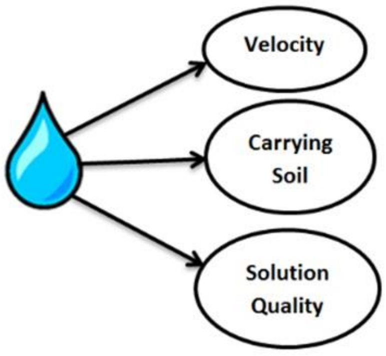


Figure 3-8. Water drop characteristics

Typically, the input to the HCA is any forms of a graph representation of the problem solution space. The graph can be a fully connected undirected graph $G = (N, E)$, where N is the set of nodes and E is the set of edges between the nodes. In order to find a solution, water drops are distributed randomly over the nodes of the graph and made to traverse the graph through the edges to find the best solution. The characteristics associated with each edge are the initial amount of soil and edge depth.

The initial amount of soil is the same for all the edges. The depth measures the distance from the water surface to the riverbed, and can be calculated by dividing the length of the edge by the amount of soil. Therefore, the depth varies and depends on the amount of soil that exists on the path and its length. The depth of the path increases when more soil is removed over the same length. A deep path can be interpreted as either the path being lengthy or there being less soil. Figure 3-9 and Figure 3-10 both illustrate the relationship between path depth, soil, and path length.

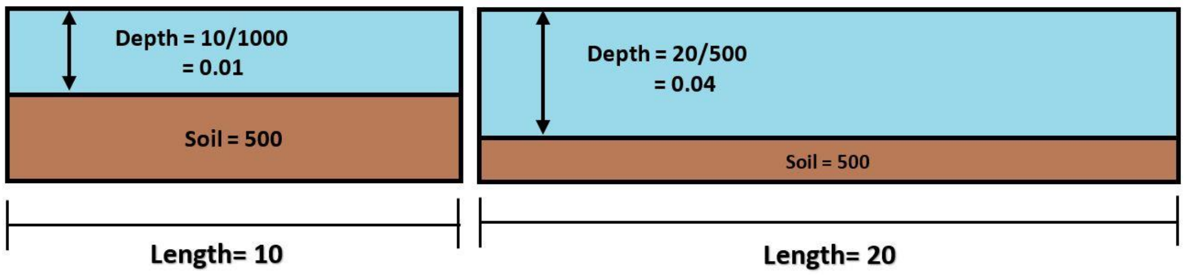


Figure 3-9. Depth increases with length increases for the same amount of soil

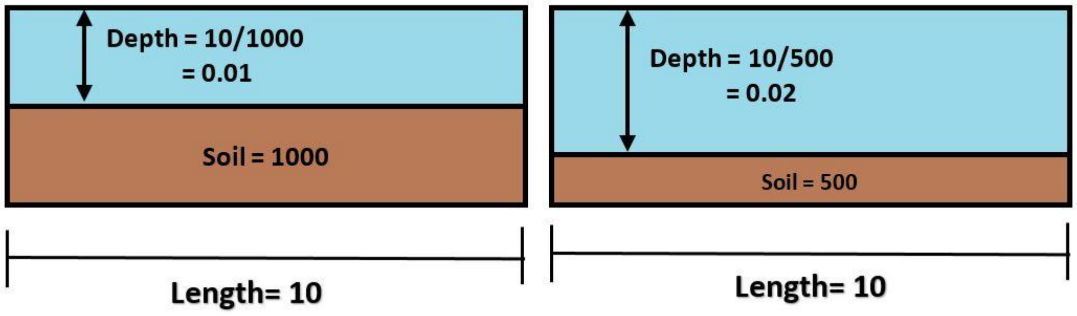


Figure 3-10. Depth increases when the amount of soil is reduced over the same length

The HCA goes through a number of cycles and iterations to find a solution to a problem. One iteration is considered complete when all water drops have generated solutions based on the problem constraints. A water drop iteratively constructs a solution for the problem by continuously moving between the nodes. Each iteration consists of specific steps (which are explained below). On the other hand, a cycle represents a varied number of iterations. The main advantage of considering a cyclic scheme in addition to the iterations is the utilisation of the execution time. Because the algorithm has a cycle, some improvements operations can be applied every cycle (i.e., every few iterations) instead of every iteration. In other algorithms, these operations have to be executed at the end of each iteration. Avoiding end of iteration operations helps in minimising the overall execution time required to find a solution. In addition, this gives the algorithm the chance to improve the results obtained from certain iterations. Finally, the cycle prevents the algorithm from being stuck in the same solutions by re-initialising some of the parameters every cycle.

A cycle starts when the temperature reaches a specific value, which makes the water drops evaporate, condense, and precipitate again. This procedure continues until the termination condition is met. The functionality of each of these stages answers one of the following questions:

- How to choose the next node to visit?
- When the water drops will evaporate?

- How many water drops will evaporate?
- Which water drops should evaporate? (How to choose among the water drops?)
- What is the purpose of the condensation stage?
- What is the purpose of the precipitation stage?

3.3.1 Flow Stage (Runoff)

This stage represents the construction stage where the HCA explores the search space. This is carried out by allowing the water drops to flow (scattered or regrouped) in different directions and constructing various solutions. Each water drop constructs a solution incrementally by moving from one point to another. Whenever a water drop wants to move toward a new node, it has to choose between different numbers of nodes (various branches). It calculates the probability of all the unvisited nodes, and chooses the highest probability node taking into consideration the problem constraints. This can be described as a state transition rule that determines the next node to visit. In the HCA, the node with the highest probability will be selected. The probability of the node is calculated using Eq. (3.6):

$$P^{WD}_{i}(j) = \frac{f\left(Soil(i,j)\right)^2 \times g\left(Depth(i,j)\right)}{\sum_{k \notin vc(WD)} \left(f\left(Soil\ (i,k)\right)^2 \times g\left(Depth(i,k)\right) \right)} \tag{3.6}$$

where $P^{WD}_{i}(j)$ is the probability of choosing node *j* from node *i*, and *vc* is the visited list of each water drop. The *f(Soil(i, j))* is equal to the inverse of the soil between *i* and *j*, and is calculated using Eq. (3.7). The g(Depth (i, k) is the inverse of the depth and calculated using Eq. (3.8).

$$f\left(Soil(i,j)\right) = \frac{1}{\mathcal{E} + Soil(i,j)} \tag{3.7}$$

Ɛ=0.01 is a small value that is used to prevent division by zero. The second factor of the transition rule is the inverse of depth, which is calculated based on Eq. (3.8).

$$g\left(Depth(i,j)\right) = \frac{1}{Depth(i,j)} \tag{3.8}$$

Depth (*i*, *j*) is the depth between two nodes *i* and *j*, and calculated by dividing the length of the path by the amount of soil. This can be expressed as follows:

$$Depth(i,j) = \frac{Length\ (i,j)}{Soil\ (i,j)} \tag{3.9}$$

The depth values might have a very small value. Therefore, it can be normalised to be within a specific range (i.e. [1–100]) to be consistent with the values of other variables. The depth of the path is constantly changing because it is influenced by the amount of existing soil, where the depth is inversely proportional to the amount of the existing soil in the path of a fixed length. The consideration of depth helps when the soil amounts are the same on different paths, but paths have different depths, especially in the first few iterations. Therefore, a water drop will choose a shallower path over a deeper path. This offers the opportunity to explore new paths and diversify the generated solutions. This also simulates the behaviour of the flowing water on the ground, where the water starts to spread over the ground until some of the paths are carved as a result of more water flowing on these paths. Let us assume the following scenario (depicted by Figure 3-11), where water drops have to choose between nodes *A* or *B* after node *S*.

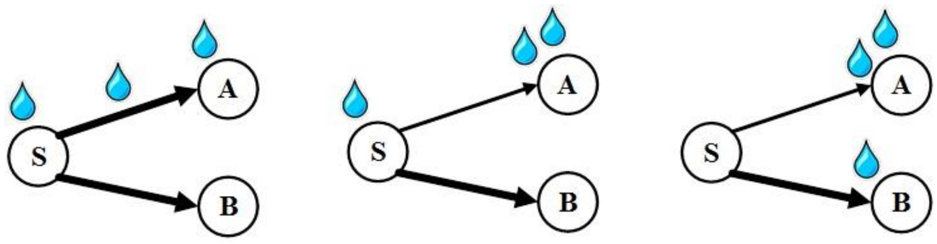


Figure 3-11. The relationship between soil and depth over the same length

Initially, both edges have the same length and same amount of soil (line thickness represents soil amount). Consequently, the depth values for the two edges will be the same. After some water drops chose node *A* to visit, the edge (*S*-*A*) as a result has less soil and is deeper. According to the new state transition rule, the next water drops will be forced to choose node *B* because edge (*S*-*B*) will be shallower than (*S*-*A*). This technique provides a way to avoid stagnation and explore the search space efficiently.

3.3.1.1 Velocity Update

After selecting the next node, the water drop moves to the selected node and marks it as visited. Each water drop has a variable velocity (*V*). Here, it is more accurate to use the term "velocity" instead of "speed", where the water drop velocity can be defined as the speed with a direction; or the rate of changing of its position (i.e., the speed of an object is the magnitude of its velocity) (Colby, 1961). The velocity of a water drop might be increased or decreased while it is moving based on the factors mentioned above. Mathematically, the velocity of a water drop at time (*t*+1) can be calculated using Eq. (3.10).

$$V_{t+1}^{WD} = [K \times V_t^{WD}] + \alpha\left(\frac{V_t^{WD}}{Soil(i,j)}\right) + \sqrt[2]{\frac{V_t^{WD}}{Soil^{WD}}} + \left(\frac{100}{\psi^{WD}}\right) + \sqrt[2]{\frac{V_t^{WD}}{Depth\,(i,j)}} \qquad (3.10)$$

Equation (3.10) defines how the water drop updates its velocity after each movement, which is illustrated in Figure 3-12.

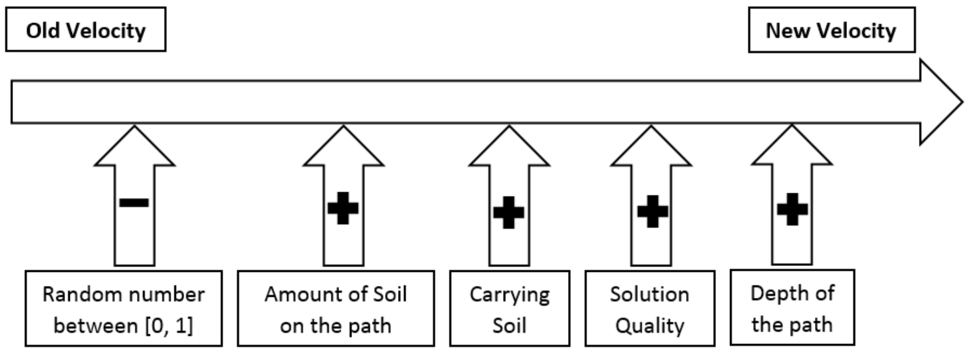


Figure 3-12. Factors that affect the new value of velocity

Each term of the equation influences the new velocity of the water drop, where V_t^{WD} is the current water drop velocity, and K is a uniformly distributed random number between [0, 1] that refers to the roughness coefficient. The roughness coefficient represents factors that may cause the water drop velocity to decrease, like path twist, hills, soil hardness, gravitational acceleration, and other obstacles that exist in their path. Owing to difficulties measuring the exact effect of these factors, some randomness is considered with the velocity.

The second term of the expression in Eq. (3.10) reflects the relationship between the amount of soil existing on a path and the velocity of a water drop. The velocity of the water drops increases when they traverse paths with less soil. Alpha (α) is a relative influence coefficient that emphasises this term in the velocity update equation and helps the water drops to emphasise and favour the path with fewer soils over the other factors. The third term of the expression represents the relationship between the amount of soil that a water drop is currently carrying and its velocity. Water drop velocity increases with decreasing amount of soil being carried. This gives weak water drops a chance to increase their velocity, as they do not hold much soil. The fourth term of the expression refers to the inverse ratio of a water drop's fitness, with velocity increasing with higher fitness. The final term of the equation indicates that a water drop will be slower in a deep path than in a shallow path.

3.3.1.2 Soil Update

Next, the amount of soil existing on the path and the depth of that path are updated. A water drop can remove (or add) soil from (or to) a path while moving based on its velocity. This is expressed by Eq. (3.11).

$$Soil(i,j) = \begin{cases} [PN * \ Soil(i,j)] - \ \Delta Soil(i,j) - \sqrt[2]{\dfrac{1}{Depth(i,j)}} & if\ V^{WD} \geq Avg\left(all_V{}^{WDS}\right)\ (Erosion) \\ [PN * \ Soil(i,j)] + \Delta Soil(i,j) + \sqrt[2]{\dfrac{1}{Depth(i,j)}} & else\ (Deposition) \end{cases} \tag{3.11}$$

PN represents a coefficient (i.e., sediment transport rate, or gradation coefficient) that may affect the reduction in the amount of soil. The soil can be removed only if the current water drop velocity is greater than the average of all other water drops. Otherwise, an amount of soil is added (soil deposition). This simulates the situation in which a water drop is able to carry more soil if its velocity is high enough, otherwise, it will deposit some the soil it's carrying. Consequently, the water drop is able to transfer an amount of soil from one place to another, and usually transfers it from fast to slow parts of the path. The increasing soil amount on some paths favours the exploration of other paths during the search process and avoids entrapment in local optimal solutions. The rate of change in the amount of soil existing between node *i* and node *j* depends on the time needed to cross that path, which is calculated using Eq. (3.12).

$$\Delta Soil(i,j) = \frac{1}{time_{i,j}^{WD}} \tag{3.12}$$

such that,

$$time_{i,j}^{WD} = \frac{Distance\ (i,j)}{V_{t+1}^{WD}} \tag{3.13}$$

Equation (3.12) shows that the amount of soil being removed is inversely proportional to time. Based on the second law of motion, time equals distance divided by velocity. Therefore, the time is proportional to velocity and inversely proportional to path length.

Furthermore, based on equation (3.11), the amount of soil being removed or added is inversely proportional to the depth of the path. This is because shallow soils will be easy to remove compared to deep soils. Therefore, the amount of soil being removed will decrease as the path depth increases. This is because the velocities of the water drops are higher in the upper layers of the channel compared to the low velocity in the lower layers, see Figure 3-13. This factor facilitates exploration of new paths because less soil is removed from the deep paths as they have been used many times before. This may extend the time needed to search for and reach best-known solutions.

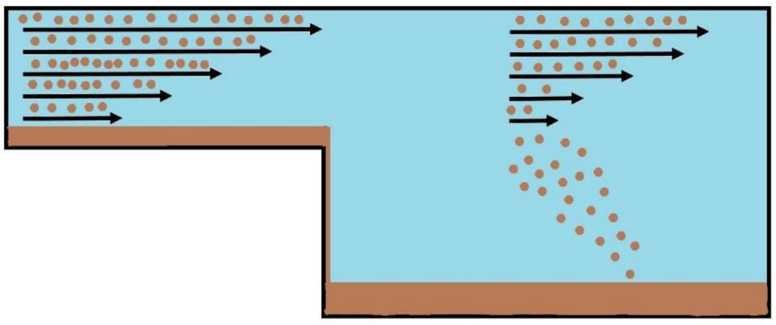


Figure 3-13.The relationship between the soil deposition and the depth as the velocity is decreased

The depth of the path increases with more soil being removed (less amount of soil over the same length will make it deeper, more soil over the same length will make it shallower). The depth of the path needs to be updated when the amount of soil existing on the path changes. It is worth clarifying that the depth factor is not limited to the relationship between the soil and the distance. The depth can be customised according to the problem specification (i.e., problem heuristic) or user preferences. For example, for the job scheduling problem, the depth can represent the time interval between the jobs.

3.3.1.3 Carrying Soil Update

In the HCA algorithm, we considered that the amount of soil the water drop carries reflects its solution quality. This can be done by embedding the quality of the water drop's solution with its carrying soil value. Figure 3-14 illustrates the fact that a water drop with more soil carried has a better solution.

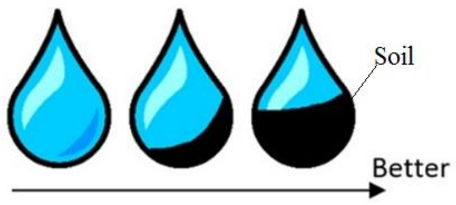


Figure 3-14. Relationship between the amount of carrying soil and its quality

To simulate this association, the amount of soil that the water drop is carrying is updated with a portion of the soil being removed from the path divided by the last solution quality found by that water drop. Therefore, the water drop with a better solution will carry more soil. This can be expressed as follows:

$$Soil^{WD} = Soil^{WD} + \frac{\Delta Soil(i,j)}{\psi^{WD}} \qquad 3.14)$$

The idea behind this step is to let a water drop preserve its previous fitness value. Later, the amount of carrying soil value is used in updating the velocity of the water drops.

3.3.1.4 Temperature Update

The final step that occurs at this stage is updating of the temperature. The new temperature value depends on the solution quality generated by the water drops in the previous iterations. The temperature will be updated as follows:

$$\mathrm{Temp}\ (\mathrm{t}+1) = \mathrm{Temp}\ (\mathrm{t}) + \Delta\mathrm{Temp} \tag{3.15}$$

where,

$$\Delta Temp = \begin{cases} \beta * \left(\dfrac{\mathrm{Temp}\ (\mathrm{t})}{\Delta D}\right) & \Delta D > 0 \\ \dfrac{\mathrm{Temp}\ (\mathrm{t})}{10} & Otherwise \end{cases} \tag{3.16}$$

and where coefficient β is determined based on the problem. The difference (ΔD) is calculated using Eq. (3.17).

$$\Delta D = MaxValue - MinValue \tag{3.17}$$

such that,

$$\begin{gathered} MaxValue = max[Solutions\ Quality(WDs)] \\ MinValue = min[Solutions\ Quality(WDs)] \end{gathered} \tag{3.18}$$

According to Eq. (3.15), the increase in temperature will be affected by the difference between the best solution (*MinValue*) and the worst solution (*MaxValue*). A lesser difference means there was not a lot of improvement in the last few iterations and, as a result, the temperature will increase (i.e., temperature is inversely proportional with solution quality). This means that there is no need to evaporate the water drops as long as they can find different solutions in each iteration. The flow stage will repeat a few times until the temperature reaches a certain value. When the temperature is sufficient, the evaporation stage begins.

3.3.2 Evaporation Stage

As in nature, evaporation plays an important role in the continuance of the water cycle. This stage is very important in preventing the algorithm from sticking with the same solution in every iteration. Therefore, in this stage, the number of water drops that will evaporate (the evaporation rate) when the temperature reaches a specific value is first determined. The evaporation rate is chosen randomly between one and the total number of water drops; this is because of the difficulty in identifying a specific rate in nature, see Eq. (3.19).

$$ER = Random\_Integer\ (1, N) \tag{3.19}$$

where *ER* is evaporation rate and *N* is number of water drops.

Alternatively, in some situation, the evaporation rate can be updated to increase gradually in a linear fashion. Therefore, the rate of evaporation can be determined based on Eq. (3.20). Identifying the best way may depend on the problem search space.

$$ER = \left\lceil \left(\frac{c}{M} \times (Pop_{size} - 1) \right) + 1 \right\rceil \tag{3.20}$$

where, ER: Evaporation rate, n = iteration number, c: current iteration, M: maximum iteration number, and Pop_{size}: population size.

Equation (3.20) shows that evaporation rate increases linearly from one up to the whole number of water drops. According to the evaporation rate, a specific number of water drops is selected to evaporate. The selection is done using the roulette-wheel selection method, taking into consideration the solution quality of each water drop. Only the selected water drops evaporate and go to the next stage. On the other hand, as the water drops evaporate the temperature value will reduce. This is similar to the evaporative cooling phenomenon in real life.

3.3.3 Condensation Stage

In this stage, as the temperature decreases, water drops draw closer to each other, then collide and combine or bounce off. These operations are useful as they allow the water drops to be in direct physical contact each other. This stage represents the collective and cooperative behaviour between all the water drops. A water drop can generate a solution without direct communication (by itself); however, communication between water drops may lead to the generation of better solutions in the next cycle. In the HCA, physical contact is used for direct communication between water drops, whereas the amount of soil on the edges and depth values can be considered indirect communication between the water drops. Direct communication (information exchange) has been proven to improve solution quality significantly when employed in other algorithms (Mavrovouniotis & Yang, 2010).

In HCA, the condensation stage is considered to be a problem-dependent stage and can be redesigned to fit the problem specification. For example, one way of implementing this stage to fit the TSP is to use local improvement methods. Using these local methods enhances the solution quality and generates better results in the next cycle (i.e., minimises the total cost of the solution); with the hypothesis that it will reduce the number of iterations needed and help to reach the optimal/near-optimal solution faster. Expression (3.21) shows the evaporated water (EW) selected from the overall population, where n represents the evaporation rate (number of evaporated water drops):

$$EW = \{ WD_1, WD_2, \cdots, WD_n\} \tag{3.21}$$

Initially, all the possible combinations (as pairs) are selected from the evaporated water drops to share their information with each other via collision. When two water drops collide, there is a chance of either the two water drops merging (coalescence) or bouncing off each other. In nature, determining which operation will take place is based on factors such as the temperature and the velocity of each water drop. In this research, we considered the similarity between the water drops' solutions to determine which process will occur. When the similarity is greater than or equal to 50% between two water drops' solutions, they merge. Otherwise, they collide and bounce off each other. Mathematically, this can be represented as follows:

$$OP(WD_1, WD_2) = \begin{cases} Bounce(WD_1, WD_2), & Similarity < 50\% \\ Merge(WD_1, WD_2), & Similarity \geq 50\% \end{cases} \tag{3.22}$$

Finding the similarity between the solutions, the measure of how close two solutions are to each other, is problem-dependent. When two water drops collide and merge, one water drop will (i.e., the collector) become more powerful by eliminating the other one and therefore acquires the characteristics of the other (i.e., its velocity). This merging operation is useful to eliminate and refine similar solutions.

$$WD1_{Vel} = WD1_{Vel} + WD2_{Vel} \tag{3.23}$$

On the other hand, when two water drops collide and bounce off, they share information with each other about the goodness of each node and how much a node contributes to their solutions. The bounce-off operation generates information that is used later to refine the water drops' solutions quality in the next cycle by emphasing on the best nodes. The information generated is available and accessible to all water drops and helps them to choose a node that has a better contribution from all the possible nodes at the flow stage. This information consists of the weight of each node. The weight measures a node contribution ratio to a solution quality, which can be customised according to the problem being solved. The idea behind sharing these details is to emphasise the best nodes found up to that point. Within this exchange, the water drops will favour those nodes in the next cycle. Mathematically, the weight can be calculated as follows:

$$Weight(node) = \frac{Node}{WD_{sol}} \tag{3.24}$$

Finally, this stage is used to update the global-best solution found up to that point. In addition to that, at this stage, the temperature value is reduced by 50° C. The lowering and rising of the temperature help to prevent the water drops from sticking with the same solution every iteration.

3.3.4 Precipitation Stage

This stage is considered the termination stage, as the algorithm checks whether the termination condition has been met. If the condition has been met, the algorithm stops with the last global-best solution. Otherwise, this stage is responsible for reinitialising all the dynamic variables, such as the amount of the soil on each edge, depth of paths, the velocity of each water drop, and the amount of soil it holds. The re-initialisation of the parameters happens after certain iterations and helps the algorithm to avoid being trapped in local optima, which may affect the algorithm's performance in the next cycle. Moreover, this stage is considered as a reinforcement stage, which is used to place emphasis on the best water drop (the collector). This is achieved by reducing the amount of soil on the edges that belong to the best water drop solution, see Eq. 3.25).

$$Soil(i,j) = 0.9 * soil(i,j)\,, \quad \forall\,(i,j) \in Best^{WD} \tag{3.25}$$

The idea behind that is to favour these edges over the other edges in the next cycle.

3.3.5 The HCA Procedure

Figure 3-15 explains the steps of the HCA algorithm, and Figure 3-16 explains the pseudocode for evaporation, condensation and precipitation stages

```
PROCEDURE: Hydrological Cycle Algorithm
% PROBLEM INPUT: Undirected graph with edge weights.
% OUTPUT: The best solution
BEGIN
Read data file
Initialisation of the parameters
Initialise static parameters
Initialise dynamic parameters
Generate initial water drops (WDs) population
Distribute WDs randomly on nodes
WHILE (termination condition is not met || iteration <
    iteration-max)
    Cycle = Cycle + 1
    % Flow stage
    REPEAT
        Iteration = iteration +1
        FOR each water drop
            Choose next Node (Eq. (3.6))
            Update velocity (Eq. (3.10))
            Update Soil and depth of the edge (Eq. (3.11))
            Update carrying soil (Eq. 3.14))
        END FOR
        Calculate the fitnesses of the solutions
        Update local optimal solution
        Update Temperature (Eq. (3.15))
    UNTIL (Temperature = Evaporation-Temperature) // end a
          cycle
    Evaporation stage (WDs)
    Condensation stage (WDs)
    Precipitation stage (WDs)
END WHILE
RETURN: solutions
END PROCEDURE
```

Figure 3-15. The HCA algorithm pseudocode

```
PROCEDURE: Evaporation (WDs)
Evaluate the solution quality
Identify Evaporation Rate (Eq.
    (3.19))
Select WDs to evaporate
END PROCEDURE
```

```
PROCEDURE: Condensation (WDs)
Information exchange (WDs)
    (Eqs. (3.21) - (3.24))
Identify The collector (WDs)
Update global solution
Update the temperature
END PROCEDURE
```

```
PROCEDURE: Precipitation (WDs)
Evaluate the solution quality
Reinitialise dynamic parameters
Global soils update (belong to best WDs) (Eq. 3.25))
Generate a new WDs population
Distribute the WDs randomly
END PROCEDURE
```

Figure 3-16.The Evaporation, Condensation and Precipitation pseudocode

3.3.6 The HCA Flowchart

Figure 3-17 represents the overall flowchart of the HCA algorithm and describes the role of each stage.

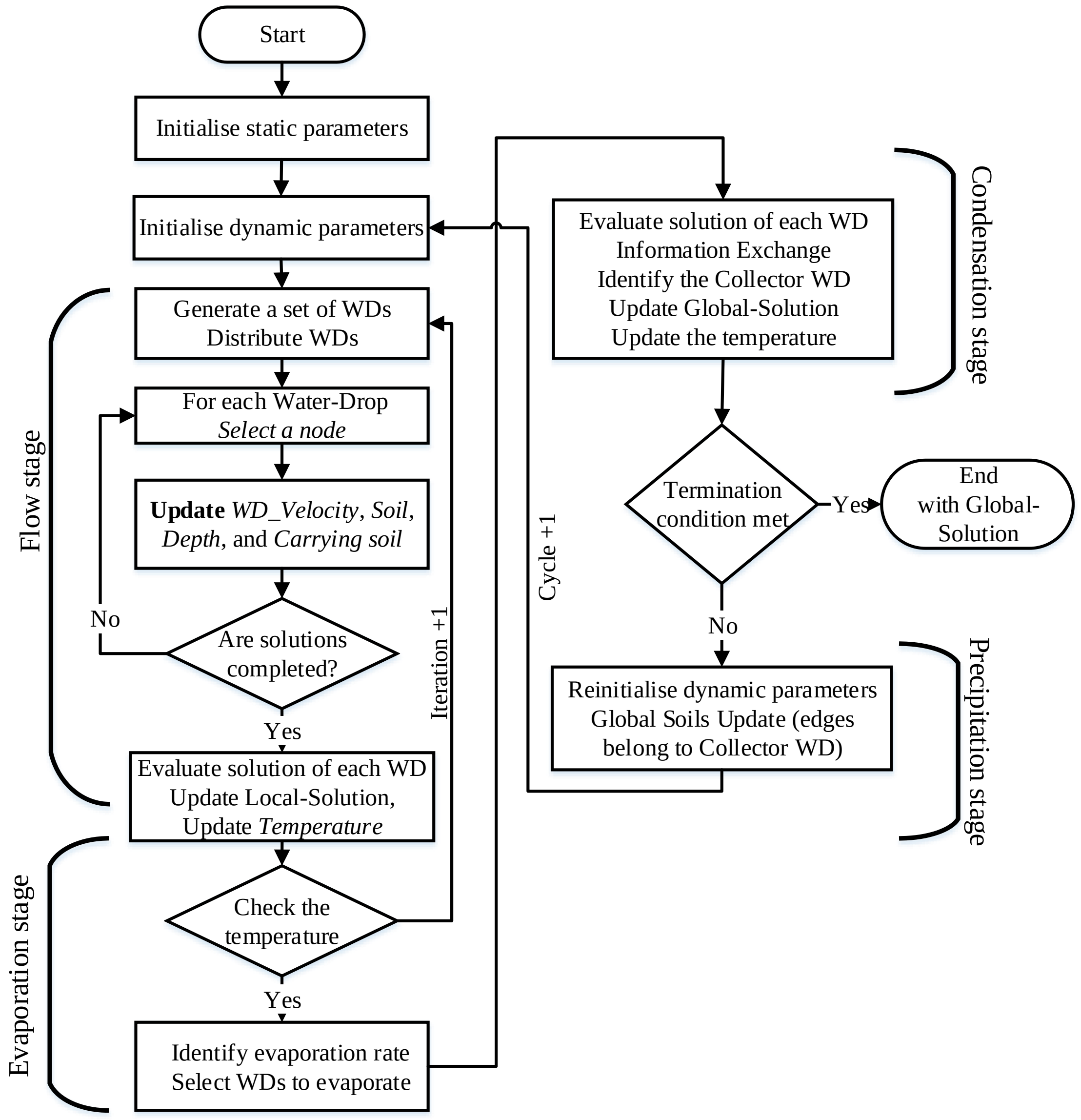


Figure 3-17. The HCA algorithm flowchart

3.3.7 Solution Generator

Basically, to find a solution, a number of artificial water drops are generated. These water drops might be placed in the same node of a graph or distributed randomly, dependant on the problem being solved. The water drops flow using the edges of the graph. The water drops build solutions in parallel, where each time a water drop chooses only one node. Then, the next water drop chooses a node, and so on. This process continues until all of the water drops visit all the nodes. Alternatively, a sequential technique can be used that allows a water drop to visit all the nodes, then the next water drop can start. Each water drop may generate a different solution during its flow. However, as the algorithm iterates, some water drops start to converge towards the best water drop solution by selecting the highest node's probability. The candidate solutions for a problem can be represented as a matrix, in which each row consists of a water drop solution.

$$Solutions = \begin{bmatrix} WD_s^1 \\ WD_s^2 \\ WD_s^i \\ \cdots \\ WD_s^k \end{bmatrix} \tag{3.26}$$

The water drop solution represents the order of the visited nodes.

$$Solutions = \begin{bmatrix} n_1 & n_2 & n_3 & \cdots & n_m \\ n_1 & n_2 & n_3 & \cdots & n_m \\ n_1 & n_2 & n_3 & \cdots & n_m \\ \vdots & & \cdots & & \\ n_1 & n_2 & n_3 & \cdots & n_m \end{bmatrix} \tag{3.27}$$

At the end of each iteration, the generated solutions are evaluated, and the best one is used to update the local-best solution. While, at the end of each cycle, the global-best solution will be updated. The edges belonging to the best solution are favoured with less amount of soil. Subsequently, the water drops start flowing and generating solutions until the termination condition is met.

3.3.8 HCA Parameters, Assumptions, and Platform

Adjusting the values of the parameters is difficult and can be considered as an optimisation problem itself. The parameter adjustment has an impact on the performance of algorithms. There are different strategies to find the best values for these parameters. For some algorithms, the parameters need to be re-tuned even if they are to be used to solve different instances of the same problem. There are two possible techniques for parameter tuning detailed in the literature. The first technique relies on conducting a series of trials, trying different combinations of the parameters values each attempt. The aim of this process is to find the best setting for these parameters. The best parameter setting is one that can produce a high-quality solution. The main drawback of the by-trial technique is that it requires a long time to achieve the best parameter values. The second technique is to use one of the well-known optimisation algorithms (i.e., genetic algorithm or neural network algorithm) to tune the new algorithm parameters (X.-S. Yang & Karamanoglu, 2013).

In the HCA, a number of parameters were considered to control the flow of the algorithm and help to generate a solution. Some of these parameters' values need to be adjusted experimentally, according to the problem domain, such as number of water drops and maximum number of iterations. In order to find the best values for these parameters, we have used structural problems that are easy to be evaluated. Table 3-1 lists the parameters and their values as used in the HCA. Some values were obtained by conducting some preliminary experiments while others are based on values used in previous studies. It is worth mentioning that these parameters and their values will be used for all experiments unless otherwise stated.

Table 3-1. HCA parameters and their values

Parameter name	Parameter value	Obtained
Number of water drops	Equal to number of nodes	experimentally
Maximum number of iterations	Double the number of nodes	experimentally
Initial soil on each edge	10000	(Shah-Hosseini, 2009)
Initial velocity	100	(Shah-Hosseini, 2007)
Initial depth	Edge length/soil on that edge	assumption
Initial carried soil	1	experimentally
Velocity updating	α=2	experimentally
Soil updating	PN= 0.99	experimentally
Initial temperature	50, β=10	experimentally
Maximum temperature	100	experimentally

Some assumptions have been made to simplify the use of this algorithm. In the first version of HCA, the width of the path has not been considered as a factor that affects the velocity value to reduce the number of parameters. Additionally, the depth value might be very small, therefore, it has been normalised to be within the range [1-100] to be within the same range of the other variables. Finally, the amount of soil has been restricted to be within a maximum and minimum value in order to avoid negative values. The maximum value set to be the initial soil value, while the minimum value is fixed at one.

The algorithm was implemented using MATLAB V.8.4 (R2014b). All the experiments were conducted on a computer with an Intel Core i5-4570 (3.20GHz) CPU and 16GB RAM, on the Microsoft Windows 7 Enterprise platform.

3.4 The Difference between HCA and IWD Algorithm

The aim of this section is to clarify the major similarities and differences between the HCA and IWD algorithms based on the mathematical formalisation of each one. Despite the fact that the IWD algorithm is used with major modifications as a subpart of the HCA, there are major differences between IWD and the HCA. Moreover, despite the two algorithms sharing the same source of inspiration – water movement in nature – they are inspired by different aspects of the water processes and events. Therefore, there are major differences between IWD and the HCA. In addition, we identify some of the IWD algorithms' weaknesses and how the HCA has addressed these issues. One of the similarities is that both algorithms use "*water drops*" as entities to generate a solution. In addition, in the HCA a water drop has velocity and can carry soil in a similar manner to IWD algorithm.

The following subsections explain, in specific detail, the differences between the algorithms. Later, the effect of the HCA modifications will be tested using structural experiments.

3.4.1 Choosing the Next Node

Choosing the next node to visit is very crucial and can have a high impact on the quality of the final solution. In particular, this can be observed when most algorithm entities keep choosing the same nodes repeatedly because there is no other factor affecting their decisions. In addition, the possibility of selecting the next node increases with an increase in the number of nodes to choose from. In the original IWD algorithm, the probability of selecting the next node is calculated based on the amount of soil (see Eq. (2.1)). Thus, a path with less soil has a better chance of being selected than a path with more soil. This is because in nature water drops always favour a path with less soil.

In contrast, in the HCA algorithm the probability of selecting the next node is an association between the soil and the depth, which enables the construction of a variety of solutions (see Eq. (3.6)). This association is useful when the same amount of soil exists on different paths; therefore, the paths' depths are used to differentiate between these edges. Moreover, this association helps to create a balance between the exploitation and exploration of the search process. The edge with a lesser depth is chosen, and this favours exploration. Alternatively, choosing the edge with less soil will favour exploitation. This advantage was found to be most notable in the first few iterations where the amount of soil is the same on all the graph edges (see section 3.4.6 experimental proof). The depth of the path can be considered to be another heuristic function, that can be customised based on the problem being solved.

3.4.2 Updating the Water Drop Velocity

The velocity of water drops plays an important role in guiding the algorithm to discover new paths, as it affects the amount of soil existing on the edges. As the velocity increases, more soil will be removed from an edge. In the original IWD algorithm, the water drops use Eq. (2.4) to update their velocity after they move from one place to another. The increase in the velocity is very small (imperceptible). Equation (2.4) of IWD uses only the amount of soil that exists on the path to increase the velocity. The update does not consider that water drop velocity might also decrease (the more soil is carried (in weight) the lower the velocity). Conversely, in HCA additional crucial factors that reflect the path situation are considered. These factors allow water drops to not only increase but also to decrease velocity (see Eq. (3.10)) and gives other water drops a chance to compete. The velocity of water drops plays an important role in guiding the algorithm to discover new paths (water drops travelling at a high velocity can carve out more paths than those at a lower velocity). Thus, the fluctuation in velocity prevents one water drop dominating the other water drops.

3.4.3 Updating the Soil Exist on Each Path

One of the main factors that affects the next edge choice is the amount of soil existing on that edge. In the original IWD algorithm, the amount of soil that exists between *i* and *j* is updated using Eq. (2.5) that removes soil. In contrast, in the HCA, the soil-update equation enables both soil removal and deposition (see Eq. (3.11)). The underlying idea is to help the algorithm to improve its exploration, avoid premature convergence, and avoid being trapped in local optima.

3.4.4 Updating the Carrying Soil for Each Water Drop

In the original IWD algorithm, the water drops use Eq. (2.8) to update the amount of soil carried by each water drop. The equation only reflects a change in the amount of soil carried. Thus, each water drop ends up carrying a huge amount of soil. In addition, the amount of soil carried (or *carrying soil*) was only used in a global soil update. In contrast, in the HCA, the carrying soil parameter is also encoded with the solution quality; more soil carried the better the solution, see Eq. 3.14). Including the solution quality in the carrying soil parameter preserves the previous quality. This carrying soil parameter, as has been explained in section 3.4.2, is also used to update water drop velocity.

3.4.5 Other Differences

The major differences between the two algorithms rely on the existence of three critical stages (i.e., evaporation, condensation, precipitation), which are associated with the changes in temperature. These stages are believed to improve the performance of the algorithm and play important roles in the construction of better solutions. The stages help the algorithm to avoid being trapped in local optima solutions and maintain the variables' values by re-initialising some variables at the end of each cycle. These stages may occur after each iteration or after a few iterations, dependant on the goodness of the flow stage. Finally, in IWD only indirect communication is considered represented by soil, while both direct and indirect communications are considered in the HCA as explained early on.

3.4.6 Experiments and Results

A number of systematic experiments were conducted to measure the HCA's performance and analyse how the variables affect each other in finding a solution. Additionally, the analysis aims to find out how the usage of the extra stages may affect the obtained results. These experiments can validate the ability of the algorithm to avoiding convergence on local optimal solutions.

The same experiments were performed for the IWD algorithm and the HCA in order to differentiate their performances.

3.4.6.1 Experiment 1

The purpose of this experiment is to explain how the values of certain variables have changed after a number of iterations. The maximum number of iterations was set to fifty. The following three variables were evaluated:

- The amount of soil on the edges.
- The velocity of the water drops.
- The amount of carrying soil.

Figure 3-18 illustrates a simple graph of five equidistant nodes. In this experiment, a number of water drops equal to the number of nodes, have been placed at node *A*, and will move toward node *E*. All the edges have the same amount of soil.

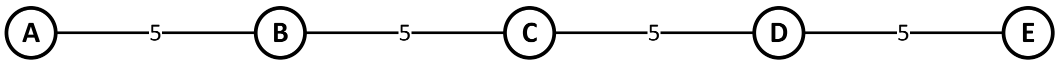


Figure 3-18. Five nodes with the same distance between them

The IWD and the HCA algorithms were executed on the same graph and their common variables were initialised with the same values. The average values of these variables were monitored and recorded at the end of each iteration. Figure 3-19 shows the average value for these variables after fifty iterations using the IWD algorithm.

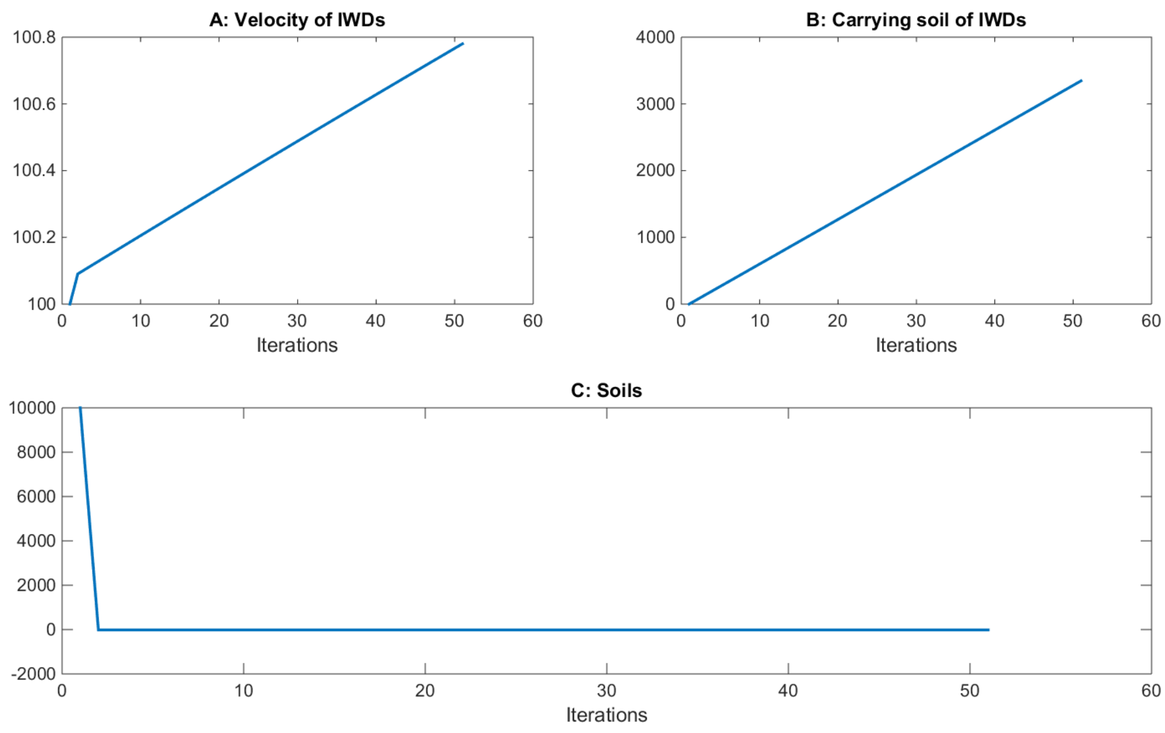


Figure 3-19. IWDs Velocity, carrying soil, and soil exist on edges using IWD algorithm

The result of the experiment revealed defects in the design of the mathematical model of the IWD algorithm. Figure 3-19-A shows that the average velocity of all the IWDs increased slightly over fifty. In Figure 3-19-B, the average amount of soil carried by IWDs is large. Finally, in Figure 3-19-C, the average amount of soil existing on all the edges decreases sharply in the first few iterations, until the amount of soil becomes negative. One of the issues that emerges from these findings is the algorithm's inability to generate diverse solutions. By rapidly removing soil from certain edges leads to the possibility of premature convergence, which has an impact on its efficiency.

Figure 3-20 shows the average values for the same variables along with the depth variable using the HCA.

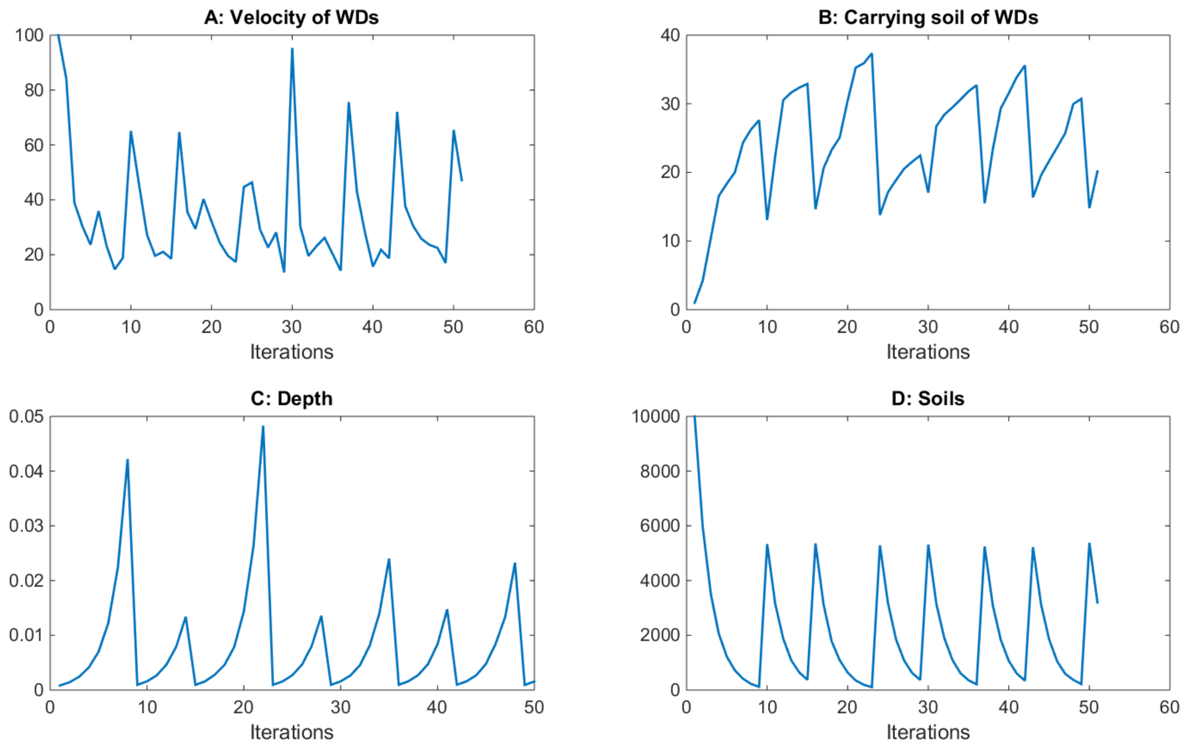


Figure 3-20. IWDs Velocity, carrying soil, and soil exist on edges using HCA.

In HCA, changing in the values of these variables implies the steadiness of the mathematical model design. For instance, the average velocity of water drops was increasing and decreasing, indicating that some water drops increased their velocity and some have decreased as intended. Similarly, the average amount of the carried soil by water drops was increasing and decreasing. It was observed that the maximum average amount of soil on the edges decreased gradually within the first few iterations. These findings explain the relatively good interrelated between the algorithm stages.

Another critical weakness of the IWD algorithm is its inability to make a different selection among a set of nodes that have similar probabilities (Alijla, Wong, Lim, Khader, & Al-Betar, 2014). To explain this problem, let us assume that a number of water drops have been placed at node *S*, and they have to choose the next node to visit between node 1, 2 and 3, as shown in Figure 3-21-A. The probability for each node is calculated based on Eq. (2.1). In this example, all the nodes have the same probability (*P*=0.33) because the same amount of soil exists on each path. Hence, water drops have no preference for any of the edges. According to the IWD algorithm procedure, once a water drop moves to a node, it will update the existing soil on that path. Let us assume that the first water drop chose the first node and it updated the soil to become 900, see Figure 3-21-B. When the second water drop has to choose, it calculates the probability for all the three nodes again. Consequently, the second water drop will choose the first node because it has the highest probability. The same process occurs for the last water drop, and it will choose the same node. This problem leads to the generation of the same solution by all the water drops, as they will continue to choose the same nodes as the first water drop. The main reason that there is no other heuristic or randomness associated with the probability calculation to influence the decision of each water drop.

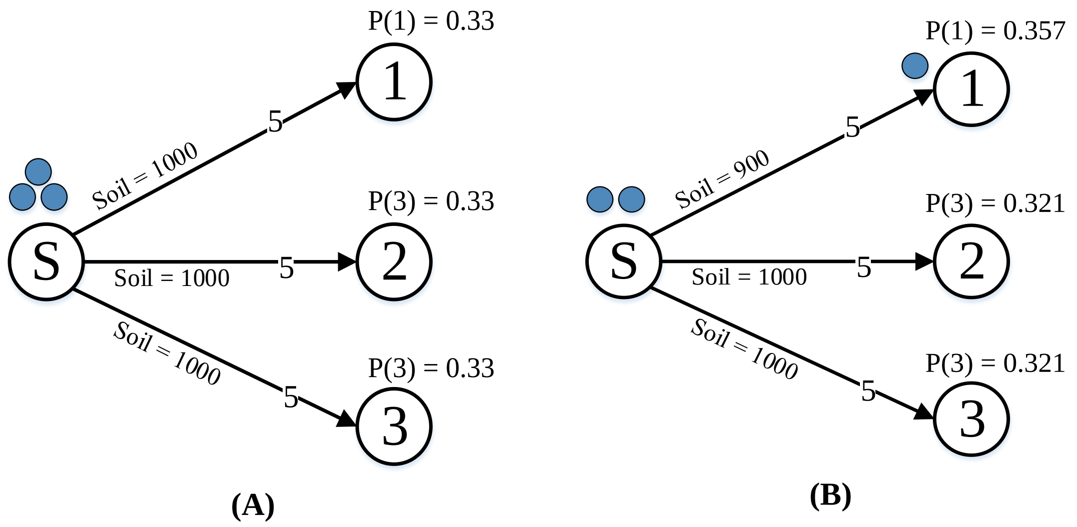


Figure 3-21. Three nodes with the same edges' lengths

The same problem persists even with varying path lengths, as shown in Figure 3-22.

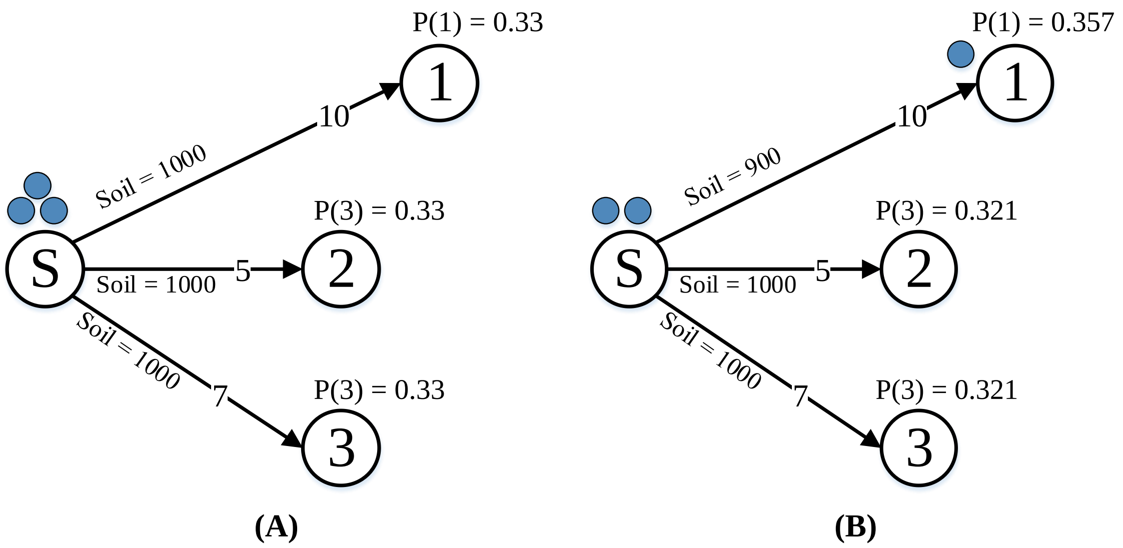


Figure 3-22. Three nodes with different edges' lengths

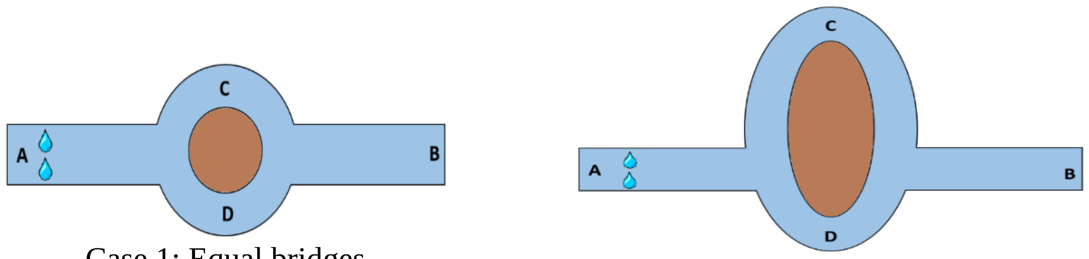
One of the more common ways to address this problem is to include another heuristic that can affect the calculation of the probabilities, such as the inverse of the path length, see Eq. (3.28).

$$P^{IWD}{}_{i}(j) = \frac{\left({}^{1}/_{\mathrm{soil}(i,j)}\right) \times \left({}^{1}/_{Dij}\right)}{\sum_{k \notin vc(iwd)} \left(\left({}^{1}/_{\mathrm{soil}(i,j)}\right) \times \left({}^{1}/_{Dij}\right)\right)} \quad (3.28)$$

However, this is not a guaranteed solution, because the water drops will always favour the shortest paths. In the previous example, all the water drops would choose node 2, as it has the shortest edge. Generally, the discussed problem is related to the design of the exploration and exploitation processes.

In the design of the HCA, it was hypothesised that it does not matter which node is chosen first, because after a few iterations the chosen path becomes deeper and influences other water drops to choose different paths. In other words, the chance of using the same path decreases the longer it is used. To test this assumption, two experiments were conducted to check the behaviour of the water drops in the HCA when it comes to choosing the next node to visit. The aim of these experiments was to validate the design of the exploration and exploitation process. Particularly, the effect of using the depth as a second factor (i.e., bias) to control the process of choosing the next node.

The experiments are designed based on the *double-bridge* experiments which were originally designed to measure the behaviour (i.e. its convergence and exploration ability in the search space) of ACO algorithm (Dorigo et al., 1996). In the double-bridge experiment, the water drops have to choose between two possible paths (two bridges) to move from one point to another. The bridges may have the same or different lengths. In Figure 3-23, water drops are placed at point *A* and they have to move to point *B* using bridge *C* or *D*. In the first case, the total cost of the two possible paths is the same. In contrast, in the second case, the cost of using bridge *A-D-B* is less than the cost of using *A-C-B*.



Case 1: Equal bridges

Case 2: Unequal bridges

Figure 3-23. A bridge experiment

One of the objectives of these experiments is to check whether the water drops will be able to discover the shortest two bridge path or will always follow the same path, regardless of bridge length. Therefore, two directed graphs were designed based on the two cases. Each graph consists of six nodes.

3.4.6.2 Double-equal Bridge Experiment:

The first graph is assumed to have two equal bridges. The main goal of this experiment is to explain how the water drops will choose the next node to visit in case the candidate nodes have the *same* probability of being chosen, see Figure 3-24. A further goal of this experiment is to demonstrate that the HCA algorithm has the ability to explore the domain of the problem as much as possible (i.e., convergence rate). Where in some problems finding the minimum/maximum values of the objective function is not always the main goal, but finding various solutions is ultimately important such as scheduling or classification problems.

In this experiment, six water drops were placed at the starting point (node *S*) and will flow towards the end point (node *E*).

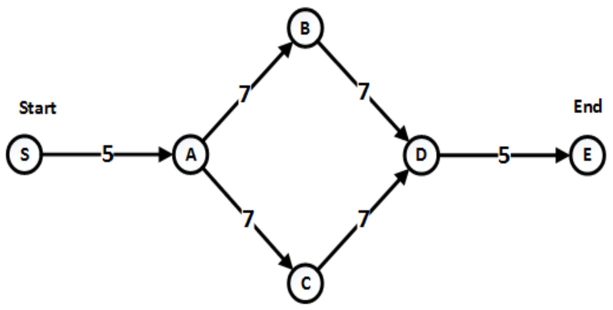


Figure 3-24. An equal bridge graph

The nodes' coordinates for this graph were entered as listed in Table 3-2. Then, the algorithms were executed.

Table 3-2. Equal bridge nodes' coordinates

Node	X coordinate	Y coordinate
S	0	5
A	5	5
B	10	10
C	10	0
D	15	5
E	20	5

In both algorithms, the edges were initialised with the same amount of soil and the same initial velocity. In the IWD algorithm, the choice of the next node is biased only by the amount of the soil. In HCA, the choice of next node is biased by the amount of the soil and the depth of the path, with more emphasis placed on the amount of soil; such that a path with less soil and less depth has a higher probability. The depth is calculated based on the amount of the soil that exists on the path and its length. Therefore, in this graph, both the amount of soil and the depth of each bridge are the same. When the first water drop reaches node *A* (the decision point), it has to choose between node *B* and *C*. Therefore, it calculates the probability of each node. In both algorithms, the probability of choosing node *B* or node *C* is the same. This probability can be interpreted as each bridge has a 50% chance of being selected. In this situation, both algorithms will choose the first node with the higher probability (i.e. node *B*), and will update the amount of soil existing on the edge between *A* and *B*. The depth is then updated accordingly in the HCA. As explained earlier, the water drops in the IWD algorithm continue to choose the same node. Table 3-3 shows the probability of choosing each bridge, and the chosen bridge using IWD algorithm.

Table 3-3. The soil and probability for each node in equal bridge using IWD algorithm

Water Drop	Bridge	Soil	Sum (Soil)	1/Soil	Probability	chosen edge
WD 1	A-B	10000	0.00020	0.000100	0.50000	A-B
	A-C	10000		0.000100	0.50000	
WD 2	A-B	988.85	0.00111	0.001011	0.91001	A-B
	A-C	10000		0.000100	0.08999	
WD 3	A-B	87.73	0.01150	0.011398	0.99130	A-B
	A-C	10000		0.000100	0.00870	
WD 4	A-B	-2.378	-0.42050	-0.420597	1.00024	A-B
	A-C	10000		0.000100	-0.00024	
WD 5	A-B	-11.408	-0.08756	-0.087657	1.00114	A-B
	A-C	10000		0.000100	-0.00114	
WD 6	A-B	-12.293	-0.08125	-0.081348	1.00123	A-B
	A-C	10000		0.000100	-0.00123	

In contrast to IWD, in the HCA, water drops will keep flowing, and the selected bridge may change each time a water drop reaches the decision point as the amount of soil and the depth are changed. Eventually, the algorithm arrives at the best solution, which is in this experiment is either one of the two bridges.

Table 3-4 shows how the amount of soil and the depth change, and how they affect the probability of choosing the next node. These values were taken during the execution of the HCA in the first iteration. It was observed that half the water drops chose bridge *A-B-D*, and the other half chose bridge *A-C-D*. Thus, unlike IWD, the HCA was able to generate different solutions by exploring both bridges.

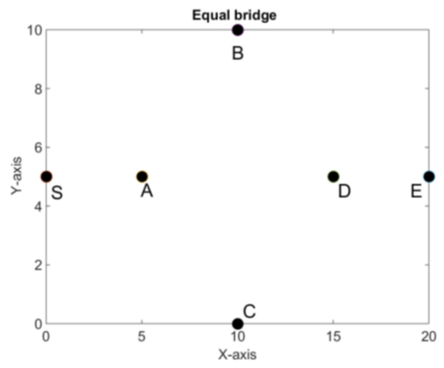
Table 3-4. The soil, depth and probability for each node in equal bridge using HCA

Water Drop	Bridge	Soil	Depth=Distance/ Soil	Normalised Depth	1/ Depth	Probability	chosen edge
WD 1	A-B	10000	0.000700	1.08783	0.91926	**0.5**	A-B
	A-C	10000	0.000700	1.08783	0.91926	0.5	
WD 2	A-B	8960.51	0.000781	1.09595	0.91245	0.49814	
	A-C	10000	0.000700	1.08783	0.91926	**0.50186**	A-C
WD 3	A-B	8960.51	0.000781	1.09595	0.91245	0.49988	
	A-C	9022.57	0.000776	1.09542	0.91290	**0.50012**	A-C
WD 4	A-B	8960.51	0.000781	1.09595	0.91245	**0.50198**	A-B
	A-C	8060.71	0.000868	1.10467	0.90525	0.49802	
WD 5	A-B	8031.73	0.000872	1.10499	0.90499	0.49993	
	A-C	8060.71	0.000868	1.10467	0.90525	**0.50007**	A-C
WD 6	A-B	8031.73	0.000872	1.10499	0.90499	**0.50208**	A-B
	A-C	7262.39	0.000964	1.11422	0.89749	0.49792	

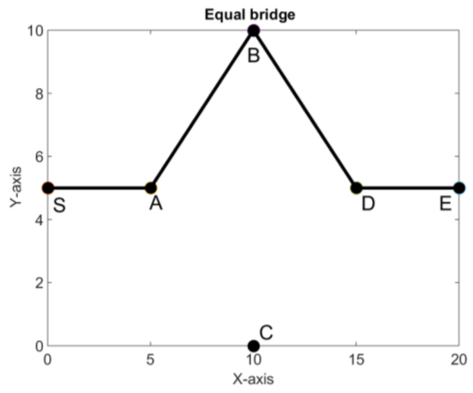
The algorithm was able to generate different solutions by exploring the two bridges, this demonstrates that each of the two bridges used 50% approximately for each. The final obtained results are presented in Figure 3-25.

Water drops	Order of nodes				
WD 1	S	A	C	D	E
WD 2	S	A	B	D	E
WD 3	S	A	C	D	E
WD 4	S	A	B	D	E
WD 5	S	A	C	D	E
WD 6	S	A	B	D	E

The final solution of each water drop



The original graph



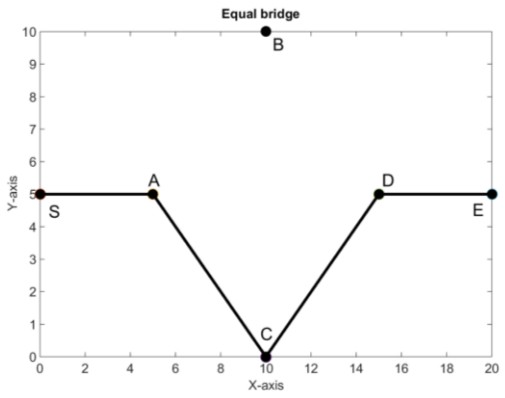


Figure 3-25. The final results of the HCA on the equal-bridge experiment

3.4.6.3 Double-unequal Bridge Experiment:

This experiment uses a graph with unequal bridge lengths as illustrated in Figure 3-26.

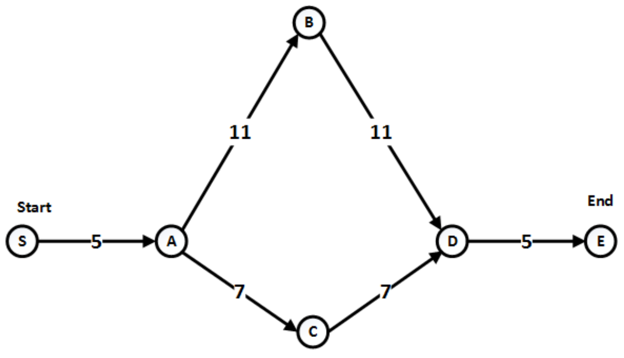


Figure 3-26. An unequal bridge graph

The nodes' coordinates of this graph as entered are listed in Table 3-5.

Table 3-5. Unequal bridge nodes' coordinates

Node	***X coordinate***	***Y coordinate***
S	0	5
A	5	5
B	10	15
C	10	0
D	15	5
E	20	5

The same initial steps were followed as in the equal bridge experiment. In the IWD algorithm, the two edges have the same probability; hence, the water drops continued choosing the first edge (*A*-*B*). When, the IWD transition rule was modified, by considering the inverse of the path length, according to Eq. (3.28). The algorithm was able to find the shortest path. However, none of the water drops chose the longer bridge. This indicates the inability of the IWD algorithm to discover more of the search space.

In HCA, the two bridges have the same amount of soil, but the depth will be different as they have different lengths. Once the first water drop has reached the decision point, it chose node *C* because it has the highest probability ($P_A(B)$= 0.491, $P_A(C)$=0.509). Table 3-6 illustrates the changes in the soil amount and the depth as the water drops cross the bridges during the first iteration of the HCA.

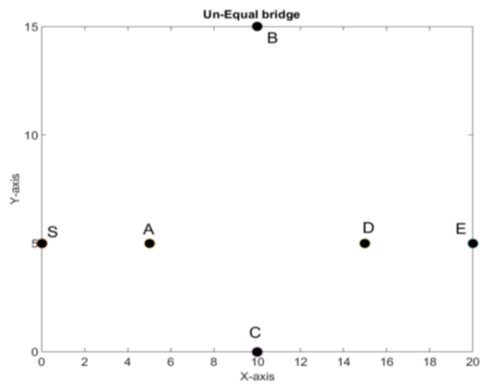
Table 3-6. The soil, depth and probability for each node in unequal bridge using HCA

Water Drop	Bridge	Soil	Depth=Distance/ Soil	Normalised Depth	1/ Depth	Probability	chosen edge
WD 1	A-B	10000	0.001100	1.12972	0.88517	0.49083	
	A-C	10000	0.000700	1.08905	0.91823	0.50917	A-C
WD 2	A-B	10000	0.001100	1.12900	0.88574	0.49260	
	A-C	9003.42	0.000777	1.09608	0.91235	0.50740	A-C
WD 3	A-B	10000	0.001100	1.12900	0.88574	0.49464	
	A-C	8068.29	0.000868	1.10509	0.90491	0.50536	A-C
WD 4	A-B	10000	0.001100	1.12900	0.88574	0.49677	
	A-C	7274.08	0.000962	1.11456	0.89722	0.50323	A-C
WD 5	A-B	10000	0.001100	1.12900	0.88574	0.49913	
	A-C	6553.87	0.001068	1.12513	0.88879	0.50087	A-C
WD 6	A-B	10000	0.001100	1.12900	0.88574	0.50182	A-B
	A-C	5883.42	0.001190	1.13730	0.87927	0.49818	

The *A-C* bridge became deeper with more soil removed, because it was used intensively by the water drops. Consequently, this allowed the use of the next bridge (*A-B*) as it is shallower. Despite the algorithm using both bridges, it succeeded in providing the best solution at the end of the execution. The final results are presented in Figure 3-27.

Water drops	***Order of nodes***				
WD 1	S	A	C	D	E
WD 2	S	A	C	D	E
WD 3	S	A	C	D	E
WD 4	S	A	B	D	E
WD 5	S	A	C	D	E
WD 6	S	A	B	D	E

The final solution of each water drop



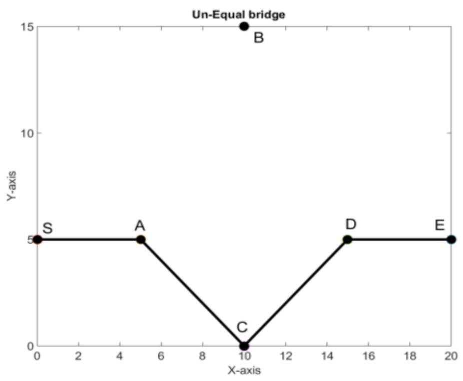


Figure 3-27. The final-results of the HCA on the unequal-bridge experiment

In summary, based on the experimental results, the HCA algorithm shows an ability to deal with the two bridge scenarios with promising performance. Moreover, the fluctuation in the depths of edges appears to be able to promote exploration of different promising areas in the search space and avoid stagnation.

3.5 Similarities and Differences to Other Water-based Algorithms

In this section, the major similarities and differences between the HCA and other water-based algorithms such as IWD, WCA, and WFA are highlighted. While these algorithms share the same source of inspiration, water movement in nature, they are inspired by different aspects of the water processes. The parameters, operations, exploration techniques, the formalisation of each stage, and solution construction differ in both algorithms. In addition, none of them takes into account the full water cycle and the activities associated with water movement. The partial simulation of a natural process may limit the algorithm performance, especially in terms of exploration and exploitation capabilities which can lead to problems such as stagnation, increased computational effort, or premature convergence.

The WCA omits some important factors in the natural water cycle. In WCA, no consideration is made for soil removal from the paths, which is considered a critical operation in the formation of streams and rivers. In WCA there is no temperature and the evaporation rate is based on a ratio based on quality of solution. There is no consideration also for the condensation stage in WCA, which is one of the crucial stages in the water cycle.

The WFA mimics the hydrological cycle in meteorology and the erosion process of water flow. The framework and working mechanism of WFA differ from HCA in many points, such as: the WFA did not consider the velocity of water drops or amount of soil that will be removed or deposited. The WFA depends only on the flow stage of the water cycle, while the other stages are optional and can be utilised if necessary for some problems. In WFA, moving the drops is based on using local search methods, and there is no specific heuristic belonging to the algorithm.

The HCA has utilised the major components of the natural water cycle. The added advantage of HCA is that all stages of the hydrological water cycle are included, leading to an overall conceptual framework under which other water-based algorithms can be placed. In addition, the inclusion of all stages allows both direct and indirect communication to take place among particles, leading to enhanced swarm intelligence. Table 1 summarises the major differences between IWD, WCA, WFA, and HCA based on some criteria.

Table 3-7. A summary of the main differences between IWD, WCA, WFA, and HCA

Criteria	IWD	WCA	WFA	HCA
Inspiration	actions and reactions that occur to water drops in rivers (Water drops flow in rivers)	streams and rivers flow towards the sea	the hydrological cycle in meteorology and the erosion phenomenon	the full hydrological cycle
Entities	intelligent water drops	streams	clouds are generated then, drops of water	artificial water drops
Flow Stage	moving water drops	changing streams' positions	moving from higher positions to lower positions based on erosion capability	moving water drops
Choosing the next node	based on the soil	based on river and sea positions	apply a local search algorithm (altitude)	based on soil and path depth
Velocity	always increases	N/A	N/A	increases and decreases
Soil update	removal	N/A	N/A	removal and deposition
Carrying Soil	equal to the amount of soil removed from a path	N/A	N/A	is encoded with the solution quality
Evaporation	N/A	when the river position is very close to the sea position	(optional) no improvement after a maximum number of iterations	based on the temperature value, which is affected by the percentage of improvements in the last set of iterations
Evaporation rate	N/A	if the distance between a river and the sea is less than the threshold	all the DOWs	random number
Condensation	N/A	N/A	N/A	enable information sharing (direct communication). Update the global-best solution, which is used to identify the collector.
Precipitation	N/A	randomly distributes a number of streams	(optional) generate new DOWs	reinitialises dynamic parameters such as soil, velocity, and carrying soil

3.6 Summary

This chapter discussed the hydrological water cycle in nature, and the continuous movement of water droplets between the earth and sky. The IWD algorithm was examined and the key aspects of its implementation discussed. Aspects of the water cycle that were omitted in IWD were identified and their potential to enhance exploration and exploitation is discussed in detail.

A new nature-inspired algorithm called the HCA is presented. This algorithm was conceptualised by delving deeper into hydrological theory. A framework was designed that describes the steps of the HCA and a mathematical model built that defines a set of variables and equations that describes the working mechanism of the HCA.

The differences between other water-based algorithms and the HCA algorithm are highlighted. The HCA has a stronger conceptual link with the natural water cycle when compared with other water based algorithms. Arguably, this closer conceptual link with the natural system has resulted in an improved model. The results of the two structured experiments showed that the HCA performed better than IWD on the two-bridge problem. It is demonstrated that this improved performance is due to the exploration and exploitation processes inherent in HCA. In summary, the experimental results reported in this section are promising and indicate the convergence ability of the HCA owing to the proper balance between exploration and exploitation. This guarantees, at least in the context of these experiments, that the HCA can proceed towards a global solution.

In conclusion, if a nature-inspired approach is to be considered as a novel and useful addition to the already large field of overlapping methods and techniques, it needs to embed the two critical concepts of self-organisation and emergentism at its core, with intuitively-based (i.e., nature-based) information-sharing mechanisms for effective exploration and exploitation, as well as demonstrate performance which is on par with related algorithms in the field. In particular, it should be shown to offer something a bit different from what is already available. We contend that the HCA satisfies these criteria and while further development is required to test its full potential, it can be confidently used by researchers dealing with optimisation problems. The performance of HCA and its convergence were promising using the double-bridge experiments. However, optimisation problems are more complicated than these experiments. Hence, the HCA must be tested and evaluated on different benchmarked optimisation problems. In order to achieve that, the next chapters demonstrate the application of HCA on different types of problems.

Chapter 4 Solving the Travelling Salesman Problem using HCA

"Problem-solving is hunting; it is savage pleasure and we are born to it."

[Thomas Harris]

This chapter reports on work that evaluates the performance of the HCA on a discrete domain problem, namely, the travelling salesman problem (TSP). Although many approaches can solve the TSP with high quality, the TSP remains an effective way of testing a new algorithm on discrete problems. Therefore, the main goal is to measure the algorithm's ability to optimise (or nearly optimise) the solution for a simple NP-hard problem. To tune the parameter values, the HCA was first tested on several structural TSP instances with different geometric shapes. These instances were designed for easy evaluation and demonstration of the optimal solution. Next, the HCA was tested on standard benchmarked symmetric TSP datasets from the literature. The results of these experiments were evaluated, and the solution quality and number of iterations were compared with those of other metaheuristic algorithms.

The rest of this chapter is organised as follows. Section 4.1 overviews the TSP and its formulation. Section 4.2 presents the configuration of HCA and explains its application to the TSP. Section 4.3 demonstrates the feasibility of solving TSP instances by HCA and compares the results with those of other algorithms. Discussion and conclusions are presented in Section 4.4.

4.1 Introduction

The TSP is a well-known classical problem in which a salesperson must visit every designated city exactly once, and return to the starting point, via the shortest possible route. Such a path is known as a Hamiltonian cycle (Held, Hoffman, Johnson, & Wolfe, 1984). For centuries, the TSP has attracted researchers' attention owing to the simplicity of its formulation and constraints. However, despite being easy to describe and understand, the TSP is difficult to solve (Hoffman & Padberg, 2013). Because a vast amount of information has been amassed on the TSP and the behaviours of TSP algorithms are easily observed, the TSP is now recognised as a standard benchmarking problem for evaluating new algorithms and comparing their performances with those of established algorithms. Many real-life problems and applications can also be formulated as TSPs, and some optimisation problems with different structures can be reduced or transformed to TSPs, such as the job scheduling problem, the knapsack problem, DNA sequencing, integrated circuit (i.e., VLSI circuits) design, drilling problem, and the satisfiability problem (Davendra, 2010; Matai, Singh, & Lal, 2010). Finally, a TSP can be classified as a combinatorial optimisation problem, as it requires finding the best solution from a finite set of feasible solutions.

Typically, a TSP is represented as a complete undirected weighted graph, where each node is connected to all other nodes. The graph *G* = (*V*, *E*) consists of a set of *V* nodes (i.e. cities) connected by a set of *E* edges (i.e. roads), where the edges are associated (assigned) with various weights. The weight is a nonnegative number reflecting the distance, the travel cost, or time of travelling that edge. Given the node coordinates (locations), the Euclidean distance between two nodes *i* and *j* can be calculated as follows:

$$Distance\ (i,j) = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \tag{4.1}$$

The TSP can be a symmetric or asymmetric weighted problem. In the symmetric problem, the path from node *A* to node *B* has the same weight as the path from node *B* to node *A*. In contrast, paths in the asymmetric problem may be unidirectional or carry different weights in each direction. Mathematically, the TSP can be formulated as Eq. (4.2) (Hoffman & Padberg, 2013), where D_{ij} represents the distance between nodes *i* and *j*.

$$Minimise \sum_{i=1}^{N} D_{ij}\, X_{ij}\ , N \geq 3 \tag{4.2}$$

subject to

$$X_{ij} \in \{0,1\},\ i,j = 1, \dots, N, i \neq j. \tag{4.3}$$

$$\sum_{i=1, i\neq j}^{N} X_{ij} = 1 \qquad i = 1, \dots, N$$

$$\sum_{j=1, j\neq i}^{N} X_{ij} = 1 \qquad j = 1, \dots, N$$

$$u_i - u_j + Nx_{ij} \leq N - 1 \qquad 2 \leq i \neq j \leq N$$

$$0 \leq u_i \leq N - 1 \qquad 2 \leq i \neq j \leq N$$

In Eq. (4.3), the decision variables X_{ij} are set to 1 if the connecting edge is part of the solution, and 0 otherwise:

$$X_{ij} = \begin{cases} 1 & the\ path\ foes\ from\ city\ i\ to\ city\ j \\ 0 & Otherwise \end{cases} \tag{4.4}$$

The TSP is considered an NP-hard problem, meaning that its complexity increases exponentially with increasing number of cities. Therefore, the number of possible solutions rises rapidly as the number of cities increases. Practically, the TSP finds the best order of the visited nodes at the lowest cost, which can be interpreted as a permutation problem. The number of possible solutions for an n-city problem is given by:

$$Number\ of\ solutions = \frac{(n-1)!}{2} \quad ,where\ ,n \geq 3. \tag{4.5}$$

Equation (4.5) calculates the number of possible ways of arranging n cities into an ordered sequence (with no repeats). As the starting node is unimportant, there are (n-1)! rather than n! possible solutions. The result is divided by two because (for example) route (1 → 2 → 3 → 4 → 5 →1) is the same as (1 → 5 → 4 → 3 → 2 →1); therefore, the reverse routes are ignored. Figure 4-1 shows a simple TSP with five nodes.

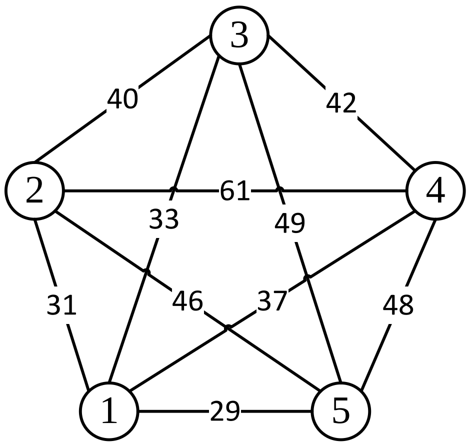


Figure 4-1. TSP instance with five nodes

In this example, there are five nodes with 120 possible solutions. The number of possible solutions can be reduced to sixty by ignoring the reverse routes, which have the same total cost. One of the best solutions is (2 → 1→ 5→4→3→ 2) with a cost of 190. Another repeated solution with the same cost but a different starting node is (1→ 5→4→3→ 2→1).

In general, small TSPs are most easily solved by trying all possibilities (i.e. exhaustive searching). This can be achieved by brute-force, branch-and-bound, or the exact algorithm (Saiyed, 2012). These methods generate all possibilities and choose the least-cost solution. Although these techniques will guarantee the optimal solution, they become impractical and expensive (i.e. require unreasonable time) when solving large TSP instances.

A simple alternative is a greedy heuristic algorithm, which solves the TSP using a heuristic function. Such algorithms cannot guarantee the optimal solution, as they do not perform an exhaustive search. However, they perform sufficiently many evaluations to find the optimal/near optimal solution. Many greedy algorithms have been developed for TSPs, such as the nearest-neighbour, insertion heuristics, and dynamic programming techniques. Metaheuristic algorithms can also provide high-quality solutions to large TSP instances. Some of these algorithms were discussed in Chapter 2.

4.2 The algorithm Setup

This section explains the specifications and the TSP-solution process of the HCA.

4.2.1 Problem Representation and Solution Generator

Typically, the input of the HCA algorithm is represented as a graph. In this research, the TSP is assumed to be symmetric, and acting on a fully connected graph. To solve the TSP problem by HCA, we initially place a number of water drops randomly on the nodes. The water drops move from node to node along the edges until each water drop has visited each node.

The candidate TSP solutions are stored in a matrix, where each row represents a different solution generated by a water drop. Therefore, a water drop solution consists of the order of the visited nodes (with no repeat visits). The length of each row (i.e. the number of columns) is denoted by n and determined by the total number of nodes (see Eq. (4.6)).

$$Solutions = \begin{bmatrix} WD_1 & 1 & 2 & \dots & n \\ WD_2 & 1 & 2 & \dots & n \\ WD_3 & 1 & 2 & \dots & n \\ \vdots & & \dots & & \\ WD_n & 1 & 2 & \dots & n \end{bmatrix} \tag{4.6}$$

After evaluating the generated solutions, the best solution is selected. Edges belong to this solution are favoured with less amount of soil. At the end of each iteration, the HCA checks whether the temperature is high enough to evaporate the water drops. At the condensation stage, the evaporated water drops share their information regarding the most promising node sequence. The similarities between the solutions of the water drops are measured by the Hamming distance. The algorithm then proceeds as described in Section 3 of Chapter 3.

4.2.2 Local Improvement Operations

The quality of generated tours can be improved by many operations, such as k-Opt (where k = 2, 3, or 4) (Helsgaun, 2000, 2006). These operations enhance the performance of the algorithm and minimise the number of iterations to reach the best-known solution. In the present problem, we apply the 2-Opt operation on the selected water drops that will evaporate at the condensation stage. The 2-Opt operation swaps the order of two edges at one time, and keeps the tour connected. The swapping results in a new tour, which is accepted if it minimises the total cost

(Blazinskas & Misevicius, 2011; Rocki & Suda, 2012). Figure 4-2 describes the pseudocode of the 2-Opt operation. This operation is repeated until a stopping criterion is met, such as no further improvements after a certain number of exchanges, or when the maximum number of exchanges is reached.

```
PROCEDURE: 2-Opt operation
INPUT: A route, distance matrix
Original-Route := route
Current-cost := CalculateTotalDistance(route)
REPEAT: until no further improvement
       FOR x := 1 to Length(route)- 1
         a := route[x]; b := route[x+1]
         FOR y := x+1 to Length(route)- 1
              c := route[y]; d := route[y+1]
              Edge1:(a, b)
              Edge2:(c, d)
              //(a, b) and (c, d) can be swapped with (a, c) and (b, d)
              // or (a, d) and (c, b)
              route := CreateNewEdges(Edge1, Edge2) → E1:=(a, d),
              E2:=(c, b)
              //Reverse Order of nodes between b and c
              New-cost := CalculateTotalDistance(route)
              IF (new-cost < current-cost)
                  New-Route := route
                  Current-cost := new-cost
              ELSE
                  route := Original-Route
              END IF
         END FOR
       END FOR
END REPEAT
RETURN: New-Route, new-cost
END PROCEDURE
```

Figure 4-2. Pseudocode of the 2-Opt operation (adapted from Burtscher, 2014)

Figure 4-3 demonstrates the operation of 2-Opt. In this example, the algorithm selects edges (2, 7) and (3, 8), and then creates new edges (2, 3) and (7, 8). The order of the nodes between the two edges must also be reversed.

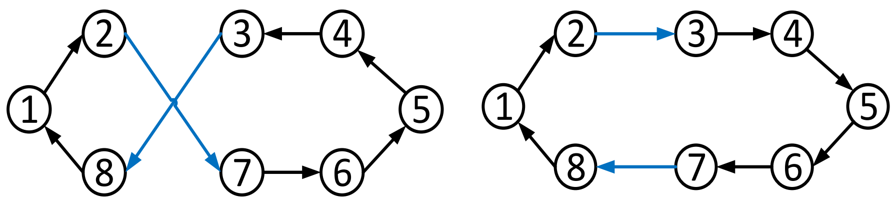


Figure 4-3. Example of removing an intersection by 2-Opt

4.2.3 HCA Parameters and Procedure

The HCA parameter values for this problem are the same as given in Section 3 of Chapter 3. The maximum number of iterations equals 3-times the number of nodes. Figure 4-4 explains the steps in solving the TSP by HCA.

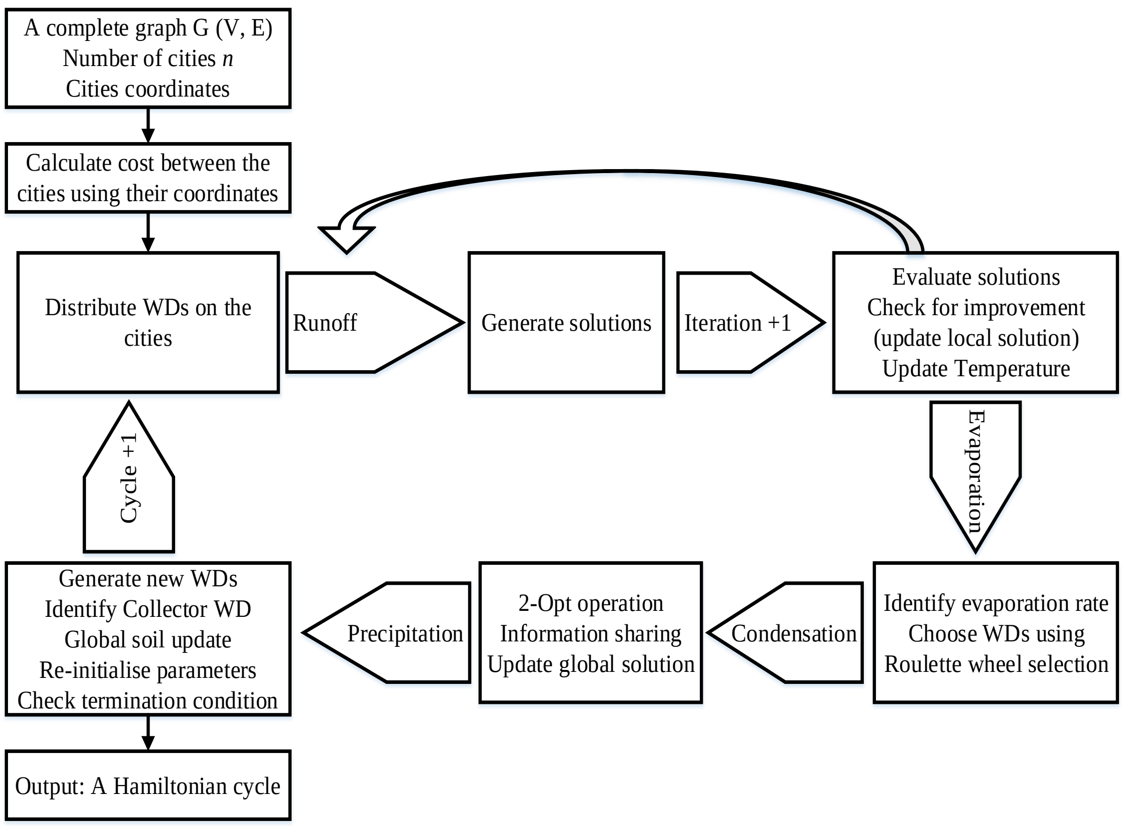


Figure 4-4. TSP solution procedure of HCA

4.3 Experimental Results and Analysis

The HCA was tested and evaluated on two groups of TSP instances; structural and benchmark. The runtime and solution quality of the benchmark results were compared with those of other algorithms.

4.3.1 Structural TSP Instances

To assess the validity of the generated output, we designed and generated synthetic TSP structures with different geometric shapes (circle, square, and triangle). The performance of the HCA is easier to be evaluated on these TSP structures than randomised instances. Several instances with different numbers of nodes were generated for each structure, and were input to the HCA algorithm with and without the 2-Opt operation. The HCA was executed ten times for each instance. The percentage difference (i.e., the deviation percentage) between the obtained and the best-known value was calculated as follows:

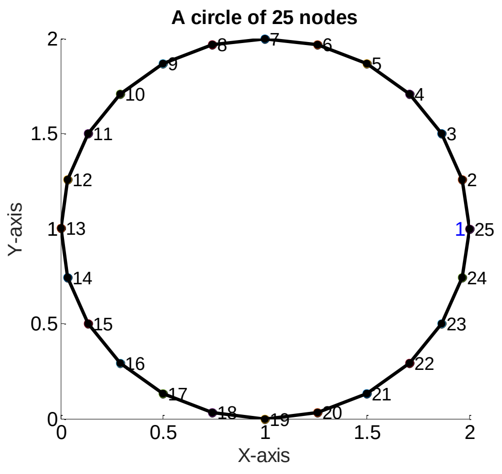
$$Difference = \frac{(Obtained\ Value - Best\_known\ Value)}{Best\_Known\ Value} \times 100\% \qquad (4.7)$$

In the circular structure, the circle circumference was divided into various numbers of nodes. Note that the number of nodes influences the inter-nodal distance, with fewer nodes increasing the distance between nodes. The node number was varied as 25, 50, 75, 100, 125, and 150, where the first and last nodes have the same coordinate. The shortest path length was calculated by the circle circumference formula (2×π×r). The circle was centred at (1, 1) and its diameter was set to 2 (i.e., r = 1). Consequently, its circumference was 6.28. The obtained results are reported in Table 4-1.

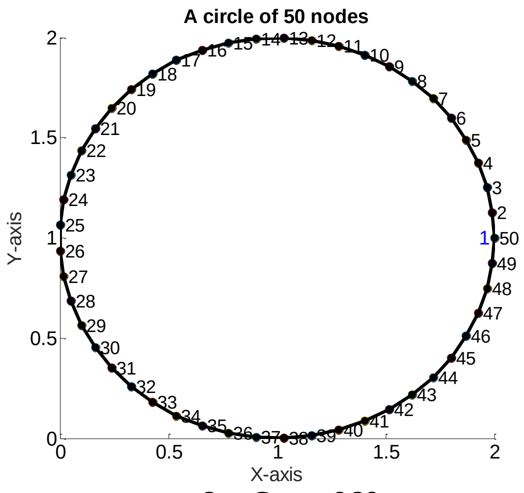
Table 4-1. TSP results on a circular structure

Instance Name	Optimal Solution	With 2-Opt			Without 2-Opt		
		Result	Avg. Time	Difference	Result	Avg. Time	Difference
Circle_25	6.28	6.28	0.72	0%	6.28	0.65	0%
Circle _50	6.28	6.28	4.48	0%	6.28	4.47	0%
Circle _75	6.28	6.28	15.43	0%	6.28	15.13	0%
Circle _100	6.28	6.28	38.60	0%	6.28	37.23	0%
Circle _125	6.28	6.28	80.49	0%	6.28	74.50	0%
Circle _150	6.28	6.28	146.68	0%	6.28	136.99	0%

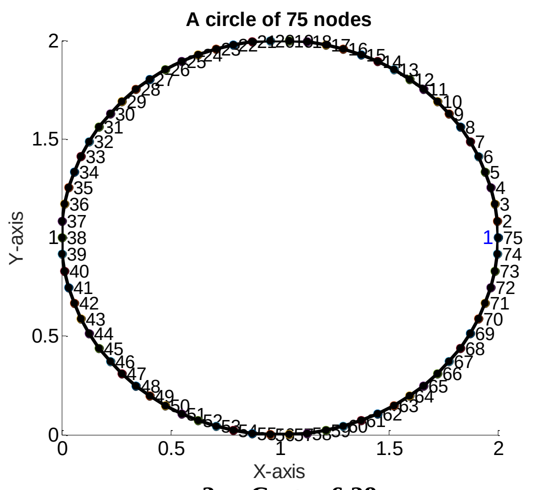
As shown in Table 4-1, the HCA found the shortest path in each instance of this structure, both with and without the 2-Opt operation. The circle instances are relatively easy to solve because the distance decreases with increasing number of nodes. Thus, the soil amount will be reduced more quickly on shorter edges than on longer edges, steering the algorithm towards the shorter edges. Figure 4-5 shows the output of the HCA on circular TSPs with different numbers of nodes.



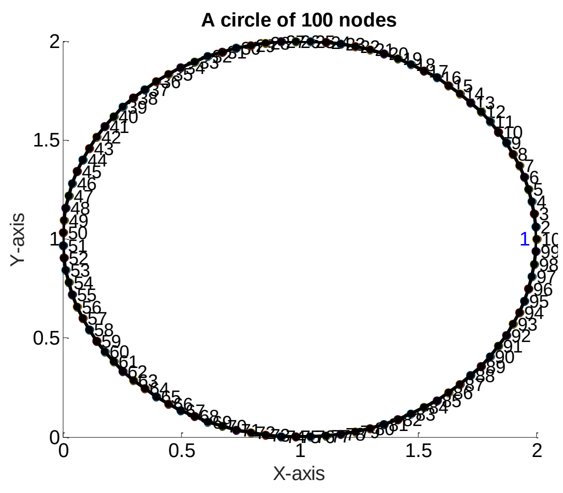
1. **Cost = 6.28**



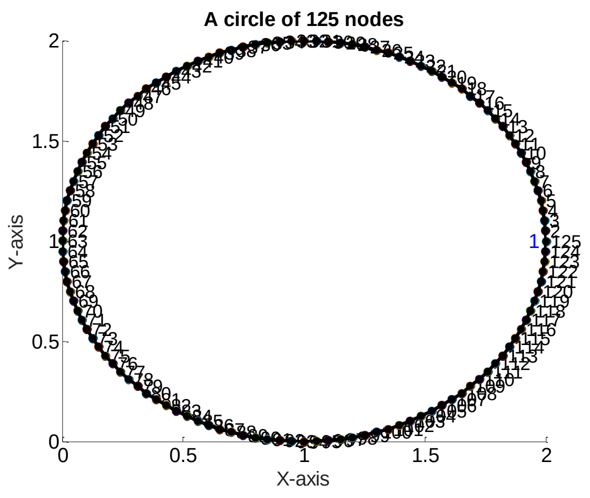
2. **Cost = 6.28**



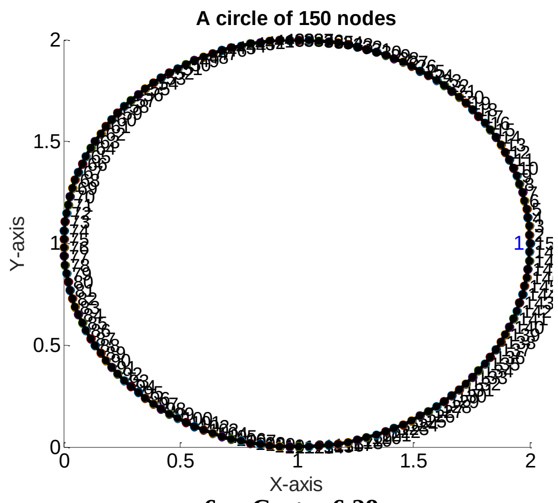
3. **Cost = 6.28**



4. **Cost = 6.28**



5. **Cost = 6.28**



6. **Cost = 6.28**

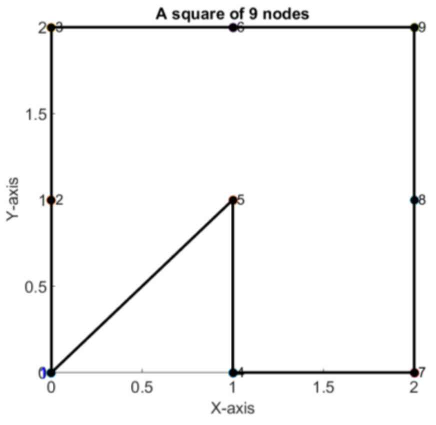
Figure 4-5. TSP solutions on circular grids

Next, the TSP was solved on a square structure. Here, the nodes were evenly spaced in an *N*×*N* grid. The shortest tour distance was the product of the number of nodes and the distance between the nodes (assumed as one unit). For example, in the 16-point (8×8) grid, the shortest path was (1×16 = 16). For an odd number of nodes, the cost of travelling to the last node was based on the length of the hypotenuse (1.41 in the present examples). Ten instances with different numbers of nodes were generated, and solved by the HCA with and without the 2-Opt operation. The results are listed in Table 4-2.

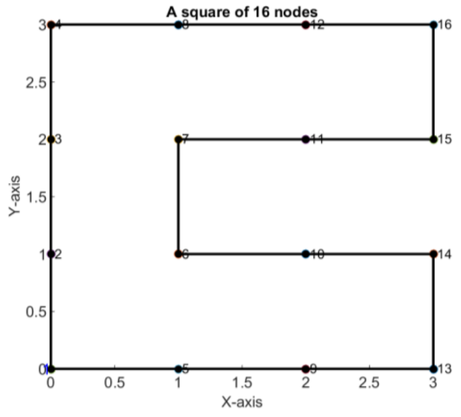
Table 4-2. TSP results on a square structure

Instance Name	Optimal Solution	With 2-Opt			Without 2-Opt		
		Result	Avg. Time	Difference	Result	Avg. Time	Difference
Square_9	9.41	9.41	0.10	0%	9.41	0.01	0%
Square_16	16	16	0.22	0%	16	0.22	0%
Square_25	25.41	25.41	0.65	0%	25.41	0.64	0%
Square_36	36	36	1.74	0%	36	1.72	0%
Square_49	49.41	49.41	4.35	0%	49.41	4.33	0%
Square_64	64	64	9.90	0%	64	9.55	0%
Square_81	81.41	81.41	20.53	0%	81.41	19.58	0%
Square_100	100	100	39.55	0%	100	39.44	0%
Square_121	121.41	121.41	74.054	0%	121.41	72.38	0%
Square_144	144	144	130.93	0%	146.89	125.18	0.02%

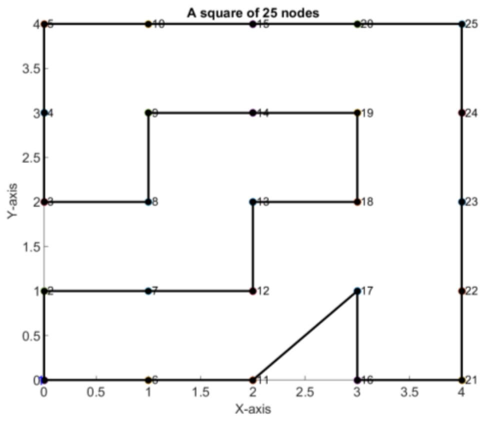
As shown in Table 4-2, the HCA obtained the optimal results (the shortest path) both with and without the 2-Opt operation. The exception was "Square_144", whose solution deviated very slightly from the optimal. The outputs of HCA with 2-Opt on square grids of different sizes are shown in Figure 4-6.



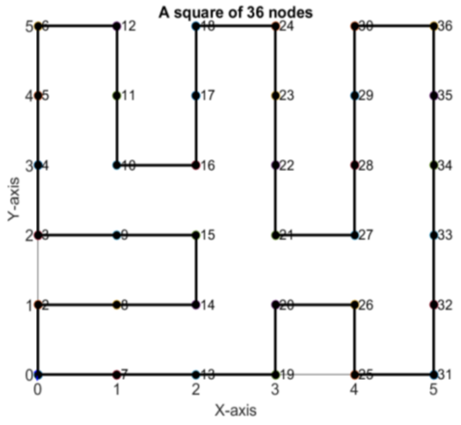
1. **Cost = 9.41**



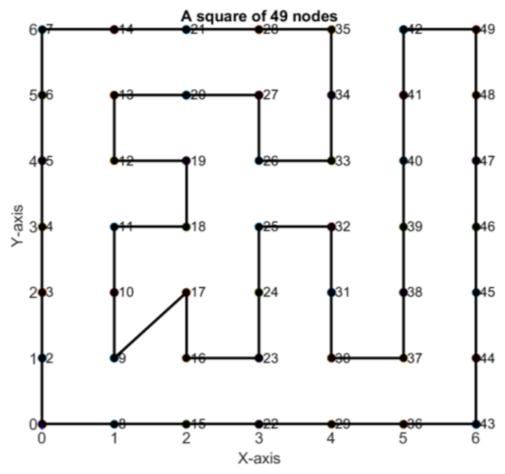
2. **Cost = 16**



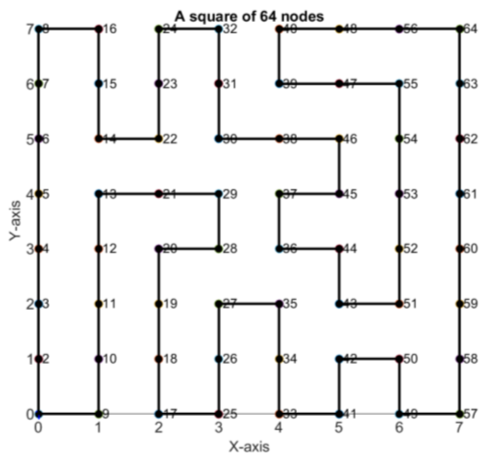
3. **Cost = 25.41**



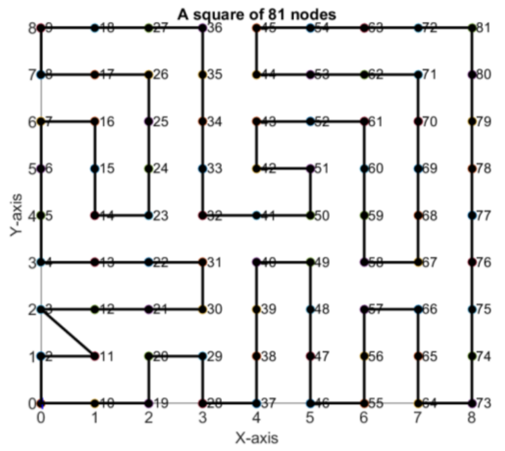
4. **Cost = 36**



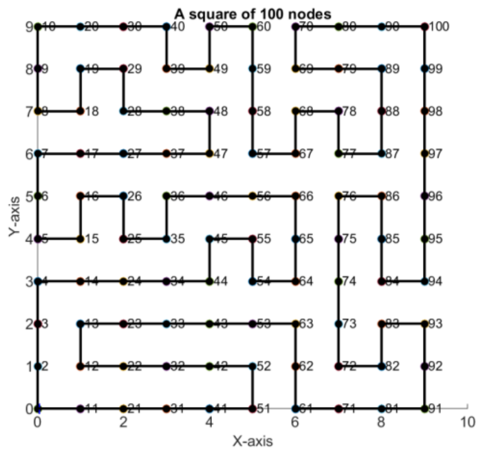
5. Cost = 49.41



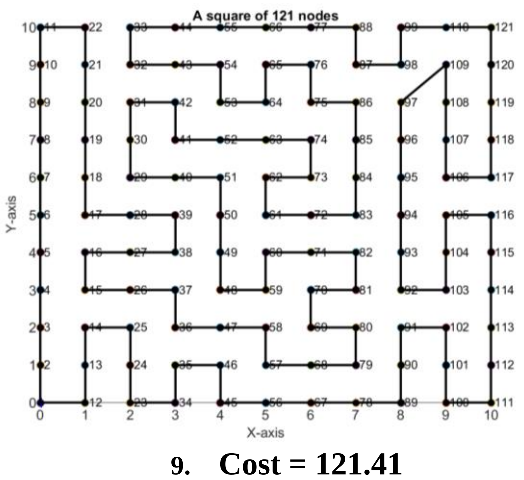
6. Cost = 64



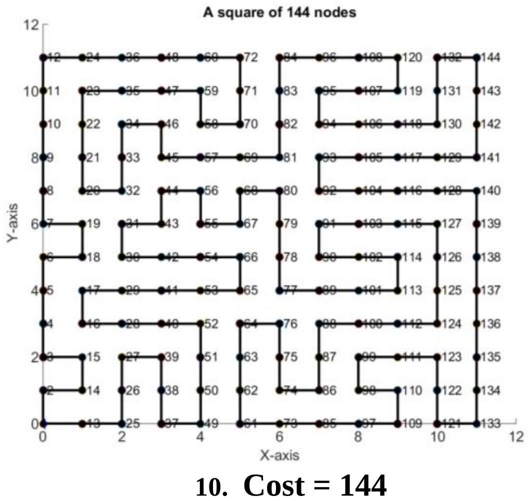
7. Cost = 81.41



8. Cost = 100



9. Cost = 121.41



10. Cost = 144

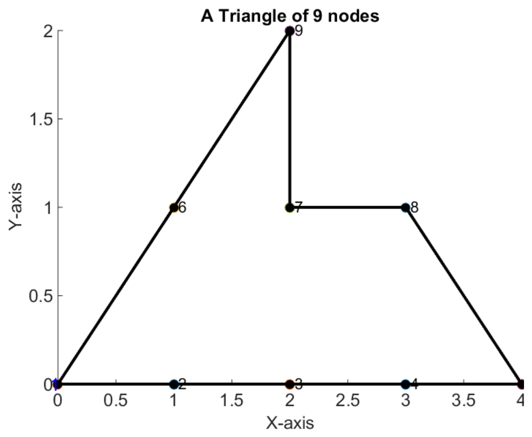
Figure 4-6. TSP solutions on square grids

Finally, the TSP was solved on an equilateral triangular grid. The number of nodes was varied as 9, 25, 49, 81, 121, and 169. Table 4-3 lists the obtained results with and without the 2-Opt operation.

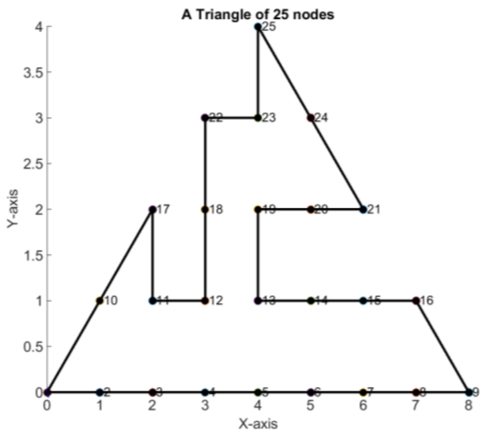
Table 4-3. TSP results on a triangular structure

Instance Name	Optimal Solution	With 2-Opt			Without 2-Opt		
		Result	**Time**	**Difference**	**Result**	**Time**	**Difference**
Triangle _9	10.24	10.24	0.08	0%	10.24	0.08	0%
Triangle _25	27.07	27.07	0.67	0%	27.07	0.66	0%
Triangle _49	51.899	51.899	4.40	0%	51.899	4.31	0%
Triangle _81	84.727	84.727	20.89	0%	86.485	19.98	0.02%
Triangle _121	125.556	125.556	74.51	0%	127.964	70.87	0.0191%
Triangle _169	174.38	174.38	230.05	0%	182.97	210.74	0.049%

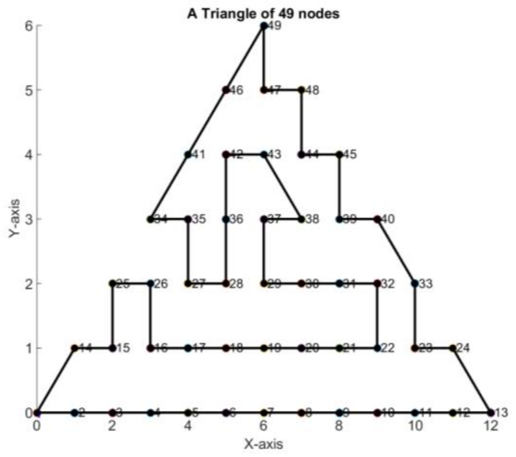
The HCA with and without 2-Opt operation produced almost similar results, except for the triangles with 121 and 169 nodes where using 2-Opt gave better results. The TSP is more difficult on the triangular structure than on the other structures, because many hypotenuses connect the nodes to different layers. The outputs of the HCA using 2-Opt on triangular grids with different node numbers are reported in Figure 4-7.



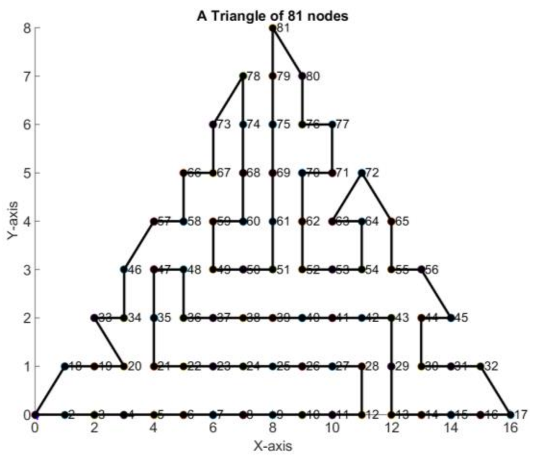
1. **Cost = 10.24**



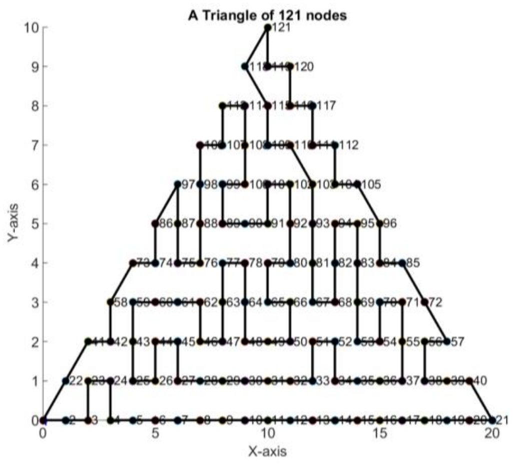
2. **Cost = 27.07**

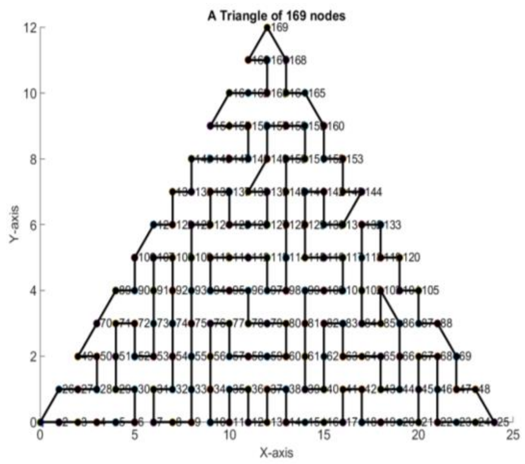


3. **Cost = 51.899**



4. **Cost = 84.727**





5. **Cost = 125.556** 6. **Cost = 174.38**

Figure 4-7. TSP solutions on equilateral triangular grids

The experimental results indicate whether the algorithm can generate different optimal solutions (if any) for the same problem in different runs. This ability depends on the nondeterministic design of the algorithm. HCA was found to sometimes generate alternate optimal solutions. For example, Figure 4-8 displays two optimal solutions for the same problem, Triangle_121, generated by HCA.

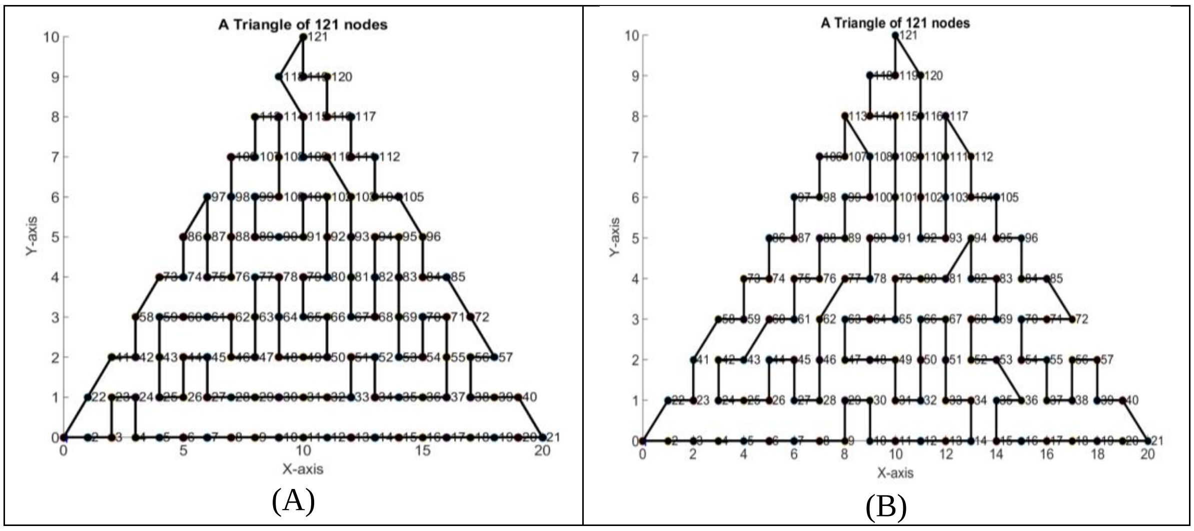


Figure 4-8. Two optimal solutions of HCA on an equilateral triangular grid

The average execution times for solving the TSP with the HCA for all problem instances are listed in Table 4-4. The execution time of the HCA increases with increasing number of nodes because of the information sharing process. With increasing number of nodes the solution space increases exponentially, a defining characteristic of NP-hard problems, this also affects execution time. The execution time also largely depends on the implementation of the algorithm, and on the compiler, machines specification, and operating system used. It is possible to reduce the execution time by improving the implementation of the algorithm.

Table 4-4. Average execution time of TSP on circular, square and triangular grids

Instance Size	***With 2-Opt Avg. time (sec)***	***Without 2-Opt Avg. time (sec)***
9	0.09	0.045
16	0.22	0.22
25	0.68	0.65
36	1.74	1.72
49	4.375	4.32
50	4.48	4.47
64	9.9	9.55
75	15.43	15.13
81	20.71	19.78
100	39.075	38.335
121	74.282	71.625
125	80.49	74.5
144	130.93	125.18
150	146.68	136.99
169	230.05	210.74

Figure 4-9 plots the average execution time as a function of node number.

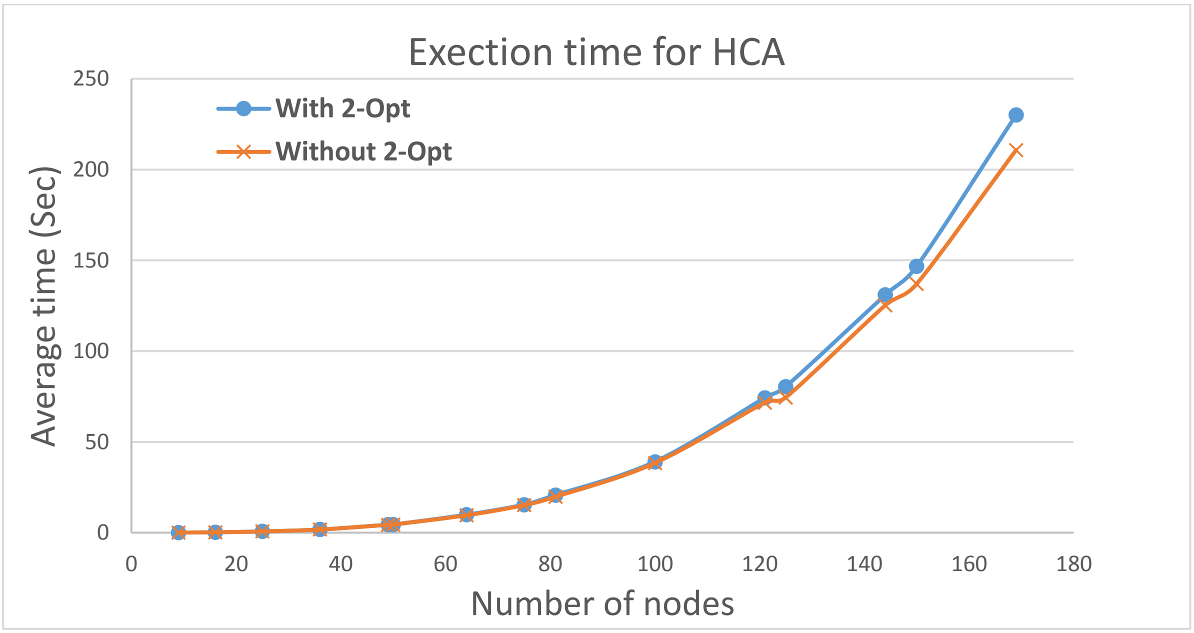


Figure 4-9. Relationship between HCA execution time and instance size

Figure 4-9 showes that the 2-Opt operation has little affect on the execution time in small instances (problems with a low node count), but noticeably increases the execution time in larger problems. However, 2-Opt was found to improve the quality of the solution for structures with a high number of nodes.

4.3.2 Benchmark TSP Instances

Next, the HCA was applied to a number of standard benchmark instances from the TSPLIB library (Reinelt, 1995). The selected instances have different structures with different numbers of cities. Some of these instances are geographical and based on real city maps; others are based on VLSI applications, drilling, and printed circuit boards. The edge-weights (distances) between the nodes were calculated by the Euclidean distance (Eq. (4.1)), and rounded to integers. The TSP file format is detailed in Reinelt (1991). Reinelt listed for every problem either the value of a provably optimal solution or an interval given by the best known lower and upper bound. The optimality of solutions has been proven by branch-and-cut or branch-and-bound algorithms.

For comparing optimisation algorithms applied to a common set of problems, a paired sample t-test is the best choice because it provides a pairwise comparison between two subjects. All the statistical comparisons in this research have been achieved using a paired samples t-test. A paired samples t-test compares two sample means from the same population regarding the same variable at two different times such as during a pre-test and post-test. A paired test is the test of the null hypothesis that the means of two subjects are equal. The paired t-test is used for correlated observations and thus is theoretically more powerful than the unpaired t-test. Furthermore, paired t-tests are more comprehensive and compelling than unpaired t-tests because they are done with subjects that have similar characteristics.

On the benchmark problems, the HCA was combined with the 2-Opt operation, which was found to have improved the solution quality in structural instances with large numbers of nodes. The results are presented in Table 4-5. In this table, the number in each instance name denotes the number of cities, and the difference column denotes the percentage difference from the best-known solution using Eq. (4.7).

Table 4-5. HCA results on benchmark TSP instances

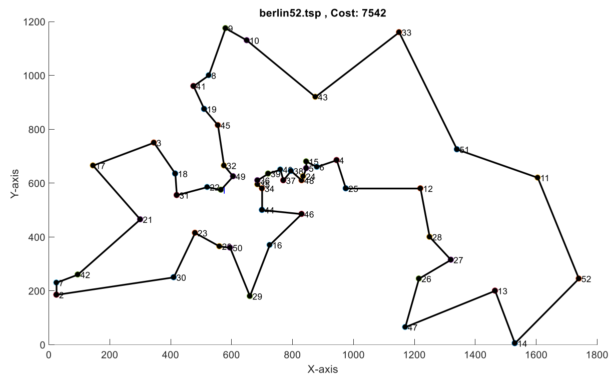
No.	***Instance name***	***Node number***	***Best-Known result***	***HCA***	***Difference %***
1	berlin52	52	7542	7542	0
2	ch130	130	6110	6110	0
3	ch150	150	6528	6528	0
7	d198	198	15780	15780	0
4	eil51	51	426	426	0
5	eil76	76	538	538	0
6	eil101	101	629	629	0
8	kroA100	100	21282	21282	0
9	kroA150	150	26524	26614	0.00339
10	kroA200	200	29368	29368	0
11	kroB100	100	22141	22141	0
12	kroB150	150	26130	26132	0.00008
13	kroB200	200	29437	29455	0.00061
14	kroC100	100	20749	20749	0
15	kroD100	100	21294	21294	0
16	kroE100	100	22068	22068	0
17	lin105	105	14379	14379	0
18	pr76	76	108159	108159	0
19	pr107	107	44303	44303	0
20	pr124	124	59030	59030	0
21	pr136	136	96772	96861	0.00092
22	rat195	195	2323	2323	0
23	st70	70	675	675	0
24	ts225	225	126643	126643	0
	Average		29534.6	29542.9	
	T-test (*P*-value)			0.12174	

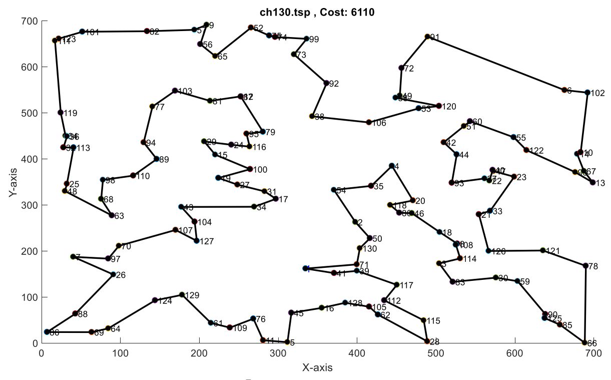
Table 4-5 shows that the HCA achieved a high performance when solving TSP. The HCA found the best-know solution in 20 of 24 instances, and the differences in the other instances were unremarkable. A T-Test paired two sample for means (student's t-tests) is used to ascertain if the null hypothesis (means of two populations are equal) can be accepted or rejected. Since the p–value is not less than our alpha, 0.05, there is no significant difference in the means between the obtained results and best-known results. It can be concluded that our approach is no worse than previous reported best results. Table 4-6 reports the minimum, average, and maximum values of the cost, time and iteration number among 10 HCA executions for each instance.

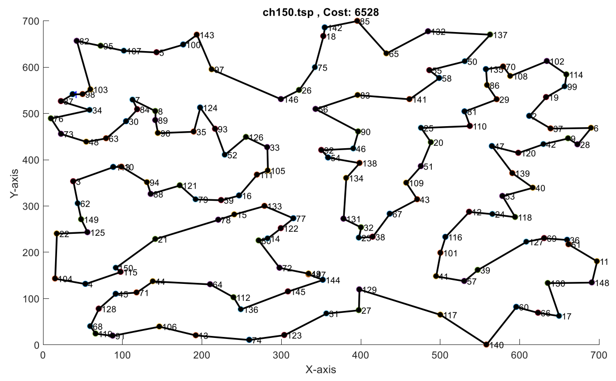
Table 4-6. Minimum, average, and maximum HCA results on benchmark TSP instances

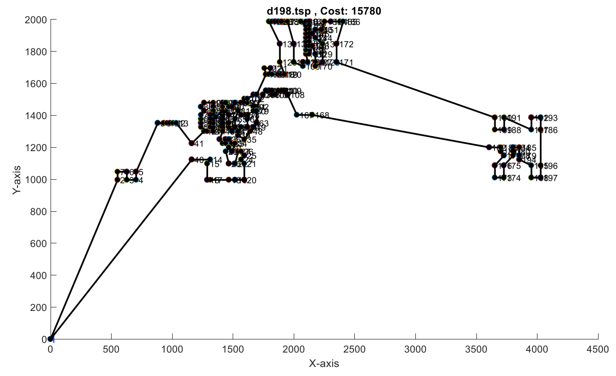
Instance name		**Cost**	**Time (s)**	**Iteration number**	**Instance name**		**Cost**	**Time (s)**	**Iteration number**
	Min	7542	5.15	5		Min	29455	455.53	35
berlin52	Avg.	7565.3	5.36	37.9	kroB200	Avg.	29519.9	464.62	200.6
	Max	7758	5.78	55		Max	29612	474.95	305
	Min	6110	93.1	53		Min	20749	39.54	11
ch130	Avg.	6128.9	95.8	168.2	kroC100	Avg.	20751	39.74	71
	Max	6177	101.6	359		Max	20769	39.96	303
	Min	6528	150	17		Min	21294	40.10	5
ch150	Avg.	6550.8	157.6	154.4	kroD100	Avg.	21416.4	40.43	151.4
	Max	6570	162.1	347		Max	21772	40.79	299
	Min	15780	415	47		Min	22068	40.27	23
d198	Avg.	15785.3	422	209.6	kroE100	Avg.	22152.9	40.63	107
	Max	15794	432.7	593		Max	22389	41.07	203
	Min	426	4.70	11		Min	14379	46.59	11
eil51	Avg.	426.85	4.73	47.2	lin105	Avg.	14385.6	47.25	111.8
	Max	430	4.79	86		Max	14412	47.75	263
	Min	538	16.19	11		Min	108159	16.47	5
eil76	Avg.	538.5	16.34	47.8	pr76	Avg.	108163.3	16.58	32
	Max	539	16.45	137		Max	108202	16.84	215
	Min	629	41.37	34		Min	44303	48.25	5
eil101	Avg.	632	41.60	99.9	pr107	Avg.	44367.5	48.84	80
	Max	638	41.85	274		Max	44438	49.35	293
	Min	21282	40.39	23		Min	59030	78.60	5
kroA100	Avg.	21308.1	40.7	112.4	pr124	Avg.	59030	79.19	47
	Max	21369	41.0	275		Max	59030	79.96	101
	Min	26614	161.4	11		Min	96861	109.96	41
kroA150	Avg.	26742.2	162.7	204.2	pr136	Avg.	96985.1	110.93	204.2
	Max	26917	163.7	371		Max	97235	113.14	371
	Min	29368	461.1	29		Min	2323	385.41	29
kroA200	Avg.	29396.3	470	150.2	rat195	Avg.	2334.6	390.44	314.6
	Max	29518	479.5	299		Max	2343	396.57	557
	Min	22141	39.7	5		Min	675	12.42	11
kroB100	Avg.	22222	40.0	19.4	st70	Avg.	676.5	12.59	77.2
	Max	22258	40.4	101		Max	681	12.71	182
	Min	26132	158	83		Min	126643	626.73	125
kroB150	Avg.	26216.2	161.3	217.4	ts225	Avg.	126788.1	636.24	336.2
	Max	26329	165.1	419		Max	126962	643.72	647

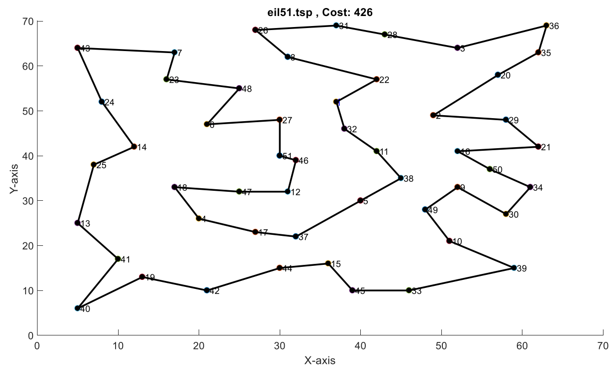
The results in Table 4-6 demonstrate the efficiency and effectiveness of the HCA algorithm. In particular, the average result and best-known solution are very close in all instances. The maximum difference was 0.00823% on the kroA150 benchmark, and zero on the pr124 benchmark. Moreover, the HCA optimised the solution on most benchmarks within a few iterations. This early convergence is attributed to information sharing among the water drops, and the use of the 2-Opt operation in the condensation stage. The solutions to the benchmark instances are displayed in Figure 4-10.

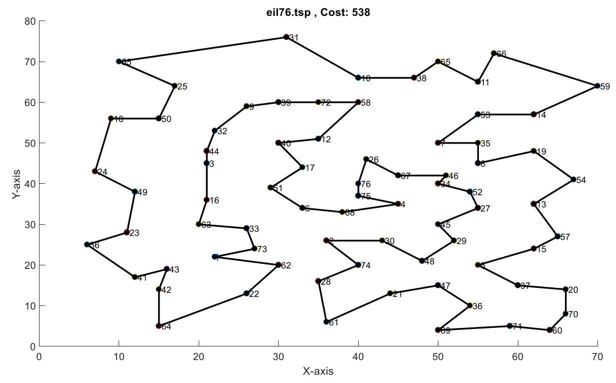

1. berlin52, Cost =7542

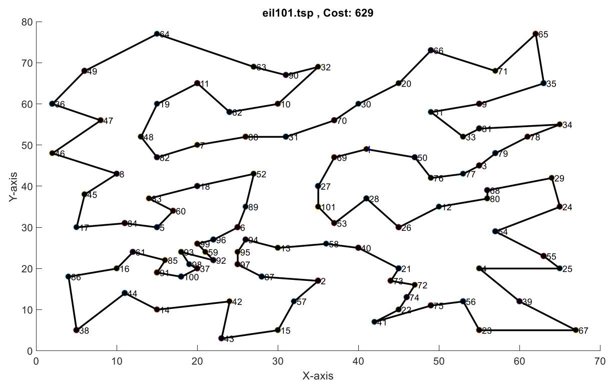

2. ch130, Cost = 6110

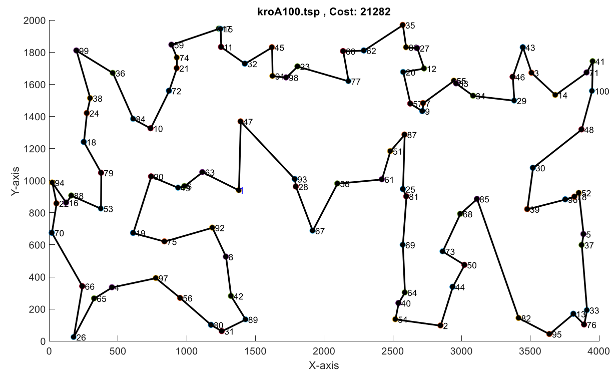

3. ch150, Cost = 6528

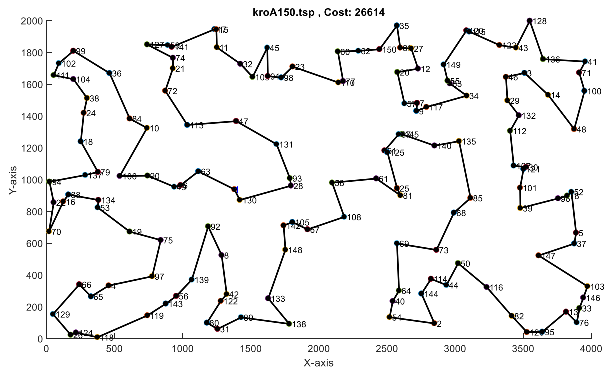

4. d198, Cost = 15780

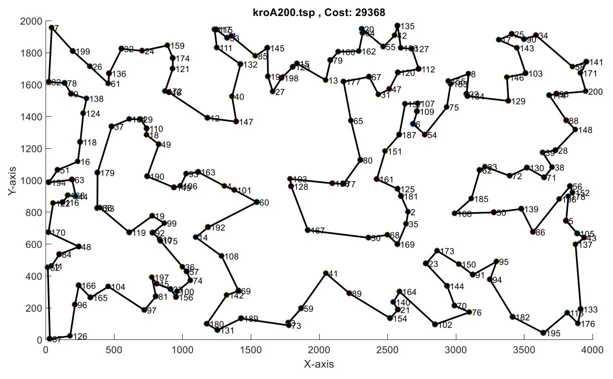

5. eil51, Cost = 426

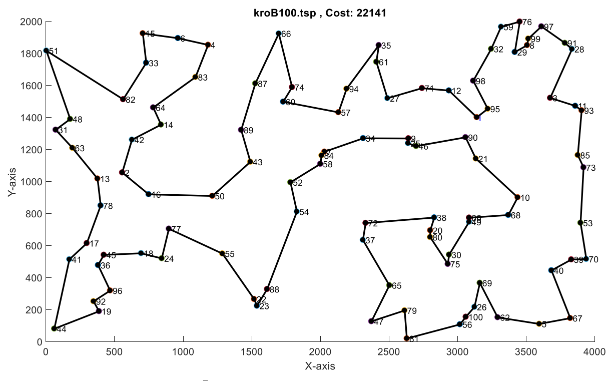

6. eil76, Cost = 538

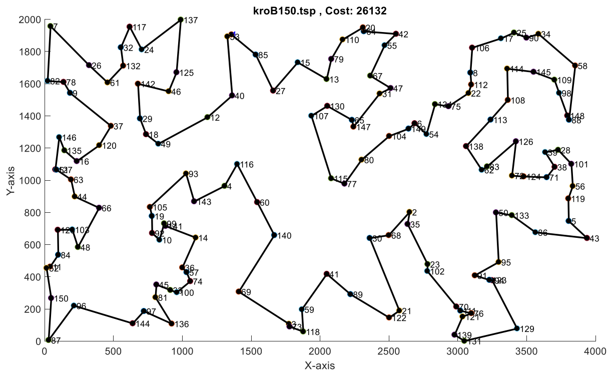

7. eil101, Cost = 629

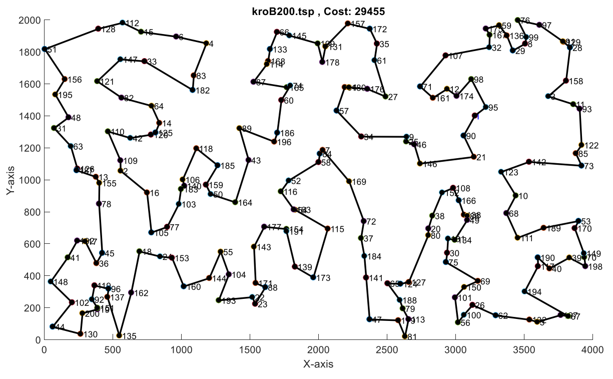

8. kroA100, Cost = 21282

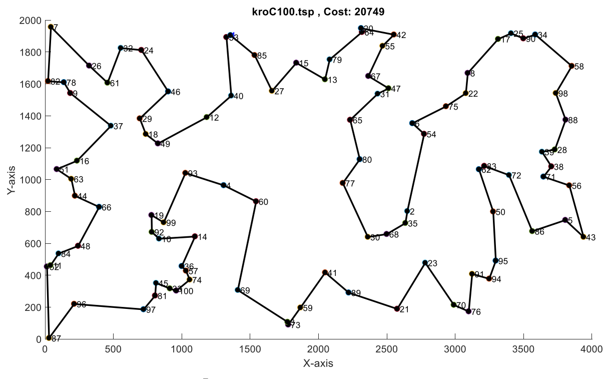

9. kroA150, Cost = 26614

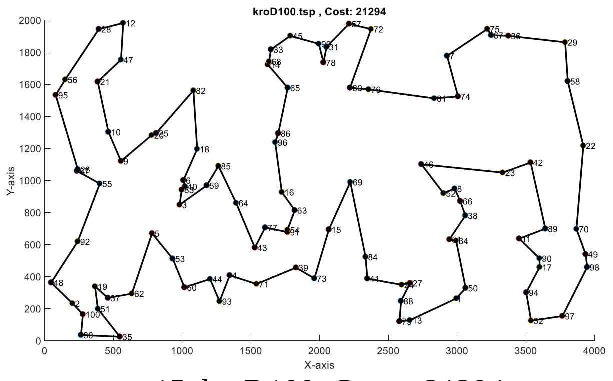

10. kroA200, Cost = 29368

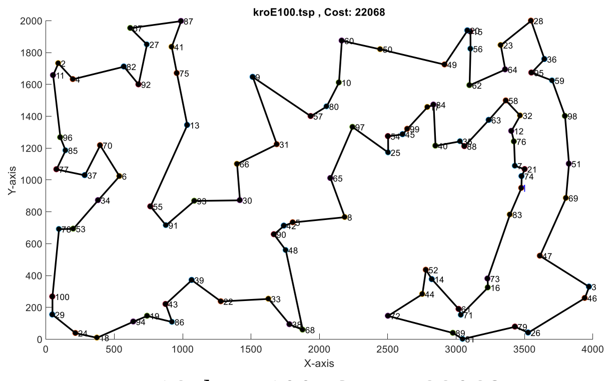

11. kroB100, Cost = 22141

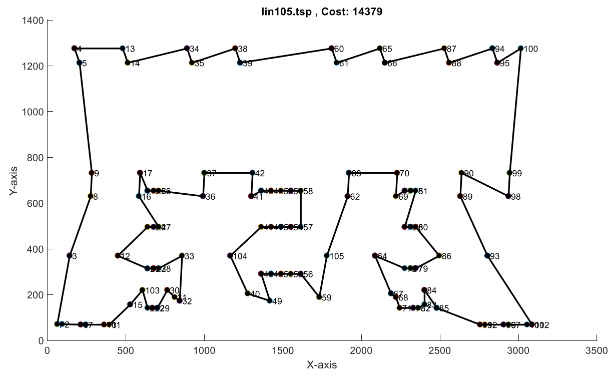

12. kroB150, Cost = 26132

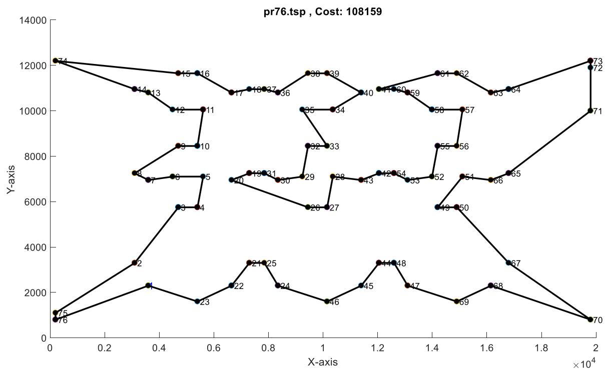

13. kroB200, Cost = 29455

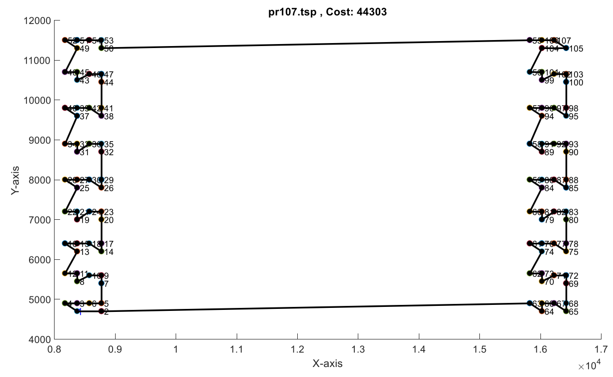

14. kroC100, Cost = 20749

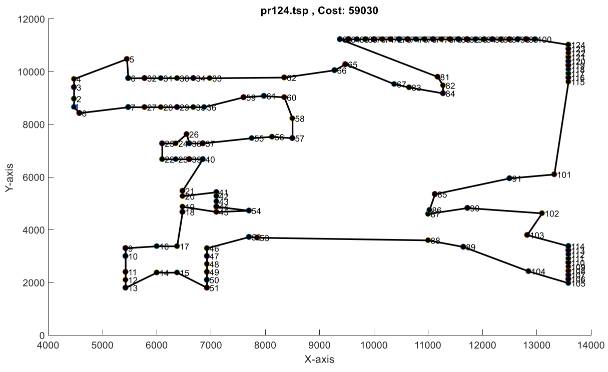

15. kroD100, Cost = 21294

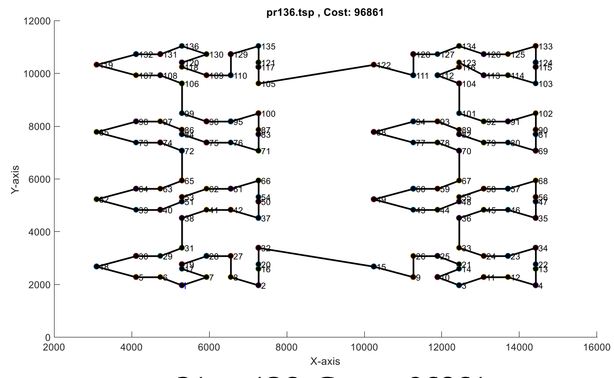

16. kroE100, Cost = 22068

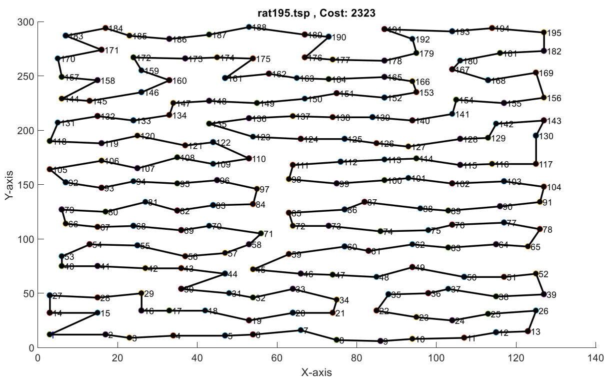

17. lin105, Cost = 14379

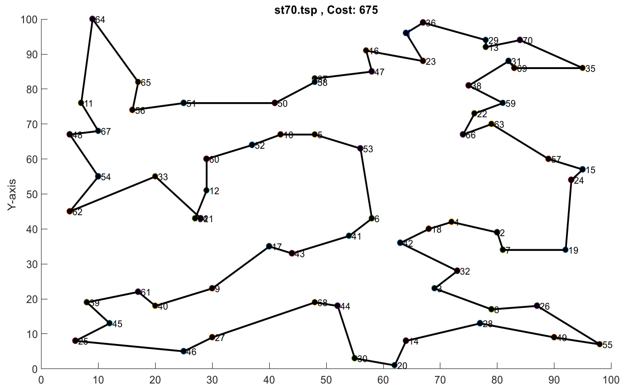

18. pr76, Cost = 108159


19. pr107, Cost = 44303


20. pr124, Cost = 59030


21. pr136, Cost = 96861


22. rat195, Cost = 2323


23. st70, Cost = 675

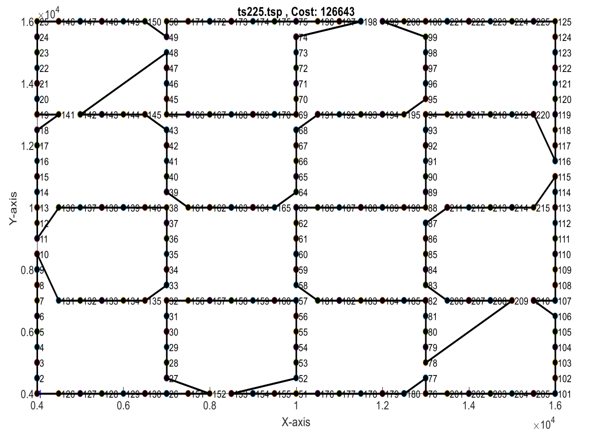

24. ts225, Cost = 126643

Figure 4-10. HCA outputs on benchmark instances

The minimal cost in HCA was compared with the reported results of other water-based algorithms, namely, the intelligent water drops (IWD) algorithm and its modifications, water wave optimisation (WWO), the water flow-like algorithm (WFA), and river formation dynamics (RFD). The comparisons are summarised in Table 4-7. The results of the original and a modified IWD (columns 4 and 5, respectively) were taken from (Shah-Hosseini, 2007) and from (Gülcü, Gülcü, Kahramanli, Campus, & Selçuklu, 2013), respectively. The results of another modified IWD, called the exponential ranking selection IWD (ERS-IWD; column 6), were extracted from (Alijla et al., 2014). The results of columns 7 and 8 were taken from (Msallam & Hamdan, 2011), who implemented the IWD and their proposed adaptive IWD on TSP instances. The WWO results (column 9) were taken from (X.-B. Wu et al., 2015). The WFA and RFD results (columns 10 and 11) were borrowed from (Srour et al., 2014) and from (Rabanal et al., 2009), respectively. The best results are marked in bold font.

Table 4-7. Best results of HCA, the original IWD, modified IWDs, WWO, WFA, and RFD

Instance name	**Best-known result**	**HCA**	**Original IWD (4)**	**IWD (5)**	**ERS-IWD (6)**	**Adaptive IWD**		**WWO (9)**	**WFA (10)**	**RFD (11)**
						IWD (7)	**AIWD (8)**			
berlin52	7542	**7542**	-	**7542**	-	-	-	-	-	-
ch130	6110	**6110**	-	-	6316	-	-	6338	**6110**	-
ch150	6528	**6528**	-	-	-	-	-	7014	**6528**	-
eil51	426	**426**	471	**426**	429	434	**426**	427	**426**	441.9
eil76	538	**538**	559	540	545	552	**538**	557	**538**	-
eil101	629	**629**	-	639	654	-	-	-	**629**	-
kroA100	21282	**21282**	23156	21429	21959	23183	21304	21668	**21282**	-
kroA150	26524	26614	-	-	-	-	-	-	**26524**	-
kroA200	29368	**29368**	-	-	31680	-	-	31064	**29368**	-
kroC100	20749	**20749**	-	20816	-	-	-	-	-	-
lin105	14379	**14379**	-	14393	14696	-	-	-	-	-
pr76	108159	**108159**	-	109608	-	-	-	-	-	-
rat195	2323	**2323**	-	-	-	2461	2338			-
st70	675	**675**	-	676	-	710	**675**	-	-	-
ts225	126643	**126643**	-	-	-	275791	127325	-	-	-
Average	24791.7	24797.7	8062	19563.2	10897	50521.8	25434.3	11178	11426	441.9
(T-test) *P*-values vs HCA			0.4025	0.2701	0.1598	0.3219	0.1878	0.1302	0.2813	-

The numbers of instances solved by these algorithms are insufficient for calculating an accurate *P*-value statistic. Moreover, some of these algorithms perform as well as HCA in certain instances. However, as confirmed in Table 4-7, HCA outperforms the original IWD algorithm and its various modifications. One plausible reason for the poor performance of the IWD algorithm is the premature convergence and stagnation in local optimal solutions. In contrast, HCA can escape from local optima by exploiting the depths of the paths along with the soil amount. These actions diversify the solutions. The most competitive opponent to HCA was WFA, which also optimised the solutions in the tested instances. In contrast, the WWO performed poorly because this algorithm was originally designed for continuous-domain problems, and its operations need adjustment for combinatorial problems. Moreover, the WWO adopts a reducing population-size strategy, which degrades its performance in some problems. Finally, the WWO suffers from slow convergence because it depends only on the altitude of the nodes.

The performances of HCA, IWD, adaptive IWD (AIWD) and modified IWD (MIWD) are further compared in Table 4-8. The best and average results of IWD and AIWD were taken from (Msallam & Hamdan, 2011), while those of MIWD were taken from (Shah-Hosseini, 2009).

Table 4-8. Best and average results of HCA, IWD, AIWD, and MIWD.

Instance Name	**HCA**		**IWD**		**AIWD**		**MIWD**	
	Best	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**
Eil51	426	426.85	434	443.2	426	428.4	428.98	432.62
St70	675	676.5	710	724.93	675	682.5	677.1	684.08
Eil76	538	538.5	552	564.43	538	542.86	549.96	558.23
KroA100	21282	21308.1	23183	23548.37	21304	21586.73	21407.57	21904.03
rat195	2323	2334.6	2461	2480.6	2338	2347.8	-	-
ts225	126643	126788.1	755791	276140.75	127325	128323.5	-	-
Average	29912.8	29947.6	130521.8	50650.4	25434.3	25652.0	5765.9	5894.7
T-test (*P*-values) vs HCA			0.3722	0.3656	0.3570	0.2867	0.3209	0.3611

This comparison aims to compare the robustness of HCA and other algorithms. Despite there being no significant differences between the results (best, average), the average results are closer to the best-known results in HCA than in the other algorithms, suggesting the superior robustness of HCA. Table 4-9 compares the runtimes of the HCA, IWD and AIWD. The best and average execution times and iteration numbers of the IWD algorithms were taken from (Msallam & Hamdan, 2011).

Table 4-9. Average execution times and best and average iteration numbers in HCA, IWD, and Adaptive IWD

Instance name	**HCA**		**IWD**		**Adaptive IWD**	
	Avg. Time (s)	**Iteration [Best, Avg.]**	**Avg. Time (s)**	**Iteration [Best, Avg.]**	**Avg. Time (s)**	**Iteration [Best, Avg.]**
eil51	4.73	[57, 47.2]	154.537	[1,509, 3000]	180.648	[190, 3000]
st70	12.59	[83, 77.2]	434.193	[960, 3,500]	453.631	[1769, 3500]
eil76	16.34	[46, 47.8]	567.208	[2147, 3,500]	571.251	[752, 3500]
kroA100	40.7	[89, 112.4]	1364.979	[3,698, 3750]	1365.752	[2397, 3,750]
rat195	390.44	[401, 314.6]	2023.162	[604, 5000]	2335.9392	[4995, ~]
ts225	636.24	[365,336.2]	3969.892	[1, 5000]	4162.92	[3,850, 5000]
Average	142.1		1419.0		1511.7	
T-test (*P*-value) for Avg. Time			0.1162		0.1200	

According to Table 4-9, HCA reaches the best solution after fewer iterations than IWD and AIWD. This result confirms the superior efficiency of HCA. Moreover, adding the other stages of the water cycle did not affect the average execution time of HCA. Figure 4-11 plots the average execution times of the three algorithms implemented on five benchmark problems.

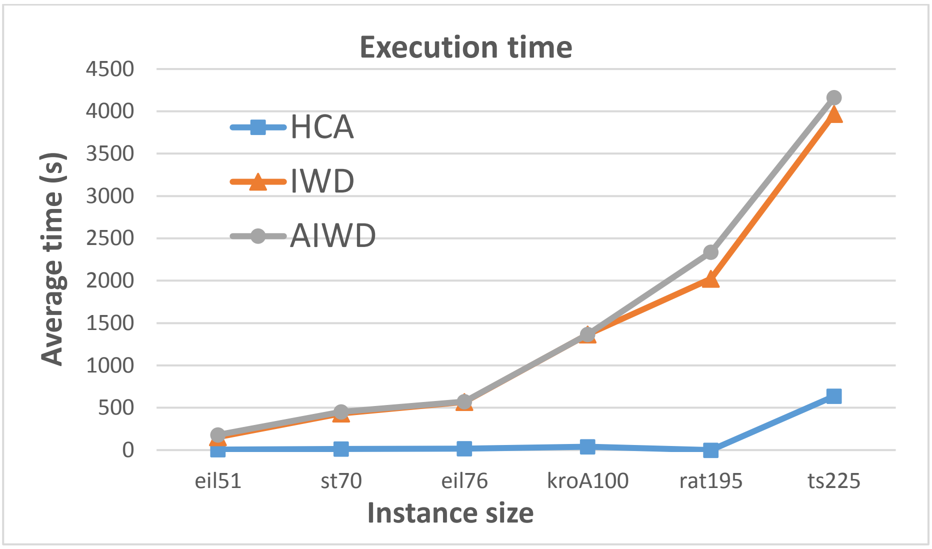


Figure 4-11. Average execution times of HCA, IWD, and Adaptive IWD

Solution searching by HCA was compared with those of other well-known algorithms, namely, an ACO algorithm combined with fast opposite gradient search (FOGS–ACO) (Saenphon, Phimoltares, & Lursinsap, 2014), a genetic simulated annealing ant colony system with PSO (GSAACS–PSO) (S.-M. Chen & Chien, 2011), an improved discrete bat algorithm (IBA) (Osaba et al., 2016), set-based PSO (S-CLPSO) (W.-N. Chen et al., 2010), a modified discrete PSO with a newly introduced mutation factor C3 (C3D–PSO); results taken from (Zhong et al., 2007), an adaptive simulated annealing algorithm with greedy search (ASA–GS) (Geng et al., 2011), the firefly algorithm (FA) (M. Wang, Fu, Tong, Li, & Zhao, 2016), a hybrid ACO enhanced with dual NN (ACOMAC–DNN) (C.-F. Tsai, Tsai, & Tseng, 2004), a discrete PSO (DPSO) (Shi et al., 2007), a self-organizing neural network using the immune system (ABNET–TSP) (Masutti & de Castro, 2009), and an improved discrete cuckoo search algorithm (IDCS) (Ouaarab et al., 2014). Table 4-10 summarises the comparison results.

Table 4-10. Best results obtained by HCA and other optimisation algorithms

Instance name	Best-known result	HCA	FOGS–ACO	GSAACS-PSO	IBA	S-CLPSO	C3D-PSO	ASA-GS	FA	ACOMAC-DNN	DPSO	ABNET-TSP	IDCS
berlin52	7542	**7542**	7546.62	**7542**	**7542**	**7542**		7544.37	7544.36	-	**7542**	**7542**	**7542**
ch130	6110	**6110**	-	6141	-	-		6110.72	-	-	-	6145	**6110**
ch150	6528	**6528**	-	**6528**	-	-		6530.9	-	-	-	6602	**6528**
eil51	426	**426**	**426**	427	**426**	**426**	**426**	428.87	428.87	430.01	427	427	**426**
eil76	538	**538**	546.83	**538**	539	**538**	**538**	544.37		552.61	546	541	**538**
eil101	629	**629**	633.40	630	634	**629**		640.212			-	638	**629**
d198	15780	**15780**		-	-	15809		15830.6		15,955.6	-		15781
kroA100	21282	**21282**	22414	**21282**	**21282**	**21282**	**21282**	21285.4	21285.4	21,408.23	-	21333	**21282**
kroA150	26524	26614	-	**26524**	-	26537		26524.9		-	-	26678	**26524**
kroA200	29368	**29368**	29717	29383	-	29399		29411.5		-	-	29600	**29382**
kroB100	22141	**22141**	-	**22141**	22140*	-		22139.1	22139.07	-	-	22343	**22141**
kroB150	26130	26132	-	**26130**	-	-		26140.7	-	-	-	26264	**26130**
kroB200	29437	29455	-	29541	-	-		29504.2	-	-	-	29637	**29448**
kroC100	20749	**20749**	-	**20749**	**20749**	20824.6		20750.8	-	-	-	20915	**20749**
kroD100	21294	**21294**	-	21309	**21294**	21405.6		21294.3	-	-	-	21374	**21294**
kroE100	22068	**22068**	-	**22068**	**22068**	-		22106.3	-	-	-	22395	**22068**
lin105	14379	**14379**	-	**14379**	-	**14379**		14383	14383	-	-	**14379**	**14379**
pr76	108159	**108159**	108864	-	-	**108159**		108159		-	108280	-	**108159**
pr107	44303	**44303**	-	-	**44303**			44301.7*	44346	-	-	-	**44303**
pr124	59030	**59030**	-	-	**59030**			59030.7	**59030**	-	-	-	**59030**
pr136	96772	96861	-	-	97547			96966.3	97182.74	-	-	-	**96790**
rat195	2323	**2323**	-	-	-			2345.22		-	-	-	2324
st70	675	**675**	678.93	-	**675**	**675**	**675**	677.11	677.11	-	**675**	-	**675**
ts225	126643	**126643**	-	-	-	-		126646	-	-	-	-	**126643**
Average	29534.6	29542.9	21353.3	16062.2	24479.2	20585.0	5730	29554	29669	9587	23494	16051	29536.5
(T-test) *P*-values versus HCA			0.1120	0.6755	0.3327	0.3088	-	0.1129	0.2684	0.1536	0.3360	0.0014	0.1890

Note. '' means incorrect value*

Although the complexity of the TSP increases with increasing number of cities, the HCA outperformed the other algorithms in most instances. However, the *P*-values indicate there are no significant differences between HCA and other algorithms, except between HCA and ABNET-TSP. The HCA competed with other algorithms such as the IDCS algorithm; indeed, the results of HCA and IDCS were not noticeably different even for large problems. The high performance of HCA was again attributed to the effective design of the HCA that included an information sharing process among the water drops. This process helps the HCA exploit the promising solutions and increases the speed of algorithm convergence. The additional stages of the HCA assist with exploring different solutions (enhancing the search capability), and prevent trapping in local optima.

4.3.3 HCA Convergence Evaluation

This section analyses the performance of the HCA and its convergence rate. As previously stated, the maximum iteration number was set to three times the number of nodes in the instance. Figure 4-12 shows the convergence of the algorithm on the berlin52 instance. The cost along the Y-axis denotes the total route length.

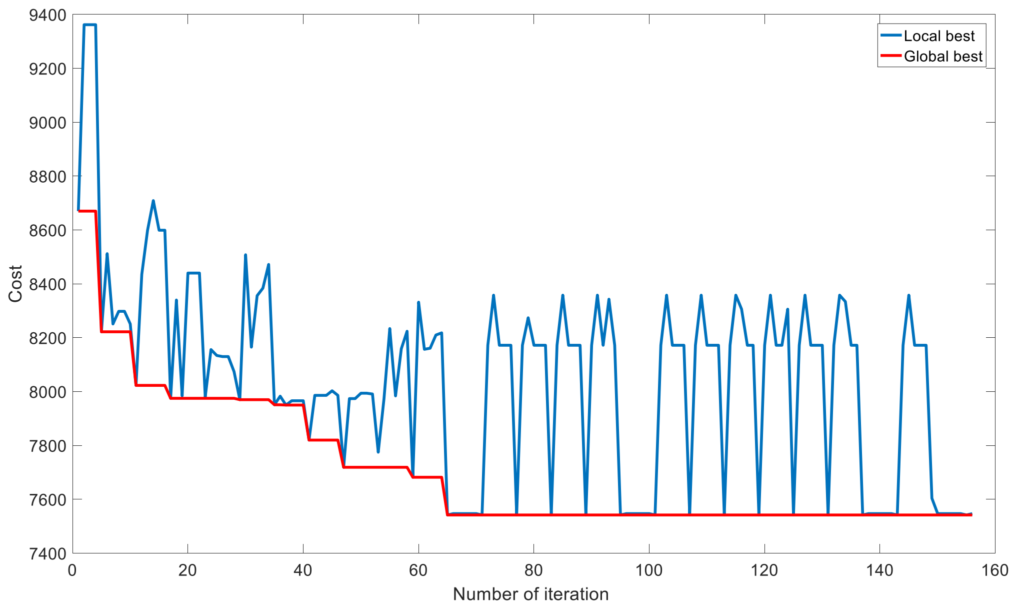


Figure 4-12. Local (blue) and global (red) best solutions on berlin52

According to Figure 4-12, the solution was optimised after 65 iterations. The berlin52 benchmark is relatively easy to solve because the node distribution reduces the possibility of falling into local optima. The local and global solutions are the best solution at the end of each iteration and the best solution among all iterations, respectively. Note that the algorithm converges towards the best-known solution. In addition, the HCA generated different solutions in every iteration, and the search process was prevented from stagnating by the depth factor and the information sharing among the water drops. The depth factor increases the chance of selecting previously unexplored or little-used paths. Figure 4-13 shows the convergence of the algorithm on the eil51 instance. The solution was optimised at the 64th iteration.

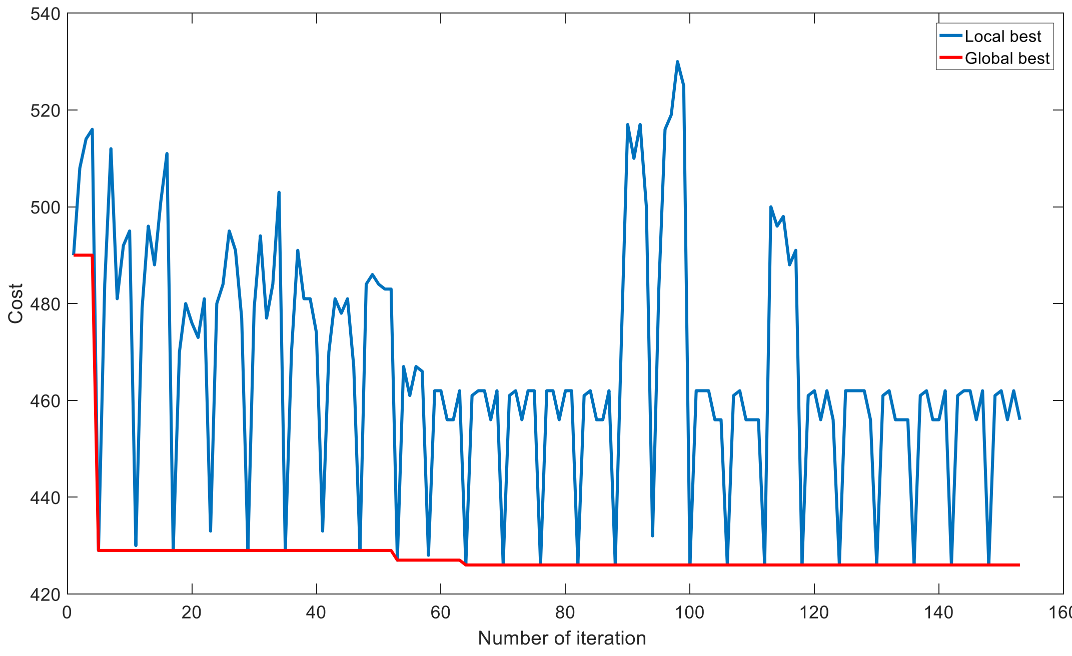


Figure 4-13. Local (blue) and global (red) best solutions on eil51

Figure 4-14 illustrates the convergence behaviour of the HCA on the eil67 instance. Here, the solution was optimised at iteration 171.

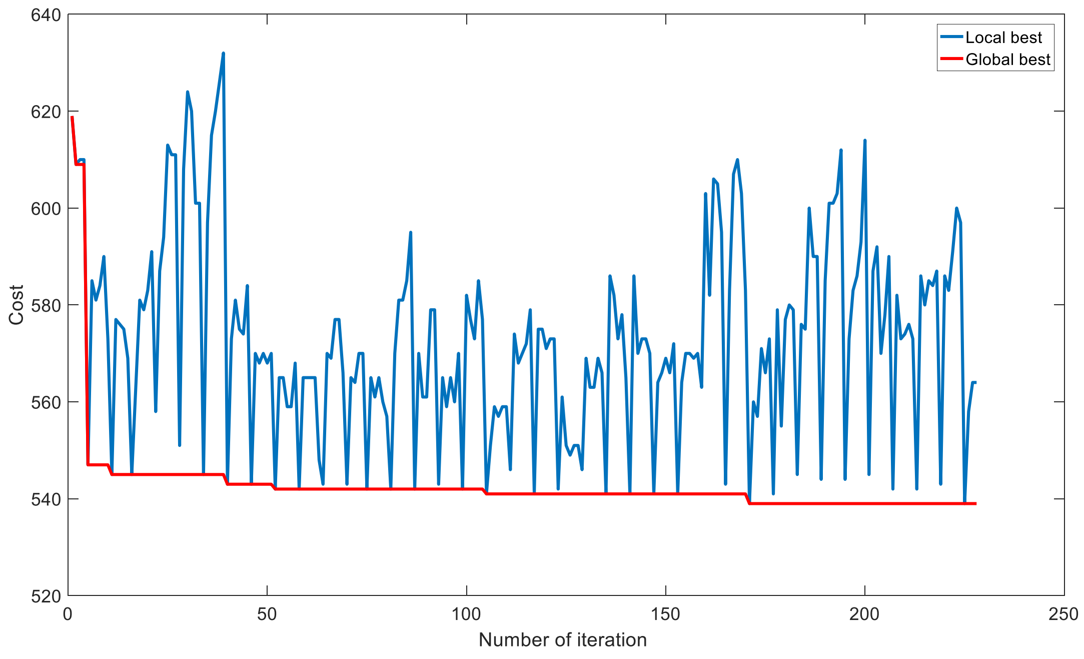


Figure 4-14. Local (blue) and global (red) best solutions on eil76

Figure 4-15 illustrates the convergence behaviour of the HCA on the eil101 instance. The best-known solution was found at iteration 99. Moreover, the smooth convergence rate confirms the good balance between the exploration and exploitation processes.

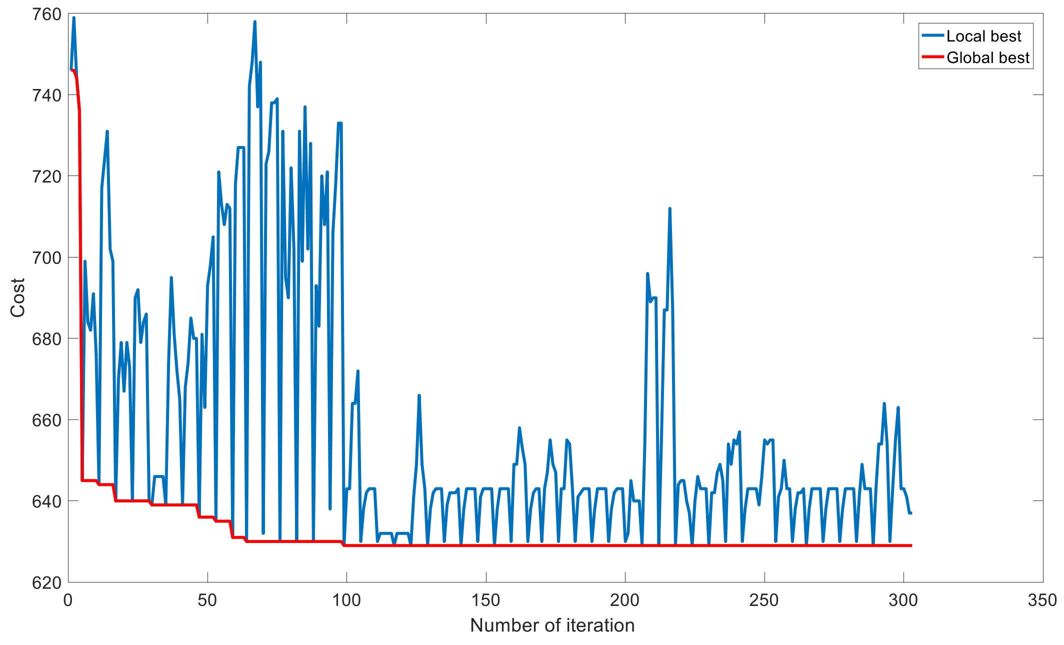


Figure 4-15. A graph for local vs. global solution on eil101

Figure 4-16 shows the convergence of the global best solution on the st70 instance. The solution was optimised at iteration 125.

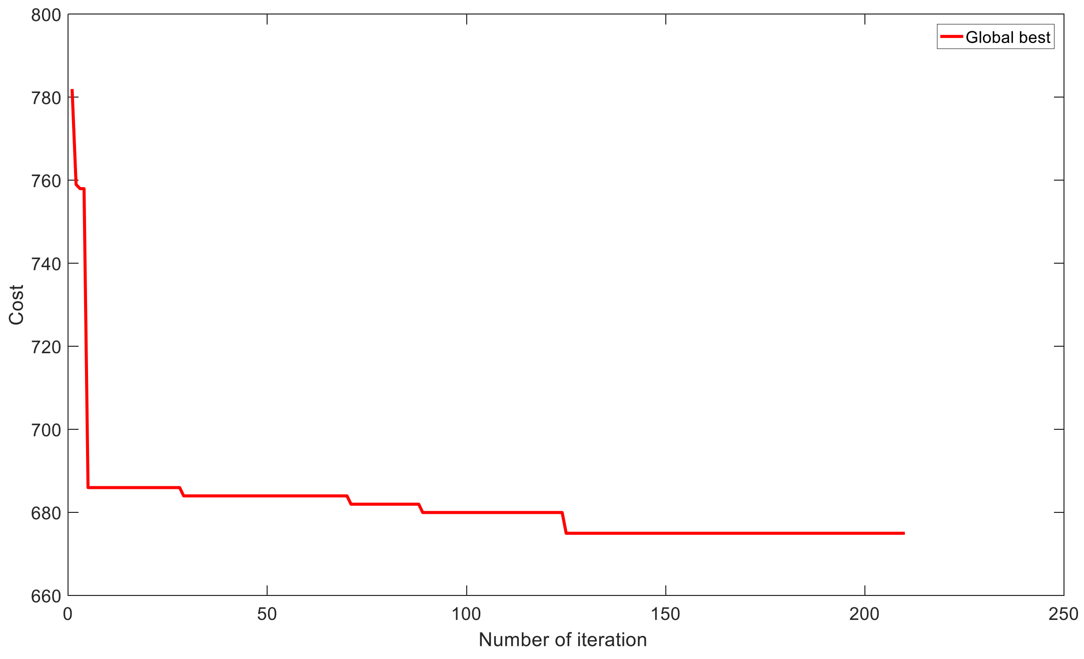


Figure 4-16. Global best solution on st70

In summary, the convergence rate of the HCA demonstrates the effectiveness of the algorithm design. The algorithm is able to search for the best-known solution until the final iterations, without stagnation in local optima. It also converges rapidly on easy instances.

4.4 Conclusion

In this chapter, HCA was applied on an archetypal NP-hard problem (the TSP). Initially, the performance of the algorithm was tested on simple geometric structures. Parameter tuning was also performed on these structures. The results obtained indicate the flexibility and capability of the algorithm in solving such problems. Moreover, the algorithm provided different same-cost solutions to the same problem (i.e. providing diverse solutions). This validates the effective design of the exploration and exploitation processes of the algorithm. Also, as confirmed by the convergence behaviour of the algorithm, the HCA successfully avoids potential stagnation in local optima. The geometric TSP instances can be considered as a minor contribution, as their simple design is useful for evaluating other new algorithms, and different shapes can be designed by the same principle.

Next, the algorithm was tested on various standard benchmarks taken from the literature. It obtained the best-known solution in most instances, confirming the effectiveness of the algorithm framework. The algorithm provided high-quality solutions and outperformed other metaheuristic algorithms in seeking the minimum path. Additionally, the HCA found the best-known solution within a few iterations and its ability to escape from local optima and find the global solution was demonstrated. The strong optimisation capability of the HCA is conferred by the efficient design of the exploration and exploitation processes. Moreover, by sharing the information among the water drops, the algorithm steers towards better solutions within a small number of iterations.

The HCA structure is a feasible approach for solving symmetric TSPs. However, the HCA performance could additionally be investigated on asymmetric TSP instances. Although the HCA optimises the TSP solution within a reasonable timeframe, further enhancements would reduce its execution time on large instances.

After demonstration the success of HCA in solving a discrete and static problem. In the next chapter, the HCA will be evaluated on continuous problems.

Chapter 5 Solving Continuous Optimisation Problems using HCA

"Premature optimisation is the root of all evil."

[Donald Knuth]

As established in Chapter 4, the HCA delivers high performance in discrete optimisation problems. This chapter demonstrates that HCA is also applicable to continuous optimisation problems (COPs). The algorithm requires no major conceptual change to its structure; instead, we apply a new solution representation proposed for COPs. The main goal here is to prove that the algorithm can solve COPs represented by numeric mathematical benchmark functions. This application also measures the algorithm's performance in finding the globally optimal solution, convergence ability, and identifies its strengths and weakness on different kinds of problems. In addition, the test functions demonstrate the ability of the algorithm to escape from local optima and explore new regions. Finally, the behaviour and convergence speed of the algorithm, and its ability to deal with multimodal functions, are evaluated. The HCA is tested on benchmark functions with diverse structures and various numbers of variables. We emphasise that proving the superiority of this algorithm over other algorithms is not our main aim. We remark only that the HCA obtained high-quality solutions on these benchmark functions.

The rest of this chapter is organised as follows. Section 5.2 briefly explains the benchmark functions, and Section 5.3 explains the experimental setup and the application of the HCA algorithm to the selected problems. Section 5.4 presents the experimental results and compares them with the results of other algorithms. Discussion and conclusions are given in Section 5.5.

5.1 Introduction

Several problems in everyday life can be represented as COPs. For instance, manufacturers must maximise their efficiency and profit by optimising the factors that reduce the operation costs. In COP, the variables may take any values within a specific range (commonly, a real number between a specified minimum and maximum) (Chong & Zak, 2013). Many of the existing mathematical benchmark functions are COPs.

5.2 Mathematical Benchmark Functions

Various continuous benchmark functions can be found in the literature (Jamil & Yang, 2013). When an optimisation algorithm is applied on one of these functions, it must find the value that minimises or maximises the function. Benchmark functions have various structures and can be unimodal (with a single optimum) or multimodal (with several local optima and one or more global optima). Each function depends on one or more variables. Typically, a unimodal function has one variable, whereas a multimodal function has several variables. Examples of a unimodal and a multimodal function are depicted in Figure 5-1.

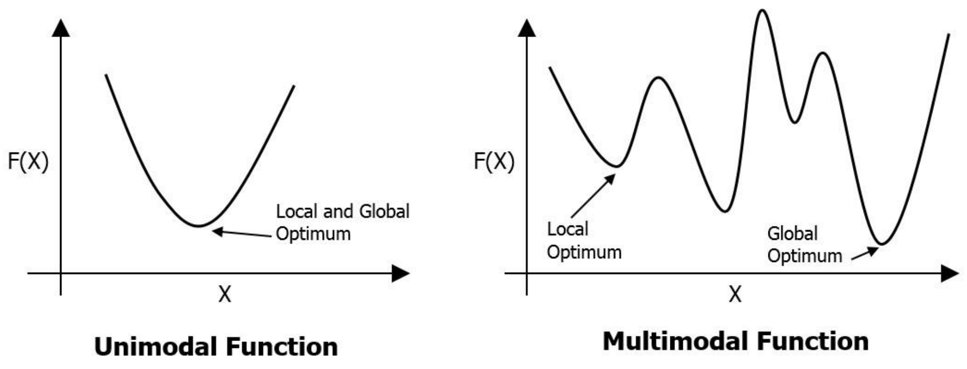


Figure 5-1. A unimodal (left) and a multimodal (right) function

Diverse benchmark functions can validate the robustness of a new optimisation algorithm. By implementing their algorithms on different benchmark functions, researchers can evaluate the algorithm's performance, convergence speed and behaviour in different situations, compete the algorithm against existing optimisation algorithms, and generalise it to different kinds of problems. For instance, unimodal functions are useful for measuring convergence speeds, whereas multimodal functions check whether an algorithm can escape local optima and explore new regions. Flat-surface functions (like Easom) are especially challenging for reaching the global optimum, as the flatness provides no directional guidance towards that optimum (Jamil & Yang, 2013).

In practise, the complexity of a function partly depends on the number of variables and the domain of each variable (the function dimensionality). Other factors, such as the number and relative distributions of the local and global optima, also critically determine a function's complexity, especially when a global minimum lies very close to a local minimum (Jamil & Yang, 2013). Some of these functions represent real-life optimisation problems, while others are artificial problems. A function can be further classified as unconstrained or constrained. An unconstrained function places no restrictions on the values of its variables. Mathematically, an unconstrained continuous objective function can be represented as follows:

$$minimise\ f(X): R^n \to \mathbb{R}\,, \tag{5.1}$$

where X is a vector of n decision variables

$$X = \{x_1, x_2, \dots, x_n\} \in \mathbb{R}\,. \tag{5.2}$$

A solution x^* is called a global minimiser of the problem $f(x)$ if $f(x^*) \leq f(x)$ for all $x \in X$. Conversely, it is called a global maximiser if $f(x^*) \geq f(x)$ for all $x \in X$.

5.3 Experimental Setup

This section explains the application of HCA to continuous problems. Unlike other algorithms, the HCA can be adapted to continuous problems without any changes in the algorithm structure. To achieve this, a new design of the solution space is proposed, as described below.

5.3.1 Problem Representation

Typically, the HCA input is represented as a graph in which water drops travel along the edges between nodes. To extend HCA to COPs, a directed graph was developed that represents continuous numbers in a topographical approach. The graph of a function with N variables, each with precision P, is composed of $(N \times P)$ layers. Each layer contains ten nodes labelled from zero to nine. Therefore, there are $(10 \times N \times P)$ nodes in the graph, as shown in Figure 5-2.

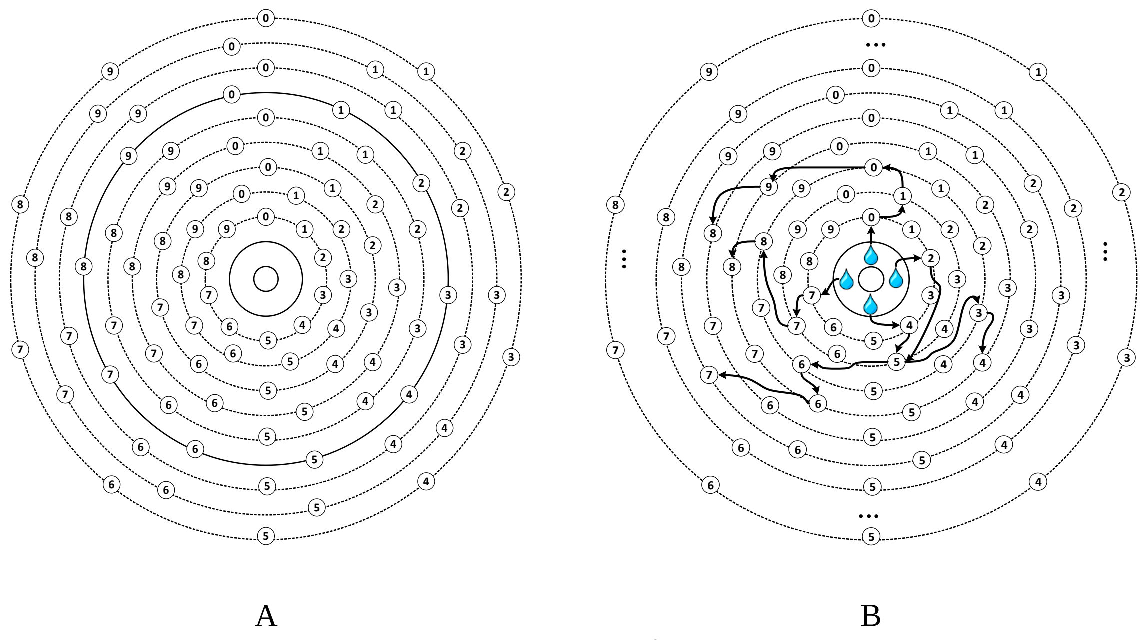


Figure 5-2. Graph representation of continuous variables

This graph can be seen as a topographical mountain with the layers represented by contour lines. To prevent a water drop from selecting two nodes from the same layer, the nodes within any given layer are unconnected. Instead, a node in one layer is connected to all nodes in the lower neighbouring layer. The value of a node is higher in layer i than in layer i+1. More specifically, the selected number in the first, second, and L^{th} layer is multiplied by 0.1, 0.01, and 10^{-L}, respectively. Mathematically, this relationship is expressed by Eq. (5.3).

$$X_{value} = \sum_{L=1}^{m} \mathrm{n} \times 10^{-L} \tag{5.3}$$

where m is the maximum layer number of each variable (representing the precision of the variable), and *n* is the node number in layer L. The water drops will generate values between zero and one for each variable. For example, if a water drop moves between node 2, 7, 5, 6, 2, and 4 respectively, then the variable value is 0.275624. A graph with real numbers was considered to speed the algorithm process instead of using binary numbers. However, the main drawback of the proposed representation is that an increase in the number of high-precision variables leads to an increase in search space and consequently increase the search time of the algorithm.

5.3.2 Variables Domain and Precision

A function has *n* variables (where *n* can be 1) and the function domain defines the set of possible values for a variable. In some functions, all variables have the same domain range; in others, the variable ranges differ. For simplicity, the values obtained by a water drop for each variable can be scaled to have values based on the domain of each variable (see Eq. (5.4)).

$$V_{value} = (U_B - L_B) \times Value + L_B \tag{5.4}$$

where V_{value} is the normalised value and U_B and L_B are the upper and lower bounds, respectively. For example, if the value obtained is 0.658752 and the variable's domain is [-10, 10], then the real value will be equal to 3.17504. Because continuous variables are expressed as real numbers, we must identify the precision (*P*) of the values, which specifies the number of digits after the decimal point. Real numbers are processed more quickly than graphs of binary numbers, because they bypass the encoding/decoding of the variables' values to and from binary values.

5.3.3 Solution Generator

To find the solution to a function, a number of water drops are placed on the peak of the continuous- variable mountain (graph). These drops flow down the mountain layers. After choosing a single number (node) in the next layer, a water drop moves to that layer. This process continues until all water drops have reached the last layer (ground level). The flows of different water drops may generate different solutions. However, as the algorithm iterates, some water drops begin converging towards the best solution by selecting the highest-probability nodes. The candidate solutions to a function can be represented as a matrix, in which each row contains one water drop solution. Each water drop generates a solution for all variables of the function.

Therefore, the water drop solution comprises a set of visited nodes based on the number of variables and the precision of each variable:

$$Solutions = \begin{bmatrix} WD_s^1 & 1 & 2 & \cdots & m_{N\times P} \\ WD_s^2 & 1 & 2 & \cdots & m_{N\times P} \\ WD_s^i & 1 & 2 & \cdots & m_{N\times P} \\ \vdots & & \cdots & & \\ WD_s^k & 1 & 2 & \cdots & m_{N\times P} \end{bmatrix} \tag{5.5}$$

An initial solution is generated randomly for each water drop based on the function dimensionality. After evaluating these solutions, the best combination is selected and favoured with less soil on the edges belonging to that solution. Governed by the algorithm procedure, the water drops subsequently start flowing and generating new solutions. To improve the performance, we distributed half of the water drops on randomly selected first-layer nodes, and the remaining half on the best first-layer nodes.

5.3.4 A Demonstrative Example

As an example, we consider a function with two variables to be minimised, as shown in Eq. (5.6). The minimum cost of this function is ($f(x, y) \approx -18.55$) at $x \approx 9.037220$ and $y \approx 8.668580$.

$$min \quad f(x,y) = (\, x \sin(4x)) + (1.1\, y \sin(2y)), \tag{5.6}$$

$$where \;\; 0 \le x \le 10, and \; 0 \le y \le 10. \tag{5.7}$$

This function has many local optima and a single global optimum within the range [0, 10], as illustrated in Figure 5-3.

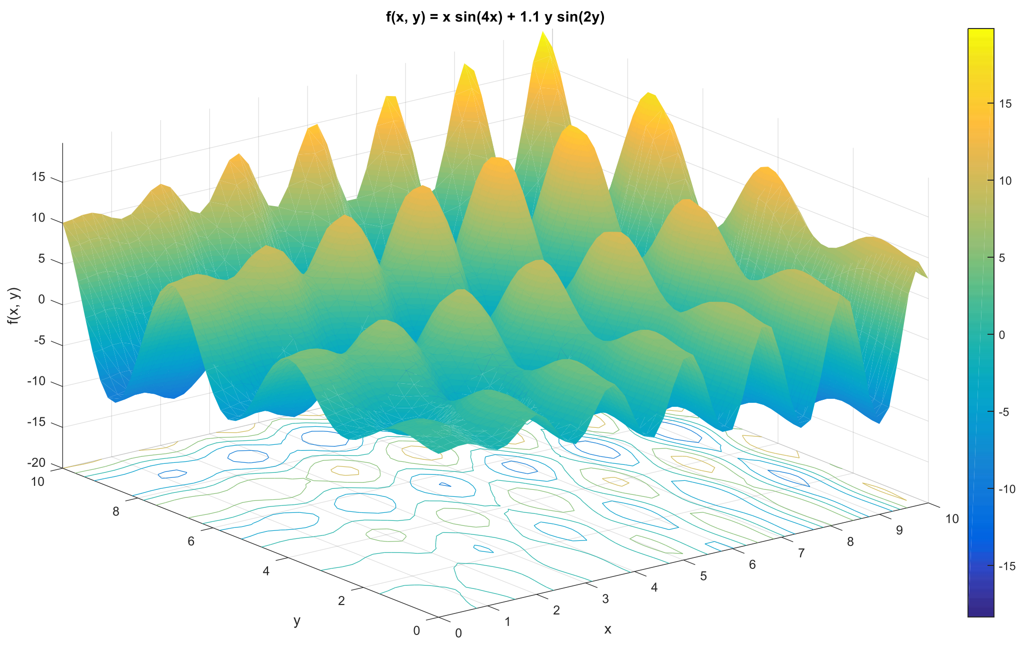


Figure 5-3. Graph of Eq. (5.6) in the range [0, 10]

Figure 5-4 shows the major steps in solving a mathematical function. Details are given in Section 3 of Chapter 3.

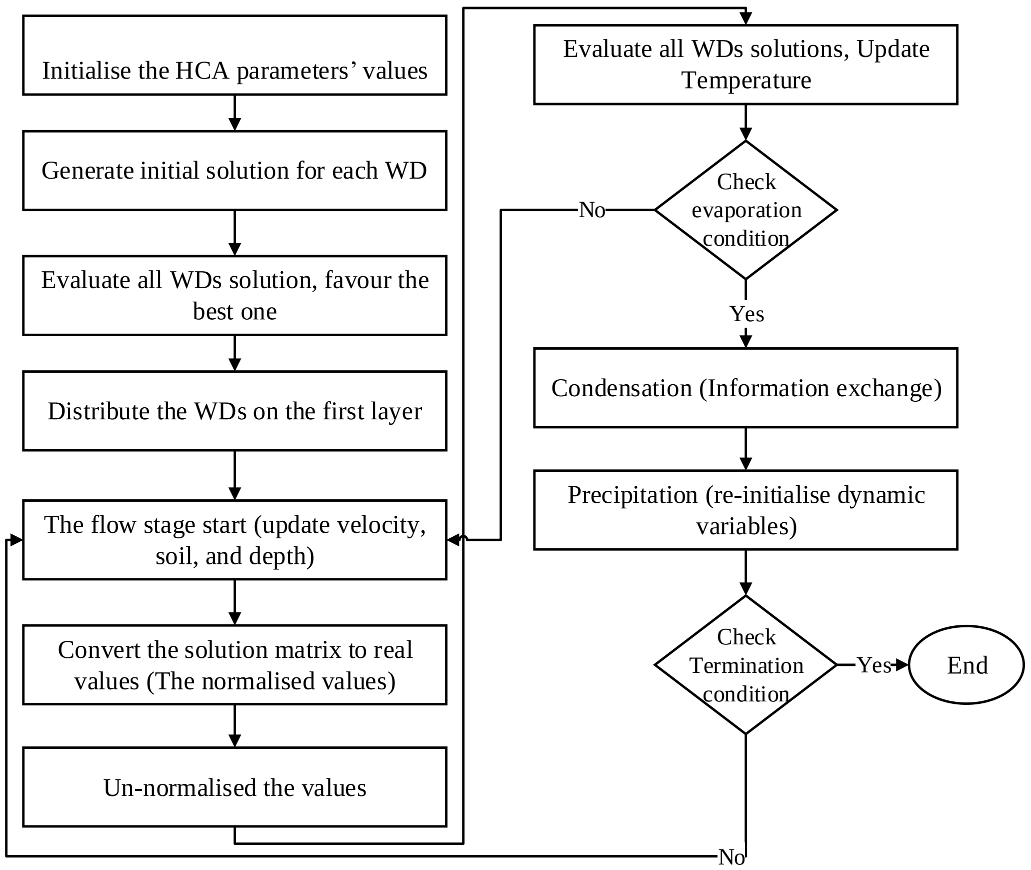

Initialise the HCA parameters' values
Generate initial solution for each WD
Evaluate all WDs solution, favour the best one
Distribute the WDs on the first layer
The flow stage start (update velocity, soil, and depth)
Convert the solution matrix to real values (The normalised values)
Un-normalised the values
Evaluate all WDs solutions, Update Temperature
Check evaporation condition
No
Yes
Condensation (Information exchange)
Precipitation (re-initialise dynamic variables)
Check Termination condition
Yes
End
No

Figure 5-4. HCA Flowchart for solving COPs

The solution of this example is assumed to be precise to six digits. Therefore, the graph consists of 12 layers. Ten water drops are generated and initialised with random values. Each water drop generates a different solution in each iteration, which is stored in a solution matrix as shown in Table 5-1.

Table 5-1. Order of the nodes visited by each water drop

Water drops	***x***						***y***					
	L1	***L2***	***L3***	***L4***	***L5***	***L6***	***L7***	***L8***	***L9***	***L10***	***L11***	***L12***
WD1	4	7	7	0	5	2	9	0	9	2	5	9
WD2	4	3	6	9	1	3	9	3	3	6	8	3
WD3	4	2	1	7	6	1	9	5	8	5	6	6
WD4	4	2	1	7	6	1	0	1	5	7	3	5
WD5	4	2	1	7	6	1	9	1	5	7	3	5
WD6	3	3	6	9	1	3	6	2	4	1	2	8
WD7	5	1	9	4	0	4	3	8	7	8	1	0
WD8	5	4	0	2	2	6	8	7	2	4	4	7
WD9	8	5	5	5	4	7	5	9	6	9	0	2
WD10	7	6	4	8	3	9	3	8	7	9	1	0

In Table 5-1, each row represents a candidate solution for the two variables generated by one water drop. The HCA was adjusted so that half of the water drops favour the nodes contained in the best solution, with some differences imposed by the depth factor. The remaining water drops generate alternative solutions through random searching. This technique helps with maintaining the balance between the exploration and exploitation phases. The sequence of nodes visited by each water drop is then converted to an equivalent real value. Finally, the obtained value of each variable is un-normalised to evaluate its fitness. These steps are repeated until the maximum number of iterations is reached. Table 5-2 shows the results of Equation 5.6 at specific iterations.

Table 5-2. Output of the algorithm after iterations 1, 2 and 150

Iteration Number	Water drops	Normalised values		Un-normalised values		*f (X, Y)*
		X	***Y***	***X***	***Y***	
1	WD1	0.472052	0.909259	4.720520	9.092590	-6.013316
	WD2	0.436913	0.933683	4.369130	9.336830	-6.081212
	WD3	0.421761	0.958566	4.217610	9.585660	-0.536365
	WD4	0.421761	0.915735	4.217610	9.157350	-9.005301
	WD5	0.421761	0.915735	4.217610	9.157350	-9.005301
	WD6	0.336913	0.624128	3.369130	6.241280	2.085529
	WD7	0.519404	0.387810	5.194040	3.878100	9.114246
	WD8	0.540226	0.872447	5.402260	8.724470	-7.443899
	WD9	0.855547	0.596902	8.555470	5.969020	-1.041478
	WD10	0.764839	0.387810	7.648390	3.878100	-1.358889
2	WD1	0.807933	0.852370	8.079330	8.523700	-2.790501
	WD2	0.893875	0.880456	8.938750	8.804560	-17.484382
	WD3	0.879256	0.874619	8.792560	8.746190	-14.458687
	WD4	0.879256	0.874619	8.792560	8.746190	-14.458687
	WD5	0.879256	0.874619	8.792560	8.746190	-14.458687
	WD6	0.321769	0.752370	3.217690	7.523700	6.042256
	WD7	0.748090	0.215735	7.480900	2.157350	-9.645766
	WD8	0.116111	0.007148	1.161110	0.071480	-1.147228
	WD9	0.534587	0.493827	5.345870	4.938270	0.680861
	WD10	0.407933	0.621281	4.079330	6.212810	-3.293494
150	WD1	0.90595	0.870747	9.0595	8.70747	-18.4946
	WD2	0.90595	0.870747	9.0595	8.70747	-18.4946
	WD3	0.90595	0.870747	9.0595	8.70747	-18.4946
	WD4	0.90595	0.870747	9.0595	8.70747	-18.4946
	WD5	0.58674	0.534539	5.8674	5.34539	-11.4513
	WD6	0.794288	0.627156	7.94288	6.27156	2.604903
	WD7	0.794288	0.961883	7.94288	9.61883	6.769391
	WD8	0.519516	0.489078	5.19516	4.89078	2.982634
	WD9	0.188429	0.961883	1.88429	9.61883	5.794613
	WD10	0.473031	0.456239	4.73031	4.56239	1.821894
		…	…	…	…	…
230				9.037220	8.668580	-18.554491

Note that the algorithm favoured the best solution in each iteration, and explored other regions that might improve the solutions. Ultimately, the algorithm found the best-known solution of the function. Figure 5-5 shows the convergence of the algorithm while evaluating this function.

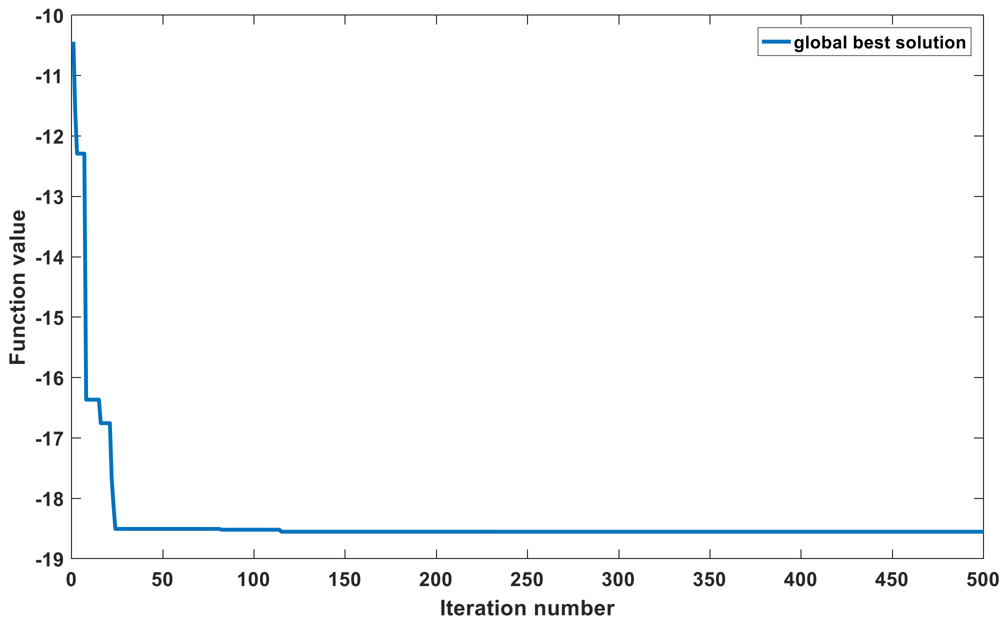


Figure 5-5. When solving Equation (5.6), the algorithm converged after 230 iterations

The HCA reached the global solution of Equation (5.6) after 230 iterations.

5.3.5 Local Mutation Improvement

The algorithm can be augmented with a mutation operation that changes the solutions of the water drops during collisions in the condensation stage. This operation enhances the performance of the algorithm and minimises the number of iterations required to reach the best-known solution. The mutation is applied randomly on selected values of the variables from the solutions of selected water drops. The mutation operation on real numbers is implemented as

$$V_{new} = 1 - (\beta * V_{old}), \tag{5.8}$$

where β is a uniform random number in the range [0, 1], and V_{old} and V_{new} are the original and new (mutated) values of the variable, respectively. This mutation can convert a large to a small value, or vice versa.

5.3.6 Parameters and Assumptions of HCA

For solving COPs, the parameters are set to constant as specified in Section 3.3.8 of Chapter 3, with the exception of the number of water drops which was set to ten, and the maximum number of iterations which was set to 500 (or 1000 for functions with more than two variables). The distance between nodes in different layers is set to equal one unit. Therefore, the depth is adjusted to equal the inverse of the number of times a node is selected. Under this setting, a path between (x, y) will deepen with increasing number of selections of node y after node x. This adjustment is required because the inter-nodal distance is constant as stipulated in the problem representation. Hence, considering the depth to equal the distance over the soil will not affect the outcome as supposed because the depth values will be same for all the edges. The depth is useful in cases where the distance of the edges are varied.

5.4 Experimental Results and Analysis

The HCA was evaluated on a number of benchmark functions selected from the literature (Surjanovic & Bingham, 2013), in order to demonstrate a variety of properties. This research is limited to unconstrained minimisation functions. The characteristics of these functions (function name, lower and upper bounds, number of variables *D*, optimum solutions, and geometric properties) are summarised in Table 5-3. The mathematical expression of each function is provided in Appendix A.

Table 5-3. Properties and characteristics of benchmark functions in the literature

No	Function Name	Lower Bound	Upper bound	*D*	*f*(*x**)	Type
1	Ackley	-32.768	32.768	2	0	Many Local Minima
2	Cross-In-Tray	-10	10	2	-2.06261	Many Local Minima
3	Drop-Wave	-5.12	5.12	2	-1	Many Local Minima
4	Gramacy & Lee (2012)	0.5	2.5	1	-0.86901	Many Local Minima
5	Griewank	-600	600	2	0	Many Local Minima
6	Holder Table	-10	10	2	-19.2085	Many Local Minima
7	Levy	-10	10	2	0	Many Local Minima
8	Levy N. 13	-10	10	2	0	Many Local Minima
9	Rastrigin	-5.12	5.12	2	0	Many Local Minima
10	Schaffer N.2	-100	100	2	0	Many Local Minima
11	Schaffer N.4	-100	100	2	0.292579	Many Local Minima
12	Schwefel	-500	500	2	0	Many Local Minima
13	Shubert	-5.12	5.12	2	-186.7309	Many Local Minima
14	Bohachevsky	-100	100	2	0	Bowl-Shaped
15	Rotated Hyper-Ellipsoid	-65.536	65.536	2	0	Bowl-Shaped
16	Sphere (Hyper)	-5.12	5.12	2	0	Bowl-Shaped
17	Sum Squares	-10	10	2	0	Bowl-Shaped
18	Booth	-10	10	2	0	Plate-Shaped
19	Matyas	-10	10	2	0	Plate-Shaped
20	McCormick	[-1.5, 4]	[-3, 4]	2	-1.9133	Plate-Shaped
21	Zakharov	-5	10	2	0	Plate-Shaped
22	Three-Hump Camel	-5	5	2	0	Valley-Shaped
23	Six-Hump Camel	[-3,3]	[-2,2]	2	-1.0316	Valley-Shaped
24	Dixon-Price	-10	10	2	0	Valley-Shaped
25	Rosenbrock	-5	10	2	0	Valley-Shaped
26	Shekel’s Foxholes (De Jong N.5)	-65.536	65.536	2	0.99800	Steep Ridges/Drops
27	Easom	-100	100	2	-1	Steep Ridges/Drops
28	Michalewicz	0	3.14	2	-1.8013	Steep Ridges/Drops
29	Beale	-4.5	4.5	2	0	Other
30	Branin	[-5, 10]	[0, 15]	2	0.39788	Other
31	Goldstein-Price I	-2	2	2	3	Other
32	Goldstein-Price II	-5	-5	2	1	Other
33	Styblinski-Tang	-5	-5	2	-78.33198	Other
34	Gramacy & Lee (2008)	-2	6	2	-0.4288819	Other
35	Martin	0	10	2	0	Other
36	Easton and Fenton	0	10	2	1.744152	Other
37	Rosenbrock D1.2	-1.2	1.2	2	0	Other
38	Sphere	-5.12	5.12	3	0	Other
39	Hartmann 3-D	0	1	3	-3.86278	Other
40	Rosenbrock	-1.2	1.2	4	0	Other
41	Powell	-4	5	4	0	Other
42	Wood	-5	5	4	0	Other
43	Sphere	-5.12	5.12	6	0	Other
44	DeJong	-2.048	2.048	2	3905.93	Maximise function

The precision of all variables in all functions was assumed to be six significant figures. For all functions, the HCA was executed twenty times with and without the mutation operation (HCA and MHCA, respectively). The maximum number of iterations was 500 for functions with one or two variables, and 1000 for functions with three or more variables.

The results are provided in Table 5-4. The algorithm numbers in Table 5-4 (headed “No”) match those of Table 5-3. For each function, Table 5-4 lists the worst, average, and best values, and the average number of iterations (“#”) needed to reach the global solution. The HCA algorithm yielded satisfactory results for all functions tested, and sometimes found the global minimum after only a few iterations. To evaluate the algorithm performance, we calculated the success rate of each function by Eq. (5.9):

$$Success\ rate\ \% = \left(\frac{Number\ of\ successful\ runs}{Total\ number\ of\ runs}\right) \times 100 \qquad (5.9)$$

The solution was accepted if the obtained solution differed from the optimum within 0.001. The algorithm was 100% successful on most of the functions. The exceptions were Powell-4v [HCA: (60%), MHCA: (90%)], Sphere -6v [HCA: (60%), MHCA: (40%)], Rosenbrock-4v [HCA: (70%), MHCA: (100%)], and Wood-4v [HCA: (50%), MHCA: (80%)]. The boldfaced values in the 'best' column represent the best solution achieved by HCA or MHCA; in all other cases, HCA and MHCA achieved the same "best" performance.

Table 5-4. Results of HCA and MHCA on various benchmark functions

No	HCA					MHCA				
	Worst	**Average**	**Best**	**SD**	**#**	**Worst**	**Average**	**Best**	**SD**	**#**
1	8.88E-16	8.88E-16	8.88E-16	0.00E+00	220.2	8.88E-16	8.88E-16	8.88E-16	0.00E+00	279.35
2	-2.06E+00	-2.06E+00	-2.06E+00	8.15E-06	362	-2.06E+00	-2.06E+00	-2.06E+00	1.14E-05	196.1
3	-1.00E+00	-1.00E+00	-1.00E+00	0.00E+00	330.8	-1.00E+00	-1.00E+00	-1.00E+00	0.00E+00	220.4
4	-8.69E-01	-8.69E-01	-8.69E-01	2.28E-16	297.65	-8.69E-01	-8.69E-01	-8.69E-01	3.55E-10	345.95
5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	212.25	0.00E+00	0.00E+00	0.00E+00	0.00E+00	284.8
6	-1.92E+01	-1.92E+01	-1.92E+01	1.51E-04	321.7	-1.92E+01	-1.92E+01	-1.92E+01	4.98E-04	302.75
7	4.23E-07	1.31E-07	1.50E-32	1.43E-07	399.5	3.60E-07	8.65E-08	1.50E-32	1.09E-07	394.8
8	1.63E-05	4.70E-06	**1.35E-31**	5.37E-06	399.45	9.97E-06	3.06E-06	1.60E-09	2.83E-06	366.3
9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	289.4	0.00E+00	0.00E+00	0.00E+00	0.00E+00	352.9
10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	284.1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	333.85
11	2.93E-01	2.93E-01	2.93E-01	5.52E-06	358.6	2.93E-01	2.93E-01	2.93E-01	9.55E-06	336.25
12	6.82E-04	4.19E-04	2.08E-04	1.41E-04	337.6	1.28E-03	6.42E-04	**2.02E-04**	2.74E-04	323.75
13	-1.87E+02	-1.87E+02	-1.87E+02	2.33E-03	368	-1.87E+02	-1.87E+02	-1.87E+02	3.30E-03	391.45
14	0.00E+00	0.00E+00	0.00E+00	0.00E+00	264.9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	288.25
15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	309.9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	291
16	0.00E+00	0.00E+00	0.00E+00	0.00E+00	283.15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	293.6
17	0.00E+00	0.00E+00	0.00E+00	0.00E+00	305.95	0.00E+00	0.00E+00	0.00E+00	0.00E+00	271.6
18	2.03E-05	6.33E-06	**0.00E+00**	6.48E-06	433.5	1.97E-05	5.78E-06	2.96E-08	6.96E-06	363.5
19	0.00E+00	0.00E+00	0.00E+00	0.00E+00	258.35	0.00E+00	0.00E+00	0.00E+00	0.00E+00	287.55
20	-1.91E+00	-1.91E+00	-1.91E+00	2.64E-04	282.65	-1.91E+00	-1.91E+00	-1.91E+00	9.23E-04	225.8
21	1.12E-04	5.07E-05	4.73E-06	3.33E-05	328.15	3.94E-04	1.30E-04	**4.28E-07**	1.43E-04	242.05
22	0.00E+00	0.00E+00	0.00E+00	0.00E+00	238.8	0.00E+00	0.00E+00	0.00E+00	0.00E+00	288.9
23	-1.03E+00	-1.03E+00	-1.03E+00	2.05E-06	348.15	-1.03E+00	-1.03E+00	-1.03E+00	1.25E-06	332.4
24	3.08E-04	9.69E-05	3.89E-06	8.58E-05	394.25	5.46E-04	2.67E-04	**4.10E-08**	1.79E-04	380.55
25	2.03E-09	2.14E-10	0.00E+00	4.53E-10	350.55	2.25E-10	3.21E-11	0.00E+00	6.93E-11	282.55
26	9.98E-01	9.98E-01	9.98E-01	1.63E-05	354.6	9.98E-01	9.98E-01	9.98E-01	5.10E-06	370.35
27	-9.97E-01	-9.98E-01	-1.00E+00	9.88E-04	354.15	-9.97E-01	-9.99E-01	-1.00E+00	9.16E-04	344.6
28	-1.80E+00	-1.80E+00	-1.80E+00	4.46E-07	428	-1.80E+00	-1.80E+00	-1.80E+00	7.78E-07	398.1
29	9.25E-05	5.25E-05	**3.41E-06**	2.53E-05	379.9	1.33E-04	7.37E-05	2.56E-05	4.05E-05	393.75
30	3.98E-01	3.98E-01	3.98E-01	1.56E-05	345.8	3.98E-01	3.98E-01	3.98E-01	2.54E-05	409.9
31	3.00E+00	3.00E+00	3.00E+00	7.33E-08	255.5	3.00E+00	3.00E+00	3.00E+00	1.53E-07	310.8
32	1.00E+00	1.00E+00	1.00E+00	0.00E+00	410.7	1.00E+00	1.00E+00	1.00E+00	0.00E+00	379.95
33	-7.83E+01	-7.83E+01	-7.83E+01	1.39E-05	351	-7.83E+01	-7.83E+01	-7.83E+01	1.58E-05	341
34	-4.29E-01	-4.29E-01	-4.29E-01	1.83E-07	262.05	-4.29E-01	-4.29E-01	-4.29E-01	1.49E-07	286.65
35	0.00E+00	0.00E+00	0.00E+00	0.00E+00	370.9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	347.1
36	1.74E+00	1.74E+00	1.74E+00	1.15E-06	385.75	1.74E+00	1.74E+00	1.74E+00	1.69E-06	408.8
37	9.22E-06	3.67E-06	**6.63E-08**	2.67E-06	382.2	4.66E-06	2.60E-06	2.79E-07	1.39E-06	351.6
38	4.43E-07	8.27E-08	0.00E+00	1.01E-07	371.3	2.13E-08	4.50E-09	0.00E+00	6.12E-09	330.45
39	-3.86E+00	-3.86E+00	-3.86E+00	3.68E-05	392.05	-3.86E+00	-3.86E+00	-3.86E+00	3.81E-05	327.5
40	1.94E-01	7.02E-02	**1.64E-03**	6.48E-02	816.5	8.75E-02	3.04E-02	2.79E-03	2.63E-02	806
41	2.42E-01	9.13E-02	3.91E-03	8.38E-02	863.95	1.12E-01	4.26E-02	**1.96E-03**	3.78E-02	840.85
42	1.12E+00	2.91E-01	7.62E-08	3.95E-01	753.95	2.67E-01	6.10E-02	**1.00E-08**	6.66E-02	694.4
43	1.05E+00	3.88E-01	**1.47E-05**	4.71E-01	879	8.96E-01	3.10E-01	6.51E-03	2.97E-01	505.2
44	3.91E+03	3.91E+03	3.91E+03	6.00E-05	**314.8**	3.91E+03	3.91E+03	3.91E+03	7.94E-05	373.85
		Mean			378.45					361.3

The results of HCA and MHCA were compared using the Wilcoxon Signed-Rank test. The *P*-values obtained in the test were 0.2460, 0.1389, 0.8572, and 0.2543 for the *worst*, *average*, *best*, and *average number of iterations* respectively. According to the *P*-values, there is no significant difference between the results. This indicates that the HCA algorithm is able to obtain best-known results without the mutation operation. However, the mutation operation reduced the average number of iterations to converge to the solution by 61%. The small standard deviation in each case shows that HCA always generated very acceptable solutions. Through this experiment, it was concluded that the inclusion of non-natural components in natural-inspired algorithms may not enhance the quality of the solution. Thus, the natural components by itself is able to generate good solutions if it is designed carefully.

The success rate was lower for functions with more than four variables because of the problem representation, where the number of layers in the representation increases with the number of variables. Consequently, the HCA performance was limited to functions with few variables. The experimental results revealed a notable advantage of the proposed problem representation, namely, the small effect of the function domain on the solution quality and algorithm performance. In contrast, the performances of some algorithms are highly affected by the function domain, because their mechanism depend on the distribution of the entities in the search space. If an entity wanders outside the function boundaries, its position must be reset by the algorithm.

The effectiveness of the algorithm is validated by analysing the calculated average values for each function (i.e., the average value of each function is close to the optimal value). This demonstrates the stability of the algorithm since it manages to find the global-optimal solution in each execution. Since the maximum number of iterations is fixed to a specific value, the average execution time was recorded for Hyper-Sphere function with different number of variables. Consequently, the execution time is approximately the same for the other functions with the same number of variables. There is a slight increase in execution time with increasing number of variables. The results are reported in Table 5-5.

Table 5-5. Average execution time per number of variables

Hyper-Sphere function	**2-variables (500 iterations)**	**3-variables (500 iterations)**	**4-variables (1000 iterations)**	**6-variables (1000 iterations)**
Average execution time (second)	2.8186	4.0832	10.716	15.962

Next, the performance of HCA was compared with that of the water cycle algorithm (WCA) on specific functions. The WCA results were taken from Sadollah et al. (2015b). The results of both algorithms are displayed in Table 5-6.

Table 5-6. Comparison between WCA and HCA

Function name	*D*	Bond	Best result	WCA				HCA			
				Worst	Avg.	Best	#	Worst	Avg.	Best	#
De Jong	2	±2.048	3905.93	3906.121	3905.940	3905.93	684	3905.93	3905.93	3905.93	**314.8**
Goldstein & Price I	2	±2	3	3.000968	3.000561	3.00002	980	3.00E+00	3.00E+00	**3.00E+00**	**345.8**
Branin	2	±2	0.3977	0.398717	0.398272	0.397731	**377**	3.98E-01	3.98E-01	3.98E-01	379.9
Martin & Gaddy	2	[0,10]	0	9.29E−04	4.16E−04	5.01E−06	**57**	0.00E+00	0.00E+00	**0.00E+00**	262.05
Rosenbrock	2	±1.2	0	1.46E−02	1.34E−03	2.81E−05	**174**	9.22E-06	3.67E-06	**6.63E-08**	370.9
Rosenbrock	2	±10	0	9.86E−04	4.32E−04	1.12E−06	623	2.03E-09	2.14E-10	**0.00E+00**	**350.55**
Rosenbrock	4	±1.2	0	7.98E−04	2.12E−04	**3.23E−07**	**266**	1.94E-01	7.02E-02	1.64E-03	816.5
Hyper sphere	6	±5.12	0	9.22E−03	6.01E−03	**1.34E−07**	**101**	1.05E+00	3.88E-01	1.47E-05	879
Shaffer	2	±100	0	9.71E−03	1.16E−03	2.61E−05	8942	0.00E+00	0.00E+00	**0.00E+00**	**284.1**
Goldstein & Price I	2	±5	3	3	3.00003	3	2400	3.00E+00	3.00E+00	3.00E+00	**345.8**
Goldstein & Price II	2	±5	1	1.1291	1.0118	1	47,500	1.00E+00	1.00E+00	1.00E+00	**255.5**
Six-Hump Came back	2	±10	−1.0316	−1.0316	−1.0316	−1.0316	3105	-1.03E+00	-1.03E+00	-1.03E+00	**348.15**
Easton & Fenton	2	[0,10]	1.74	1.7441	1.7441	1.7441	650	1.74E+00	1.74E+00	1.74E+00	**385.75**
Wood	4	±5	0	3.81E−05	1.58E−06	**1.30E−10**	15,650	1.12E+00	2.91E-01	7.62E-08	**753.95**
Powell quartic	4	±5	0	2.87E−09	6.09E−10	**1.12E−11**	23,500	2.42E-01	9.13E-02	3.91E-03	**863.95**
Mean							7000.6				**463.8**

The results presented in Table 5-6 show that HCA and WCA produced best-known results with differing accuracies. The HCA outperformed the WCA in some cases, and reached the global optimum after fewer iterations than WCA in 10 of 15 cases. However, the WCA outperformed the HCA on functions with more than two variables. Significance values of 0.75 (*worst*), 0.97 (*average*) and 0.86 (*best*) were found using Wilcoxon test, indicating no significant difference in performance between WCA and HCA.

In addition, the *average number of iterations* are also compared, and the *P*-value (0.03078) indicates a significant difference, which confirms that the HCA was able to reach the global-optimal solution with fewer iterations than WCA. However, the solutions' accuracy of the HCA over functions with more than two variables was lower than WCA.

The average number of iterations was compared between HCA and other optimisation algorithms. The algorithms compared were continuous ACO based on discrete encoding CACO–DE (H. Huang & Hao, 2006), an ant colony system (ANTS), a genetic algorithm (GA), a bee algorithm (BA), and grenade explosion method (GEM). The results of ANTS, GA, BE and GEM were taken from (Ahrari & Atai, 2010). The HCA results were also compared with the WCA and evaporation rate based WCA (ER–WCA) results reported by (Sadollah et al., 2015b). All algorithms were 100% successful on most of the evaluated functions. HCA was less than 100% successful only on Sphere-6v (60%) and Rosenbrock-4v (70%). This variation in the success rate is attributed to the surface structure of the functions domain and number of variables. The comparison results are listed in Table 5-7 .

Table 5-7. Comparison between HCA and other algorithms

No	Function name	D	B	CACO-DE	ANTS	GA	BA	GEM	WCA	ER-WCA	HCA
1	De Jong	2	±2.048	1872	6000	10,160	868	746	684	1220	**314.8**
2	Goldstein & Price I	2	±2	666	5330	5662	999	701	980	480	**345.8**
3	Branin	2	±2	-	1936	7325	1657	689	377	**160**	379.9
4	Martin & Gaddy	2	[0,10]	340	1688	2488	526	258	**57**	100	262.05
5a	Rosenbrock	2	±1.2	-	6842	10,212	631	572	174	**73**	370.9
5b	Rosenbrock	2	±10	1313	7505	-	2306	2289	623	**94**	350.55
6	Rosenbrock	4	±1.2	624	8471	-	28,529	82,188	**266**	300	816.5
7	Hyper sphere	6	±5.12	270	22,050	15,468	7113	423	101	**91**	879
8	Shaffer	2			-	-	-	-	8942	1110	**284.1**
	Mean			847.5	7477.8	8552.5	5328.6	10983.3	1356.0	403.1	444.8

As seen in Table 5-7, the HCA results are very competitive and optimised the solutions after fewer iterations. The Wilcoxon test was applied to identify the significance of differences between the results. The T-test was also applied to obtain *P*-values along with the *W*-values. The results of the *P/W*-values are reported in Table 5-8.

Table 5-8. Comparison between P/W-values for HCA and other algorithms (based on Table 5-7)

Algorithm versus HCA	Using T-test	Using Wilcoxon test		Is the result significant at $P \leq 0.05$
	***P*-value**	***Z*-value**	***W*-value (at critical value of w)**	
CACO-DE vs HCA	0.324033	-0.9435	6 (at 0)	no
ANTS vs HCA	0.014953	-2.5205	0 (at 3)	yes
GA vs HCA	0.005598	-2.2014	0 (at 0)	yes
BA vs HCA	0.188434	-2.5205	0 (at 3)	yes
GEM vs HCA	0.333414	-1.5403	7 (at 3)	no
WCA vs HCA	0.379505	-0.2962	20 (at 5)	no
ER-WCA vs HCA	0.832326	-0.5331	18 (at 5)	no

Based on Table 5-8, there are significant differences between the results of ANTS, GA, BA and HCA, where HCA reached the best-known solutions in fewer iterations than the ANTS, GA, and BA. Although the *P*-value for "BA vs HCA" indicates that there is no statistically significant difference, the *W*-value shows there is a significant difference between the two algorithms. Further, the performance of HCA CACO-DE, GEM, WCA and ER-WCA are very competitive. Overall, the results also indicate that HCA is highly efficient for functions optimisation.

The number of iterations for best-known solution searching in HCA, M-HCA, continuous GA (CGA), and enhanced continuous tabu search (ECTS) were compared on various other functions. The CGA and ECTS results were taken from (Chelouah & Siarry, 2000). The success rate of all algorithms was 100%. The results are listed in Table 5-9, where numbers in boldface indicate the lowest value.

Table 5-9. Comparison between CGA, ECTS, HCA, and MHCA

No	Function Name	D	CGA	ECTS	HCA	MHCA
1	Shubert	2	575	-	**368**	391.45
2	Bohachevsky	2	370	-	**264.9**	288.25
3	Zakharov	2	620	**195**	328.15	242.05
4	Rosenbrock	2	960	480	350.55	**282.55**
5	Easom	2	1504	1284	**354.6**	370.35
6	Branin	2	620	**245**	379.9	393.75
7	Goldstein-Price 1	2	410	**231**	345.8	409.9
8	Hartmann	3	582	548	392.05	**327.5**
9	Sphere	3	750	338	371.3	**330.45**
	Mean		710.1	474.4	350.6	337.4

Note that both HCA and MHCA optimised the solutions in fewer iterations than CGA and ECTS. This is confirmed using the Wilcoxon test and T-test on the obtained results, see Table 5-10.

Table 5-10. Comparison between P/W-values for CGA, ECTS, HCA, and MHCA

Algorithm versus HCA	Using T-test	Using Wilcoxon test		Is the result significant at $P \leq 0.05$
	***P*-value**	***Z*-value**	***W*-value (at critical value of *w*)**	
CGA vs HCA	0.012635	-2.6656	0 (at 5)	yes
CGA vs MHCA	0.012361	-2.6656	0 (at 5)	yes
ECTS vs HCA	0.456634	-0.3381	12 (at 2)	no
ECTS vs MHCA	0.369119	-0.8452	9 (at 2)	no
HCA vs MHCA	0.470531	-0.7701	16 (at 5)	no

Despite there being no significant difference between the results of HCA, MHCA and ECTS, the means of HCA and MHCA results are less than ECTS, indicating competitive performance.

Figure 5-6 illustrates the convergence of the global and local solutions of the Ackley function found by HCA as the iterations proceed. The local results (red lines) represent the quality of the solution obtained after each iteration. The algorithm continued generating different solutions in each iteration, reflecting the proper design of the flow stage. We postulate that such solution diversity was maintained by the depth factor and fluctuations of the water-drop velocities. The HCA minimised the Ackley function after 327 iterations.

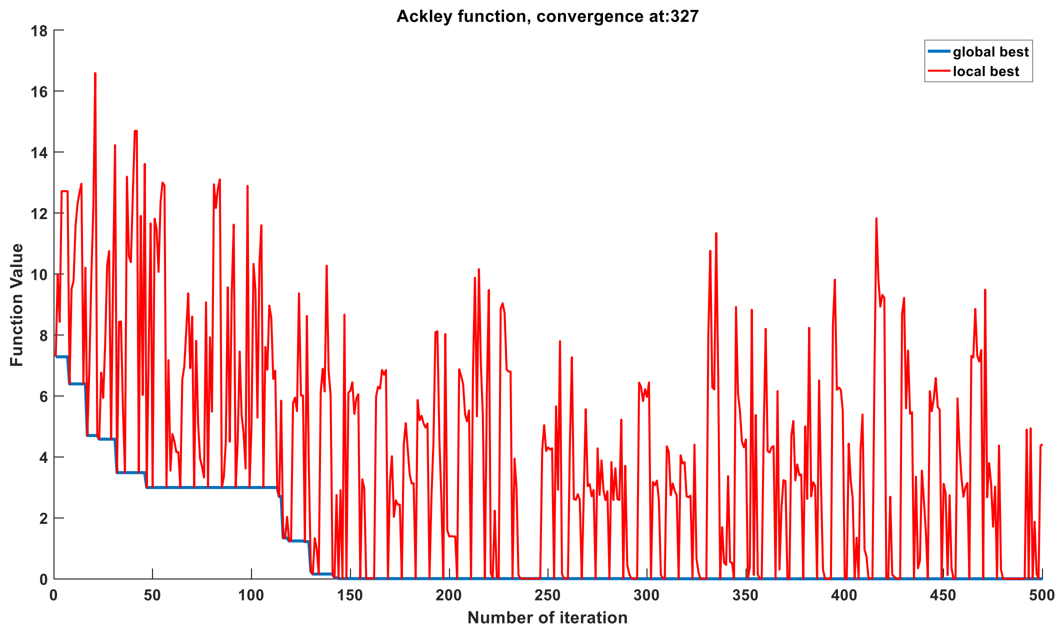


Figure 5-6. Global and local convergence of HCA throughout the iterations

Figure 5-7 illustrates the convergence of global solutions of the McCormick function found by HCA.

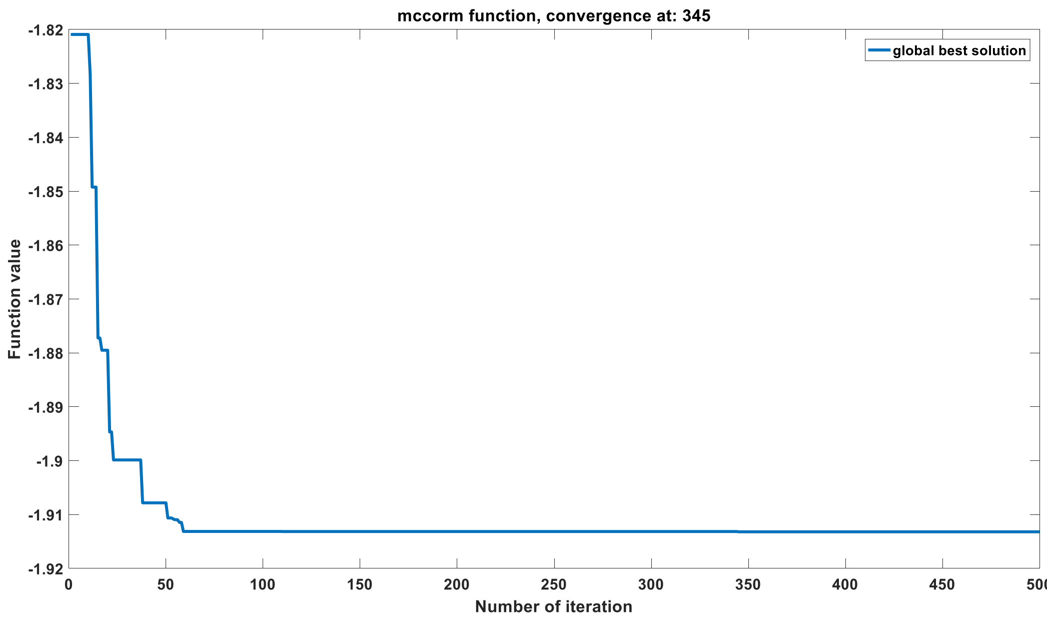


Figure 5-7. Global HCA convergence on the McCormick function

Figure 5-8, 5-9, 5-10, 5-11 and 5-12 show the convergences of the HCA on the Easom function, the Goldstein–Price function, the 4-variable Rosenbrock function, the 4-variable Hyper-Sphere function and the 6-variable Hyper-Sphere function, respectively. The functions were minimised after 123, 404, 421, 634 and 994 iterations, respectively.

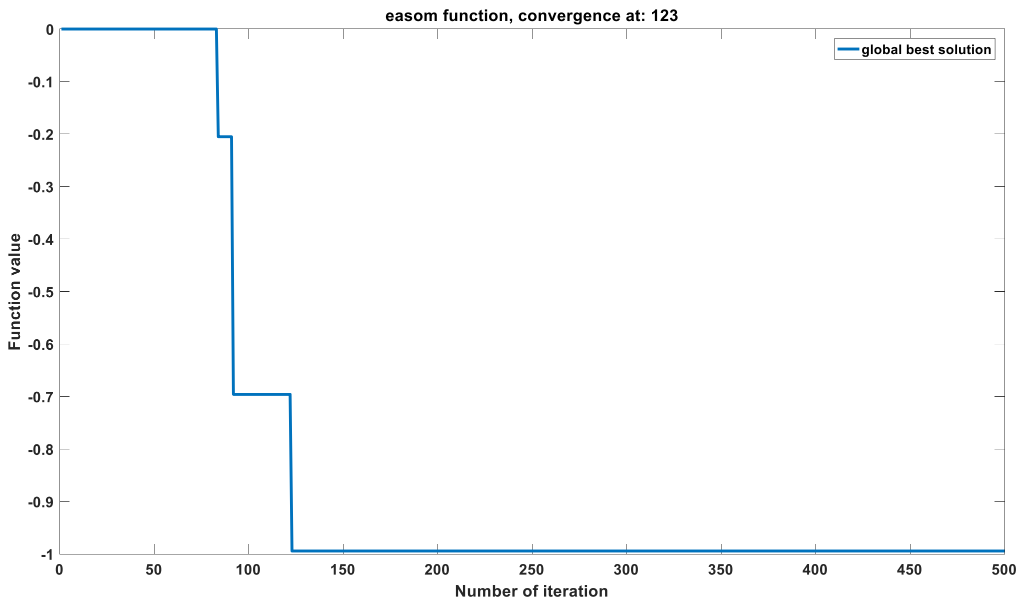


Figure 5-8. Global HCA convergence on the Easom function

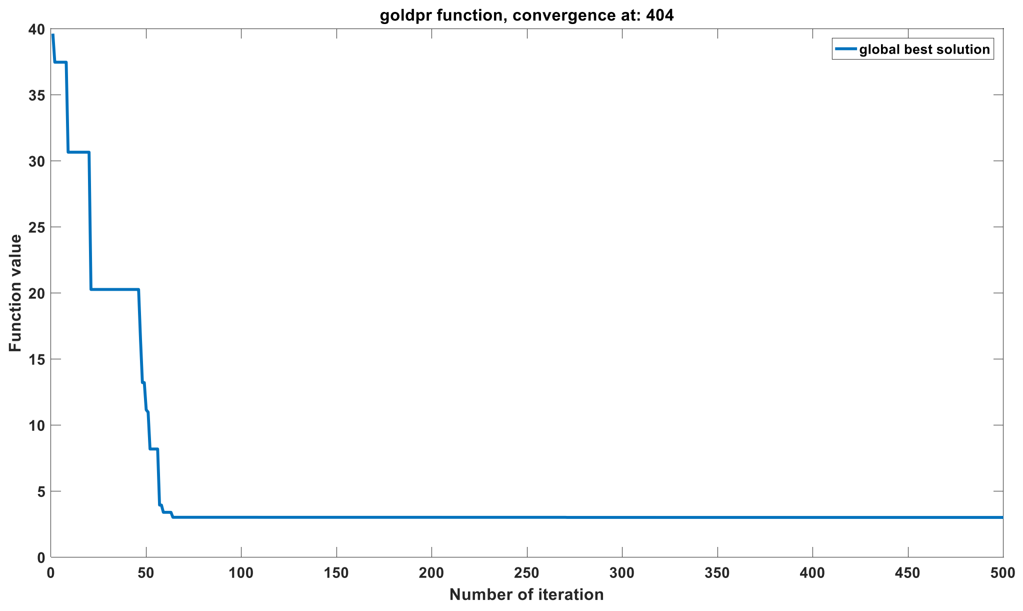


Figure 5-9. Global HCA convergence on the Goldstein–Price function

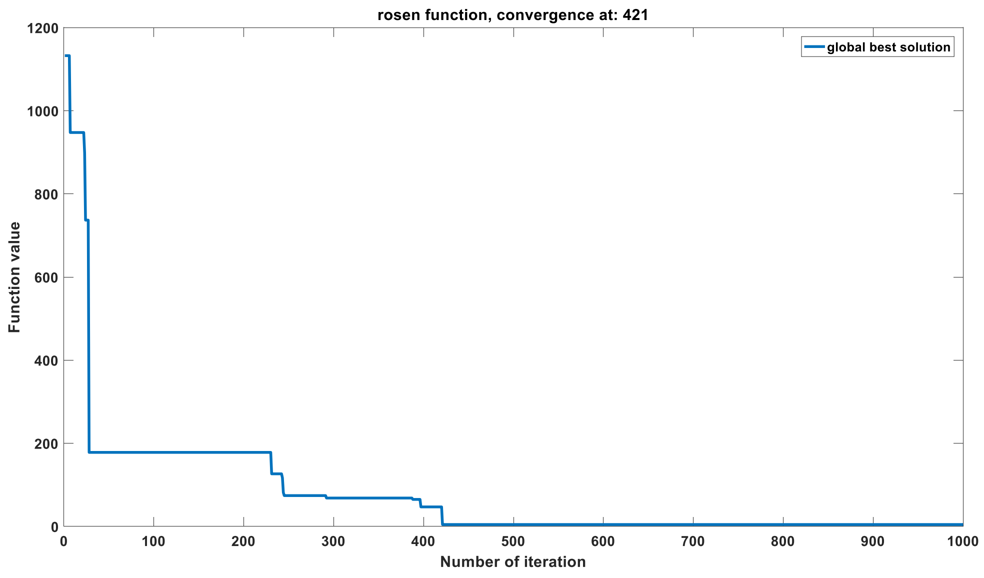


Figure 5-10. Global HCA convergence on the Rosenbrock function with four variables

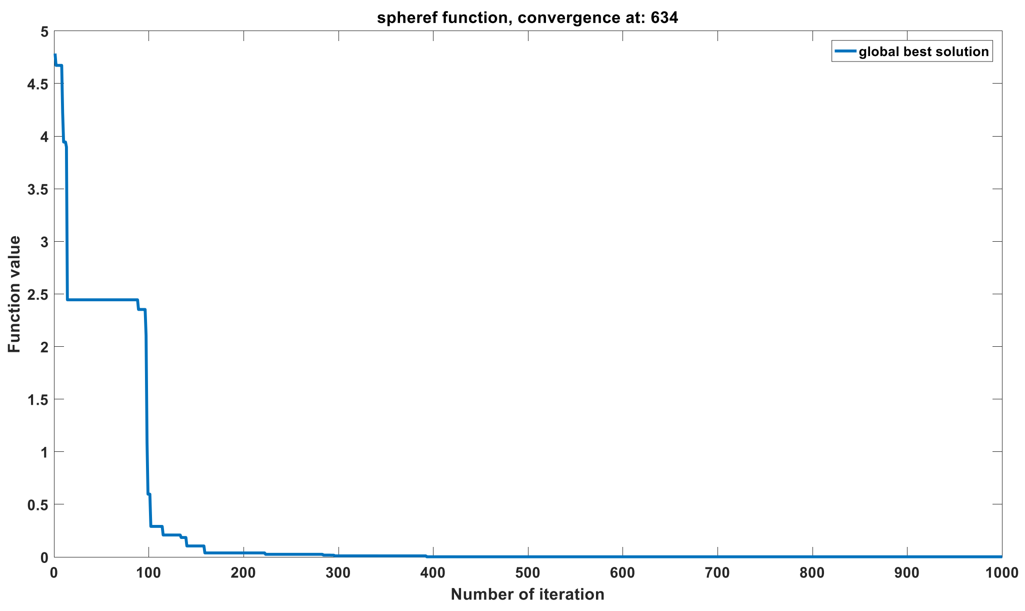


Figure 5-11. Global convergence of HCA on the Hyper-sphere function with four variables

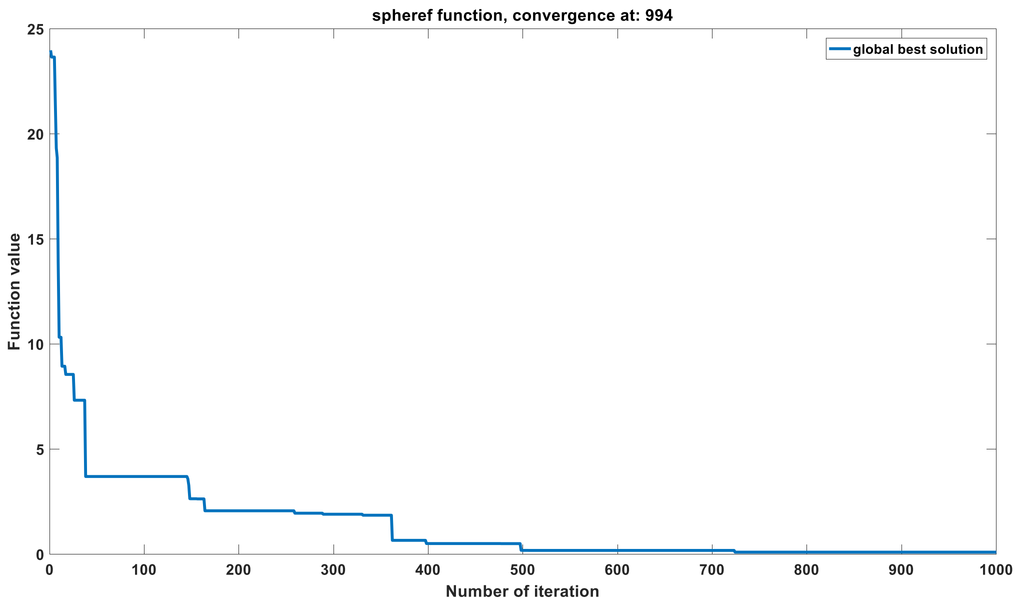


Figure 5-12. Global convergence of HCA on the Hyper-sphere function with six variables

As shown in the above figures, the HCA required more iterations to minimise functions with more than two variables.

5.5 Conclusion

This chapter demonstrated that HCA can solve COPs without any major changes to its structure. The HCA was shown to provide high-quality solutions for a variety of unconstrained benchmarked test functions. In addition, the HCA was confirmed to escape local optima and find the global optimum, this validates the convergence capability and effectiveness of the exploration and exploitation processes in this algorithm. Once again the HCA's high performance is considered to be partly attributable to the direct and indirect communication among the water drops which enables information sharing. Consequently, the search space is diversified and the overall performance is improved. This research confirmed that the depth value can be customised to fit the problem settings. Furthermore, it was found that the mutation operation further improved the performance of HCA in term of reducing the number of iterations required to optimise the solution for some functions.

The proposed representation of COPs is easily adaptable to other problems. For instance, the representation can be regarded as a topology in gravity-based approaches, with the swarm particles starting at the highest point. If the graph vertices are weighted, the representation becomes a form of optimised route finding. Overall, the problem representation is efficient and can be easily adapted to classification problems (see Appendix B).

In the benchmarking experiments, the HCA performed at least as well as other optimisation algorithms. However, there is room for further improvement. In particular, the algorithm's performance need to be improved on problems involving two or more variables, functions with constraints, and multiple-objective functions. Additionally, the HCA might be improved by including other techniques to dynamically control the evaporation and condensation occurrences. The impact of number of water drops on the solution quality is also worth investigating.

Chapter 6 Solving the Capacitated Vehicle Routing Problem using HCA

"Every minute you spend in planning saves 10 minutes in execution; this gives you a 1,000 percent Return on Energy!"

[Brian Tracy]

In this chapter, the HCA is applied to a well-known routing problem called the capacitated vehicle routing problem (CVRP). Solutions to the CVRP attempt to find the optimal routes by which a fleet of vehicles can serve a number of customers. A route constitutes the best sequence of customer service by one vehicle without violating the capacity. This chapter first investigates whether the HCA algorithm can solve the CVRP, a challenging optimisation problem with highly practical and relevant applications in real-life. The objectives of the CVRP are minimising the cost and the number of vehicles. Next, the important factors related to the superior performance of the HCA over combinatorial multi-objective NP-hard problems, and the flexibility of the HCA algorithm, are assessed in empirical results. The HCA implementation used is based on the general framework described in Chapter 3. The algorithm is tested on a number of structural and benchmark CVRP problems. In the computational experiments, the HCA produced high-quality solutions that were competitive with the solutions of other approaches.

The remainder of this chapter is structured as follows. Section 6.1 briefly introduces and formulates the CVRP. Section 6.2 configures the HCA and explains its application to the CVRP. The computational results of the HCA and other algorithms are compared in Section 6.3. Section 6.4 discusses the content and concludes the chapter.

6.1 The Capacitated Vehicle Routing Problem

The operations of supply chains and logistics have in recent years expanded and evolved and have become increasingly more complex. To optimise their processes under these challenges, many companies rely on technology-based strategies such as distribution management systems (DMSs). The DMS is designed to monitor, facilitate, and control the distribution process with efficiency and reliability. In general, efficient computerised programs assist the management and control of the vehicles' workflow, ensuring a smooth flow of goods from supplier to customers. This management reduces the transportation cost of companies, in turn reducing the price burden on consumers. Reducing the number of vehicles involved in transportation can also reduce the costs. Thus, optimising transportation provides a reliable and flexible distribution of goods to customers at the lowest cost and highest quality of service, thereby improving the competitiveness of the business.

As the travelled distance increases, fuel consumption and consequently the costs of freight transportation generally increase. Obviously, the total travelling time is proportional to the length of the paths, and increasing the travelling time in turn increases labour costs. The transportation costs are also affected by other factors, such as the size and utilisation of the truck capacity, road and traffic conditions, and fuel prices (Kulović, 2004). According to Rodrigue et al. (2006), transportation accounts for approximately 10% of a product’s total cost. In New Zealand, transport contributes approximately 60% percent of the total logistics costs. In particular, the cost of road freight transport has inflated by approximately 10% per annum over the last ten years because of the increased oil prices (Financial and Economic Analysis Team, 2010).

An important classical problem related to transportation, distribution management and logistics is the VRP. The VRP was first formulated by Dantzig and Ramser in (1959) under the title “The Truck Dispatching Problem”. In their version, a fleet of gasoline delivery trucks was routed between a central terminal and a number of service stations supplied by that terminal. In the VRP, a fleet of vehicles must deliver or collect items from a depot to a set of geographically distributed sites at the lowest possible cost. The VRP is classified as a combinatorial nondeterministic polynomial-time hard (NP-hard) optimisation problem; that is, its execution time increases exponentially with problem size (number of nodes).

Over time, researchers have proposed various complex variants of the conventional VRP, each with unique additional constraints. These variants include capacitated VRP (CVRP), VRP with time window (VRPTW), multiple depot VRP (MDVRP), VRP with pick-up and delivery (VRPPD), periodic VRP (PVRP), VRP with backhauls (VRPB), and dynamic VRP (DVRP), as shown in Figure 6-1. A problem may involve a combination of these variants. The specifications and details of these variants are described elsewhere (see Han, 2010; S. N. Kumar & Panneerselvam, 2012; Toth & Daniele, 2002).

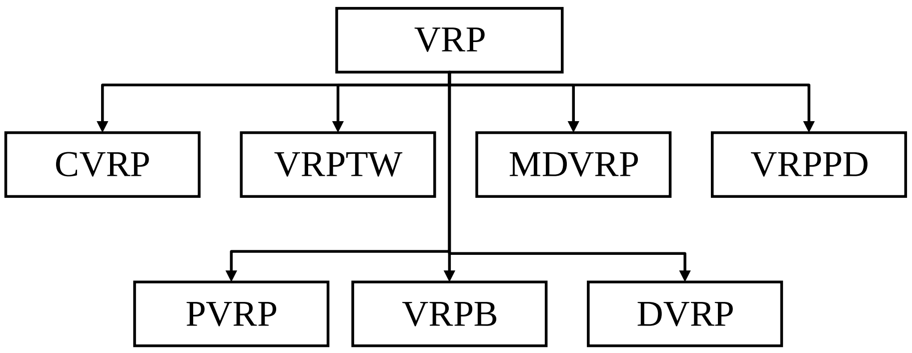


Figure 6-1. Variants of the vehicle routing problem

Among the most studied variants is the CVRP (Toth & Vigo, 2002). As its name implies, vehicles in the CVRP are additionally constrained by their carrying capacity. Each vehicle can load goods (customer's demands) until it reaches its maximum capacity. The vehicle fleet in a CVRP can be homogeneous or heterogeneous. In a homogeneous fleet, all vehicles are of similar type and have the same capacity (uniform fleet). In contrast, vehicles in a heterogeneous fleet are of different types with possibly different capacities (mixed fleet). The CVRP describes many real-life problems such as newspaper distribution, food and drinks distribution to grocery stores, milk collection from farmers, garbage collection, mail distribution, and school bus routing.

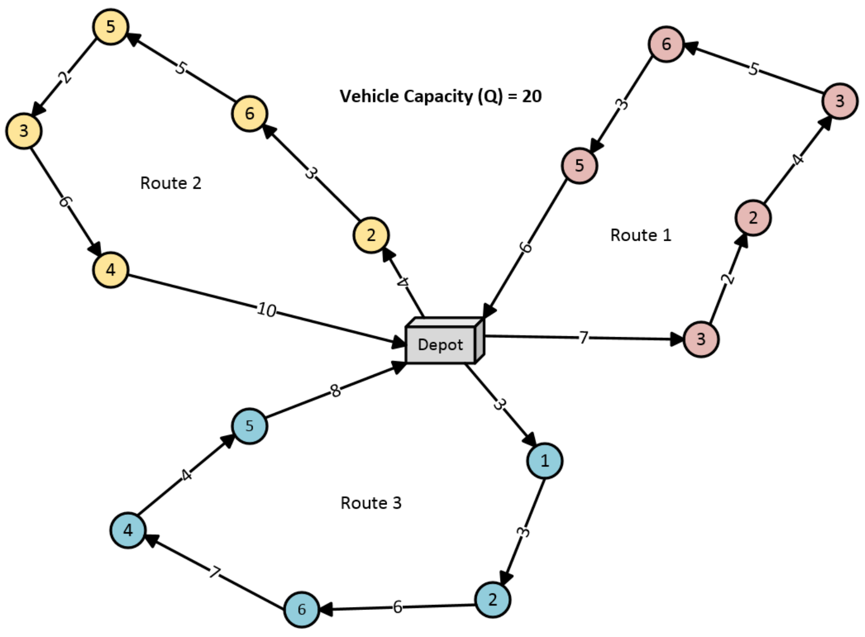


Figure 6-2. Example of the capacitated vehicle routing problem

Figure 6-2 shows a simple example of a solved CVRP with one depot and 15 nodes (customers). The numbers in the circles and on the arrows indicate the customers' demands and the travelling cost, respectively. The assumed vehicle capacity is 20 units. Solving this problem requires three vehicles, one for each route. The first, second and third vehicles carried 19 units, 20 units and 18 units of their total capacity, respectively, along their assigned routes.

The CVRP is a multi-objective problem that simultaneously minimises the total travelling cost and the number of required vehicles. Reducing the number of vehicles is achieved by maximising the vehicle load. The CVRP is also considered as a complex optimisation problem as it combines two well-known problems, the bin packing problem (BPP) and TSP (Ralphs, Kopman, Pulleyblank, & Trotter, 2003). The BPP solution assigns the customers to vehicles, assuming no travelling cost between the nodes, while the TSP solution gives the order of customer visits, assuming infinite vehicle capacity. As both of these problems are NP-hard, the CVRP is also NP-hard, and therefore difficult to solve.

A typical CVRP involves a fleet of homogeneous delivery vehicles (each with limited capacity), a single depot, and a number of customers. Each customer is geographically located at specific coordinates, and demands a predetermined positive quantity of some commodity that is known in advance. The total demands of all customers cannot exceed the total capacity of the fleet, and the demand of any node cannot exceed the vehicle capacity. The distance between customers can be symmetric (identical in both directions) or asymmetric (longer in one direction than the other).

The CVRP must design a feasible route for each vehicle that minimises the total distance travelled without exceeding the vehicle capacity. A route represents the order of customers to be visited by a particular vehicle. A route is feasible only if the total demand of all customers on that route does not exceed the vehicle capacity. A CVRP solution is considered feasible if all customers are served, and each customer is served exactly once by one vehicle. Moreover, each vehicle must begin from the depot and visit a specific number of customers before returning to the depot.

6.1.1 Problem Formulation

Similar to TSP, the CVRP can be described as a fully connected undirected graph $G = (N, E)$ where N represents the nodes (customers), and E represents the edges (roads) between the nodes. Each edge is assigned a positive value that equals the travelling cost.

- $N = \{1, 2, \ldots, n\}$ represents a set of nodes (customers), with node 1 representing the depot and $n \geq 2$.

- $E = \{i, j\}$, where $1 \leq i, j \leq n$ and $i \neq j$, represents a set of edges (roads) between the nodes. Each edge is assigned a positive value representing its travelling cost or the distance between its two connected nodes.

- $C(i, j)$ stores the distance or the cost value of each edge $(i, j) \in E$, where $C(i, j) > 0$ and $C(i, i) = 0$. $C(i, j)$ is symmetric; that is, $C(i, j) = C(j, i)$ for each $i, j \in N$.

- $K = \{1, 2, \ldots, m\}$ with $m > 0$ represents a set of vehicles, each with capacity Q.

- Q: represents the vehicle capacity (identical for all vehicles; i.e., a homogeneous fleet is assumed).

- d_i: represents the demand for each $i \in N$, where $d_1 = 0$ and $0 < d_i \leq Q$.

- $R = \{1, 2, \ldots, n, 1\}$ represents the route for each vehicle, or the order of nodes to be visited by a particular vehicle. Each vehicle starts and finishes at the depot. A route is considered feasible if the total demand of all nodes on that route does not exceed Q, i.e., $\sum_{\mathrm{i}}^{\mathrm{n}} di \ \leq Q$.

- $S = \{R_1, R_2, \ldots, R_k\}$ represents the solution of the VRP, where k is the number of routes.

Mathematically, the CVRP can be stated and modelled based on the integrity constraints (Caric & Gold, 2008; S. N. Kumar & Panneerselvam, 2012) as follows:

$$min \sum_{k \in K} \sum_{i \in N} \sum_{j \in N} C_{ij} X_{ij}^{K} \tag{6.1}$$

subject to:

$$\sum_{k \in K} \sum_{i \in N} X_{ij}^{k} = 1 \ \ , \forall j \in N \backslash \{1\} \tag{6.2}$$

$$\sum_{k \in K} \sum_{j \in N} X_{ij}^{k} = 1 \ , \forall i \in N \backslash \{1\} \tag{6.3}$$

$$\sum_{i \in N} di \sum_{j \in N} X_{ij}^{k} \ \leq Q \ , \forall k \in K \tag{6.4}$$

$$\sum_{i \in N} X_{ih}^{k} - \sum_{j \in N} X_{hj}^{k} = 0 \ , \forall h \in N \backslash \{1\}, \forall k \in K \tag{6.5}$$

$$\sum_{j \in N \setminus \{1\}} X_{1j}^{k} = 1 \quad, \forall k \in K \tag{6.6}$$

$$\sum_{i \in N \setminus \{1\}} X_{i1}^{k} = 1 \quad, \forall k \in K \tag{6.7}$$

$$X_{ij}^{k} \in \{0, 1\} \;, \forall i \in N, \forall j \in N, \forall k \in K \tag{6.8}$$

$$\text{where, } X_{ij}^{K} \begin{cases} 1: & \text{if vehicle } k \text{ visits node } j \text{ immediately after node } i, i \neq j \\ 0: & \text{otherwise} \end{cases} \tag{6.9}$$

$\forall$: for each , \ : except

The objective function in (6.1) minimises the total travelling distance. Constraints (6.2) and (6.3) guarantee that each node is served exactly once by one vehicle. Constraint (6.4) ensures that the total demand for any vehicle route does not exceed the vehicle capacity. Constraint (6.5) states that a vehicle must leave the node that it has just entered. Constraints (6.6) and (6.7) indicate that each vehicle must leave and return to the depot. The last constraint checks the integrity.

6.1.2 A Classical Heuristic Algorithm

The CVRP has received considerable attention by researchers. Throughout the past few decades, many exact, heuristic and metaheuristic algorithms have been proposed for solving this problem. The heuristic algorithms optimise the solution based on heuristic values for small-sized instances. One popular heuristic algorithm is the Savings Algorithm (SA) proposed by Clarke and Wright (Clarke & Wright, 1964). This is considered one of the most constructive heuristic algorithms for solving CVRPs. To minimise the total cost, the SA tries merging two separate routes into a single route. The savings of the merging process are calculated as follows:

$$Savings\ (i,j) \ = \ Cost(1,i) \ + \ Cost\ (1,j) - Cost\ (i,j), \tag{6.10}$$

where *i*, *j* denote two customers, and 1 represents the depot. Figure 6-3 (a) shows the initial situation, in which both customers *i* (demand = 5) and *j* (demand = 5) need a separate vehicle (capacity = 10). After applying the SA and merging the routes of customers *i* and *j*, we obtain Figure 6-3 (b). As the vehicle capacity is sufficient for both nodes, the route shortens if the vehicle moves directly to *j* after visiting *i*.

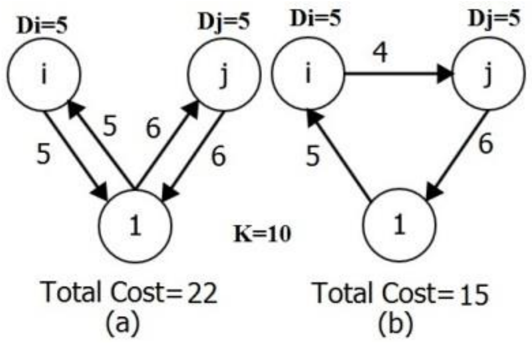


Figure 6-3. Concept of the SA (Lysgaard, 1997)

Besides the SA, the CVRP has been solved by the IWD algorithm. The experimental results are published in (Wedyan & Narayanan, 2014).

6.2 HCA for Solving CVRP

In this thesis, the CVRP was solved by the HCA framework (described in Section 3 of Chapter 3). The algorithm begins by initialising all static and dynamic parameters, and generating an initial set of artificial water drops. The number of water drops is assumed to equal the number of existing nodes. The water drops are then randomly distributed over the graph nodes, representing their first cities to be visited from the depot.

Next, every water drop begins moving from one point to another such that the selected point has the highest probability among the unvisited nodes, without violating the problem constraints. The next node is decided by assessing the amount of soil on the path leading to the node, and the path depth. The savings amount is used as another heuristic that reduces the total cost while enhancing the solution quality; see Eq. (3.6). If the savings value is high, visiting the second customer from the first customer is a good choice.

$$P^{WD}{}_{i}(j) = \frac{f\left(Soil(i,j)\right)^{2} \times g\left(Depth(i,j)\right) \times \boldsymbol{Sav}\ (\boldsymbol{i},\boldsymbol{j})}{\sum_{k \notin vc(WD)} \left(f\left(Soil\ (i,k)\right)^{2} \times g\left(Depth(i,k)\right) \times \boldsymbol{Sav}\ (\boldsymbol{i},\boldsymbol{j}) \right)} \tag{6.11}$$

Each water drop maintains a visited list that tracks the visited nodes and guarantees that no node is visited twice (visiting a node more than once would violate one of the CVRP constraints). This process continues until the water drop reaches its maximum load. The water drop then returns to the depot and starts a new tour, thus completing its route for a vehicle. Candidate solutions to this problem are constructed incrementally. All of the water drops cooperate simultaneously (in parallel) to generate solutions, and each water drop generates a different solution. This means that the water drops sequentially choose a new node to visit, at one time several routes can be built. One iteration is completed when each water drop has visited all nodes and generated its own solution. At the end of each iteration, the local-best solution found so far is updated by the current least-cost solution. The temperature after each iteration is updated based on the total solution quality of all water drops and the previous temperature.

The evaporation rate is randomly chosen between one and the total number of water drops. The water drops to be evaporated are selected by the roulette-wheel technique, considering the solution quality of each water drop. At the condensation stage, the water drops approach each other, then collide and combine. These operations enable information exchange among the particles using local improvement methods, which minimise the number of required iterations and enhance the solution quality in subsequent iterations. Moreover, depending on the solution quality, the use of these methods varies in each cycle of the algorithm (i.e., will not be used at the end of each iteration). In contrast, in other algorithms these methods must be implemented at the end of each iteration. The condensation stage also updates the best global solution found so far. After evaluating the solutions, the current best water drop is selected and rewarded by minimising the amount of soil on each edge in its solution set. The algorithm stops with the latest global best solution when the termination condition is met. Otherwise, all dynamic variables (amount of soil on each edge, the velocity of each water drop and the amount of soil that the water drop can hold) are reinitialised.

6.2.1 Problem Representation

The CVRP problem is naturally representable as a graph of nodes (customers) connected by edges (roads between customers). The Euclidean distances between the nodes represent the travelling costs. Initially, all of the problem data are read from a text file that holds the number of customers, the vehicle capacity, and the coordinates and demands of the customers. The depot is assigned as node one. The Euclidean distance between the locations of all customers is then calculated by Eq. (6.12),

$$C_{ij} = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}. \tag{6.12}$$

This study assumes a symmetric CVRP and rounds the distance result to the nearest integer. Thus, the travelling cost between two nodes is the same in both directions. Consequently, the cost matrix of the problem is symmetric ($C_{ij} = C_{ji}$). The demand of each customer is stored in an additional matrix called the *demand matrix*. The vehicle capacity and number of customers are stored in other variables.

6.2.2 Solution Representation

To construct a solution, the water drops travel among the nodes of the graph. The CVRP solution consists of multiple routes, one for each vehicle. Suppose there are N customers requiring K vehicles to satisfy their demands. The solutions generated by the water drops are presented as follows:

Table 6-1. Solution representation in the CVRP problem

Water Drops	Vehicle	Depot	Customers order					Depot
WD_1	1	1	C_1	C_2	C_3	…	C_n	1
	2	1	C_1	C_2	C_3	…	C_n	1
	…	…						
	K	1	C_1	C_2	C_3	…	C_n	1
WD_2	1	1	C_1	C_2	C_3	…	C_n	1
	2	1	C_1	C_2	C_3	…	C_n	1
	…	…						
	K	1	C_1	C_2	C_3	…	C_n	1
….								
WD_n								

For example, the solution generated by one water drop might be represented as follows:

- Route1: depot → customer 2 → customer 6 → customer 8 → customer 10 → depot
- Route2: depot → customer 3 → customer 7 → customer 5 → depot
- Route3: depot → customer 4 → depot

As shown in the above example, the solution is composed of a set of vectors. Each vector represents one route travelled by one vehicle. The vector contains the start node of the vehicle (the depot), the sequence of customer identities to be visited, and the end node (the depot). The number of customers per vehicle will depend on the topology and constraints of the problem.

6.2.3 The Local Improvement Methods

Various local improvement methods have been described and applied in the CVRP literature (Bräysy & Gendreau, 2005; Kindervater & Savelsbergh, 1997; M. Qi, Zhang, Zhang, & Miao, 2013; Vidal, Crainic, Gendreau, & Prins, 2013). These improvement methods can be categorised into intra-route (one route) or inter-route (two routes) methods; alternatively, they can be considered as node-exchanging or arc-exchanging operators. Node-exchanging operators move a node (customer) to another location, whereas arc-exchanging operators move or exchange a number of arcs to another location. In both interpretations, the node can be moved within the same route or passed to a different route. The efficiency and effectiveness of these methods depends on the way they are implemented and the execution period. Considering all possible permutations will improve the solution quality but increases the execution time. Below, we report on the five most commonly used improvement methods in the literature.

6.2.3.1 The Shift (Relocate) Operator

The shift operator is an inter-route operator that deals with two routes. After checking that the capacity constraint on the new route fits the new customer demand, the shift operator moves one customer from one route to another, as shown in Figure 6-4 (Bräysy & Gendreau, 2005). If the move is valid, the location (order) of the customer in the sequence is determined by the least cost (i.e., by checking all possible insertion positions). If the cost of the modified routes is better than the cost of the old ones, the routes are updated. This operation is important as it may reduce the number of required vehicles used, which improves the solution.

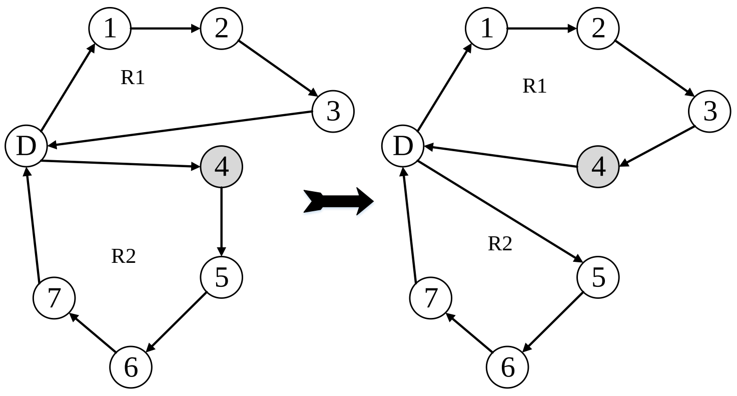


Figure 6-4. An example of shift operation

In Figure 6-4, node 4 is moved from R2 to R1, assuming that the vehicle capacity that serves the nodes of R1 can also accommodate the demand of node 4. In addition, the order of the nodes is rearranged to achieve the lowest cost.

6.2.3.2 The Swap (Exchange) Operator

The swap operation improves two routes by simultaneously exchanging two customers from two different routes while preserving their positions from the old routes (Bräysy & Gendreau, 2005). An example is shown in Figure 6-5. It then checks whether both routes remain feasible and whether the swap improves the total cost of the two routes. If the modified routes are more economical then the old ones, the routes are updated.

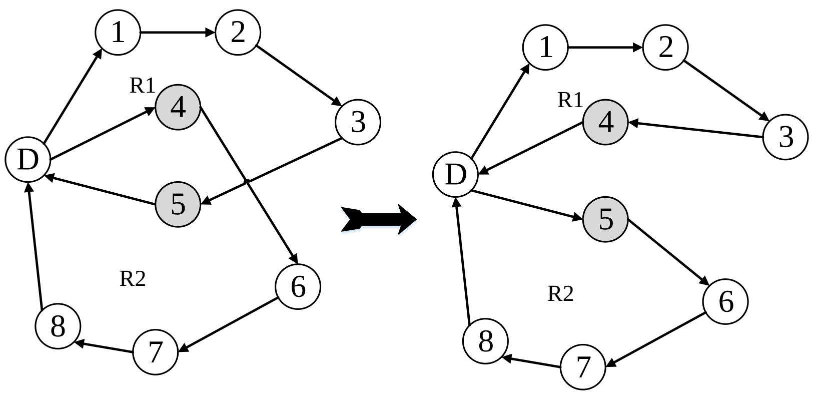


Figure 6-5. Example of a swap operation

In Figure 6-5, swapping nodes 4 and 5 minimises the total cost, assuming that the capacity constraint is not violated.

6.2.3.3 The 2-Opt Method

This method exchanges two edges in one route to remove intersections. More details can be found in Section 2 of Chapter 4.

6.2.3.4 The 3-Opt Exchange Method

The 3-Opt operator works similarly to 2-Opt, but removes three edges at the same time within the same route and reconnects them in all possible ways to shorten the tour (S. Lin, 1965). In each reconnection, 3-opt evaluates the route cost to find the lowest one (Blazinskas & Misevicius, 2011).

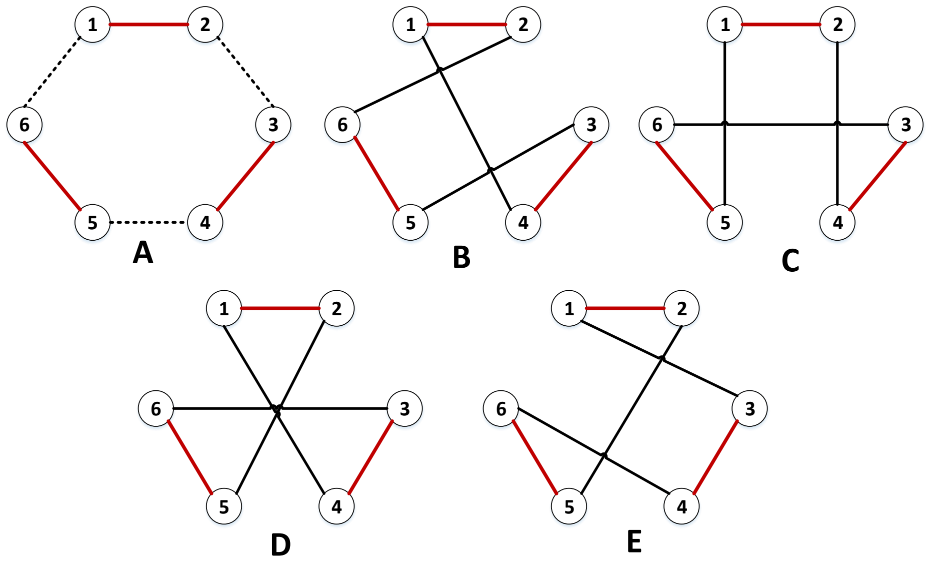


Figure 6-6. Example of 3-Opt moves for six nodes (https://i.stack.imgur.com/PAhkl.jpg)

When edges (2, 3), (4, 5), and (6, 1) are removed from Figure 6-6 (A), there are 8 different ways to reconnect the tour (7 excluding the original order). However, three movements lead to the same results as the 2-Opt operation, so are not shown in Figure 6-6.

6.2.3.5 The 2H-Opt Operation

The 2H-Opt heuristic (also known as 2.5-Opt) is a local search operation. It works by moving one node to another location within the same route, provided that the move reduces the cost (Stattenberger, Dankesreiter, Baumgartner, & Schneider, 2007).

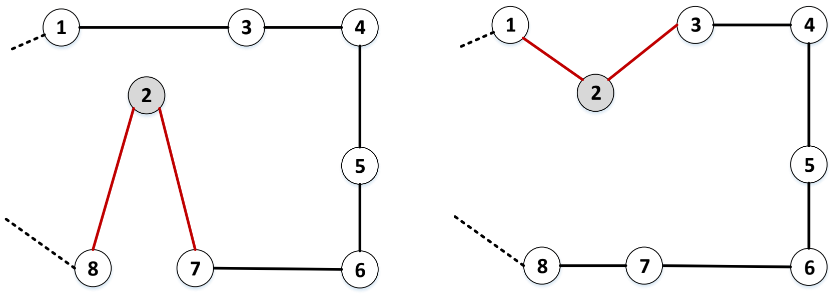


Figure 6-7. Example of the 2H-opt operator

In Figure 6-7, node 2 was relocated between nodes 1 and 3 to minimise the route cost.

6.3 Computational Experiment

This section explains the experimental setup, the dataset used, and the experimental results. This study assumes that goods are delivered but not collected. Moreover, all customers' demands are known in advance, and the delivery cannot be split between two vehicles. Several identical homogeneous vehicles are available at a single depot. The travel cost between the locations of each pair of customers is the same in both directions (i.e., the CVRP problem is symmetric). There is no time-window constraint in the CVRP. The main and secondary objectives are to minimise the total cost and minimise the number of vehicles needed to serve all customers, respectively. The parameter values are given in Chapter 3, Section 3.5. To emphasise, the number of water drops equals the number of customers, and the maximum number of iterations is four times the number of customers. The cost between customers is calculated using Eq. (4.1) and the results were rounded to integer value.

6.3.1 Experimental Data

The HCA was tested and evaluated on two types of CVRP instances (structural and benchmark). The structural instances were generated as circles of nodes with one central node (the depot). The structural instances are easily evaluated, and useful for testing the algorithm validity. The benchmark instances were some well-known CVRP datasets. Among the several standard benchmark datasets for CVRP in the literature, the most common datasets consist of five categories of instances with different structures and various sizes. The present experimental study is performed on set *A*, set *B*, and set *P* (Augerat et al., 1998); set *E* (Christofides & Eilon, 1969); and set *F* (Fisher, 1994). The instances in these datasets represent real-life data. The locations of the customers and the depot are randomly distributed in set *A*, whereas the customers are clustered in set *B*. Set *P* consists of modified instances from the *A*, *B*, and *E* instances. The instances in these series have different values of tightness, defined as the ratio of the total number of demands to the total capacity of the vehicles. Therefore, tight instances are more difficult to solve optimally than looser instances. Further details about these series can be found in Uchoa et al. (2017). These datasets are downloadable from several websites[3,4]. The HCA was executed ten times per instance, and the best results were recorded. The results obtained were compared with the currently best-known solution for each instance, based on the minimum cost and number of vehicles required. The results of the proposed algorithm were evaluated and compared with those of the other algorithms.

6.3.2 Computational Results

In this section, we reported the obtained results.

6.3.2.1 Structural Instances

Three circular instances with different numbers of customers (30, 50, and 100) were generated. In each instance, the radius was set to 5, and the coordinates of the customers were equally distributed on the circle's circumference. Initially, we calculated the theta angles that distribute the coordinates evenly between 0 and 2π, which depend on the number of customers, by Eq. (6.13):

$$\theta = \frac{(2 \times \pi)}{Number\ of\ points} \tag{6.13}$$

A point (x, y) at angle θ on a circle with known radius r can be expressed in polar coordinates, as shown in Figure 6-8.

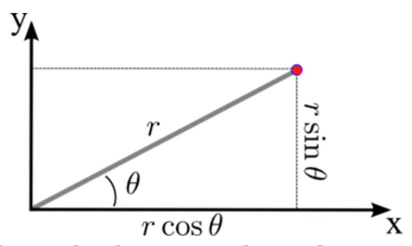


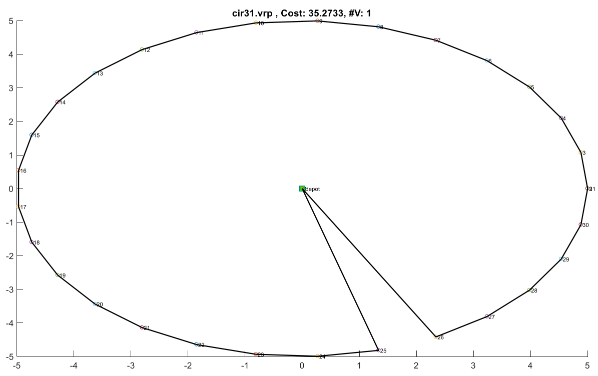
Figure 6-8. Relationship between polar and Cartesian coordinates[5]

[3] http://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/index.htm
[4] http://neo.lcc.uma.es/vrp/vrp-instances/
[5] From Math Insight. http://mathinsight.org/image/polar_coordinates_cartesian

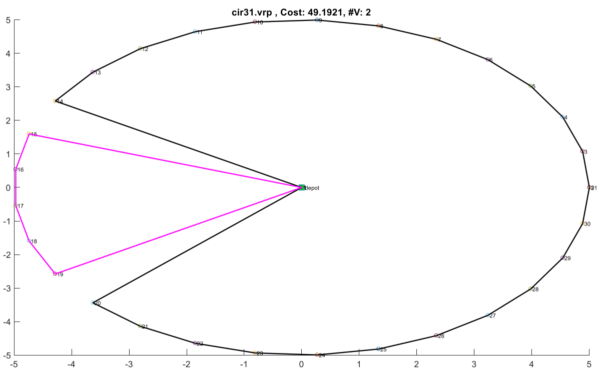
The Cartesian coordinates of node *i* on a circle centred at (x_0, y_0) are given by

$$\begin{aligned} X_i &= X_0 + (r \times \cos(\theta)) \\ Y_i &= Y_0 + (r \times \sin(\theta)). \end{aligned} \tag{6.14}$$

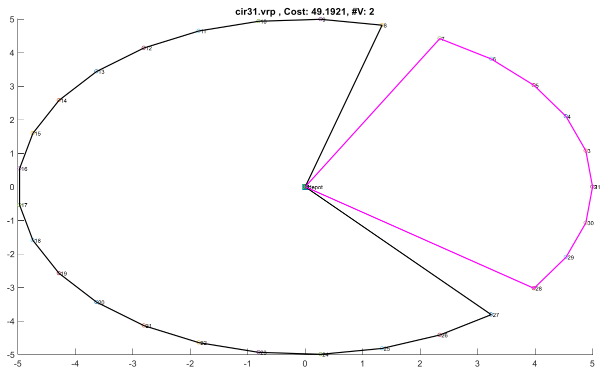
In each instance there is one depot located at the centre of the circle (0, 0). The demand of each customer was set to 1 unit. The vehicle capacity was initialised as the total demand of all customers, requiring one vehicle in the first run. The vehicle capacity was decreased in each subsequent run. Figure 6-9 shows the result of applying the HCA on the first circular instance with 30 customers. The "Cir***X***-Capacity:***X***, Cost=***X***, #*V*=***X***" indicates instance name-number of nodes, vehicle capacity, total cost, and number of required vehicles respectively.



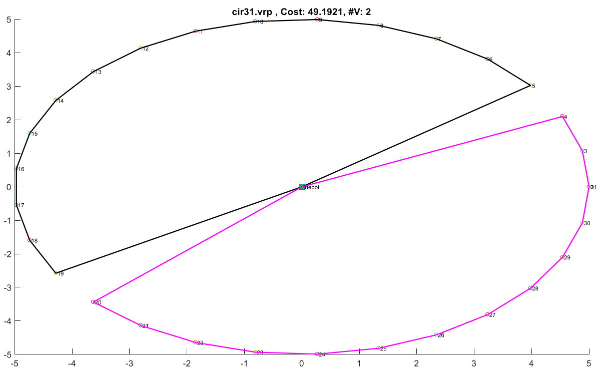
1. Cir31-Capacity: 30, Cost= 35.2733, #*V*=1



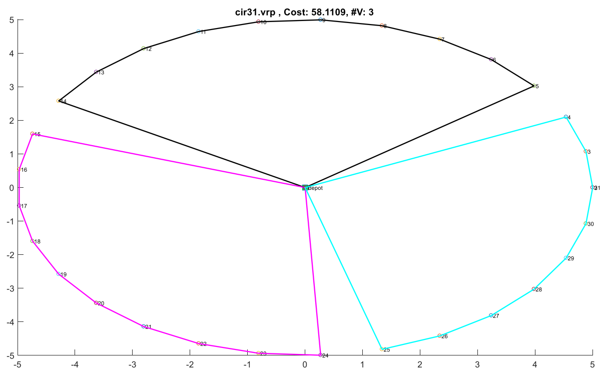
2. Cir31-Capacity: 25, Cost= 49. 1921, #*V*=2



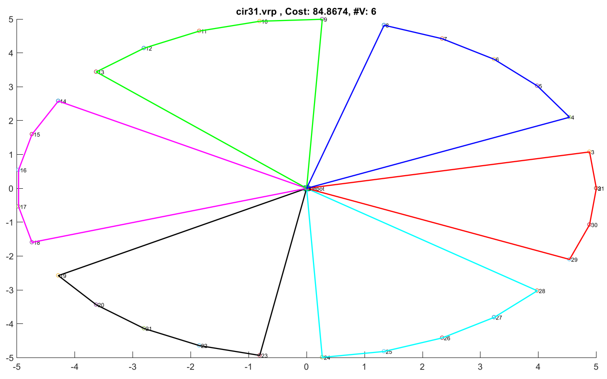
3. Cir31-Capacity: 20, Cost= 49.1921, #*V*=2



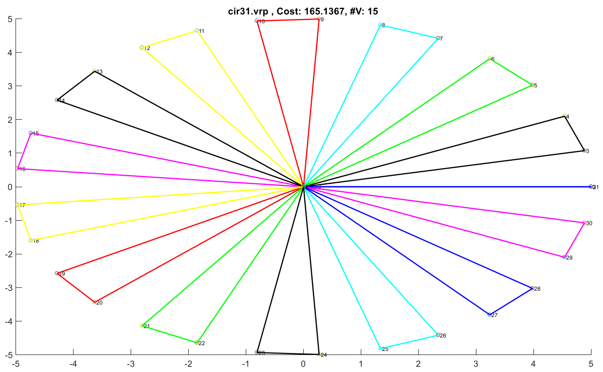
4. Cir31-Capacity: 15, Cost= 49.1921, #*V*=2



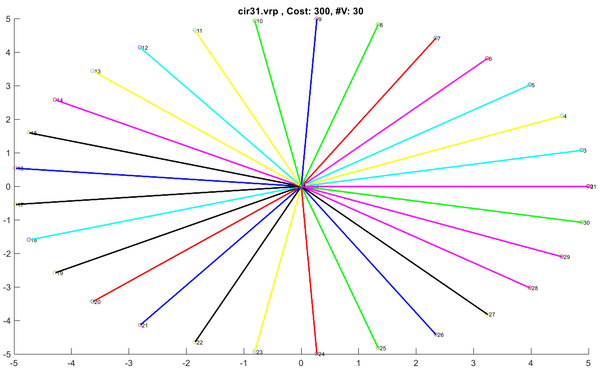
5. Cir31-Capacity: 10, Cost= 58.1109, #*V*=3



6. Cir31-Capacity: 5, Cost= 84.8674, #*V*=6



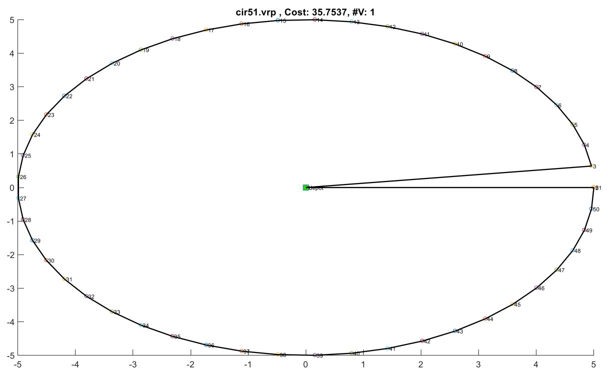
7. Cir31-Capacity: 2, Cost= 165.1367, #*V*=15



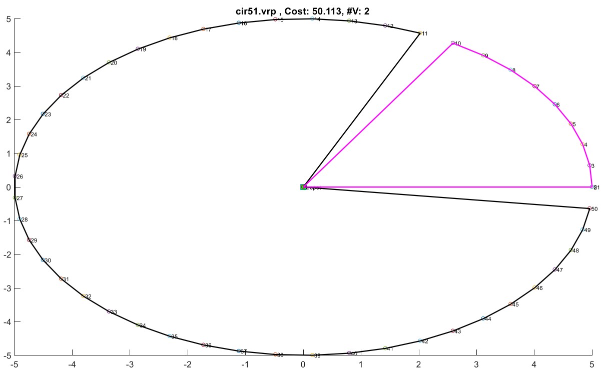
8. Cir31-Capacity: 1, Cost= 300, #*V*=30

Figure 6-9. Circular instance with 30-customers and one depot

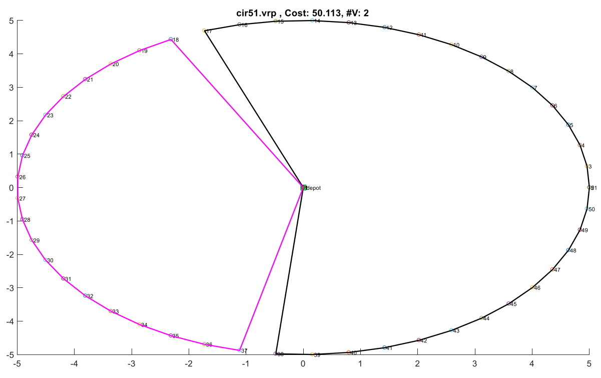
Note that an additional vehicle is needed whenever the capacity is reduced. Also, the HCA manages to find the shortest path with the maximum vehicle load in each run. Figure 6-10 shows the output of the circular instance with 50 customers.



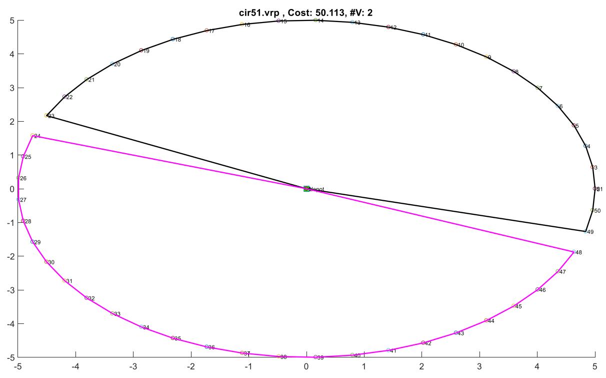
1. Cir50-Capacity: 50, Cost= 35.7537, #*V*=1



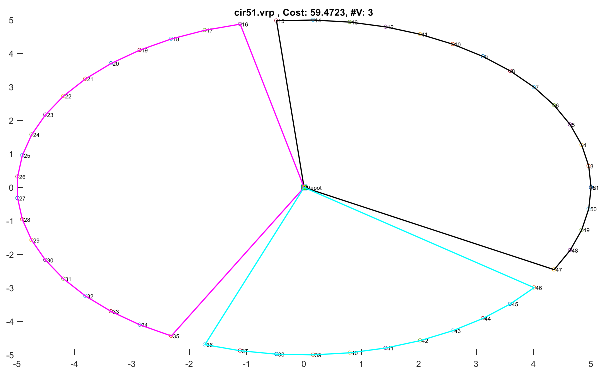
2. Cir50-Capacity: 40, Cost= 50.113, #*V*=2



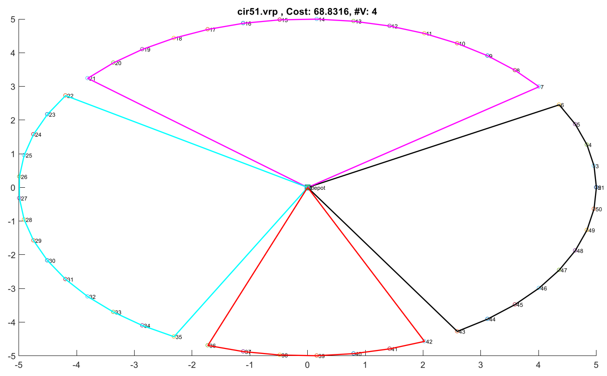
3. Cir50-Capacity: 30, Cost= 50.113, #*V*=2



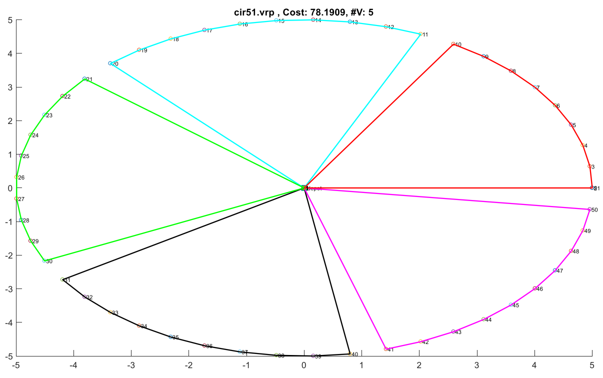
4. Cir50-Capacity: 25, Cost= 50.113, #*V*=2



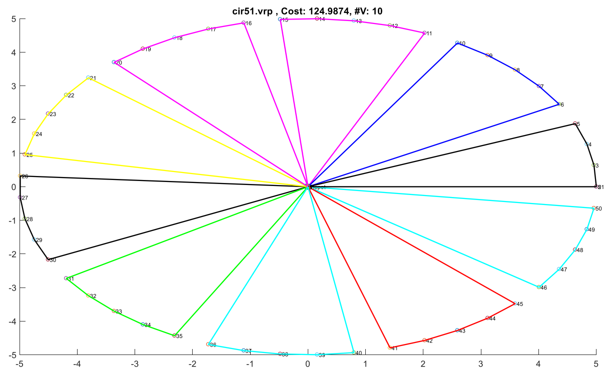
5. Cir50-Capacity: 20, Cost= 59.4723, #*V*=3



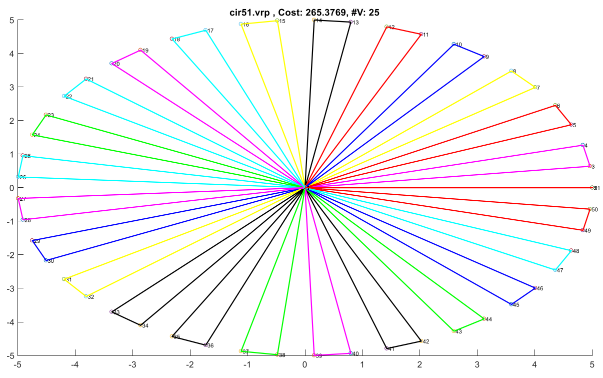
6. Cir50-Capacity: 15, Cost= 68.8316, #*V*=4



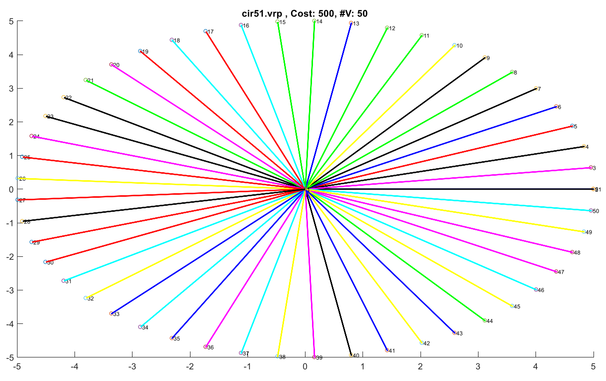
7. Cir50-Capacity: 10, Cost= 78.1909, #*V*=5



8. Cir50-Capacity: 5, Cost= 124.9874, #*V*=10



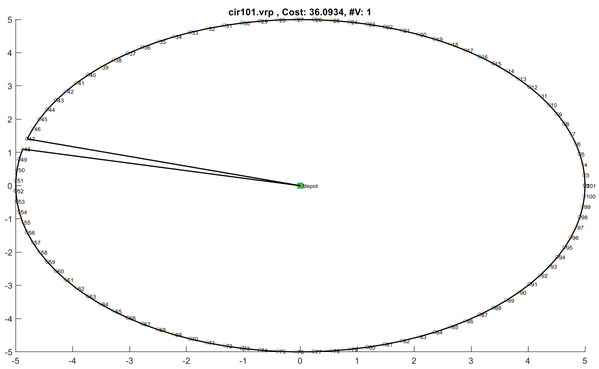
9. Cir50-Capacity: 2, Cost= 265.3769, #*V*=25



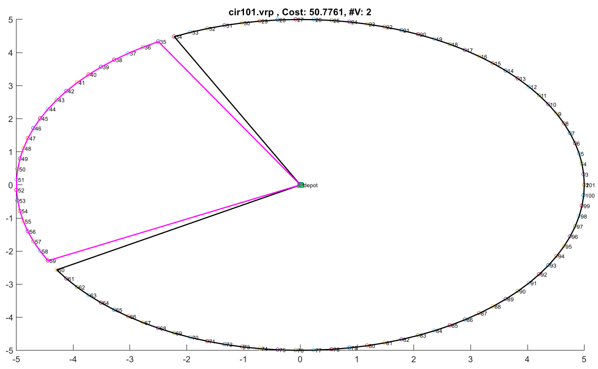
10. Cir50-Capacity: 1, Cost= 500, #*V*=50

Figure 6-10. Circular instance with 50-customers and one depot

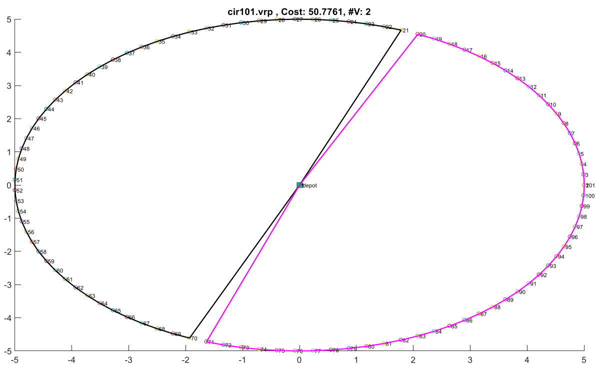
Figure 6-11 shows the output of the circle instance with 100 customers.



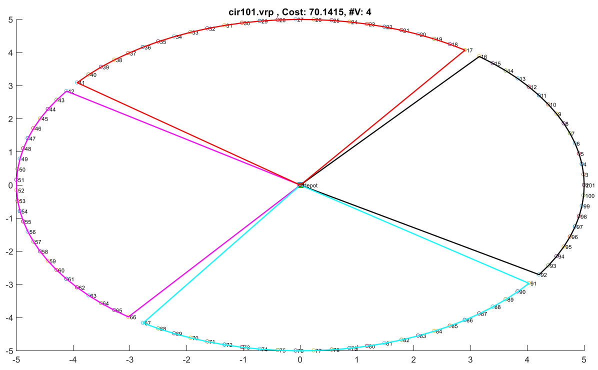
1. Cir101-Capacity: 100, Cost=36.088, #*V*=1



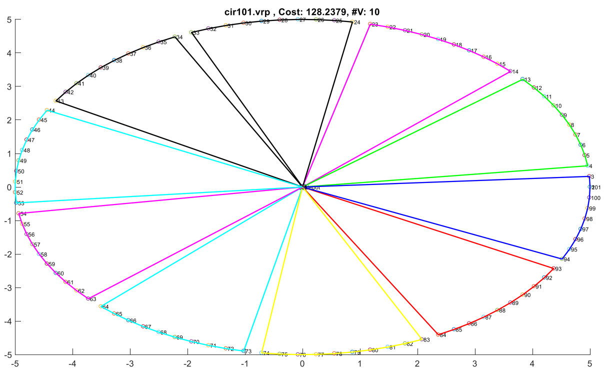
2. Cir101-Capacity: 75, Cost=50.7761, #*V*=2



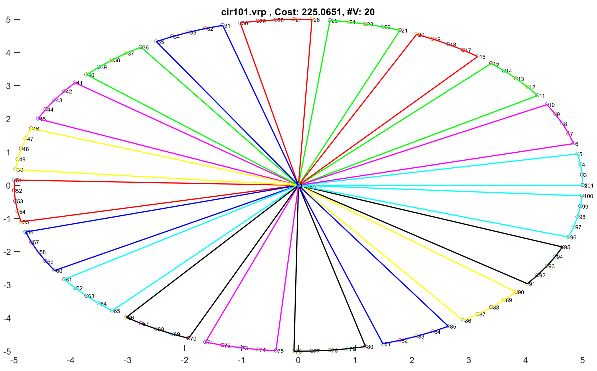
3. Cir101-Capacity: 50, Cost=50.7761, #*V*=2



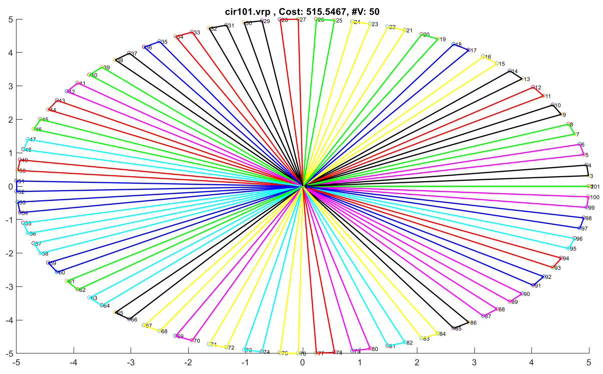
4. Cir101-Capacity: 25, Cost= 70.1415, #*V*=4



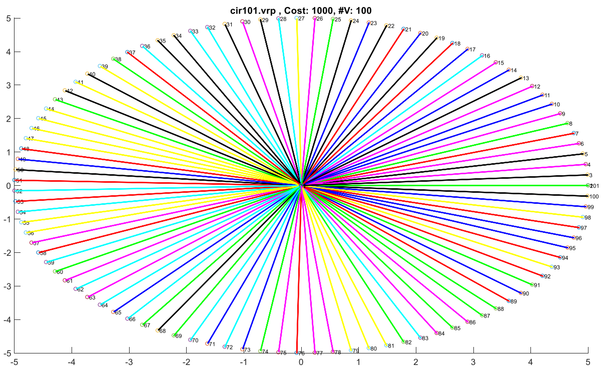
5. Cir101-Capacity: 10, Cost= 128.24, #*V*=10



6. Cir101-Capacity: 5, Cost= 225.065, #*V*=20



7. Cir101-Capacity: 2, Cost= 515.547, #*V*=50



8. Cir101-Capacity: 1, Cost= 1000, #*V*=100

Figure 6-11. Circular instance with 100-customers and one depot

These results show that the HCA solved these instances in every run and that vehicle number increased with decreasing capacity of the vehicle.

6.3.2.2 Benchmark Instances

In the second experiment, we tested the HCA on benchmark instances from the standard CVRP library. The results of applying HCA on datasets *A*, *B*, *P*, *E*, and *F* are reported in the following tables. The second column of each table lists the names of the instances. The names are formatted as *X*-*n*##*k*#, where *X*, *n* and *k* denote the dataset type, number of nodes and number of vehicles, respectively. The instances marked with "**" indicate that increasing the number of vehicles by one decreases the best-known value. Columns 3 and 4 list the number of customers and number of vehicles required to solve this instance, respectively. The maximum vehicle capacity in each instance is listed in the "V.C" column. The "BKS" and "HCA" columns list the currently best-known solutions in each instance, as reported by (Uchoa et al., 2017) and the HCA, respectively. The "no v." column represents the number of vehicles required using HCA. "Avg. time" reports the average time in seconds, and "Avg. #", "Max #" and "Min #" report the minimum, average, and maximum iteration number, respectively. The "Diff" column shows the actual difference between the obtained and best-known solutions. The percentage gap (i.e., the error percentage) between the obtained and best-known values was calculated as follows:

$$Gap\% = \frac{(Obtained\ ressult - best\_known\ result)}{best\_known\ result} \times 100\% \qquad (6.15)$$

In Table 6-2, the boldface results indicate that HCA found the best-known solution on set *A*.

Table 6-2. Results on dataset A

No.	**Instance name**	**#C**	**#V**	**V.C**	**BKS**	**HCA**	**no v.**	**Avg. time (s)**	**Min #**	**Avg. #**	**Max #**	**Diff**	**% Gap**
1	A-n32-k5	31	5	100	784	**784**	5	3.85	18	51.36	96	0	0.00%
2	A-n33-k5	32	5	100	661	**661**	5	5.16	6	22.57	111	0	0.00%
3	A-n33-k6	32	6	100	742	**742**	6	5.16	13	44.00	90	0	0.00%
4	A-n34-k5	33	5	100	778	**778**	5	5.12	7	49.00	123	0	0.00%
5	A-n36-k5	35	5	100	799	**799**	5	6.17	84	84.00	84	0	0.00%
6	A-n37-k5	36	5	100	669	**669**	5	7.53	23	30.00	35	0	0.00%
7	A-n37-k6	36	6	100	949	**949**	6	5.78	74	82.00	90	0	0.00%
8	A-n38-k5	37	5	100	730	**730**	5	6.88	7	47.60	110	0	0.00%
9	A-n39-k5	38	5	100	822	**822**	5	7.13	31	62.00	95	0	0.00%
10	A-n39-k6	38	6	100	831	**831**	6	8.73	28	42.00	56	0	0.00%
11	A-n44-k6	43	6	100	937	**937**	6	11.27	26	26.00	26	0	0.00%
12	A-n45-k6	44	6	100	944	948	6	12.60	59	59.00	59	4	0.42%
13	A-n45-k7	44	7	100	1146	1151	7	12.11	131	131.00	131	5	0.44%
14	A-n46-k7	45	7	100	914	**914**	7	13.89	75	92.00	109	0	0.00%
15	A-n48-k7	47	7	100	1073	1074	7	16.44	71	71.00	71	1	0.09%
16	A-n53-k7	52	7	100	1010	1017	7	23.45	41	96.14	144	7	0.69%
17	A-n54-k7	53	7	100	1167	1300	**9**	24.42	129	129.00	129	133	11.40%
18	A-n55-k9	54	9	100	1073	1074	9	26.95	143	143.00	143	1	0.09%
19	A-n60-k9	59	9	100	1354	1361	9	35.89	142	142.00	142	7	0.52%
20	A-n61-k9	60	9	100	1034	1044	**10**	43.08	116	137.00	175	10	0.97%
21	A-n62-k8	61	8	100	1288	1322	8	41.66	38	38.00	38	34	2.64%
22	A-n63-k9	62	9	100	1616	1641	9	42.32	187	187.00	187	25	1.55%
23	A-n63-k10	62	10	100	1314	1324	10	44.55	36	69.00	102	10	0.76%
24	A-n64-k9	63	9	100	1401	1421	9	47.45	124	124.00	124	20	1.43%
25	A-n65-k9	64	9	100	1174	1181	9	50.35	32	32.00	32	7	0.60%
26	A-n69-k9	68	9	100	1159	1170	9	64.46	99	99.00	99	11	0.95%
27	A-n80-k10	79	10	100	1763	1822	10	112.25	165	165.00	165	59	3.35%
Average					1041.9	1054.3							
(T-test) *P*-value						0.0276							

There is a significant difference (*P* = 0.0276) between the HCA and BKS. However, the HCA did find the best-known solution in approximately 44% (12 out of 27) of the instances. For the 66% of the remaining instances, it obtained a near-optimal solution (with less than 1% gap). The HCA solved all instances with the same number of vehicles as in the best-known solution, except in instances “A-n54-k7” and “A-n61-k9”, where it required 2 and 1 extra vehicles, respectively. The “A-n54-k7” result (with an 11% gap) deviated most widely from the best-known solution. However, the average iteration numbers in most of the instances were relatively small when compared with the instance size. Table 6-3 shows the evaluation results of dataset *B*.

Table 6-3. Results on dataset B

No.	Instance name	#C	#V	V.C	BKS	HCA	no v.	Avg. Time (s)	Min #	Avg. #	Max #	Diff	% Gap
1	B-n31-k5	30	5	100	672	**672**	5	4.17	2	4.10	32	0	0.00%
2	B-n34-k5	33	5	100	788	**788**	5	5.09	14	36.75	93	0	0.00%
3	B-n35-k5	34	5	100	955	**955**	5	5.85	37	49.33	59	0	0.00%
4	B-n38-k6	37	6	100	805	**805**	6	7.87	21	35.50	57	0	0.00%
5	B-n39-k5	38	5	100	549	**549**	5	8.13	26	61.50	97	0	0.00%
6	B-n41-k6	40	6	100	829	**829**	6	8.68	67	67.00	67	0	0.00%
7	B-n43-k6	42	6	100	742	**742**	6	12.21	63	76.00	89	0	0.00%
8	B-n44-k7	43	7	100	909	916	7	11.39	37	37.00	37	7	0.77%
9	B-n45-k5	44	5	100	751	**751**	5	14.29	37	89.50	125	0	0.00%
10	B-n45-k6	44	6	100	678	691	6	15.55	97	97.00	97	13	1.92%
11	B-n50-k7	49	7	100	741	**741**	7	22.57	28	82.28	141	0	0.00%
12	B-n50-k8	49	8	100	1312	1326	8	18.45	55	72.00	89	14	1.07%
13	B-n51-k7 **	50	7	100	1032	1016	**8**	22.11	51	83.66	137	-14	-1.36%
14	B-n52-k7	51	7	100	747	748	7	22.98	149	149.00	149	1	0.13%
15	B-n56-k7	55	7	100	707	712	7	30.27	32	100.42	136	5	0.71%
16	B-n57-k7 **	56	7	100	1153	1145	**8**	31.82	86	86.00	86	-8	-0.69%
17	B-n57-k9	56	9	100	1598	1612	9	29.65	32	32.00	32	14	0.88%
18	B-n63-k10	62	10	100	1496	1545	10	43.36	137	149.50	162	49	3.28%
19	B-n64-k9	63	9	100	861	881	9	51.44	82	111.00	140	20	2.32%
20	B-n66-k9	65	9	100	1316	1329	9	52.43	55	55.00	55	13	0.99%
21	B-n67-k10	66	10	100	1032	1059	10	54.26	51	51.00	51	27	2.62%
22	B-n68-k9	67	9	100	1272	1291	9	57.59	124	136.50	149	19	1.49%
23	B-n78-k10	77	10	100	1221	1250	10	100.86	54	54.00	54	29	2.38%
	Average				963.7	971.9							
	(T-test) *P*-value					0.0110							

In this case, the *P*-value indicates there is a significant difference between the obtained result and BKS. Nonetheless, the HCA found the best-known solution in approximately 39% (9 out of 23) of the instances, and near-optimal solutions (below 1% gap) in 36% of the remaining instances. In two instances (B-n1-k7 and B-n57-k7), the solutions found by HCA were lower-cost than the best-known solution but required one extra vehicle. In HCA, the objective function minimising the distance is prioritised over that minimising the vehicle number. When these objectives conflict, prioritisation removes the difficulty of optimising both at the same time. Table 6-4 shows the evaluation results of dataset *P*.

Table 6-4. Results on dataset P

No.	**Instance name**	**#C**	**#V**	**V.C**	**BKS**	**HCA**	**no v.**	**Avg. Time (s)**	**Min #**	**Avg. #**	**Max #**	**Diff**	**% Gap**
1	P-n16-k8	15	8	35	450	**450**	8	0.72	7	13.83	18	0	0.00%
2	P-n19-k2	18	2	160	212	**212**	2	1.46	8	25.00	54	0	0.00%
3	P-n20-k2	19	2	160	216	**216**	2	1.80	10	10.00	10	0	0.00%
4	P-n21-k2	20	2	160	211	**211**	2	1.85	4	4.36	7	0	0.00%
5	P-n22-k2	21	2	160	216	**216**	2	2.55	3	4.65	12	0	0.00%
6	P-n22-k8**	21	8	3000	603	**590**	**9**	2.19	8	29.58	59	-13	-2.16%
7	P-n23-k8	22	8	40	529	**529**	8	2.67	30	32.50	35	0	0.00%
8	P-n40-k5	39	5	140	458	**458**	5	14.42	10	27.94	69	0	0.00%
9	P-n45-k5	44	5	150	510	**510**	5	20.25	32	67.17	92	0	0.00%
10	P-n50-k7	49	7	150	554	555	7	28.78	555	555.00	555	1	0.18%
11	P-n50-k8**	49	8	120	631	629	9	26.60	63	63.00	63	1	-0.32%
12	P-n50-k10	49	10	100	696	697	10	26.96	24	24.00	24	1	0.14%
13	P-n51-k10	50	10	80	741	745	10	28.51	20	67.00	114	4	0.54%
14	P-n55-k7	54	7	170	568	573	7	38.00	88	106.33	136	5	0.88%
15	P-n55-k10	54	10	115	694	695	10	36.87	131	131.00	131	1	0.14%
16	P-n55-k15**	54	15	70	989	**945**	**16**	43.41	62	62.00	62	-44	-4.45%
17	P-n60-k10	59	10	120	744	745	10	50.34	111	111.00	111	1	0.13%
18	P-n60-k15	59	15	80	968	971	15	55.79	158	158.00	158	3	0.31%
19	P-n65-k10	64	10	130	792	792	10	65.95	95	95.00	95	0	0.00%
20	P-n70-k10	69	10	135	827	834	10	83.41	198	198.00	198	7	0.85%
21	P-n76-k4	75	4	350	593	593	4	176.26	133	133.00	133	0	0.00%
22	P-n76-k5	75	5	280	627	630	5	133.08	77	147.00	217	3	0.48%
23	P-n101-k4	100	4	400	681	682	4	629.24	199	199.00	199	1	0.15%
	Average				587.4	586.0							
	(T-test) *P*-value (p < 0.05)					0.5094							

In this case, the *P*-value indicates there is no significant difference between the obtained result and BKS. The HCA found the best-known solution in approximately 43% (10 out of 23) of the instances, and near-optimal solutions (below 1% gap) in 76% of the remaining instances. In three instances (P-n22-k8, P-n50-k8, and P-n55-k15), HCA found lower-cost solutions than the best-known solution, but required one extra vehicle. Table 6-5 shows the evaluation results of dataset *E*.

Table 6-5. Results on dataset E

No.	Instance name	#C	#V	V.C	BKS	HCA	no v.	Avg. Time (s)	Min #	Avg. #	Max #	Diff	% Gap
1	E-n22-k4	21	4	6000	375	**375**	4	1.61	3	3.45	6	0	0.00%
2	E-n23-k3	22	3	4500	569	**569**	3	2.57	6	16.16	36	0	0.00%
3	E-n30-k3**	29	3	4500	534	503	**4**	4.55	5	27.77	75	-31	-5.81%
4	E-n33-k4	32	4	8000	835	**835**	4	4.35	24	24.00	24	0	0.00%
5	E-n51-k5	50	5	160	521	**521**	5	30.39	122	122.00	122	0	0.00%
6	E-n76-k7	75	7	220	682	683	7	125.79	217	217.00	217	1	0.15%
7	E-n76-k8	75	8	180	735	738	8	111.99	177	177.00	177	3	0.41%
8	E-n76-k10	75	10	140	830	839	**11**	108.68	131	131.00	131	9	1.08%
9	E-n76-k14	75	14	100	1021	1042	**15**	111.42	56	121.00	216	21	2.06%
10	E-n101-k8	100	8	200	815	825	8	377.32	88	88.00	88	10	1.23%
11	E-n101-k14	100	14	112	1067	1100	14	323.25	211	211.00	211	33	3.09%
Average					725.8	730.0							
(T-test) *P*-value						0.40003							

In dataset *E*, the *P*-value indicates there is no significant difference between the obtained result and BKS. the HCA found the best-known solution in approximately 36% (4 out of 11) of the instances, and near-optimal solutions (below 1% gap) in 28% of the remaining instances. In the E-n30-k3 instance, the HCA again reduced the cost but required one extra vehicle. The best-known solution for E-n30-k3 will be 503 if it is solved using four vehicles. Table 6-6 shows the evaluation results of dataset *F*.

Table 6-6. Results on dataset F

No.	Instance name	#C	#V	V.C	BKS	HCA	no v.	Avg. Time (s)	Min #	Avg. #	Max #	Diff	% Gap
1	F-n45-k4	44	4	2010	724	**724**	4	16.44	125	125.00	125	0	0.00%
2	F-n72-k4	71	4	30000	237	**237**	4	234.57	16	64.00	136	0	0.00%
3	F-n135-k7	134	7	2210	1162	1223	7	1382.34	116	116.00	116	61	5.25%
Average					707.7	728.0							
(T-test) P-Value						0.4226							

In dataset *F*, the HCA found the best-known solution in 2 out of 3 instances, with a small percentage gap in the third instance. The overall results show that the HCA can optimise the CVRP solutions in various instances from different categories. The HCA performed especially well in instances from category *P*, where the gap was always within 1% of the optimum.

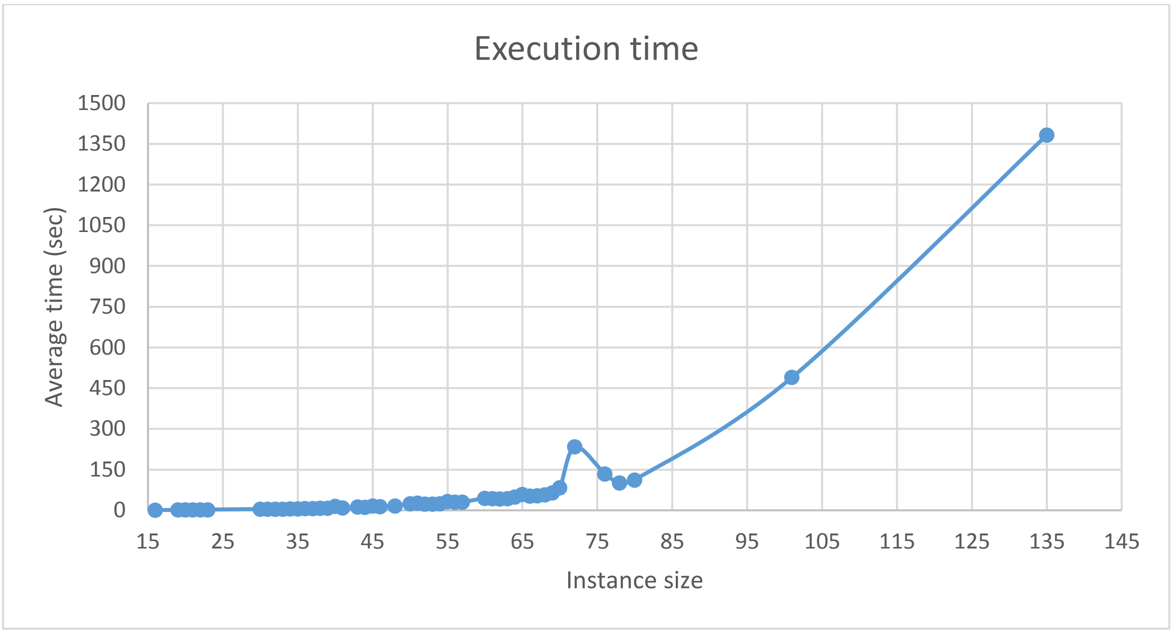


Figure 6-12. Execution time (averaged over all datasets) versus instance size

Figure 6-12 plots the execution time (averaged over all datasets) as a function of instance size. As the instance size increased, the average execution time increased at an acceptable rate. The maximum execution time was approximately 23 minutes for instance with 135 customers. Note that the total execution time includes the execution time of the improvement methods.

Table 6-7 shows the results of thirty-five CVRP instances from different datasets solved by the SA, IWD (Wedyan & Narayanan, 2014), and HCA algorithms. The best results are indicated in boldface.

Table 6-7. Comparison of SA, IWD, and HCA results on various CVRP instances

Instance name	# vehicle	Capacity	Best-known Results	SA		IWD		HCA	
				Cost	# vehicle	Cost	# vehicle	Cost	# vehicle
A-n32-k5	5	100	784	842	5	806	5	**784**	5
A-n33-k5	5	100	661	716	5	673	5	**661**	5
A-n33-k6	6	100	742	774	6	753	7	**742**	6
A-n36-k5	5	100	799	815	5	815	5	**779**	5
A-n37-k5	5	100	669	705	5	705	5	**669**	5
A-n61-k9	9	100	1034	1106	10	1103	10	**1044**	10
A-n80-k10	10	100	1763	1840	11	1889	10	**1822**	10
B-n35-k5	5	100	955	978	5	987	5	**955**	5
B-n38-k6	6	100	805	837	6	824	6	**805**	6
B-n41-k6	6	100	829	901	6	840	7	**829**	6
B-n43-k6	6	100	742	777	6	748	7	**742**	6
B-n50-k7	7	100	741	745	7	773	7	**741**	7
B-n57-k7	7	100	1153	1242	8	1230	8	**1145***	8
B-n66-K9	9	100	1316	1419	10	1381	10	**1329**	9
B-n78-k10	10	100	1221	1262	10	1311	10	**1250**	10
P-n16-k8	8	35	450	482	9	478	9	**450**	8
P-n19-k2	2	160	212	237	2	215	2	**212**	2
p-n21-k2	2	160	211	236	2	211	2	**211**	2
p-n20-k2	2	160	216	234	2	217	2	**216**	2
P-n22-k2	2	160	216	240	2	217	2	**216**	2
P-n22-k8	8	3000	590	590	10	660	9	**590**	9
P-n40-k5	5	140	458	516	5	480	5	**458**	5
P-n45-k5	5	150	510	569	5	532	5	**510**	5
P-n50-k7	7	150	554	583	7	573	7	**555**	7
P-n50-k8	8	120	631	650	9	655	9	**629***	9
P-n51-k10	10	80	741	784	11	780	11	**745**	10
E-n22-k4	4	6000	375	387	4	379	4	**375**	4
E-n23-k3	3	4500	569	621	3	570	3	**569**	3
E-n30-k3	3	4500	534	532	4	551	4	**503***	4
E-n33-k4	4	8000	835	841	4	851	4	**835**	4
E-n51-k5	5	160	521	582	5	538	6	**521**	5
E-n76-k10	10	140	830	896	11	904	11	**839**	11
F-n45-k4	4	2010	724	743	4	834	5	**724**	4
F-n72-k4	4	30000	237	254	4	251	5	**237**	4
F-n135-k7	7	2210	1162	**1204**	7	1243	7	1223	7

As evidenced in Table 6-7, the HCA outperformed the other algorithms in all instances except F-n135-k7. The HCA gave the best-known result in approximately 65% (23 out of 35) of the instances. The HCA results were statistically significant compared with the SA and IWD (*P*-values = 1.25E-09 and 9.04E-08 respectively) using T-test. In contrast, there is no significant difference between IWD and SA (*P*-value = 0.42398). In addition, the HCA optimised some instances with fewer vehicles than the IWD solutions, because its flow stage and complementary stages were properly designed. These tests inspire confidence that the algorithm phases will produce high-quality results in practical applications.

Although many solutions to the CVRP have been published in the literature, these solutions are not easily compared with the HCA solutions. First, this problem embraces a wide variety of dataset types; second, some researches have evaluated their algorithms on only a few instances, which invalidates any comparison. Nonetheless, the HCA results are initially compared with the results of the water flow algorithm (WFA) (Tran, 2011) and the water flow-like algorithm (WFLA) (Zainudin et al., 2015). The results of these algorithms are compared against the best-known results in Table 6-8, where the symbol "—" indicates that no records were found.

Table 6-8. Results of HCA, WFA, and WFLA

Instance name	BKS	HCA	WFA	WFLA
E-n13-k4	247	—	**247**	—
E-n22-k4	375	**375**	**375**	375.28
E-n23-k3	569	**569**	**569**	—
E-n30-k3	534	503*	**534**	568.56
E-n33-k4	835	**835**	**835**	837.67
E-n51-k5	521	**521**	**521**	545.24
E-n76-k7	682	**683**	687	—
E-n76-k8	735	**738**	—	774.41
E-n76-k10	830	**839**	—	876.78
E-n101-k8	815	**825**	853	878.55
E-n101-k14	1067	**1100**	—	1144.11
F-n45-k4	724	**724**	**724**	—
F-n72-k4	237	**237**	241	—
Mean	628.5	662.4	558.6	750.1

Although the WFA and WFLA algorithms were not extensively tested on CVRP instances, the comparisons confirm the superior quality of HCA. There is a significant difference between HCA results and WFLA results (*P*-value = 0.00478) using T-test. Although there is no significant difference between HCA results and WFA results (*P*-value = 0.11524), the high quality of the WFA solution was conferred mainly through using LINGO software, which provides an initial solution for the WFA optimisation.

Table 6-9 compares the results of HCA and some other heuristic methods, namely, the enhanced sweep nearest algorithm (SWNA) (Na, Jun, & Kim, 2011) (column 4), the centroid-based 3-phase (CB3P) algorithm (column 5), a sweep algorithm with cluster adjustment (SCA) (column 6), an electrostatic-based algorithm called ALGELECT (Faulin & del Valle, 2008) (column 7), a heuristic algorithm based on tabu search (TS) and adaptive large neighbourhood search (TALNS) (Kir, Yazgan, & Tüncel, 2017) (column 8), and a simulated annealing (SA) algorithm (Harmanani et al., 2011) (column 9). CB3P is a three-phase heuristic algorithm that constructs clusters, adjusts them, and establishes the routes. The values in columns 5 and 6 are taken from (Shin & Han, 2011). The results in Table 6-9 are collated from reported instances. The best results among these algorithms are highlighted in boldface.

Table 6-9. Comparison of results by HCA and other heuristic methods

Instance	**BKS**	**HCA**		**SWNA [4]**	**CB3P [5]**		**SCA [6]**		**ALGELECT [7]**		**TALNS [8]**	**SA [9]**
		Cost	# V		Cost	# V	Cost	# V	Cost	# V		
A-n32-k5	784	**784**	5	810	881	5	872	5	1032	5	**784**	791.0789
A-n33-k5	661	**661**	5	686	728	5	788	5	789	5	**661**	685.577
A-n33-k6	742	**742**	6	743	770	6	829	7	834	7	**742**	748.08
A-n34-k5	778	**778**	5	785	812	5	852	6	835	5	**778**	802.614
A-n36-k5	799	**799**	5	826	814	5	884	5	908	5	**799**	837.477
A-n37-k5	669	**669**	5	670	756	5	734	5	783	5	**669**	697.943
A-n37-k6	949	**949**	6	962	1027	7	1050	1	1046	6	**949**	963.995
A-n38-k5	730	**730**	5	749	819	6	874	6	861	6	**730**	733.945
A-n39-k5	822	**822**	5	—	864	5	971	6	990	5	**822**	848.322
A-n39-k6	831	**831**	6	856	881	6	966	6	861	6	**831**	859.147
A-n44-k6	937	**937**	6	957	1037	7	1092	7	1028	6	939	951.802
A-n45-k6	944	**948**	6	991	1040	7	1043	7	1040	7	955	982.829
A-n45-k7	1146	**1151**	7	1173	1288	7	1281	7	1258	7	1153	1199.50
A-n46-k7	914	**914**	7	946	992	7	1013	7	1062	7	915	—
A-n48-k7	1073	1074	7	1113	1145	7	1143	7	1162	7	**1073**	—
A-n53-k7	1010	**1017**	7	—	1117	8	1116	8	1191	7	1026	1044.91
A-n54-k7	1167	1300	9	—	1209	8	1320	8	1291	7	**1169**	1192.69
A-n55-k9	1073	**1074**	9	1095	1155	10	1192	10	1191	9	**1074**	1144.85
A-n60-k9	1354	**1361**	9	1420	1430	9	1574	9	1503	9	1366	1455.63
A-n61-k9	1034	**1044**	10	1100	1201	11	1184	11	1181	10	1045	1064.76
A-n62-k8	1288	1322	8	1359	1470	9	1559	9	1408	8	**1302**	1484.81
A-n63-k9	1616	**1641**	9	1712	166	10	1823	10	1745	9	1644	1698.63
A-n63-k10	1314	**1324**	10	1386	1405	11	1523	11	1409	10	1325	—
A-n64-k9	1401	**1421**	9	1499	1587	10	1597	10	1521	9	1442	—
A-n65-k9	1174	**1181**	9	1223	1276	10	1351	10	1357	9	1189	—
A-n69-k9	1159	1170	9	1207	1283	10	1254	10	1389	9	**1169**	—
A-n80-k10	1763	1822	10	1866	1883	11	2014	11	1901	10	**1790**	—
B-n31-k5	672	**672**	5	677	700	5	713	5	—	—	—	—
B-n34-k5	788	**788**	5	802	851	6	845	6	—	—	—	—
B-n35-k5	955	**955**	5	962	969	5	1002	5	—	—	—	—
B-n38-k6	805	**805**	6	817	834	6	863	6	—	—	—	—
B-n39-k5	549	**549**	5	575	620	5	560	5	—	—	—	—
B-n41-k6	829	**829**	6	843	862	7	881	6	—	—	—	—
B-n43-k6	742	**742**	6	746	857	6	812	6	—	—	—	—
B-n44-k7	909	**916**	7	942	963	7	1097	8	—	—	—	—
B-n45-k5	751	**751**	5	797	807	6	803	6	—	—	—	—
B-n45-k6	678	**691**	6	732	743	7	756	7	—	—	—	—
B-n50-k7	741	**741**	7	779	772	7	763	7	—	—	—	—
B-n50-k8	1312	**1326**	8	1349	1431	9	1446	8	—	—	—	—
B-n52-k7	747	**748**	7	758	754	7	765	7	—	—	—	—
B-n56-k7	707	**712**	7	726	741	7	832	7	—	—	—	—
B-n57-k9	1598	**1612**	9	1642	1673	9	1807	9	—	—	—	—
B-n64-k9	861	**881**	9	1161	910	10	1023	10	—	—	—	—
B-n66-k9	1316	**1329**	9	1363	1468	10	1483	10	—	—	—	—
B-n67-k10	1032	**1059**	10	1080	1108	10	1134	11	—	—	—	—
B-n68-k9	1272	**1291**	9	1308	1338	9	1362	9	—	—	—	—
B-n78-k10	1221	**1250**	10	1268	1276	10	1479	11	—	—	—	—
P-n16-k8	450	**450**	8	513	497	9	568	10	—	—	—	—
P-n19-k2	212	**212**	2	219	256	3	236	2	—	—	—	—
P-n20-k2	216	**216**	2	217	240	2	238	2	—	—	—	—
P-n21-k2	211	**211**	2	211	240	2	238	2	—	—	—	—
P-n22-k2	216	**216**	2	216	245	2	237	2	—	—	—	—
P-n22-k8	603	**590**	9	560*	672	10	687	10	—	—	—	—
P-n23-k8	529	**529**	8	554	703	12	645	11	—	—	—	—
P-n40-k5	458	**458**	5	467	505	5	499	5	—	—	—	—
P-n45-k5	510	**510**	5	—	533	5	525	5	—	—	—	—
P-n50-k7	554	**555**	7	—	583	7	585	7	—	—	—	—
P-n50-k8	631	**629**	9	—	669	9	675	9	—	—	—	—
P-n50-k10	696	**697**	10	—	740	11	779	11	—	—	—	—

Instance	**BKS**	**HCA**		**SWNA [4]**	**CB3P [5]**		**SCA [6]**		**ALGELECT [7]**		**TALNS [8]**	**SA [9]**
		Cost	# V		Cost	# V	Cost	# V	Cost	# V		
P-n51-k10	741	**745**	10	—	779	11	806	11	—	—	—	—
P-n55-k7	568	**573**	7	—	610	7	611	7	—	—	—	—
P-n55-k10	694	**945**	16	—	749	10	763	10	—	—	—	—
P-n60-k10	744	**745**	10	—	786	11	823	11	—	—	—	—
P-n60-k15	968	**971**	15	—	1006	16	1086	16	—	—	—	—
P-n65-k10	792	**792**	10	—	836	10	856	11	—	—	—	—
P-n70-k10	827	**834**	10	—	891	11	902	11	—	—	—	—
P-n76-k4	593	**593**	4	612	685	4	603	4	—	—	—	—
P-n76-k5	627	**630**	5	—	737	5	647	5	—	—	—	—
P-n101-k4	681	**682**	4	715	698	4	702	4	—	—	—	—
E-n22-k4	375	**375**	4	**375**	—	—	—	—	—	—	—	—
E-n23-k3	569	**569**	3	**569**	—	—	—	—	—	—	—	—
E-n30-k3	534	**503***	4	543	—	—	—	—	—	—	—	—
E-n33-k4	835	**835**	4	852	—	—	—	—	—	—	—	—
E-n51-k5	521	**521**	5	532	—	—	—	—	—	—	—	555.14
E-n76-k7	682	**683**	7	703	—	—	—	—	—	—	—	—
E-n76-k8	735	**738**	8	746	—	—	—	—	—	—	—	—
E-n76-k10	830	**839**	11	907	—	—	—	—	—	—	—	910.32
E-n76-k14	1021	**1042**	15	1072	—	—	—	—	—	—	—	—
E-n101-k8	817	**825**	8	850	—	—	—	—	—	—	—	1051.34
E-n101-k14	1071	**1100**	14	1152	—	—	—	—	—	—	—	—
F-n45-k4	721	**724**	4	750	—	—	—	—	—	—	—	—
F-n72-k4	237	**237**	4	241	—	—	—	—	—	—	—	—
F-n135-k7	1162	1223	7	**1221**	—	—	—	—	—	—	—	—
Average	834	845		**886**	904		957		1169		1050	987

As seen in the table, the HCA outperformed the SWNA, SCA, ALGELECT, and SA algorithms and very competitive with CB3P and TALNS. The HCA and TALNS provided the best-known results in 12 and 11 out of 27 instances, respectively. In nine of the remaining instances, HCA outperformed TALNS, while in six of the remaining instances, TALNS yielded the best solutions. Table 6-10 shows the *P*-values for the statistical analysis using T-test.

Table 6-10. The P-values between the algorithms results (based on Table 6-9)

Algorithms	**T-test (*P*-value)**
HCA- SWNA	3.03E-08
HCA- CB3P	0.1251
HCA- SCA	1.18E-16
HCA- ALGELECT	1.68E-11
HCA- TALNS	0.3778
HCA-SA	0.00359

Table *6-11* compares the results and the average execution time in HCA and other nature-inspired algorithms, namely, the cuckoo search algorithm (CS) (Alssager & Othman, 2016), a hybrid algorithm based on ACO and PSO (PACO) (Kao, Chen, & Huang, 2012), a hybrid discrete PSO algorithm with SA (DPSO–SA), a genetic algorithm (GA) with 2-Opt results (A. Chen et al., 2006), PSO with two-solution representations (PSO–SR2) (Ai & Kachitvichyanukul, 2009).

Table 6-11. Comparison of CVRP results by HCA and various nature-inspired algorithms

Instance name	**BKS**	**HCA**		**CS [1]**		**SA from CS [2]**		**PACO [3]**		**DPSO-SA [4]**		**GA with 2-opt [5]**		**PSO-SR2 [6]**	
		Cost	**T (s)**	**Cost**	**T (s)**	**Cost**	**T (s)**	**Cost**	**T (s)**	**Cost**	**T (s)**	**Cost**	**T (s)**	**Cost**	**T(s)**
A-n33-k5	661	**661**	5.16	688	3.391	**661**	38.2	**661**	0.12	**661**	32.2	**661**	39.6	**661**	13
A-n46-k7	914	**914**	13.89	973	8.782	931	143.8	**914**	0.16	**914**	128.9	928	136	**914**	23
A-n60-k9	1354	1464	35.89	1414	12.72	1363	286.3	**1354**	14.15	**1354**	308.8	1360	296	1355	40
B-n35-k5	955	**955**	5.85	962	5.31	960	58.4	**955**	0.06	**955**	37.6	**955**	46.9	**955**	14
B-n45-k5	751	**751**	14.29	770	5.62	760	123.5	**751**	1.12	**751**	134.2	762	129	**751**	20
B-n68-k9	1272	1291	57.59	1320	24.44	1298	409.2	1275	87.19	**1272**	344.2	1296	396	1274	50
B-n78-k10	1221	1250	100.86	1284	27.18	1256	483.3	**1221**	54.38	1239	429.4	1248	—	1223	64
P-n76-k4	593	**593**	176.26	679	66.65	612	489.6	**593**	8.61	602	496.3	605	528	594	48
P-n101-k4	681	**682**	629.24	758	167.5	715	1964.9	683	25.7	694	977.5	706	1213	683	86
E-n30-k3	534	**503***	4.55	565	6.67	534	69.3	534	0.05	534	28.4	534	30.5	534	16
E-n51-k5	521	**521**	30.39	590	14.38	541	362.4	**521**	0.51	528	28.4	531	290	**521**	22
E-n76-k7	682	**683**	125.79	746	56.62	704	619.3	685	18.95	688	526.5	697	499	**682**	60
F-n72-k4	237	**237**	234.57	255	80.41	253	604.6	**237**	6.28	244	398.3	246	469	**237**	53
F-n135-k7	1162	1223	1382.34	1301	1166	1243	2533.9	**1170**	246.85	1215	1526.3	1246	1894	1162	258
Average	824.14	863.46	201.19	878.93	117.55	845.07	584.76	825.29	33.15	832.21	385.50	841.07	459.00	824.71	54.79

In finding the best-known solution, HCA outperformed CS and competed well with the other algorithms. In terms of execution time, HCA was outperformed by PACO, PSO–SR2, and CS but was more computationally efficient than SA, DPSO–SA, and GA.

Table 6-12 shows the *P*-values for the statistical analysis by comparing the cost and time of the algorithms using T-test.

Table 6-12. The P-values between the algorithms results and execution time (based on Table 6 11)

Algorithms	**T-test (*P*-values)**	
	Cost	**Time**
HCA- CS	0.00103	0.02856
HCA- SA	0.41396	0.00293
HCA- PACO	0.19152	0.07338
HCA- DPSO	0.53464	0.00026
HCA- GA with 2-opt	0.70394	0.00030
HCA- PSO-SR2	0.17867	0.10866

Table 6-13 compares the results of HCA and other metaheuristic algorithms (Mazidi, Fakhrahmad, & Sadreddini, 2016). These authors solved the CVRP by seven metaheuristic algorithms (SA, stochastic hill climbing (SHC), ACO, TS, PSO, and an improved GA). Their improved GA combines the GA and ACO to improve the performance of the original GA. The algorithms were tested on a number of CVRP instances (Table 6-13).

Table 6-13. Comparison of results by HCA, improved GA and other algorithms

Instance name	**BKS**	**SA**	**SHC**	**AC**	**TS**	**PSO**	**GA**	**Improved GA**	**HCA**
A-n32-k5	784	787	787	787	787	790	787	787	**784**
A-n45-k7	1146	1167	1163	1155	1151	1157	1150	**1148**	1151
A-n55-k9	1073	1098	1102	1085	1075	1085	1075	1075	**1074**
A-n61-k9	1034	1094	1167	1170	1045	1110	1065	1050	**1040**
A-n63-k9	1616	1663	1665	1680	1640	1700	1655	1650	**1641**
A-n80-k10	1763	1868	1870	1690	1825	1830	1820	1814	**1822**
P-n19-k2	212	212	212	212	212	220	212	212	**212**
P-n45-k5	510	542	545	515	530	540	514	514	**510**
P-n55-k15	989	1025	1030	1250	1150	1174	1050	1020	**945***
P-n101-k4	681	892	915	800	990	980	725	725	**682**
Average	980.8	1034.8	1045.6	1034.4	1040.5	1058.6	1005.3	999.5	986.1
T-test (P-values)		**0.0339**	**0.0289**	**0.2179**	**0.1503**	**0.0559**	**0.1025**	**0.13399**	**Vs. HCA**

The HCA outperformed the improved GA in 9 out of 10 instances, and outperformed all of the other algorithms. Although there are no significant differences between HCA and AC, TS, PSO, GA and IGA, but HCA has the least average among the other algorithms. Table 6-14 compares the results of HCA and another GA, the fitness aggregated GA (FAGA), taken from (V. S. Kumar, Thansekhar, & Saravanan, 2014). The best results are highlighted in boldface.

Table 6-14. Comparison of results by HCA and FAGA

No	Instance name	BKS	HCA		FAGA	
			Cost	# V	Cost	# V
1	A-n32-k5	784	**784**	5	804	5
2	A-n33-k5	661	661	5	661	5
3	A-n39-k5	822	**822**	5	839	5
4	A-n45-k6	944	**948**	6	957	6
5	A-n48-k7	1073	**1074**	7	1101	7
6	A-n55-k9	1073	**1074**	7	1089	9
7	A-n62-k8	1288	1322	9	**1288**	8
8	A-n64-k9	1401	1421	9	**1401**	9
9	A-n80-k10	1763	1822	10	**1777**	10
10	B-n31-k5	672	672	5	672	5
11	B-n35-k5	955	**955**	5	968	5
12	B-n39-k5	549	549	5	549	5
13	B-n45-k5	751	751	5	751	5
14	B-n50-k7	741	741	7	741	7
15	B-n56-k7	707	**712**	7	721	7
16	B-n57-k9	1598	1612	9	**1598**	9
17	B-n64-k9	861	881	9	**861**	9
18	B-n78-k10	1221	1250	10	**1239**	10
19	P-n19-k2	212	212	2	212	2
20	P-n20-k2	216	216	2	216	2
21	P-n23-k8	529	529	8	529	8
22	P-n45-k5	510	510	5	510	5
23	P-n50-k8	631	**629***	9	648	8
24	P-n51-k10	741	**745**	10	760	10
25	P-n60-k10	744	745	10	**744**	10
26	P-n65-k10	792	**792**	10	806	10
Average		855.3	860.9		863.2	
T-test *P*-values (Vs BKS)			0.00812		0.00018	
T-test P-value vs. HCA					0.88053	

Although both algorithms reached the best-known solution in some instances, HCA outperformed FAGA in nine instances, whereas FAGA outperformed HCA in eight instances. The two algorithms are very competitive and there are no significant differences. Finally, Table 6-15 compares the results of HCA and GA on sixteen CVRP instances. Here, the GA results were taken from (Najera, 2007).

Table 6-15. Comparison of results by HCA and GA

No	Instance name	BKS	HCA	GA
1	A-n32-k5	784	**784**	787
2	A-n38-k5	730	**730**	735
3	A-n54-k7	1167	1300	**1196**
4	A-n60-k9	1354	**1361**	1372
5	A-n69-k9	1159	**1170**	1190
6	A-n80-k10	1763	**1822**	1875
7	B-n38-k6	805	**805**	809
8	B-n41-k6	829	**829**	835
9	B-n45-k5	751	**751**	755
10	B-n57-k7	1153	1145	**1143**
11	B-n78-k10	1221	**1250**	1264
12	E-n22-k4	375	**375**	375
13	E-n76-k7	682	**683**	703
14	E-n76-k8	735	**738**	771
15	E-n76-k10	830	**839**	874
16	E-n76-k14	1021	**1042**	1052
Average		959.9	976.5	983.5
(T-test) Vs BKS (*P*-value)			0.07793	0.00518
(T-test) Vs. HCA (p-value)				0.41150

As confirmed in Table 6-15, although there are no significant differences between the algorithms, but the HCA outperformed the GA in all instances except A-n54-k7 and B-n57-k7.

In conclusion, based on the comparisons between the HCA and other algorithms, it can be inferred that HCA is a promising alternative optimisation tool for the CVRP problem. In addition, different implementations of the same algorithm may give results of different quality and performance. Therefore, high-quality solutions to the CVRP require an appropriate algorithm design, correct implementation techniques, and strong programming skills. The topology or distribution of the customers and the depot position also affect the solution quality, as they increase the complexity of the problem. For this reason, understanding why an algorithm cannot provide the best-known solution in a particular instance is a nontrivial process. Moreover, CVRP instances have many locally optimal solutions, which are difficult to escape in some algorithms. Therefore, an adequate balance between exploration and exploitation will lead to better results.

6.3.2.3 The HCA convergence on CVRP

The following figures show the convergence speed of the algorithm and the best-known solution in instances from different categories. Figure 6-13 plots the convergence of the HCA on the A-n32-k5 instance.

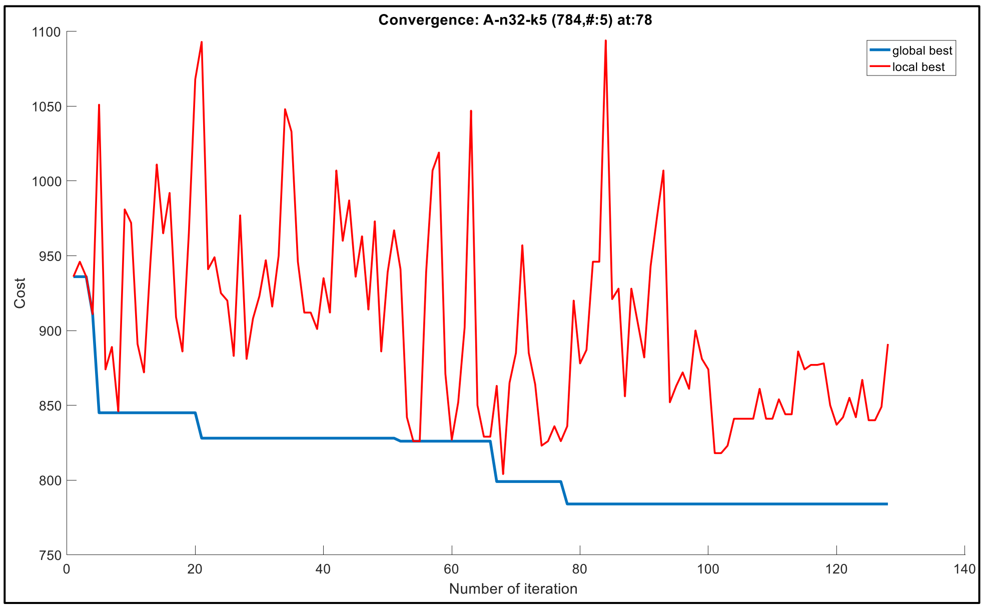


Figure 6-13. HCA convergence on A-n32-k5

The HCA reached the best-known solution after 78 iterations. Thereafter, the algorithm continued to generate different solutions in each iteration without becoming trapped in local optima. Figure 6-14 shows the best-known solution in the A-n32-k5 instance.

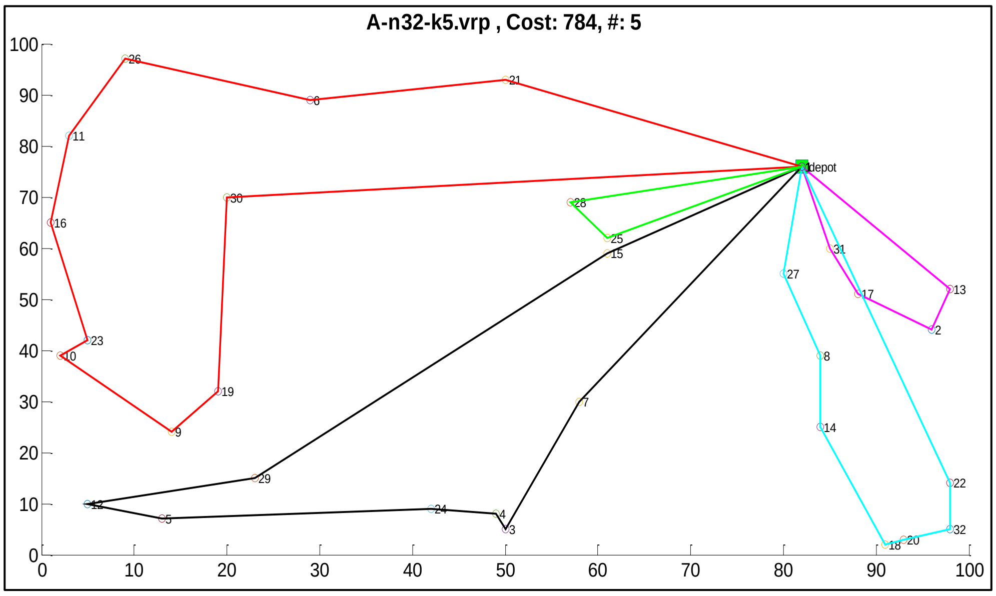


Figure 6-14. Best-known solution on A-n32-k5 (cost = 784)

Figure 6-15 shows the convergence of the HCA in the B-n50-k7 instance, where the best-known solution was reached after 143 iterations.

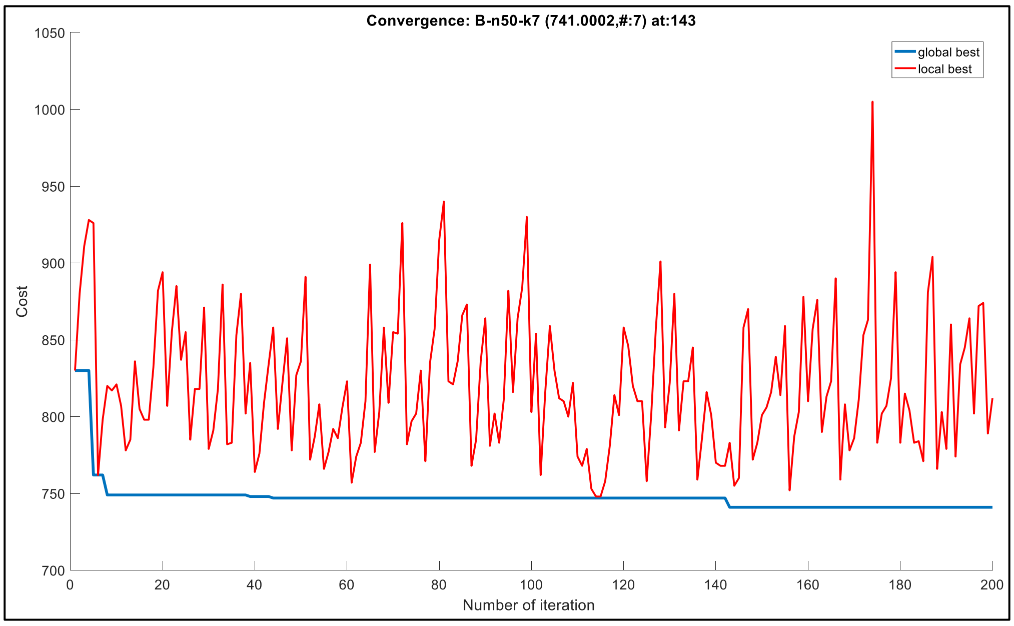


Figure 6-15. HCA convergence on B-n50-k7

Figure 6-16 shows the best-known solution in the B-n50-k5 instance, with a cost of 741.

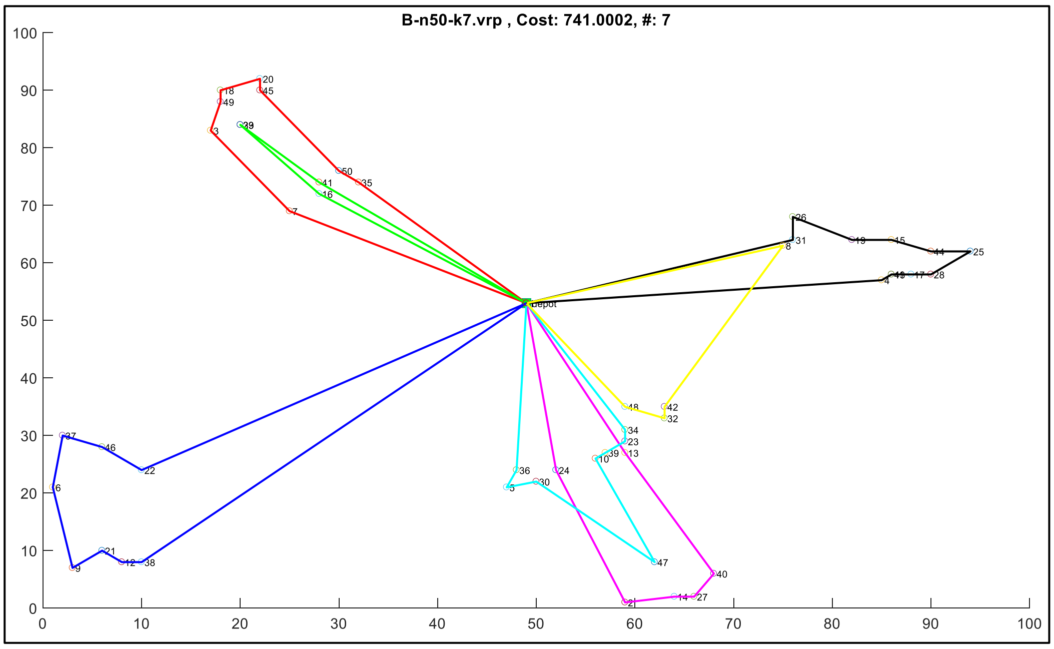


Figure 6-16. The best-known solution for B-n50-k7

Figure 6-17 shows the solution in the B-n51-k7 instance. The cost equals 1016, which is less than the best-known solution, and an extra vehicle is required.

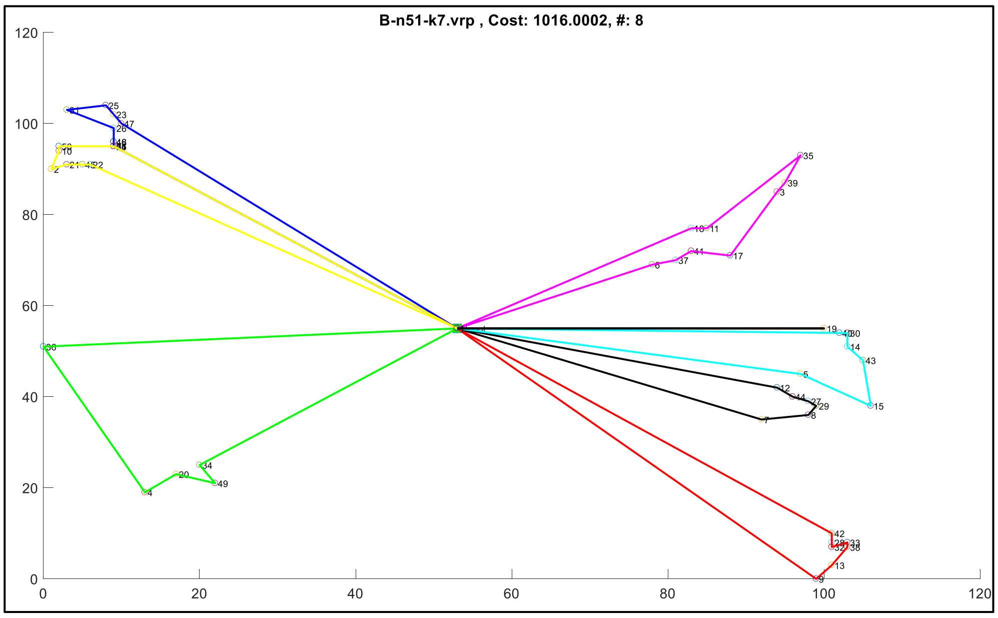


Figure 6-17. A solution on B-n51-k7

Figure 6-18 shows the convergence of the HCA in the P-n55-k15 instance, where the best-known solution was reached after 210 iterations.

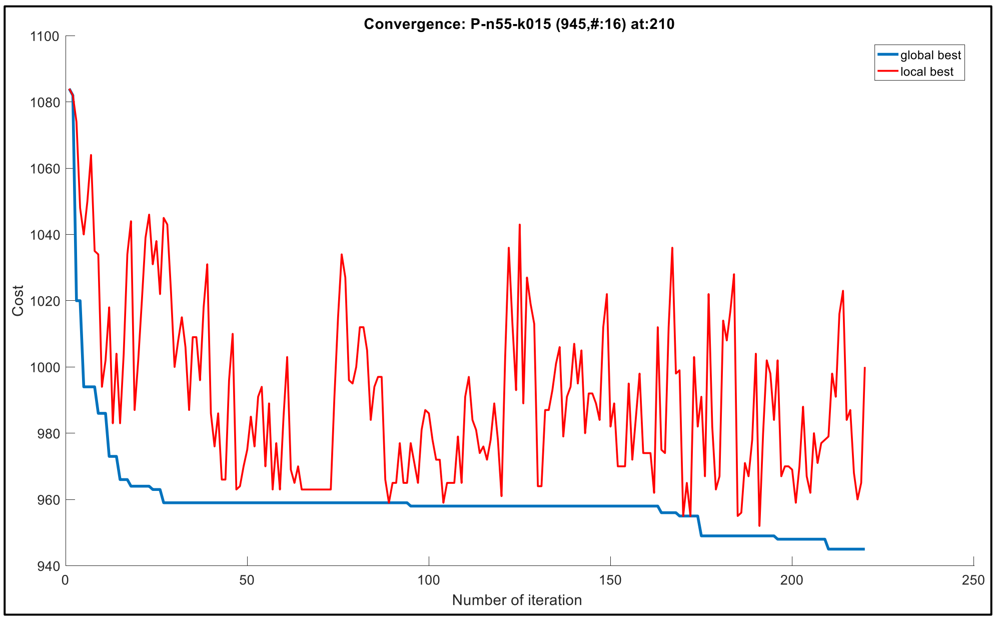


Figure 6-18. HCA convergence on P-n55-k15

Figure 6-19 shows the solution in the P-n55-k15 instance. The cost is less than the best-known (equalling 945), and an extra vehicle is required.

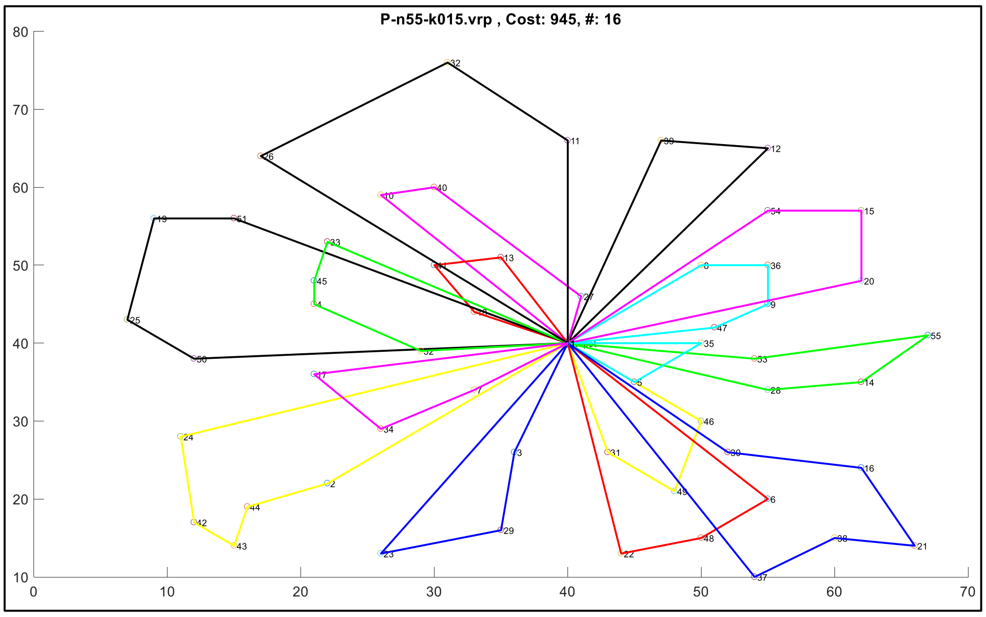


Figure 6-19. A solution on P-n55-k15 (cost = 945)

Figure 6-20 shows the convergence of HCA in the E-n51-k5 instance, where the best-known solution was reached after 174 iterations.

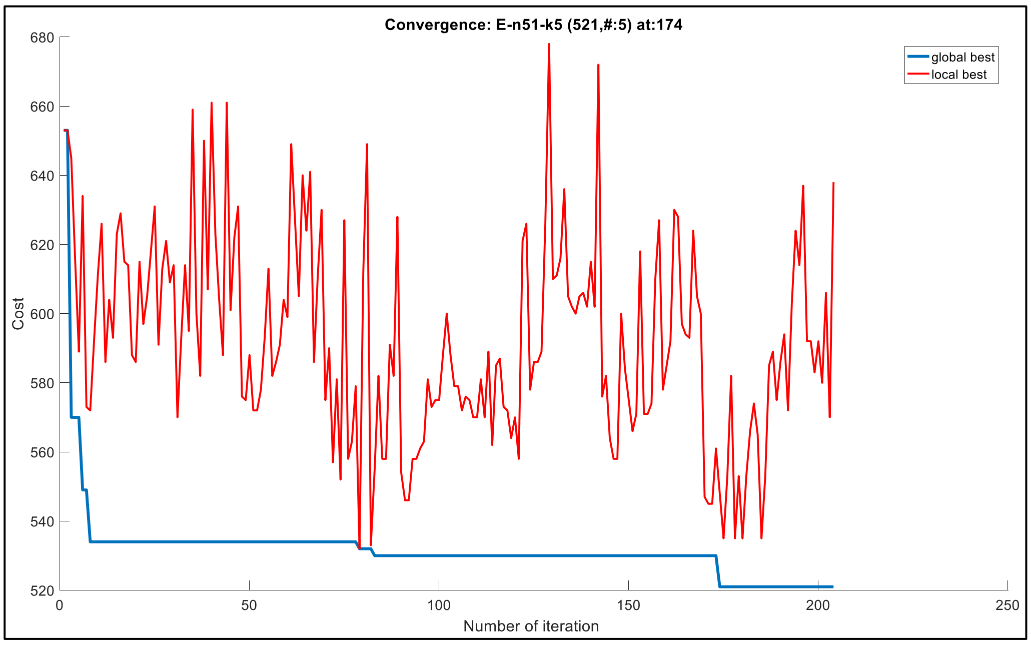


Figure 6-20. HCA convergence on E-n51-k5

Figure 6-21 shows the best-known solution in the E-n51-k5 instance, with a cost of 521.

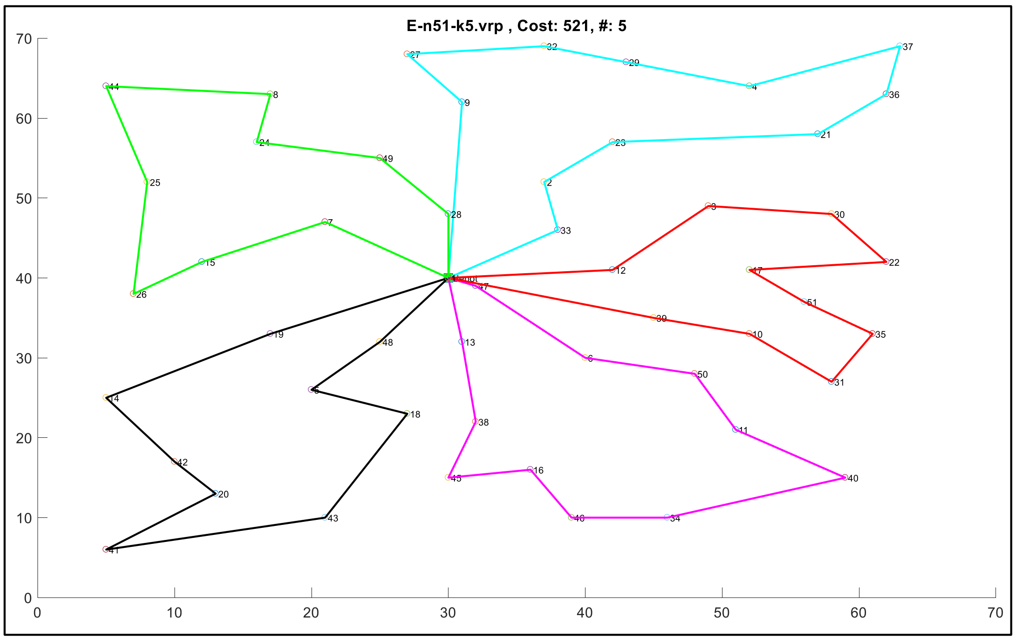


Figure 6-21. Best-known solution on E-n51-k5 (cost = 521)

Figure 6-22 shows the convergence of the HCA in the F-n72-k4 instance. Here, the algorithm reached the best-known solution after 186 iterations.

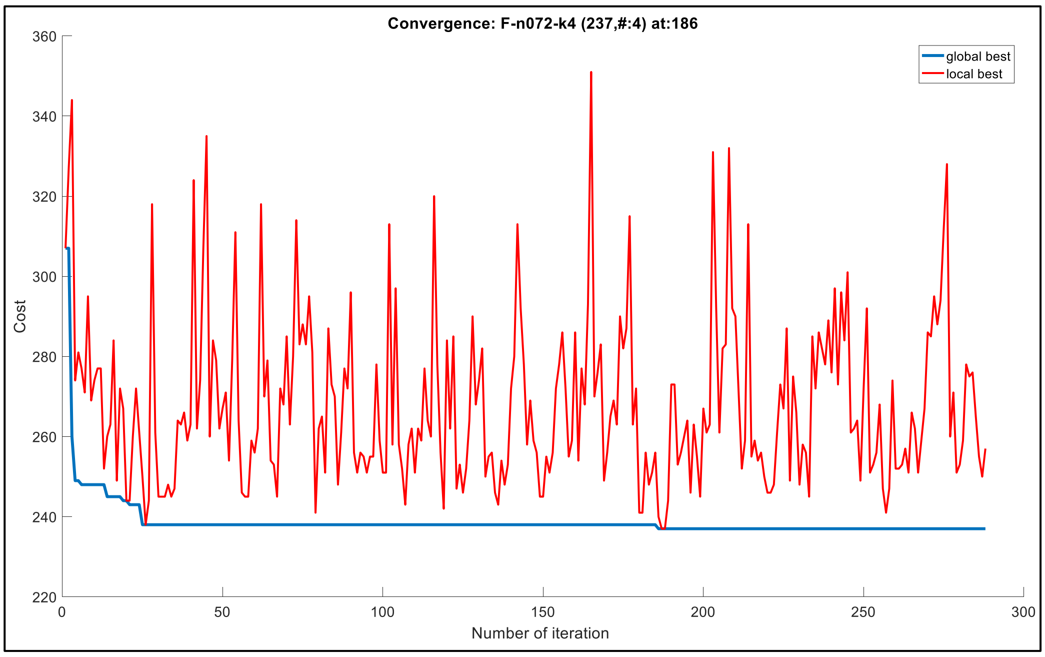


Figure 6-22. HCA convergence on F-n72-k4

In Figure 6-22, the HCA reached the best-known solution after 186 iterations and the search never stagnated. Figure 6-23 shows the best-known solution in the F-n72-k4 instance with a cost of 237.

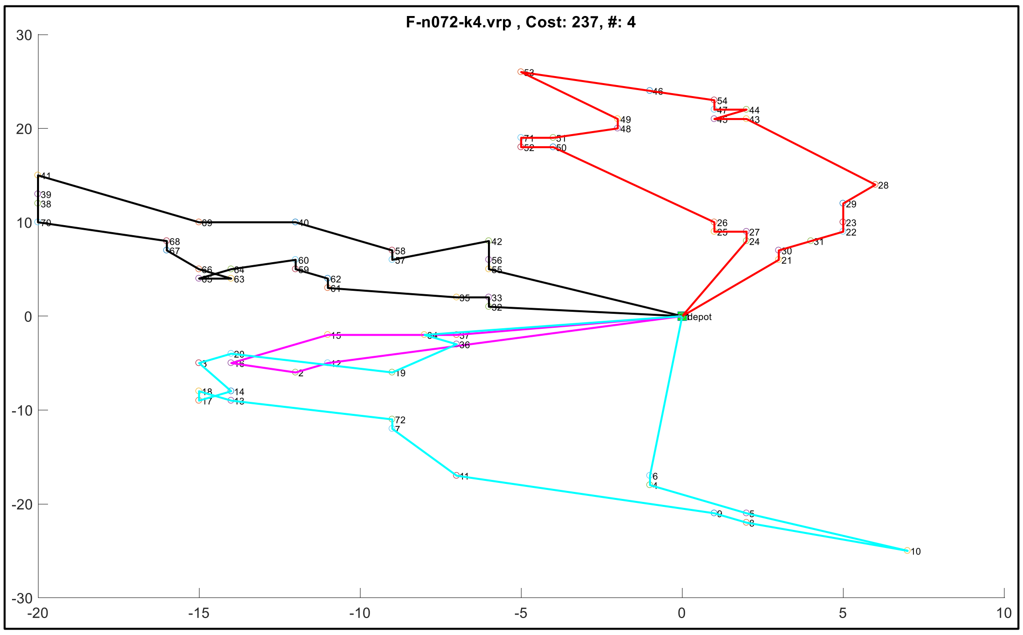


Figure 6-23. Best-known solution on F-n72-k4 (cost = 237)

The above convergence plots demonstrate that the algorithm can generate different solutions in every iteration. This property is conferred by the depth factor, which operationally is the direct opposite of the soil amount on the paths. Therefore, these experiments provide further confirmation that the HCA explores different areas of the solution space while also focusing on promising areas. Furthermore, the information sharing that occurs in the condensation phase, represented by the improvement methods, crucially enhances the obtained solutions and reduces the required number of iterations to reach the best-known solution.

6.4 Conclusion

This chapter reported on experiments where HCA was applied to CVRP a well-known routing problem. The algorithm performance was evaluated on two types of CVRP instances. The experimental results demonstrate that the HCA can find high-quality solutions for the CVRP. Furthermore, HCA was found to be competitive with other algorithms, especially water-based algorithms. In many instances, HCA outcompeted the other algorithms because it utilises other phases of the water cycle (i.e., evaporation and condensation) that largely influence the efficiency of problem-solving. These phases help the algorithm to escape local optima by exploiting and exploring stronger solutions; therefore, they improve the solution quality. The evaporation stage prevents the algorithm from sticking at the same solution in successive iterations. Furthermore, activating the information sharing at the condensation phase using certain local improvement methods enhanced the obtained solutions while reducing the required number of iteration.

More experiments are needed on other categories with large numbers of customers to guarantee the performance robustness of the HCA. In addition, the influence of the parameter values on the generated solutions should be investigated. Such parameter tuning will enhance the efficiency and accuracy of the algorithm. Finally, the model should be tested and extended on experiments with additional constraints, such as multi-depot problems, heterogeneous vehicles, time windows, and delivery and pickup demand.

Due to the nature of the CVRP, it can be extended to represent a problem with a dynamic environment (like, DCVRP). Therefore, the next chapter address how to solve DCVRP with HCA.

Chapter 7 Solving the Dynamic Capacitated Vehicle Routing Problem using HCA

"We cannot solve our problems with the same level of thinking that created them."

[Albert Einstein]

7.1 Introduction

In Chapter 6, we explained how HCA solves the static CVRP. This chapter assesses the performance of HCA on discrete dynamic optimisation problems (DOPs). In the last few decades, the increased focus on the realism of optimisation problems has paved the way for new dynamic variants of classical optimisation problems. Owing to the temporally dynamic nature of many real world problems, DOPs have become a popular and challenging topic of study in operations research (Nguyen, Yang, & Branke, 2012). DOPs can naturally model many problems whose inputs change over time. Such temporal changes influence the objective functions or the constraints. Therefore, DOPs require techniques that are able to optimise their objective functions with time-dynamic variables. Solving a DOP requires two major steps: identifying the temporal changes in the problem and finding a new better solution.

DOPs are typically solved by handling the variable changes and optimising the last solution based on the newly available information. To obtain a high-quality solution, the problem statuses before and after these changes must be connected. If the changes are not interrelated, then the algorithm needs to completely reset and solve the problem from scratch. If the changes in the problem are relatively small, periodic, or recurrent, reusing the good properties of previously found solutions will reduce the computational time. Like conventional static optimisation problems, the No-Free-Lunch theorems support DOPs (Wolpert & Macready, 1997).

Dynamic CVRPs (DCVRPs) are typical examples of DOPs. The DCVRP, also known as *online* or *real-time* CVRP, extends the VRP to time-varying problems. In a DCVRP, at least one portion of the input changes as time passes (i.e., one or more parameters are time-dependent). DCVRP differs from static CVRP mainly by the occurrence of events over time. In other words, the relevant information in a DCVRP is not completely available to the planner before the routing process, or may change while executing the distribution. In contrast, all information in a static CVRP is known in advance and remains unchanged during the execution. Therefore, when scheduling a DCVRP, the routs must be continuously evolved and re-optimised.

Furthermore, the DCVRP is subjected to the traditional constraints of the conventional VRP (e.g., limited vehicle capacity). The DCVRP can be regarded as a set of time-dependent CVRPs. Therefore, like the conventional static CVRP, the DCVRP is NP-hard and its optimal solutions may not be found within a reasonable computational time. DCRVPs model many real-life problems such as routing problems, transportation systems, courier mail services, taxicab services, Uber services, distribution of oil, and emergency systems. In all of these examples, the service can be triggered by different kinds of events.

The vehicle rescheduling problems (VRSPs) can be considered to be a special class of DCVRP. In a VRSP, the schedule needs to be updated immediately due to unavoidable dynamic events that affect the continuity of the current distribution schedule. Events such as vehicle break down or road closure. The recovery schedule should ideally minimise the deviation costs and disruption rate from the original schedule.

This chapter examines the performance of HCA under different circumstances (i.e., in dynamic environments). An HCA-based framework is developed that continuously updates the DCVRP solution with the arrival of new customers. The HCA framework is further tested with a road closure event in the VRSP. Different experiments are conducted to determine whether HCA can be generalised to typical DCVRP problems. Additionally, based on these experiments, the relationship between the HCA performance and degree of dynamism (i.e., problem complexity) is elucidated. Finally, an investigation is presented that determines whether or not the degree of dynamism is related to the number of time slices.

In many dynamic systems or problems, changing inputs must be handled in the best possible way. To maintain balance, natural systems must similarly adapt to unexpected environmental changes or events caused by human actions. The hydrological water cycle is a natural dynamic system in which water movement is affected and triggered by changes in factors at various stages. The water cycle is maintained by adaption or counter-response to these changes. For example, water evaporates when the temperature increases, and flows as result of a precipitation event. Based on this concept, the DCVRP seeks a suitable and efficient framework in which the events in the problem resemble the changes in the water cycle. To this end, we use soil and depth heuristics to propagate or preserve the good properties of previous tentative solutions between time slices, and to enhance the exploitation of new information.

This chapter is organised as follows. Section 7.2 introduces the DCVRP and briefly describes its solution approaches before formulating the DCVRP. Section 7.3 customises the HCA for solving the DCVRP. The experimental results and performance comparisons with other algorithms are shown in Section 7.4. Conclusions are presented in Section 7.5.

7.2 Dynamic Capacitated Vehicle Routing Problem

In the last few decades, the number of research papers on DCVRPs has increased because many distribution companies must solve complex problems in real-time. In a DCVRP, different types of disruptions often occur after planning or while executing a logistic scheduling. Unexpected disruptions include the appearance of new customers (i.e., requests for new orders), order cancelliations, changes in location, road congestion or closure due to accidents (i.e., altered travelling time), changes in the service time-window, vehicle breakdowns (i.e., reduced vehicle availability), spilt or damaged cargoes caused by accidents, and changes in the demands of known customers (i.e., decreases or increases in the order quantities) (Álvarez, Díaz, & Osaba, 2014).

In DCVRPs, the most studied source of dynamism is the appearance of new customers (i.e. new orders) during service process (Pillac, Gendreau, Guéret, & Medaglia, 2013). This problem is of interest because disruptive events must be handled efficiently and quickly. Distribution companies require fast and efficient re-optimisation algorithms (i.e., intelligent distribution systems) to control their fleets in real-time, maximise the efficiency of the distribution, and reduce the distribution cost. Moreover, to ensure customer satisfaction, companies must immediately accommodate and serve the increased volume of new and urgent requests from customers. As the distribution costs comprise a non-negligible fraction of the purchase price of a product or service, an effective distribution system improves the competitiveness of a company.

Similar to DCVRP, most papers in the current literature consider the arrival of new orders to be as an urgent event in VRSP. Although these papers have successfully solved the VRSP, they have not directly or explicitly addressed road-closure events. Therefore, such events remain unexplored in the literature. Besides new-order placement, the review of the literature found VRSP studies investigating vehicle breakdown, travel time delay, and order cancellation events (Bock, 2010; Cortés, Núñez, & Sáez, 2008; Dávid & Krész, 2017; El Rhalibi & Kelleher, 2003; Haghani & Yang, 2007; J.-Q. Li, Mirchandani, & Borenstein, 2007; Lieshout, 2016; Mu, Fu, Lysgaard, & Eglese, 2011; X. Wang, Wu, & Hu, 2010). Other researchers were found to have focussed on train and airline rescheduling problems (Fekete, Kroeller, Lorek, & Pfetsch, 2011). Methods applied in real-time vehicle schedule recovery are reviewed in (Visentini, Borenstein, Li, & Mirchandani, 2014).

The burgeoning interest in DCVRPs is also driven by advances in technology, telecommunications, smartphones, global positioning systems (GPSs), and geographical information systems (GISs). These systems are essential in dynamic environments, as companies can track and communicate with vehicle drivers and assign them with new orders or new instructions in real-time (Pillac et al., 2013). By utilising these systems, companies can exploit and process all of the necessary information as it appears, and thereby maximising their profits. In such systems, information processing can be centralised or decentralised. In a centralised system, the current information on the drivers' next destinations is updated through communication between the drivers and the dispatching centre. In contrast, a decentralised system allows the drivers to process the new data and take appropriate actions without deferring to the dispatch centre. Figure 7-1 depicts an example of a simple DCVRP with eight static customers and two dynamic customers. The routes have been altered to serve the new customers without violating the vehicle capacity.

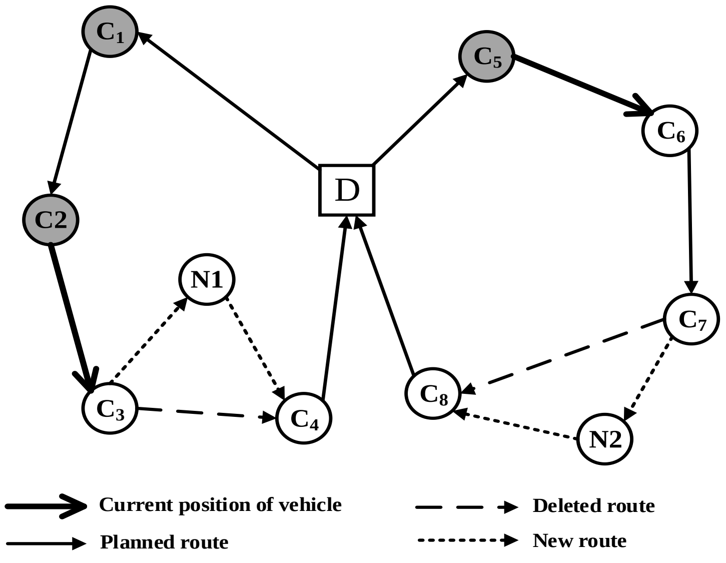


Figure 7-1. A DCVR with eight advanced and two dynamic requests

The main objective of a DCVRP solution is to reduce the cost and minimise the number of vehicles on the route. This is achieved by redirecting a vehicle to service new nearby, rather than distant, requests. Therefore, a DCVRP inherits the traditional objectives defined in the standard CVRP, but must also optimise other objective functions such as: maximise the ratio of served to neglected customers; maximise the service level (i.e., minimise the response time); minimise the wait for service (i.e., maximise the satisfaction of customers); maximise the profit or improve the productivity; and reduce the environmental impact (congestion, noise, pollution). These objectives depend on the degree of dynamism and the nature of the problem (Khouadjia, 2011; Larsen, 2001).

Further details and taxonomies of DCVRPs are presented in several review papers, for example see: (Abbatecola, Fanti, & Ukovich, 2016; Bektas, Repoussis, & Tarantilis, 2014; Ferrucci, 2013; Golden, Raghavan, & Wasil, 2008; Larsen, 2001; Larsen, Madsen, & Solomon, 2007; Pillac, 2013; Pillac et al., 2013; Psaraftis, 1995; Psaraftis, Wen, & Kontovas, 2016).

7.2.1 Road Closure Events

DCVRPs are largely concerned with changes in road conditions. In real-life situations, road availability can be reduced by blockage or closure. The following typical events have an undesirable impact on the distribution network:

- Major accidents or heavy congestion.
- Roadworks (road construction or maintenance).
- Weather conditions (e.g., snow, fog, heavy rain, typhoon).
- Parades, protest/public events, or emergency conditions.
- Natural disasters (such as road collapses in earthquakes, floods and slips).

These situations cause either delays or the complete closure of some roads, leading to inconsistencies in the planned schedules and routes. Consequently, employees must work overtime, this increased labour cost in turn increases the distribution cost. To avert this problem, companies require an immediate reschedule recovery plan, in other words, a VRSP. However, finding an alternative schedule can itself incur high additional costs.

Some distribution companies routinely perform manual rescheduling. The dispatcher reschedules the affected vehicle and gives it a new alternative route. In rescheduling, the dispatcher aims to reduce the impact of the event on the next customers and to minimise the costs of deviating from the original schedule (Gang Yu & Qi, 2004). However, the best reaction to such events depends on factors such as the time and location of the closure, the current location of the vehicles, the remaining capacity of each vehicle, and the locations of the affected customers. Considering the policies of the distribution company, staff must also decide whether to accept some delays in customer deliveries or reschedule some of the routes. In making this decision, they calculate the cost of each option and choose the least expensive option. According to Spliet, Gabor, and Dekker (2014), the costs incurred by deviation depend on the time of the event; specifically, the costs are lower if the event happens at a later stage of the plan execution, when large portions of the main schedule have remained intact.

The quality of a new solution can be measured with a simple mathematical metric called the *Deviation cost*, defined by Eq. (7.1):

$$Deviation\ cost = \ new\ schedule\ cost - \ original\ schedule\ cost. \qquad (7.1)$$

The deviation costs can be used as an objective to be minimised during the recovery schedule without violating the capacity constraint. Another quality metric of a new solution is the *Disruption rate*, which calculates the number of customers affected by a road closure:

$$Disruption\ rate = \sum_{i=1}^{r} \sum_{k=1}^{n} x_k, \qquad (7.2)$$

$$such\ that \quad x_k = \begin{cases} 1 & if\ the\ customer\ is\ affected \\ 0 & otherwise \end{cases}$$

In Eq. (7.2), r is the number of routes, and *n* is the number of customers in route *i*. For example, given a solution (1, 2, 3, 4, 5, 6, and 7), there are two alternative solutions (*S1*=1, 2, 7, 6, 5, 4, and 3) and (*S2* =1, 2, 4, 3, 5, 6, and 7) with disruption rates of 2 and 5, respectively. In this case, *S2* would be selected over $S1$ due to its lower disruption rate. Minimising the two metrics, disruption rate and deviation cost, preserves the initial schedule as much as possible and also help to assess the stability of the algorithm (i.e., the degree to which the new schedule approximates the original). Thus, the favoured solution will minimise both the deviation cost and the disruption rate.

7.2.2 Degree of Dynamism

In general, measuring an algorithm's performance is simpler and easier in static problems than in dynamic problems. In static problems, the evaluation is commonly based on criteria such as the running time, number of iterations, and solution quality. However, measuring the performance on a dynamic problem is nontrivial and requires new metrics for a proper evaluation (Pillac et al., 2013).

In both static and dynamic CVRPs, the locations of the known customers are important inputs that might affect the performance and solution quality of the algorithm. The performance of a DCRVP solver also depends on the number and locations of new customers, the types of dynamic events, the timing of the events (events distribution), and the current values of the problem variables.

Lund et al. (1996) proposed a metric called the *degree of dynamism* (DoD) that measures and determines the complexity of a DCVRP. The DoD defines the ratio between the number of dynamic requests (orders received while executing the route schedule) and the number of total requests (known and dynamic):

$$DoD \ = \frac{Dynamic\ requests}{Total\ requests} \times 100\% \qquad (7.3)$$

Thus, according to Eq.(7.3), when dynamic requests are received from two of ten customers the DoD is 20%. The DoD metric assumes that events are evenly distributed along the time horizon and this is not normally the case. To overcome this limitation, Larsen (2001) modified the DoD metric to account for the arrival time of requests. This metric, called the *effective degree of dynamism* (EDoD), is defined by Eq. (7.4):

$$EDoD = \left(\frac{1}{total\ number\ of\ requests} \times \sum_{i \in R} \frac{t_i}{T}\right) \times 100\% \qquad (7.4)$$

such that $0 < t_i \leq T$

Here, T is the length of the working day, R is the set of dynamic requests, and t_i is the arrival time of request *i*. The arrival times of the requests are important, as they may affect the solution quality. The EDoD measure lies between 0 (purely static; simple problem) and 1 (purely dynamic; difficult problem) (Ferrucci, 2013). Figure 7-2 depicts two potential scenarios of arrival times (scenarios *A* and *B*) and two scenarios of order distributions, i.e., arrival rates (scenarios *C* and *D*). In scenario *A*, all nine of the new requests arrive at the beginning of the planning horizon, so are easily reacted to and accommodated. In scenario *B*, all requests arrive late in the day, creating difficulties in handling these requests.

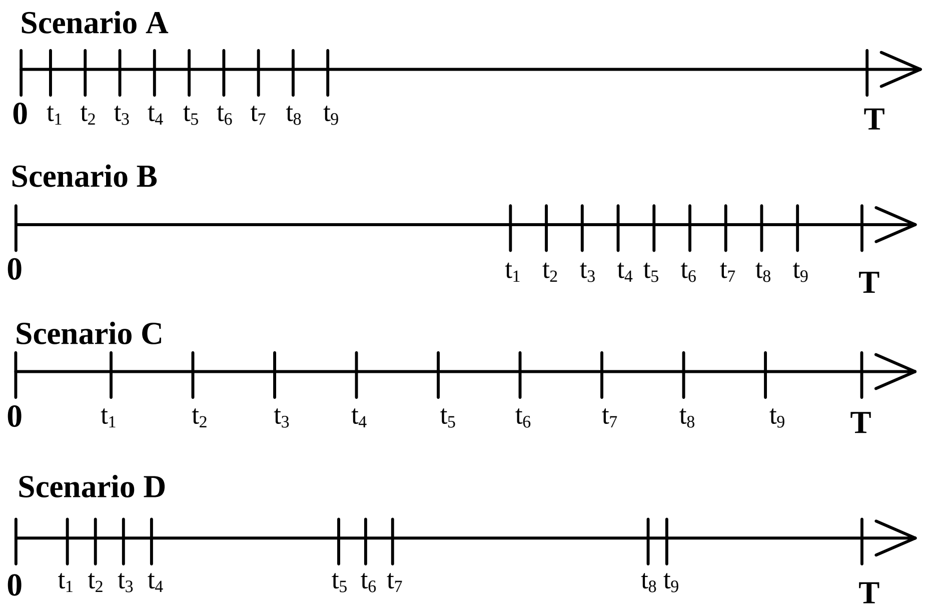

Scenario A
0
t1 t2 t3 t4 t5 t6 t7 t8 t9
T
Scenario B
0
t1 t2 t3 t4 t5 t6 t7 t8 t9
T
Scenario C
0
t1 t2 t3 t4 t5 t6 t7 t8 t9
T
Scenario D
0
t1 t2 t3 t4
t5 t6 t7
t8 t9
T

Figure 7-2 Relationship among the arrival time, rate of dynamic requests and the time horizon (Larsen, 2001)

Ichoua, Gendreau, and Potvin (2007) identified two factors that affect the dynamism level: the frequency of changes and the urgency of requests. The arrival rate also crucially determines the complexity of the problem. The frequency of update the problem information affects the time available for optimisation. In scenario *C*, the constant bursts of requests must be handled by continually restarting the optimisation algorithm (Kilby et al., 1998). In contrast, widely spread requests (scenario *D*) provide longer time windows for re-optimising or improving the current plan. The performance of serving the new requests (response time) is also affected by the geographical locations of the new requests, the current locations, the loads of the vehicles, and the travelling times (vehicle speeds) between the customers. Furthermore, the company's service policies may influence the handling of new requests and increase the complexity of the problem. Some companies will delay or deny the service if they cannot accommodate (serve) a new request immediately or within economically feasible limits.

7.2.3 Approaches for Solving DCVRPs

The literature proposes a number of different approaches for solving DCVRPs. Wilson and Colvin first solved a simple DVRP in 1977. In their scenario, one vehicle processed requests from dynamic customers while moving between a start point and a destination point (dial-a-ride problem). To accommodate the new customers, they updated the current route using a heuristic insertion method. Psaraftis (1980) extended the dial-a-ride problem to include immediate requests. This problem, in which a customer requesting service must be serviced as soon as possible, was solved by a dynamic programming approach.

In many DCVRPs, an initial solution called a *tentative schedule* (Psaraftis, 1980) is generated for the known customers. During the day, the tentative schedule is continuously updated to incorporate new requests or to further optimise the solution. The dispatcher then updates the drivers with the new optimised routes and new information about their next destination. The DCVRP is most commonly solved by the insert-and-improve technique which inserts the new requests in the minimal-cost position and optimises the solution by continuously improving the schedule between the new requests.

Gendreau, Guertin, Potvin, and Taillard (1999) solved the DVRP with a Tabu search. Their technique restarted the optimisation algorithm whenever a new order received. This technique is time-consuming; moreover, because the optimisation time is not specified the optimiser may be interrupted before reaching a high-quality solution. However, it is suitable for problems with urgent requests requiring immediate attention. In an alternative approach, at each update, the re-optimisation can be replaced by a batching strategy that collectively handles and re-optimises a number of new requests.

Kilby et al. (1998) proposed another simple approach that divides a DCVRP into a sequence of static CVRPs. Basically, a day is divided into several time slices of equal length. Each time slice represents a static CVRP with different vehicle statuses (i.e., different remaining capacities and locations). All requests arriving during a time slice are delayed until the subsequent time slice. Based on the newly available information, the optimisation algorithm is executed once during each time slice (each sub-problem). Any other existing constraints are considered during the schedule update. The main advantage of this technique is the equal computational effort for each time slice. However, determining the ideal length and number of time slices is critical to solution quality. The number of time slices determines when new information is sent to the optimisation algorithm. Dividing the day into many time slices increases the computational effort (as the optimisation process must be restarted many times), but allows quick reaction to the newly arrived customers. If the time slices are too few, the new customers will be postponed for a long time. In an extension of Kilby's method, Montemanni et al. (2002) introduced a user-decided cut-off time, after which all requests are postponed to the next time slice.

One of the issues in the previous approaches is when a vehicle (i.e., a driver) can/cannot be diverted from its current schedule. The routing schedule must commit the drivers to predetermined destinations (i.e., no diversion is allowed). Under this constraint, the drivers must serve their next customers at particular times without interruption, i.e., once a driver is committed to a customer, this task cannot be changed. However, setting the commitment time is a vital question in DCVRPs. In one technique, the schedule is fixed for a specific period of time (the *advance commitment time*). During this period, the dispatcher cannot change the schedule (Montemanni et al., 2002).

Some previous approaches simplified the solution for DCVRP by solving a series of static problems. However, in order to improve the final solution quality, transferring the characteristics of a good solution to the subsequent static problems is important. Additionally, the optimisation needs to track the served and unserved customers. Therefore, the optimisation algorithm may need to be facilitated by supplementary components, such as that of Montemanni et al. (2002). Other assumptions and policies, such as the travelling time between the customers (which is based on the vehicle speed) must also be considered. In the absence of congestion, the travel times are equivalent to the corresponding real road distances or the Euclidean distances.

7.2.4 Problem Formulation

The DCVRP is strongly related to the static CVRP. Thus, the mathematical formulation and constraints of the DCVPR are similar to those of static CVRP. In most DCVRP implementations, the objectives are to minimise the total travelling cost and the required number of vehicles while serving all requests. The total travelling cost in a DCVRP is calculated as follows:

$$Cost\ (DCVRP)\ =\sum_{i=1}^{n} cost\ (solution(CVRP_i)) \tag{7.5}$$

where n is the number of static CVRP instances generated in the DCVRP. It is difficult to determine the exact number of vehicles used because the vehicle can return to the depot when it completes the task even if it still has a remaining load. In our experiments, a vehicle will be counted if it has been used regardless of its remaining capacity. Therefore, the number of vehicles used in a DCVRP is obtained by summing the numbers of vehicles used in each static CVRP.

7.2.5 Pareto Frontier for Multi-Objective Problems

In many practical applications, an optimisation problem must optimise multiple objective functions simultaneously (Stadler, 1988), see Eq. (7.6). However, multiple objective functions often conflict with each other (Coello Coello, Lamont, & Veldhuizen, 2007). Therefore, the best solution to a multi-objective problem requires a trade-off between the objective functions.

$$min\ or\ max\ \ F(x)\ =\{\,f_1(x), f_2(x), \ldots, f_n(x)\}, \tag{7.6}$$

$$\text{such that: } x \in X \text{ and } n\ \geq 2.$$

Here, *X* is a set of constraints or decision variables, and *n* is the number of objective functions. For example, suppose that a CVPR instance admits different solutions with different qualities, as shown in Table 7-1. Which solution should we select? We can choose to minimise the number of vehicles or reduce the total cost.

Table 7-1. Candidate CVRP solutions with different qualities

Solution	Vehicle numbers	Total cost
A	10	170
B	9	200
C	8	180
D	7	210

Obviously, the chosen solution should minimise both objectives. When multiple solutions minimise each objective, choosing the best solution is a challenging task. For example, comparing solutions *B* and *C*, we find that both objectives are lower in *C* than in *B*. Hence, solution *C* dominates solution *B* and solution *B* is an inferior option. The remaining solutions (*A*, *C*, and *D*) are called non-dominated solutions, as they have a trade-off between the number of vehicles and total cost.

Conflicting objectives can be optimised through the *Pareto frontier* technique, named after Vilfredo Pareto (Marler & Arora, 2004). In this technique, many candidate solutions maybe located in the feasible region. The chosen solutions must lie as close as possible to the Pareto front line (see Figure 7-3). To apply the Pareto technique, the feasible solutions are plotted (shown graphically) between axes representing the objective functions. After defining the feasible region, the possible points are marked by a line (Ngatchou, Zarei, & El-Sharkawi, 2005).

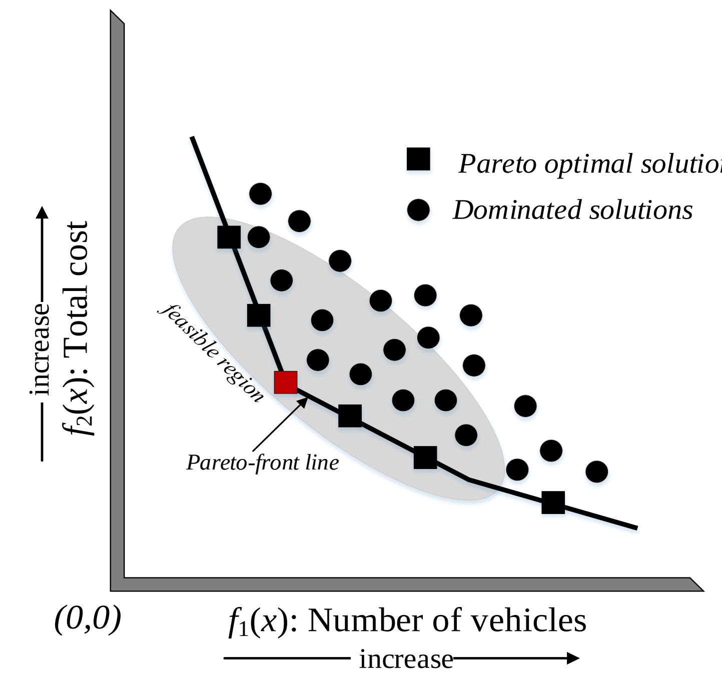


Figure 7-3. Conceptual illustration of Pareto solutions in a two-objectives search space (Ngatchou et al., 2005).

Figure 7-3 shows the Pareto front of two objective functions. The solutions on this line (represented by squares) are called Pareto-optimal solutions. Each circle in Figure 7-3 denotes a dominated solution. Note that all of the Pareto-optimal solutions are non-dominated. Only one preferred solution should be selected from the Pareto-optimal solutions (e.g., the red square).

In some problems when the Pareto line is difficult to generate, each objective can be assigned a numerical weight that balances the opposing objective (see Eq. (7.7)). This weighting helps to prioritise one objective over the other.

$$Solution\ total\ weight = (w_1 \times Objective\ 1) + (w_2 \times Objective\ 2) + \cdots + (w_n \times Objective\ n), \quad (7.7)$$
$$\text{such that } w_1 + w_2 + \cdots + w_n = 1.$$

The solution with the highest total weight will be selected as a Pareto optimal solution. For the DCVRP, we set the weight as follows:

$$Solution\ total\ weight = (0.6 \times total\ cost\ ) + \ (0.4 \times number\ of\ vehicles). \qquad (7.8)$$

In Eq. (7.8), by setting the coefficients to 0.6 and 0.4, minimising total cost is prioritised over minimising the number of vehicles.

7.3 The HCA–DCVRP Approach

Similar to the conventional CVRP, the DCVRP can be represented as a fully connected graph (see Chapter 6 for details). However, the graph in a DCVRP changes as new nodes are added to accommodate new customers, and an edge is deleted when a road closure occurs. To solve the DCVRP by HCA, we can incorporate the HCA with an additional component called an *events manager* (aka *controller*). In previous research the events manager has been used to facilitate and coordinate similar optimisation tasks (Housroum, Hsu, Dupas, & Goncalves, 2006; Montemanni, Gambardella, Rizzoli, & Donati, 2005; Oliveira, Souza, Am, & Silva, 2008; Pillac, Guéret, & Medaglia, 2012; Rizzoli et al., 2007).

The events manager updates of scheduling information, controls the inputs to the HCA and uses the HCA outputs to update routes. Updating the information involves collecting new orders, queuing pending orders, tracing served customers, identifying the committed customers, determining the current routes (sequence of customers to be served) and position, and evaluating the remaining capacity of each vehicle. The events manager produces a sequence of static CVRPs based on the available information, and sends it to the HCA. The HCA then solves the static CVRP and returns the results to the events manager for updating the current routes. The new details are sent to the designated drivers.

Similar to Montemanni et al. (2005), we solve the DCVRP using a centralised approach. For this purpose, we require the length of the working day T, which denotes the maximum number of working hours in one day. T is divided into a number of time slices (*nts*). The length *Lts* of each time slice is given by Eq. (7.9):

$$Lts\ = \frac{T}{nts} \qquad (7.9)$$

For example, if the day length is 1000 unit, and the number of time slices is set to 50 unit, the length of each time slice is 20 unit. The working day is divided into time slices as follows:

$$Day\ = \{TS_1, TS_2, TS_3, \dots,\ TS_{nts}\} \qquad (7.10)$$

Depending on the available time of the customers, each time slice can accommodate N customers (see Figure 7-4).

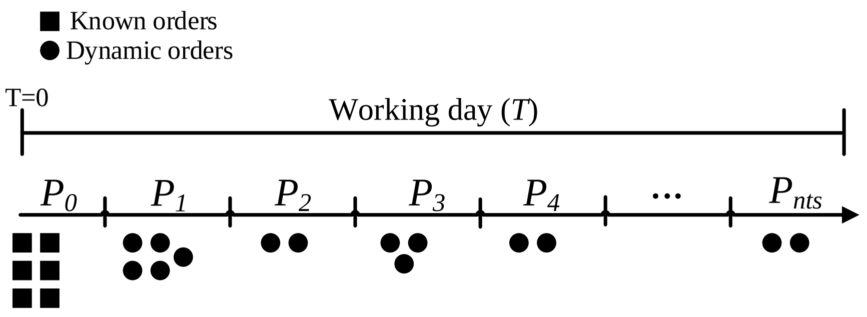


Figure 7-4. A time-slice sequence of static CVRPs (Khouadjia, 2011)

In each time slice, the information and conditions satisfy a static CVRP, but the statuses of the vehicles (i.e., the locations and remaining capacities) change. Whereas Montemanni et al. (2005) specified a cut-off time and an advance-commitment time, we simply commit a driver to a specific customer. When the driver is on route to serve that customer, this commitment is inviolable. In addition, instead of specifying the cut-off time, we address all customers arriving within the current time slice before starting the next time-slice.

$$nts_i < k \leq nts_{i+1} \tag{7.11}$$

where k is the available time of the customer, and nts_i and nts_{i+1} are the current and next time slices, respectively. Any new requests arriving during a time slice will be processed in the next time slice. The events manager calls the HCA at the end of each time slice. The pseudocode of the events manager is given in Figure 7-5.

```
BEGIN
PROCEDURE: Events Manager
      Time = 0 // represent current time (the time-step = 1)
      T = length of day
      // Read data file (customers locations, demands, and available
      time)
      //Initially, all the vehicles are at the depot
      List of unserved customers = {customers with available time = 0}
      //Build an initial feasible solution
      Initial-Solution = HCA (list of unserved known customers)
      //Vehicles leave the depot to serve the customers
      WHILE (Time < T or Orders ≠ empty)
            //New requests are received and sorted in a queue (based on
            the available time).
            //The vehicles are moving and serving the customers
            IF (Time = the end the time-slice) // at the end of each time
            slice (T/nts) do
                  //Obtain the new requests that correspond to the
                  current time slice.
                  ObtainVehiclesPosition()
                  ObtainVehilcesCapacity()  //check vehicle's residual
                  capacities
                  CheckCommitedOrders()  //remove committed customers
                  from planned routes.
                  UpdateServerdCustomers(); //remove served customers
                  from planned routes.
                  Static CVRP = GenerateStaticProblem()
                  Solution = HCA (Static CVRP)
                  UpdateRoutes(Solution)
            END-IF
            Time = Time + 1
      END-WHILE
      Vehicles return to the depot
END-PROCEDURE
```
Figure 7-5. Pseudocode of the events manager procedure

Another component called the *events generator* generates different events that simulate real-life scenarios. In this study, the dynamism is sourced from the arrival of new customers and road closures. Figure 7-6 shows the system architecture of the HCA-based solution method for the DCVRP.

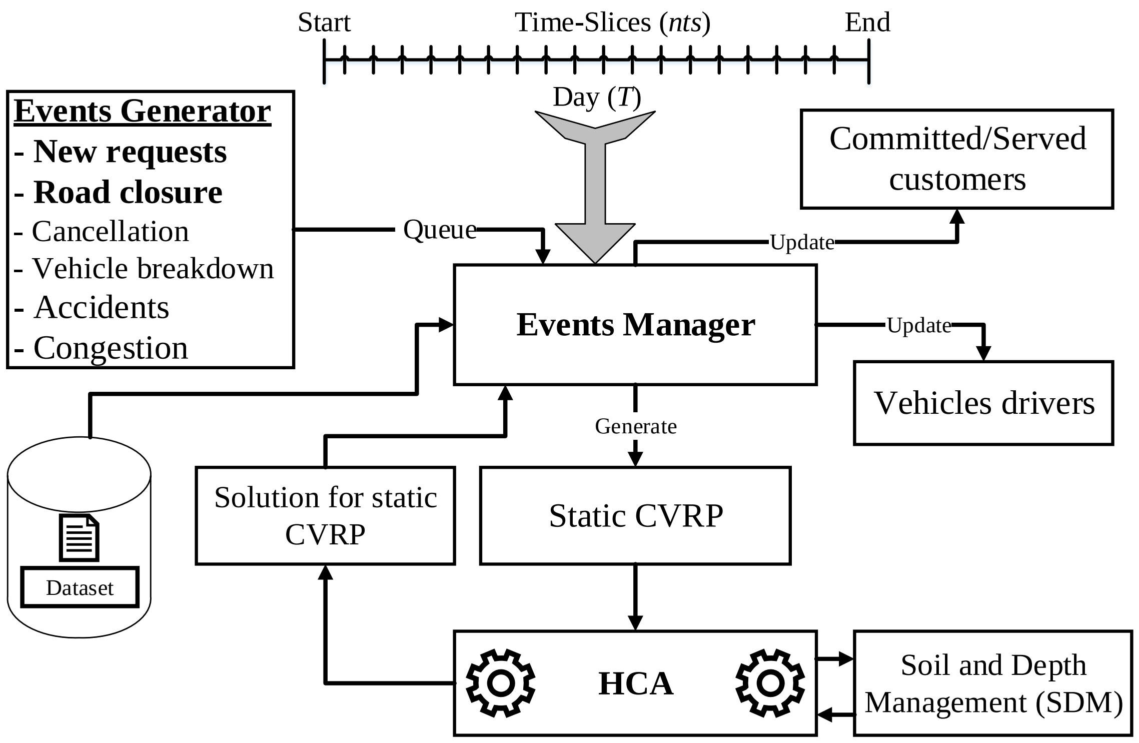


Figure 7-6. Architecture of HCA-based system for solving the DCVRP

To transfer or preserve the good properties of the previously solved static CVRP to the next CVRP in the sequence, we employ a special auxiliary technique called *soil and depth management* (SDM). The SDM component in the architecture preserves the good properties of previous solutions between successive time slices, thereby reducing the computational effort and obtaining better solutions in the next time slices. The SDM working mechanism is inspired by paths with different depths in natural environments. Deeper paths will remain carved even when water evaporates. Here we assume that edges representing deeply carved paths will not be affected by the arrival of new customers, whereas those representing shallow paths will locate new customers. This strategy is consistent with (supports) the design of the transition rule that controls the water movement. This rule allows water drops to spread in different directions while maintaining the edges belonging to good previous solutions.

The events manager first reads a data text file and initialises all of the required data structures. An important datum is the available time, which is associated with each request and indicates the disclosure time of that request to enter the optimisation process. Prior to the disclosure time, nothing is known about the request. If the available time is zero, the order is known at the beginning of the execution (i.e., is a static or known order). The available times are spread throughout the working day. Before beginning each day, the events manager sends all details of the known customers to the HCA, which finds an initial solution. During the day, the arrivals of the new customers are accumulated and sorted according to their available times and the current time slice. Only those customers arriving between the beginning and end of a time slice will be considered. At the end of each time-slice and before the next time-slice begins, the events manager checks the customers that become available, removes the served and committed customers from the previous route plan, and updates the vehicle statuses. The current location of each vehicle is considered to be the location of its last served customer. Again, the events manager sends all newly available details to the HCA for optimising the previous solution. The new optimised solution is used to update the current schedule and the new vehicle destinations. The vehicles committed to specific customers follow fixed routes and cannot be diverted.

7.3.1 HCA Procedure for Solving DCVRP

The main HCA procedure is that described in Chapter 6, with minor modifications (namely, implementing the Pareto frontier technique in the condensation stage, and connecting the HCA with the SDM component).

The Pareto-optimal solution is found among the solutions obtained by the evaporated water drops. The quality of the selected solution is then enhanced by improvement methods, and the global-best solution is updated by the current-best solution.

The SDM component controls the soil and depth values. Consequently, the role of the precipitation stage is limited to the redistribution of water drops over the nodes, with intense precipitation on the new nodes. The SDM reinitialises the soil and depth values when the HCA obtains a solution in a specific time slice, not after each cycle. The edges leading to new customers are shallower than the established edges, and those belonging to previous solutions have less soil than the current edges. The HCA procedure is described in Figure 7-7.

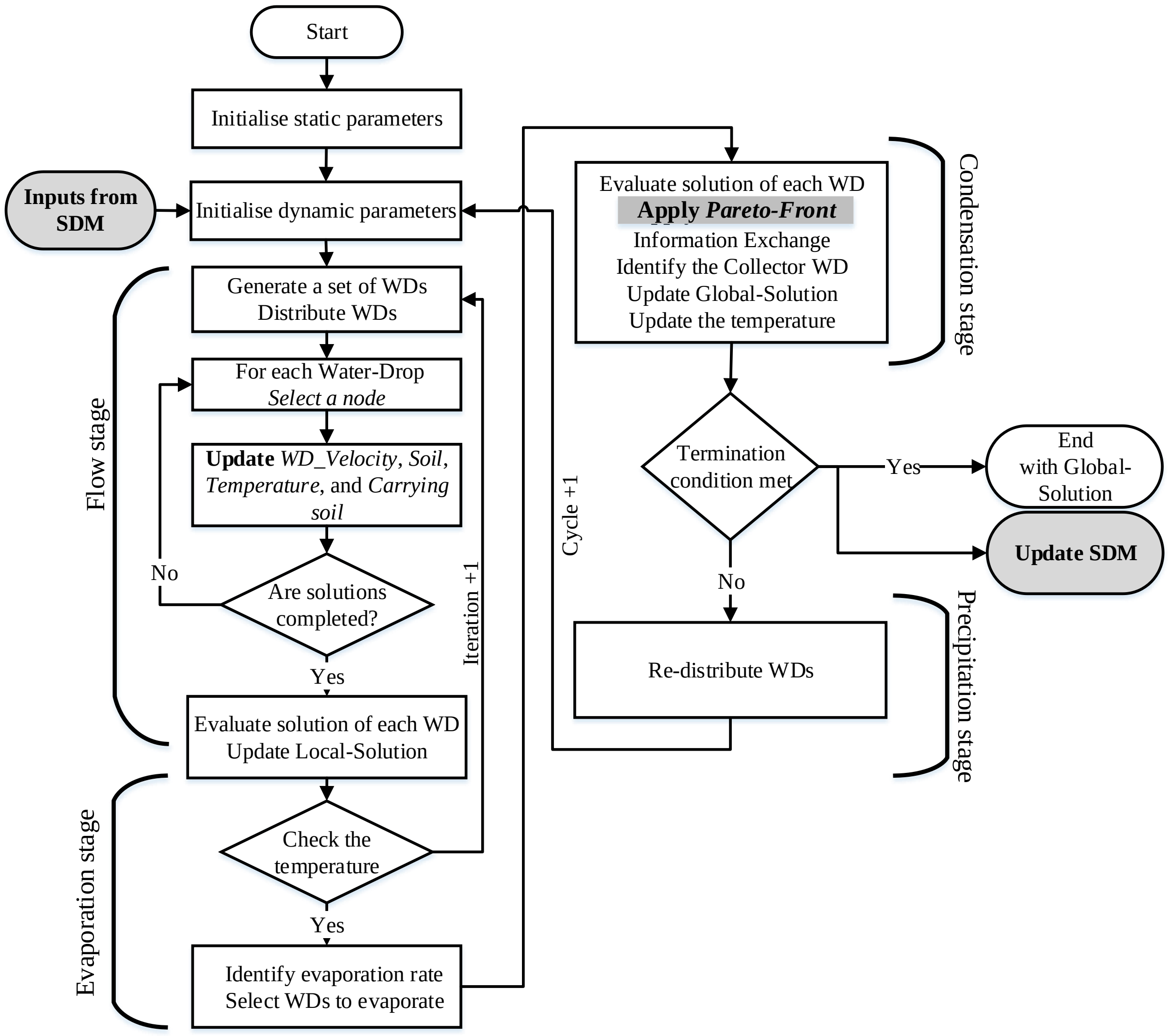


Figure 7-7. HCA procedure for solving the DCVRP

7.3.2 Assumption and Constraints

Owing to the complexity of the DCVRP and the various factors that may affect the evaluation of any result, we must clarify some assumptions to ensure the validity and replicability of the research results. The assumptions adapted from Psaraftis et al (2016) are:

- There is only one depot.
- Vehicles have a limited capacity.
- At the beginning, all vehicles are loaded to their maximum capacity.
- Delivery only; no pickup is required.
- The dynamic elements are new orders and a road closure.
- There is no time-window constraint.
- The service time (time required to serve a customer) is zero.

- The distances are the Euclidian distances.
- The depot holds an unlimited number of homogenous vehicles.
- The objective functions are: minimise total cost, number of vehicles, serve all requests.
- All received orders should be accepted.
- Existing customers cannot amend or cancel their orders.
- There is simultaneous communication between the vehicles and the dispatcher
- There is no stochastic information or probability distribution on future orders.
- The vehicle travelling speed is fixed (1 unit).

A vehicle returns to the depot after completing its duties (i.e., serving all customers on its designated route), after dispensing its load (using its whole capacity), or when the current load is not enough to serve a new customer. When a vehicle returns to the depot, it is reloaded to its maximum capacity. All vehicles must return to the depot before the depot closes at the end of the day. In the present research, we also assume that the vehicle speed (travelling time) is constant and equal to one time-unit per distance-unit (i.e., a distance of 1 unit is driven in 1 time unit) in all experiments. The current time is incremented by a fixed time step of one.

Most problems with a time-window constraint also specify a service time (i.e., task duration), which extends the working day. Especially in small problems, the service time improves the realism of the simulation (Pillac et al., 2013; Pillac, Guéret, & Medaglia, 2010). However, we chose to exclude the service time to reduce the number of factors affecting the final results and obtain more precise empirical results.

If the demand by new customers exceeds the capacity constraint of all vehicles, another vehicle is added to the fleet. Alternatively, another vehicle might replace the current one to reduce the total cost. In some static CVRP instances, adding an extra vehicle has proven to reduce the cost. Furthermore, once a vehicle is on route to serve a specific customer, it cannot be diverted.

7.3.3 Benchmark Dataset

Benchmark datasets provide a basis for assessing and comparing an algorithm’s performance on a particular problem. To our knowledge, no adequate benchmark datasets are publicly available for the DCVRP. Pillac et al. (Pillac et al., 2013), Gath (2016), and Maciejewski et al. (2017) stated that “*there is no reference for a standard benchmark for the DVRP*”. Most researchers have referred to Kilby et al. (1998) and Montemanni et al. (2002), who generated DVRP datasets by adding more parameters (i.e., available time, service time, and day length) to static VRP datasets. The available times for customers are generated from a uniform random distribution function. Previous studies refer to data available on certain websites[6][7][8], but unfortunately, these links are defunct. The unavailability of a benchmark dataset precludes a fair comparison of the study results.

The experiments in many published studies were conducted on randomly generated instances or real-life problems. Random data are easily obtained and can be constructed in various ways to cover different scenarios. Therefore, they enable in-depth analyses of approximately real-life problems. The main drawback of random datasets is the difficulty in evaluating the results, especially for instances involving many customers. Real-life problems require highly specific DVRPs with certain objectives and various constraints, and the data must be taken from real applications. For example, studies allowing pick up and delivery did not consider the vehicle capacity (Fleischmann, Gnutzmann, & Sandvoß, 2004; J. Yang, Jaillet, & Mahmassani, 2004).

DCVRPs are commonly studied by simulation models that update the problem variables and control their dependencies and interactions over time. For example, the model updates the system state variables including the locations and loads of the vehicles, the time increment, and customer statuses. Our present simulation model is represented by the events manager and the events generator. One impediment to comparing different solutions is the different settings and specifications in the simulation procedures of different studies. For example, the vehicle speed, time increment, and whether the distances are real-valued or rounded to integers, differ among studies. Some studies adopt a waiting strategy, in which a vehicle idles at a location before moving towards the next customer. The waiting strategy is particularly used to fill the gap between orders in a problem with a time-window constraint (Branke, Middendorf, Noeth, & Dessouky, 2005). These settings are crucial and may affect the quality of the final results, but are not completely described or clarified in some of the previous studies. In our simulation, the time is incremented by a fixed unit of one second.

In summary, to conduct an appropriate and consistent comparison for a DCVRP, we require a unified independent simulation model and a standard benchmark dataset. However, such a model must support various approaches, which is extremely difficult. For this reason, a significant comparison between different approaches is seemingly impossible.

For our experiments, we adapt three benchmark instances from Lackner (2004). The original dataset was designed for DVRP with a time-window constraint, and involved five degrees of

[6] http://www.cs.strath.ac.uk/~apes/apedata.html
[7] http://www.fernuni-hagen.de/WINF/inhalte/benchmark_data.htm
[8] http://natcomp.liacs.nl/index.php?page=code

dynamicity (10%, 30%, 50%, 70%, and 90%). Lackner adapted his dataset from the classical Solomon's benchmark (Solomon, 1987), adding the availability time for each customer based on the dynamicity degree. The dataset comprises three categories: *C*, *R*, and *RC*, in which the nodes are clustered, randomly distributed, and mixed from the *R* and *C* categories, respectively. Every instance contains 101 nodes (including the depot) distributed in a 100×100 Euclidean space. To ensure reproducibility of our results, we provide the benchmark instances used in Appendix C for interested readers.

7.4 Computational Experiments

This section evaluates the performance of HCA–DCVRP and analyses its computational results. To simplify the analysis, we studied two sources of dynamism (the arrival of new customers and road closure) in separate experiments.

7.4.1 Experiment 1: New customers

As mentioned earlier, our approach divides the DCVRP into a sequence of static CVRPs. For this purpose, we divide the day into a number of time slices and solve one static CVRP per time slice. The length of the working day (a user-controlled parameter) was set to 1000 seconds in all instances. The critical issue in DCVRPs is deciding the number of time-slices. A preliminary experiment aims to investigate whether the number of time slices is related to the DoD values, and their influence on the quality of the final solution. To achieve the desired objective, we executed the algorithm on DCVRP instances with different *nts* and DoD values. The *nts* was set to 10, 25, 50, 75, or an *exponentially increasing rate* (EIR). The DoDs were assumed as 10%, 30%, 50%, 70%, and 90%. Instances in category *RC* have relatively higher EDoDs than instances in the *C* and *R* categories.

By varying the *nts*, we can analyse the algorithm performance in different situations. For instance, when *nts* is set to equal 75 (i.e., the day is divided into 75 time slices), we can see if the algorithm can quickly process new requests, as more static CVRPs are generated for large *nts*. On the other hand, instead of fixing the number of time slices, we can trigger the events manager to process the next time slice when the number of new customers reaches a threshold ratio (e.g., 100%). For this scenario, we designed a new exponentially increasing function that controls the rate of change in a time-dependent manner:

$$\omega_{t+1} = \omega_t + \alpha \times (1 + r * t)^n, where\ \alpha \neq 0, n \geq 1, and\ \omega_0 = 1. \tag{7.12}$$

In Eq. (7.12), ω_t and ω_{t+1} respectively represent the previous and new increase rates at *t*+1, where *t* is the current time. The coefficients *r* (= 0.001) and α (= 10) control the increase rate subject to the time and the urgency rate of addressing new customers, respectively, and *n* is the number of new customers arriving at *t*. When α is large, the events manager will manage new customers more quickly. According to Eq. (7.12), the rate of change is increased by increasing the number of new customers per unit time, and accelerates as time passes. Consequently, the time slices vary in length, which affects the total number of time slices per day. This technique is useful when the number of new customers increases at certain times of the day (i.e., during rush hours), and declines at other times. Furthermore, this technique prevents the delay in processing new customers, thereby enhancing the customers' satisfaction.

Figure 7-8 shows the customer distribution in the C101-50-50 instance. The squares and circles represent the known and dynamic customers, respectively.

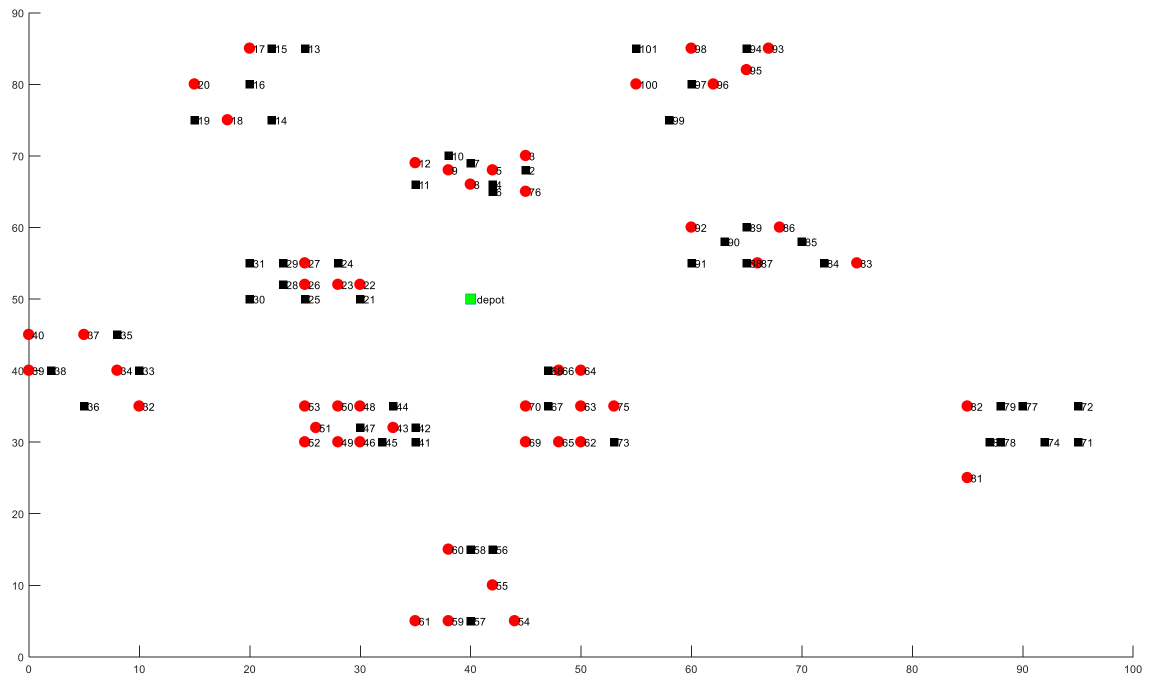


Figure 7-8. Distribution of customers in the C101-50-50 instance

The HCA–DCVRP was executed 20 times in each instance in all experiments, and the results were recorded. The number of HCA iterations was set to to five times the number of customers in the given time slice, ensuring similar computational effort in each time slice.

7.4.2 Computational Results

In this experiment, we evaluated three classes of instances dealing with the arrival of new customers. The classes, C101, R101 and RC101, are named by their category and DoD value. The instances are named as category (number of nodes)-static%-dynamic%. For example, instance C101-10-90 belongs to category *C* and contains 101 nodes (including the depot), among which 10% represent static customers and 90% represent dynamic customers. A sample of the outputs is presented in Appendix D. Table 7-2 lists the obtained HCA results on instances in category *C*. The "#V" columns list the numbers of required vehicles. Bold entries indicate the best solutions among the *nts* values.

Table 7-2. HCA-DCVRP results of Category C

Instance name-DoD	DoD	EDoD	Number of time slices (*nts*)									
			10		25		50		75		EIR	
			Cost	#V	Cost	#V	Cost	#V	Cost	#V	Cost	#V
C101-10-90												
Min	90%	24%	1804	14	1796	13	1844	14	1801	14	**1740**	12
Avg.			1848	13.5	1893	12.8	1917	13.3	1927	13.0	**1818**	11.9
Max			1878	15	1962	16	1976	13	1974	15	**1855**	12
C101-30-70												
Min	70%	20%	1646	11	1773	14	1739	13	1771	13	**1637**	11
Avg.			1706	11.4	1824	12.1	1862	12.1	1855	12.2	**1704**	11.0
Max			1771	13	1875	11	1927	12	1922	15	**1738**	11
C101-50-50												
Min	50%	16%	1655	11	1639	14	1679	12	1649	12	**1506**	10
Avg.			1725	12.0	1741	12.9	1774	12.7	1763	13.3	**1560**	10.2
Max			1764	13	1784	13	1839	16	1819	16	**1591**	11
C101-70-30												
Min	30%	9%	1493	14	1520	13	1511	13	1546	12	**1343**	10
Avg.			1514	13.2	1572	13.3	1587	13.8	1613	13.5	**1405**	10.1
Max			1529	14	1610	13	1619	14	1655	13	**1438**	10
C101-90-10												
Min	10%	4%	1346	13	1334	14	1301	14	1346	13	**1176**	11
Avg.			1399	13.0	1401	14.0	1412	14.2	1393	13.1	**1229**	11.1
Max			1432	13	1441	14	1447	15	1432	14	**1254**	11
Mean – Min			1589		1612		1615		1623		1480	
Mean – Avg.			1638		1686		1710		1710		1543	
Mean – Max			1675		1734		1762		1760		1575	

Contrary to expectation, increasing the number of time slices per day did not reduce the cost of instances with high DoD values (Table 7-2). It is worth noting that the HCA is called only when new customers become available in a specific time slice. Therefore, dividing the day into many time slices is pointless unless the system handles many new customers that frequently arrive in every time slice. In terms of minimising the total cost and number of vehicles, the EIR technique outperformed the other *nts* values. Figure 7-9 shows the relationship between the minimum costs and DoD values in *C* instances.

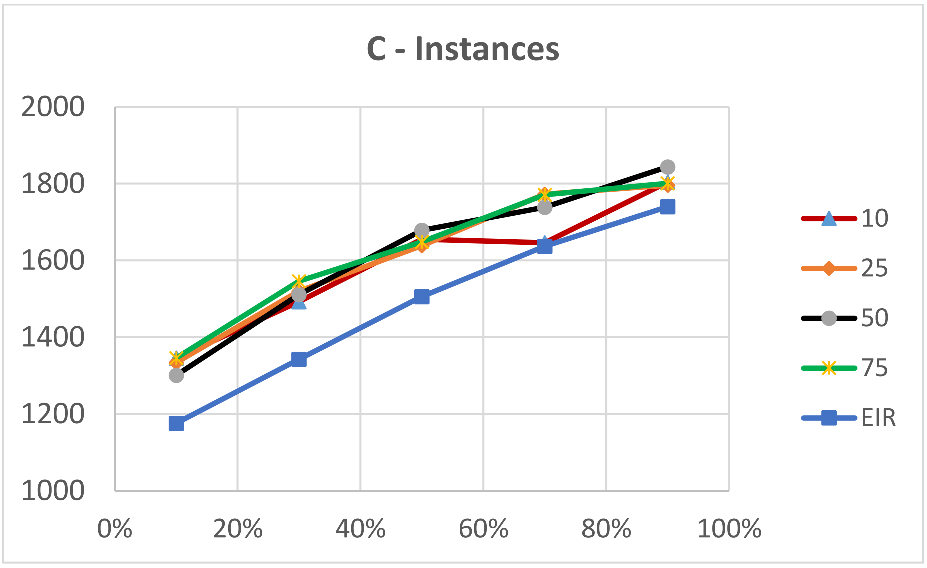


Figure 7-9. Relationship between cost and DoD values in the C instances listed in Table 7-2

The EIR, which minimised the solutions, was the best choice for these instances (Figure 7-9). Table 7-3 shows the results of instances in the *R* category with different DoD values.

Table 7-3. The HCA-DCVRP results of Category R

Instance name-DoD	DoD	EDoD	Number of time slices (*nts*)									
			10		25		50		75		EIR	
			Cost	#V	Cost	#V	Cost	#V	Cost	#V	Cost	#V
R101-10-90												
Min	90%	25%	**1009**	8	1088	9	1047	8	1064	9	1026	8
Avg.			**1073**	8.1	1126	8.3	1102	8.4	1110	8.4	1100	8.3
Max			**1094**	8	1156	8	1128	9	1147	8	1149	8
R101-30-70												
Min	70%	19%	1054	9	1063	8	1055	8	1095	8	**1047**	8
Avg.			**1079**	8.9	1110	8.1	1130	8.6	1133	8.4	1112	8.1
Max			**1096**	9	1140	8	1169	9	1168	8	1142	9
R101-50-50												
Min	50%	13%	**1019**	9	1055	8	1051	8	1045	8	1054	8
Avg.			**1051**	8.6	1091	8.4	1112	8.2	1126	8.3	1105	8.3
Max			**1069**	9	1127	9	1162	9	1166	8	1128	9
R101-70-30												
Min	30%	8%	**983**	8	1012	8	1025	9	1050	8	1030	8
Avg.			**1030**	8.3	1057	8.7	1090	9.1	1106	8.4	1090	8
Max			**1048**	8	1085	8	1135	12	1149	9	1129	8
R101-90-10												
Min	10%	4%	**978**	8	998	8	987	8	978	8	982	8
Avg.			**1009**	8	1042	8.4	1059	8.3	1079	8.6	1046	8.3
Max			**1028**	8	1071	8	1109	8	1124	8	1068	8
Mean – Min			1009		1043		1033		1046		1028	
Mean – Avg.			1048		1085		1099		1111		1091	
Mean – Max			1067		1116		1141		1151		1123	

As evident in Table 7-3, the results were fairly similar, but best when *nts* was equal to 10. It was also found that the EIR technique required fewer vehicles than any of the *nts* scenarios tested. Furthermore, in category *R*, setting *nts* to 10 yielded better results than setting *nts* to 25, 50, or 75. Figure 7-10 shows the relationship between the minimum costs and DoD values for the different *R* instances.

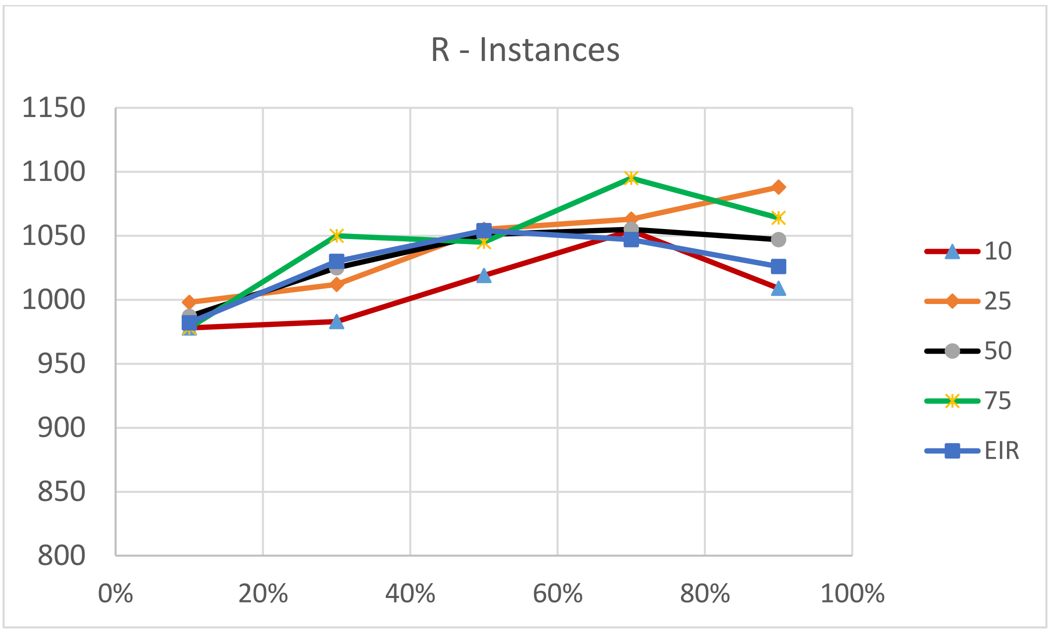


Figure 7-10. Relationship between cost and DoD values in the R instances listed in Table 7-3

Figure 7-10 clarifies that setting *nts* to10 minimises the cost of the HCA, regardless of the DoD values. Table 7-4 shows the results of instances in the *RC* category with different DoD values.

Table 7-4. The HCA-DCVRP results of Category RC

Instance name-DoD	DoD	EDoD	Number of time slices (*nts*)									
			10		**25**		**50**		**75**		**EIR**	
			Cost	**#V**	**Cost**	**#V**	**Cost**	**#V**	**Cost**	**#V**	**Cost**	**#V**
RC101-10-90												
Min			1231	10	1247	10	**1221**	9	1282	9	1260	10
Avg.	90%	29%	1285	9.7	**1314**	9.5	1341	9.6	1341	9.5	1324	9.4
Max			**1312**	10	1353	10	1391	9	1376	10	1356	9
RC101-30-70												
Min			1321	10	**1215**	10	1261	9	1275	9	1263	9
Avg.	70%	23%	1392	10.1	**1307**	9.6	1338	9.4	1325	9.5	1354	9.5
Max			1416	10	**1366**	9	1403	9	1377	9	1402	10
RC101-50-50												
Min			1344	11	1282	9	1247	9	1293	9	**1200**	9
Avg.	50%	16%	1402	10.4	1341	9.6	1325	9.5	1342	9.5	**1324**	9.5
Max			1441	10	1384	9	1402	10	1391	9	**1365**	10
RC101-70-30												
Min			1285	10	1230	9	1276	9	1271	10	**1215**	9
Avg.	30%	10%	1353	9.5	1351	9.6	1323	9.3	1329	9.5	**1318**	9.7
Max			1393	10	1421	10	1364	9	1383	9	**1357**	10
RC101-90-10												
Min			1230	9	1258	10	1259	9	1211	9	**1200**	9
Avg.	10%	6%	**1279**	9.2	1331	9.7	1302	9.7	1293	9.7	1282	9.1
Max			**1303**	9	1376	11	1347	10	1345	10	1331	9
Mean – Min			1282		1246		1253		1266		1228	
Mean – Avg.			1342		1329		1326		1326		1320	
Mean – Max			1373		1380		1381		1374		1362	

As shown in the table, the EIR technique minimised the results in two instances. Figure 7-11 shows the relationship between the minimum costs and DoD values for different *RC* instances. Again, the EIR technique yielded lower costs than the HCA, regardless of number of time slices in the latter.

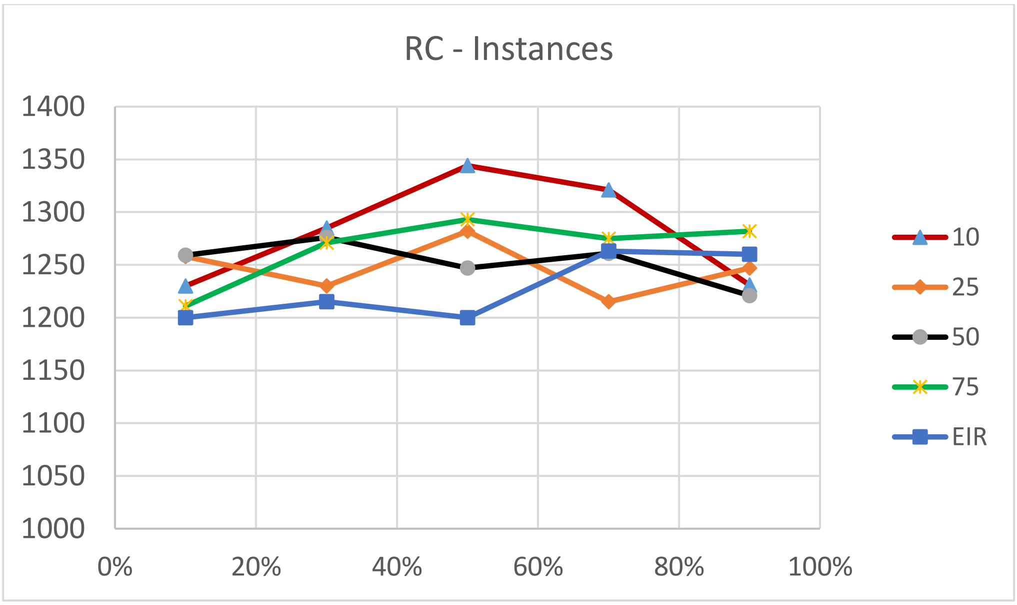


Figure 7-11. Relationship between total cost and DoD values in the RC instances listed in Table 7-4

To determine the best parameter setting for DCVRPs, we statistically compared the results obtained in all categories. Table 7-5 shows the *P*-values computed in the *t*-test. Note that the results of small sample sizes cannot be reliably generalised, so the analysis requires careful interpretation.

Table 7-5. Comparison of P-values obtained by t-test

Comparison	***P*-values using t-test (at P < 0.05)**								
	C101			**R101**			**RC101**		
	Min	**Avg.**	**Max**	**Min**	**Avg.**	**Max**	**Min**	**Avg.**	**Max**
10 vs 25	0.4308	0.0770	**0.0345**	**0.0447**	**0.0013**	**0.0005**	0.2290	0.6344	0.7995
10 vs 50	0.3052	**0.0377**	**0.0185**	**0.0406**	**0.0010**	**0.0021**	0.2530	0.5409	0.7307
10 vs. 75	0.2520	0.0530	**0.0328**	**0.0318**	**0.0011**	**0.0008**	0.4344	0.5220	0.9530
25 vs. 50	0.8982	**0.0092**	0.0658	0.3084	0.2409	0.1431	0.7290	0.8323	0.9448
25 vs. 75	0.0924	0.0487	0.0726	0.8206	0.0840	0.0497	0.3402	0.8288	0.6685
50 vs. 75	0.6923	0.9815	0.9090	0.2234	**0.0154**	0.0523	0.5196	0.9720	0.4067
10 vs. EIR	**0.0024**	0.0502	**0.0396**	0.1234	**0.0026**	**0.0013**	0.1243	0.3345	0.6446
25 vs. EIR	**0.0031**	**0.0020**	**0.0006**	0.3088	0.6030	0.4702	0.4680	0.6459	0.3693
50 vs. EIR	**0.0009**	**0.0010**	**0.0008**	0.3243	0.0760	0.1967	0.2700	0.4478	0.0575
75 vs. EIR	**0.0038**	**0.0008**	**0.0006**	0.1725	**0.0059**	**0.0455**	0.0703	0.5580	0.2711

From the results in Table 7-5, we can draw the following conclusions:

- In the *C* and *RC* instances, the results were statistically independent of *nts* (all *P* >0.05), but setting *nts* to 10 yielded better results than setting *nts* = 25, 50 and 75 especially in the *R* instances. The average results in the *R* instances were also significantly different between 10 and the other *nts* values. Furthermore, there was no significant difference between setting *nts* to 50 or 75; the sole exception was the average results in the *R* instances. These findings may be explained by the distribution of customers and the arrival-time rate of the new customers. When the EDoD values are low (as in these instances), dividing the day into many time slices is ineffective.
- Diverting a vehicle to serve nearby new customers is not always possible under the capacity constraint. In such cases, a new vehicle is required to serve the new customers, which increases the total cost and number of vehicles.
- After analysing the results, we determined that *nts* should lie between 10 and 50 if this method is used. This finding is consistent with Montemanni et al. (2005), who proved that dividing the day into more time slices does not always improve the results.
- The EIR results significantly differed from the *nts* = 10, 25, 50, and 75 results in the *C* and *RC* instances, and from the *nts* = 75 results in the *R* instances. Therefore, dividing the day into a fixed number of time slices may be impractical for some DCVRPs. The EIR is a good choice for problems with low EDoD values. Techniques such as EIR,

which prohibit the accumulation of many new customers and support the quick processing of new arriving customers, will generally improve the quality of the final solution. The proper technique allows the serving of new nearby customers while reducing the total cost, especially in systems with high DoD values.

- As expected, the results were higher in high-DoD instances than in low-DoD instances, because in the former instances, customers are continuously inserted into current routes (continuous diversion of vehicles). Consequently, the cost of finding a new solution will always increase. In instances with high DoD values, dividing the day into many time slices (e.g., *nts* $\geq$ 50) shortens the time slice, allowing the quick processing of new requests. However, because this solution increases the urgency of addressing the requests and does not utilise the full capacity of the vehicles, new vehicles may be needed for serving new customers. Therefore, the early insertion of customers into the current solutions does not guarantee better solutions.
- The Pareto front can direct the search in other directions where better solutions may exist. This was confirmed when the algorithm produced high-cost solutions with fewer required vehicles, or low-cost solutions with more required vehicles.

Table 7-6 compares the average execution time of each instance for *nts* = 10, 25, 50, 75 and EIR.

Table 7-6. Average execution times of instances with different nts and DoD values

Instance name-DoD	**10**	**25**	**50**	**75**	**EIR**
C101-10-90	53.5	11.4	15	21	10.7
C101-30-70	15	21.9	30.5	37.5	21.5
C101-50-50	28.6	71	143.4	233.5	52.2
C101-70-30	73.5	191	414.1	687.2	145.6
C101-90-10	147.8	259.8	710.6	883.3	275.9
R101-10-90	698.2	446.6	414.3	641.5	113.8
R101-30-70	314.2	307.2	462.7	579.8	118.1
R101-50-50	190.6	296.6	534.7	792.5	146.8
R101-70-30	87.9	229.2	594	948.6	186.4
R101-90-10	167	311.3	814.8	1218.1	298.5
RC101-10-90	761.5	438.5	551.8	712.8	145.2
RC101-30-70	345.7	229.4	435.9	523.3	110
RC101-50-50	160.7	263.2	472.6	730.3	141.6
RC101-70-30	85	347.4	702.7	1040.2	194.7
RC101-90-10	153.3	422.7	468.1	930.5	243.6
Mean	218.8	256.5	451.0	665.3	147.0
(T-test) *P*-values versus 10		0.39936	0.00759	0.00056	0.26701
(T-test) *P*-values versus 25			0.00047	0.00003	0.00155
(T-test) *P*-values versus 50				0.00003	0.000004
(T-test) *P*-value versus 75					0.000003

Not unsurpirsingly our results showed that dividing the day into many time slices increases the execution time, because the algorithm must continuously update the current solution. The *P*-values (Table 7-6) indicate that the EIR technique significantly reduces the average execution time.

As a secondary aim of these experiments, we determined whether the cost of the initial solution for known customers is related to the final cost. Table 7-7 lists the initial and final costs in selected instances with different DoD values.

Table 7-7. Relationship between initial and final cost of serving new customers

Instance	***nts***	**Minimum solution**		**Maximum solution**	
		Initial cost	**Final cost**	**Initial cost**	**Final cost**
C101-30-70	10	407	1646	399	1771
C101-30-70	75	409	1771	397	1922
C101-50-50	25	562	1639	572	1784
C101-50-50	75	579	1649	573	1819
R101-50-50	10	543	1019	537	1069
R101-70-30	75	699	1050	683	1149
RC101-90-10	75	1018	1211	988	1345

As shown in Table 7-7, finding the least-cost solution at the beginning does not guarantee finding the least-cost solution at the end, especially when the DoD is low (i.e., there are many static customers). Thus, a low-quality initial solution offers more opportunities for accommodating more requests than a high-quality solution, as it may not utilise the full vehicle capacity. Therefore, the quality of the first solution of the known customers is unrelated to the quality of the final solution.

Experiments, using the same problem instances, were also conducted to solve the DCVRPs using the IWD algorithm. The same setting was used in both algorithms. A Soil-Management techinque is only used with IWD algorithm because there is no consideration for depth in IWD algorithm. The HCA and IWD results were compared using the EIR technique. The same setting was used in both algorithms. Table 7-8 shows the minimum results of both algorithms.

Table 7-8. Comparison between HCA and IWD results

No	**Instance name-DoD**	**HCA**					**IWD**				
		Min	**#V**	**Avg.**	**#V**	**Time (s)**	**Min**	**#V**	**Avg.**	**#V**	**Time (s)**
1	C101-10-90	**1740**	12	**1818**	11.9	11	1840	11	1947	11.1	68
2	C101-30-70	**1637**	11	**1704**	11	22	1822	11	1827	11	164
3	C101-50-50	**1506**	10	**1560**	10.2	52	1560	10	1599	10.2	507
4	C101-70-30	1343	10	1405	10.1	146	**1329**	10	**1374**	10.1	1303
5	C101-90-10	1176	11	1229	11.1	276	**1086**	11	**1148**	11.2	2921
6	R101-10-90	**1028**	8	1100	8.3	114	1045	8	**1096**	8.3	966
7	R101-30-70	**1051**	8	**1112**	8.1	118	1084	8	1139	8.5	956
8	R101-50-50	**1056**	8	**1105**	8.3	147	1075	8	1125	8.5	1236
9	R101-70-30	**1030**	8	**1090**	8	186	1060	9	1106	8.3	1662
10	R101-90-10	990	8	1046	8.3	299	**984**	9	1046	8.5	2928
11	RC101-10-90	**1265**	**10**	**1324**	9.4	145	1320	10	1396	10	1176
12	RC101-30-70	**1263**	**9**	**1354**	9.5	110	1290	10	1342	10.1	765
13	RC101-50-50	**1200**	**9**	**1324**	9.5	142	1259	10	1368	9.8	1140
14	RC101-70-30	**1215**	**9**	**1318**	9.7	195	1296	10	1367	10.4	1660
15	RC101-90-10	**1200**	**9**	**1282**	9.1	244	1211	9	1269	9.3	2029
	Average	1247	9.3	1318	9.5	147	1284	9.6	1343	9.7	1299
	P-values using t-test						0.03094	0.10381	0.09833	0.06321	0.00004

There is a significant difference between the HCA and IWD minimum results as listed in Table 7-8. In terms of minimum cost, the HCA outperformed the IWD algorithm in 12 out of 15 instances. Furthermore, the execution time was significantly shorter in HCA than in IWD. This suggests that the depth and the additional water cycle stages implemented in HCA play an important role in finding a high-quality solution for DCVRPs.

7.4.3 Experiment 2: Road closure

In this simulation experiment, the events manager was triggered by a road closure, and the time-slice technique was omitted. Figure 7-12 is a flowchart of the simulation experiments.

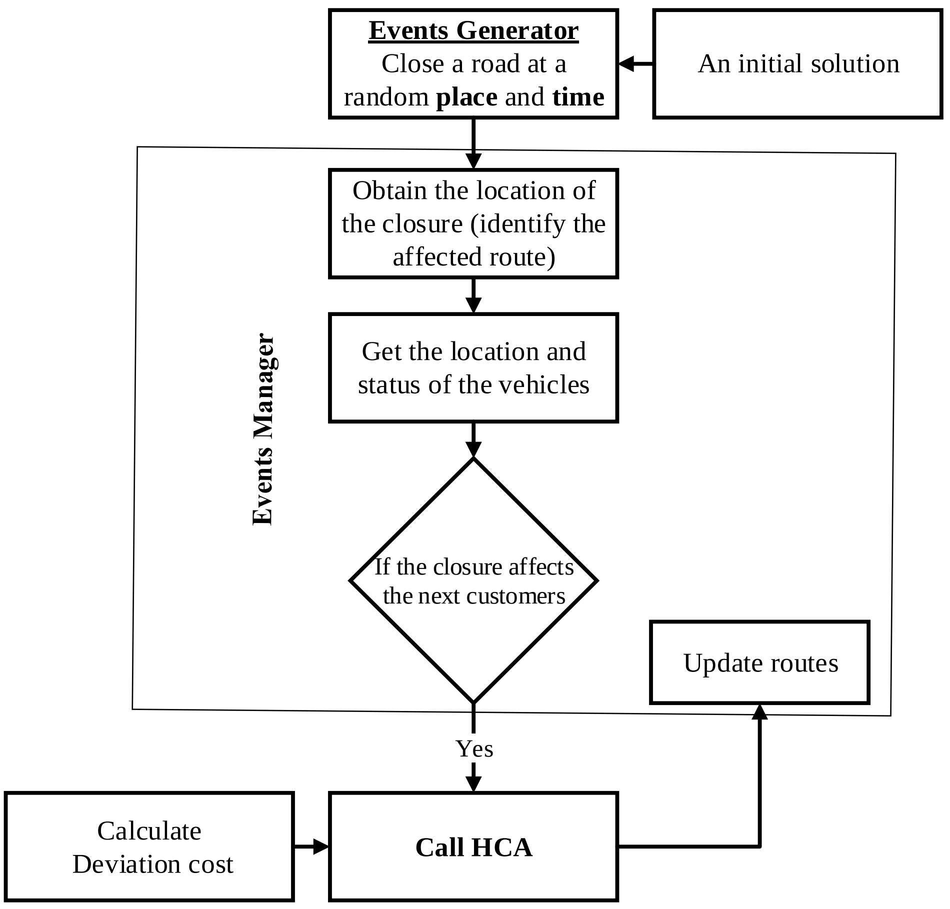

Events Generator
Close a road at a random place and time
An initial solution
Events Manager
Obtain the location of the closure (identify the affected route)
Get the location and status of the vehicles
If the closure affects the next customers
Update routes
Yes
Calculate Deviation cost
Call HCA

Figure 7-12. System procedure in a road closure event

The procedure starts with an initial solution generated by HCA. The events generator closes a road at a random place and time. The events manager locates the closure and assesses the vehicles' status. The HCA then finds a recovery solution and updates the current routes. The performance is evaluated by comparing the costs before and after the road closure, and computing the relative increase in the final cost. The best-known solution is then used as a benchmark.

The number of possible scenarios in the road-closure simulation is endless, and addressing all of them is impractical. Figures 7-13–7-17 illustrate some simple possible scenarios and their avoidance solutions. In these figures, the numbers inside and outside the circles represent a customer's id and demand, respectively. The colour of the circle indicates the status of the customer (blue: served, red: unserved). The numbers on the lines represent the distance/travelling costs. The crossed circle marks the road-closure site. The triangles mark the locations of the vehicles.

A road closure in a route can be avoided by different approaches: re-ordering the sequence of the remaining customers in the affected route only (i.e. solving an asymmetric TSP), re-allocating (shifting) the affected customer to another route to be served by another vehicle, exchanging (swapping) the affected customer with another customer from a different route that is supposed to be served by a different vehicle, or adding a new vehicle that serves only the affected customer. The shift and exchange operations are not executed if they violate the vehicle's capacities. The advantage of each approach depends on the situation and the event time, and the most suitable approach depends on the deviation cost and disruption rate. In an exchange operation, the number of affected customers is always two because both of the swapped customers are after the swap served by a different vehicle. However, the customer’s order is preserved in the routes. In Figure 7-13, the road was closed after vehicle 1 had passed the closure site, so the schedule does not need updating. Such trivial scenarios were not considered in our experiments.

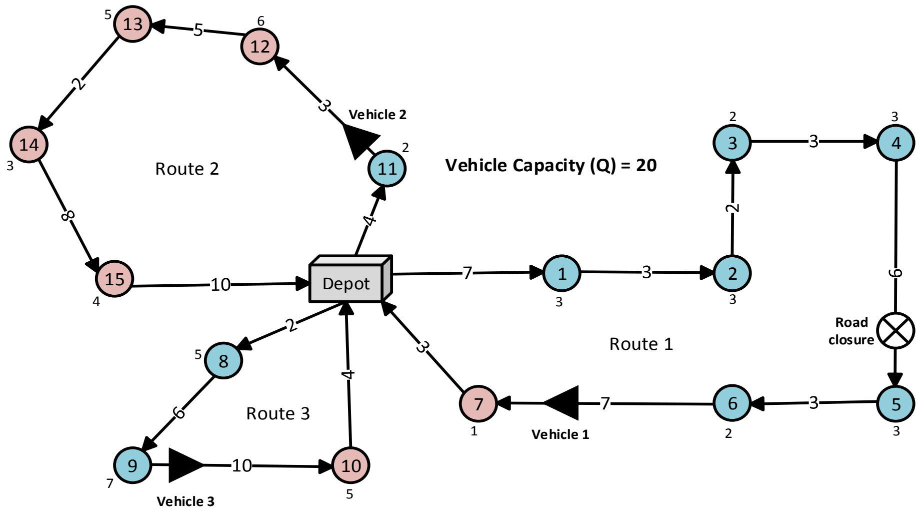


Figure 7-13. Road closure does not affect the schedule

Figure 7-14 shows a possible closure ahead of vehicle 1. The schedule is unaffected because the vehicle can serve customer 7, the last customer on vehicle 1’s route, without rescheduling. However, the vehicle must find an alternative path back to the depot (represented by the dashed line). This simple scenario was also excluded from our experiments.

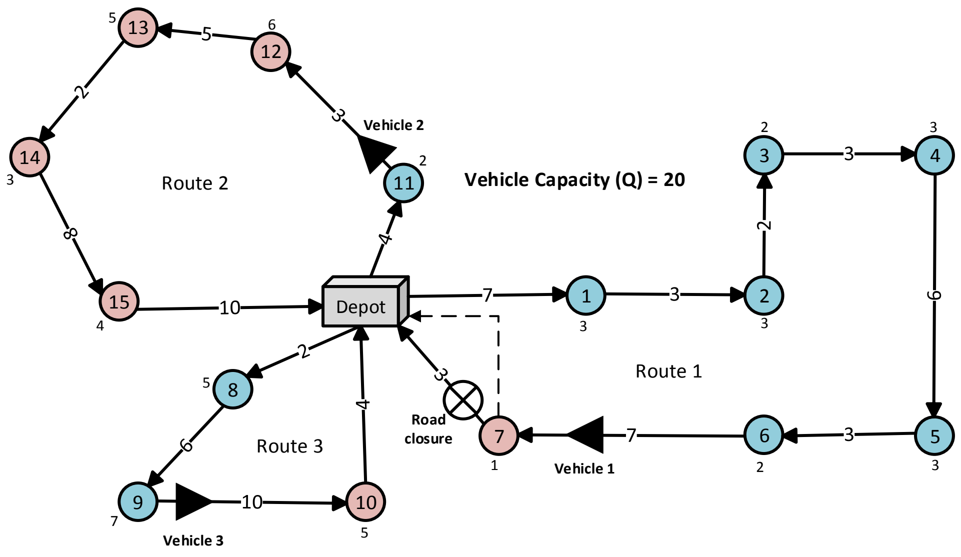


Figure 7-14. Schedule is unaffected by the road closure

In Figure 7-15, a road closure occurs in route 1 between customers 1 and 2. This closure affects the distribution schedule and the customers, especially that of customer 2 (the affected customer).

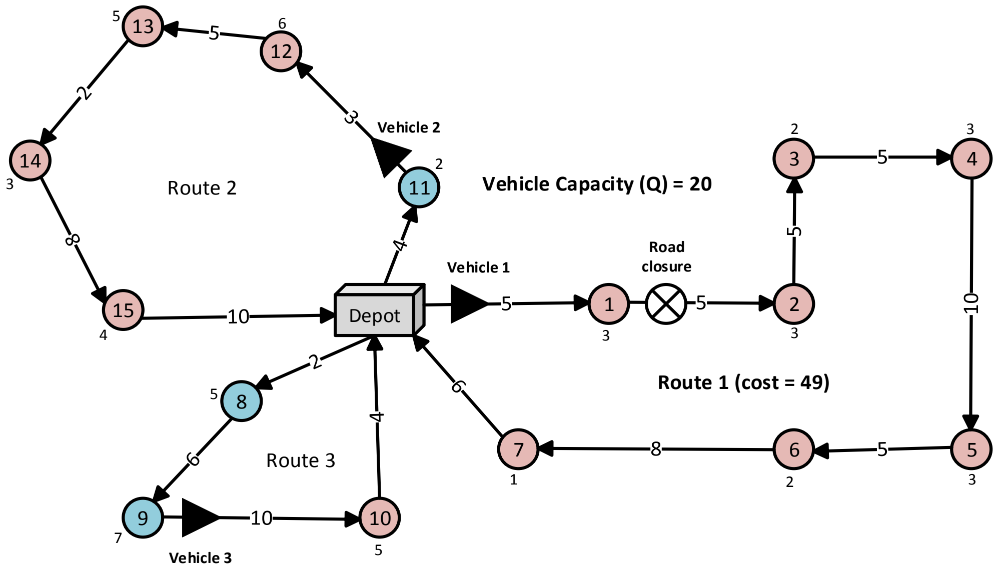


Figure 7-15. Road closure forces a re-scheduling of a customer's order

In this scenario, the procedure may simply reorder Route 1 (the customer's sequence). Two possible re-ordering solutions are illustrated in Figure 7-16.

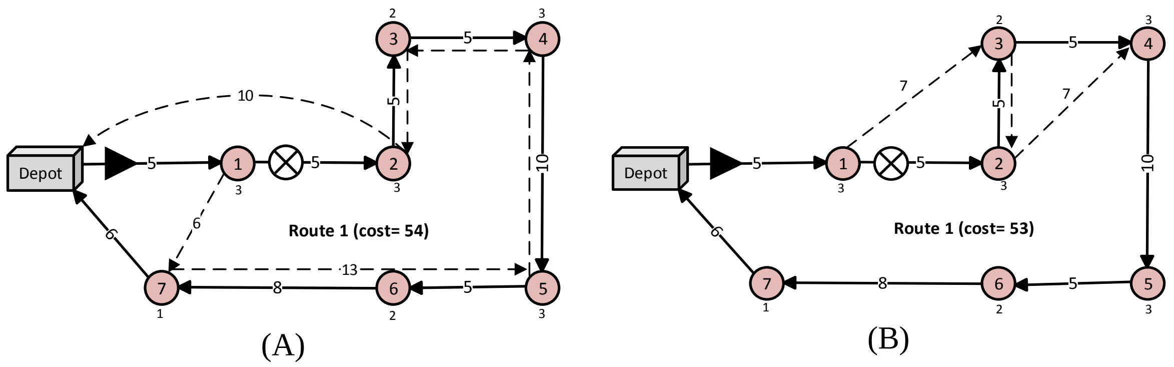


Figure 7-16. Two possible solutions for a road closure blocking access to customer 2.

Whether we choose route *A* or *B*, in Figure 7-16, depends on the deviation cost and the disruption rate (both should be reduced as far as possible). Calculating the total cost of the two solutions, we find that solution *B* is more economical than solution *A*. The deviation cost, computed by Eq. (7.1), is 5 for solution *A* and 4 for solution *B*. The disruption rates in solutions *A* and *B* are 6 and 3, respectively. Therefore, in this scenario solution *B* is selected as the recovery schedule.

Figure 7-17 shows a closure on route 2 between customers 11 and 12. In this case, re-ordering may be ineffective as the vehicle has already served most of the customers. Provided that the capacity constraints are unviolated, the affected customers can be shifted to the route of another nearby vehicle.

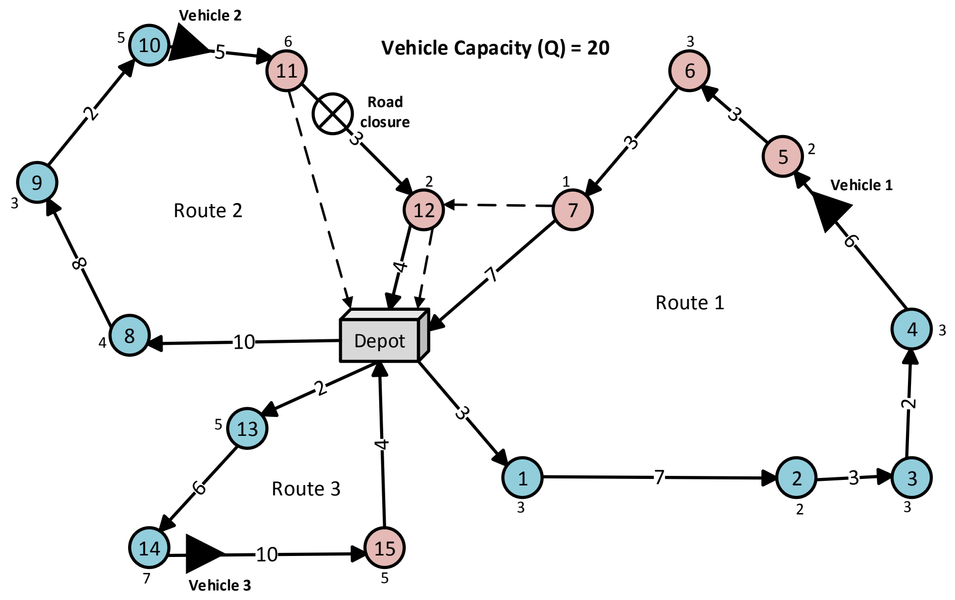


Figure 7-17. Re-allocating customers between vehicles

In Figure 7-17, customer 12 could be served by vehicle 1 without violating the capacity constraint, and vehicle 2 can return to the depot after serving customer 11. Alternatively, customers 12 and 7 can be exchanged such that vehicle 1 serves customer 12 and vehicle 2 serves customer 7. This swap must consider the customers' demands and the vehicle capacities. Obviously, all of the previous examples could be rectified by adding a new vehicle that serves only the affected customer.

7.4.4 Computational Results

To gain insight into the quality of the solutions obtained, the HCA performance was evaluated empirically on several static CVRP instances with different structures. In these experiments, we assumed that a single road may close in any path ahead of a vehicle executing its schedule. To this end, we removed an edge connecting two customers in the original schedule. The road closure was solved by four operations: re-ordering, shifting, exchanging, and deploying a new vehicle. In each operation, the deviation cost (DC) and disruption rate (DR) were calculated by Eqs. 7.1 and 7.2, respectively. The DR is always one for adding a new vehicle and two for exchanging two customers. The simulations were executed 100 times in each instance. As the shift and exchange operations may be excluded by the capacity constraint, the results were retained only when all operations were permissible to facilitate the comparison of results. It is worth noting that because road-closure events were not explicitly considered in previous studies, a result comparison with previous studies is not possible.

In the first instance (A-n32-k5), the cost of the best-known solution was 784, and the vehicle capacity was 100. Figure 7-18 shows the best-known solution in this instance with a road closure between nodes 24 and 5. The stars indicate the current vehicle locations, and the filled and open circles indicate the locations of the served and unserved customers, respectively. The site of the road closure is marked by a blue cross.

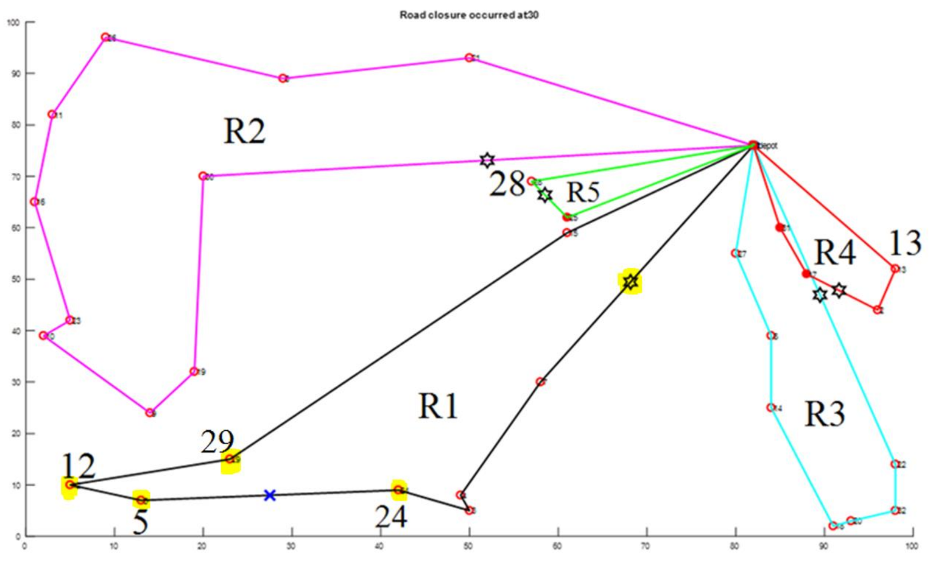


Figure 7-18. A-n32-k5 instance with a road closure occurring at time 30

In Figure 7-18, the solution is minimised by re-ordering customer 5 (the affected customer) and customer 12. After serving customer 24, the vehicle visits customer 12 before visiting customer 5. The final cost is 785, one unit higher than the original cost (784). The number of affected customers is one. The cost of the shift operation, i.e., serving customer 5 after serving customer 28 (in route 5), is 931. The exchange operation, achieved by swapping customer 13 (from route 4) with customer 5, costs 1071. Finally, the cost of adding a new vehicle is 979 units. Figure 7-19 shows the best-known solution in the A-n32-k5 instance with a road closure between nodes 30 and 19 on route 2.

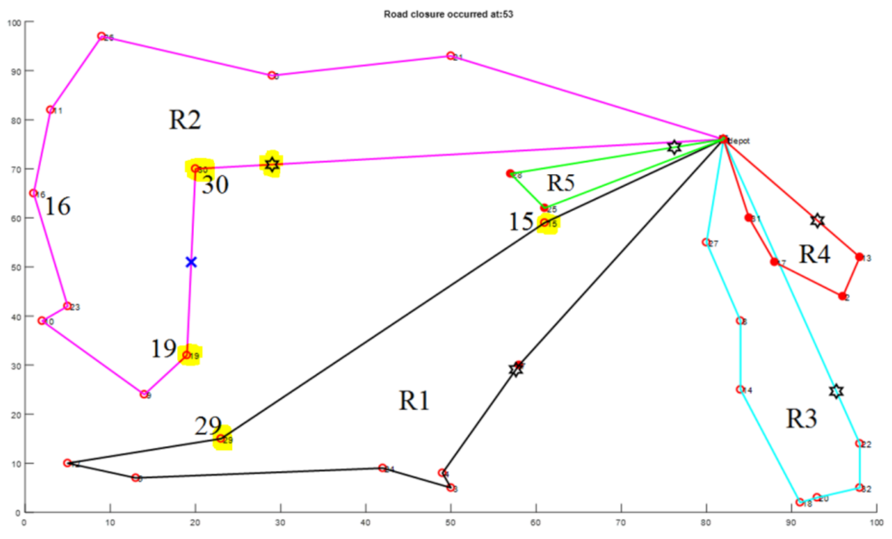


Figure 7-19. A-n32-k5 instance with a road closure occurring at time 53

In Figure 7-19, the road closure is optimally avoided by shifting customer 19 to route 1 between customer 29 and 15 (1-…-5-12-29-19-15-1). The cost of this solution is 792, whereas the re-ordering approach costs 820. The exchange operation, which swaps customer 15 with customer

19, costs 847 (1-...-4-3-24-5-12-29-19-1, and 1-30-15-9-10-23-16-11-26-6-21-1). Finally, adding a new vehicle costs 937. Table 7-9 shows the results in the A-n32-k5 instance for different road closure events.

Table 7-9. Results of A-n32-k5 instance with different road closures

No	Re-order			Shift			Exchange		New vehicle		Time(s)
	Cost	DC	DR	Cost	DC	DR	Cost	DC	Cost	DC	
1	**785**	1	3	790	6	1	810	26	838	54	0.2
2	**807**	23	4	893	109	4	1042	258	942	158	0.21
3	**785**	1	5	895	111	6	908	124	935	151	0.18
4	**785**	1	3	897	113	2	923	139	951	167	0.19
5	816	32	2	**808**	24	2	923	139	858	74	0.18
6	**785**	1	3	924	140	4	1024	240	979	195	0.19
7	**794**	10	6	895	111	6	908	124	935	151	0.2
8	816	32	2	**811**	27	1	923	139	858	74	0.19
9	816	32	2	**799**	15	3	803	19	858	74	0.2
10	**787**	3	2	879	95	4	1177	393	933	149	0.2
11	**785**	1	3	904	120	1	923	139	951	167	0.19
12	**807**	23	4	893	109	3	1042	258	942	158	0.21
13	**790**	6	3	873	89	5	972	188	928	144	0.19
14	**785**	1	3	931	147	3	1071	287	979	195	0.19
15	**801**	17	2	897	113	4	859	75	939	155	0.2
16	**790**	6	3	857	73	7	972	188	928	144	0.2
17	**816**	32	8	893	109	6	910	126	951	167	0.43
18	**801**	17	2	893	109	5	859	75	939	155	0.19
19	**787**	3	2	863	79	6	936	152	933	149	0.18
20	**790**	6	3	860	76	6	972	188	928	144	0.18
Average	**796.4**	**12.4**	**3.3**	872.8	88.8	4.0	947.9	163.9	925.3	141.3	0.21

As evident in Table 7-9, the re-ordering operation minimised the solution in 17 out of 20 cases, while the shift operation minimised the solution in 3 cases. Re-ordering also minimised the average cost, DC and DR values. In the B-n50-k7 instance, the cost of the best-known solution was 741, and the vehicle capacity was 100 (see Table 7-10).

Table 7-10. Results of B-n50-k7 instance with different road closures

No	Re-order			Shift			Exchange		New vehicle		Time(s)
	Cost	DC	DR	Cost	DC	DR	Cost	DC	Cost	DC	
1	**747**	6	6	856	115	6	883	142	859	118	0.32
2	**745**	4	2	828	87	2	925	184	847	106	0.31
3	**747**	6	6	837	96	4	912	171	865	124	0.35
4	**746**	5	5	836	95	5	921	180	863	122	0.33
5	**747**	6	4	766	25	3	934	193	819	78	0.3
6	**745**	4	6	837	96	5	887	146	863	122	0.32
7	**745**	4	6	861	120	5	905	164	863	122	0.33
8	**750**	9	4	802	61	5	751	10	844	103	0.3
9	**750**	9	4	761	20	5	903	162	822	81	0.34
10	**748**	7	2	803	62	3	762	21	849	108	0.29
11	**747**	6	6	863	122	4	912	171	865	124	0.33
12	**750**	9	2	822	81	2	906	165	835	94	0.29
13	**747**	6	6	833	92	6	883	142	859	118	0.31
14	**747**	6	6	833	92	3	906	165	861	120	0.33
15	**747**	6	6	859	118	3	906	165	861	120	0.33
16	**745**	4	2	803	62	4	761	20	847	106	0.31
17	**747**	6	5	779	38	6	760	19	804	63	0.33
18	**743**	2	2	805	64	1	894	153	841	100	0.3
19	**748**	7	2	752	11	4	760	19	833	92	0.31
20	**752**	11	6	770	29	6	952	211	825	84	0.33
Average	747.2	6.2	4.4	815.3	74.3	4.1	871.2	130.2	846.3	105.3	0.32

Among the four operations, re-ordering again minimised the costs, DCs and their averages. However, the average DR was minimised by the shift operation. In the next instance (E-n51-k5), the cost of the best-known solution was 521, with a vehicle capacity of 160. Figure 7-20 shows a road closure between nodes 48 and 5 in the E-n51-k5 instance.

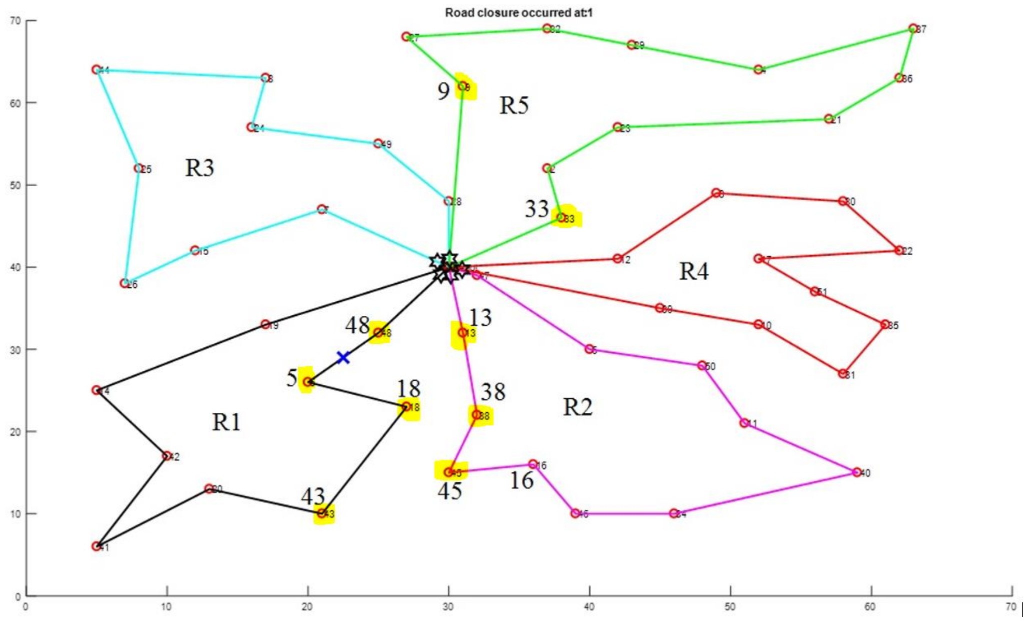


Figure 7-20. E-n51-k57 instance with a road closure occurring at time 1

In Figure 7-20, the road closure is best solved by exchanging customers 38 and 5 (total cost = 533). The vehicle taking route 1 will serve customers 1-48-38-18-43-20-...-1, while that of route 2 will serve 1-13-5-45-16-…-1. The re-ordering and shift approaches cost 543 and 548, respectively (where the latter serves customer 5 after customer 33 in route 5). The cost of serving node 5 by a new vehicle is 548. Table 7-11 shows the obtained results in different road closure events.

Table 7-11. Results of E-n51-k5 instance with different road closures

No	**Re-order**			**Shift**			**Exchange**		**New vehicle**		**Time(s)**
	Cost	**DC**	**DR**	**Cost**	**DC**	**DR**	**Cost**	**DC**	**Cost**	**DC**	
1	**530**	9	2	531	10	2	540	19	548	27	0.28
2	**526**	5	2	543	22	3	543	22	562	41	0.32
3	535	14	4	**531**	10	2	540	19	548	27	0.3
4	**541**	20	3	543	22	3	543	22	562	41	0.28
5	**535**	14	2	569	48	2	569	48	578	57	0.3
6	**526**	5	2	583	62	6	571	50	601	80	0.32
7	**528**	7	3	580	59	4	680	159	581	60	0.29
8	535	14	2	**533**	12	5	569	48	578	57	0.28
9	**526**	5	2	540	19	6	563	42	577	56	0.31
10	**543**	22	4	574	53	4	608	87	581	60	0.31
11	**543**	22	6	548	27	7	533	12	548	27	0.31
12	**523**	2	10	576	55	6	585	64	581	60	0.34
13	**528**	7	3	580	59	4	595	74	581	60	0.31
14	**536**	15	8	580	59	4	580	59	581	60	0.37
15	**536**	15	9	574	53	3	599	78	576	55	0.4
16	**542**	21	8	551	30	8	582	61	573	52	0.35
17	**537**	16	6	580	59	4	580	59	581	60	0.32
18	**529**	8	2	540	19	6	563	42	577	56	0.35
19	**526**	5	2	543	22	3	557	36	562	41	0.34
20	**541**	20	3	543	22	3	557	36	562	41	0.3
Average	533.3	12.3	4.2	557.1	36.1	4.3	572.9	51.9	571.9	50.9	0.32

As shown in Table 7-11, the re-ordering operation minimised the solution in 18 out of 20 cases. It also minimised the average cost, DC, and DR.

The cost of the best-known solution in the P-n65-k10 was 792, with a vehicle capacity of 130. The results of different road closure events in this instance are displayed in Table 7-12.

Table 7-12. Results of P-n65-k10 instance with different road closures

No	**Re-order**			**Shift**			**Exchange**		**New vehicle**		**Time(s)**
	Cost	**DC**	**DR**	**Cost**	**DC**	**DR**	**Cost**	**DC**	**Cost**	**DC**	
1	**803**	11	2	816	24	4	826	34	837	45	0.44
2	**807**	15	4	831	39	3	830	38	842	50	0.43
3	**806**	14	6	810	18	9	816	24	848	56	0.44
4	**797**	5	6	843	51	5	812	20	851	59	0.44
5	**811**	19	6	827	35	6	851	59	856	64	0.46
6	**803**	11	2	818	26	3	811	19	837	45	0.55
7	810	18	2	**797**	5	1	793	1	816	24	0.43
8	**807**	15	4	822	30	4	844	52	835	43	0.47
9	**795**	3	2	821	29	1	801	9	824	32	0.46
10	**803**	11	2	819	27	1	816	24	819	27	0.49
11	**811**	19	3	831	39	2	809	17	832	40	0.47
12	**795**	3	2	804	12	4	801	9	824	32	0.44
13	**801**	9	4	832	40	5	813	21	856	64	0.43
14	**803**	11	2	808	16	3	795	3	820	28	0.44
15	**795**	3	2	812	20	2	801	9	824	32	0.46
16	**803**	11	2	835	43	2	811	19	837	45	0.43
17	**795**	3	2	817	25	1	801	9	824	32	0.49
18	**797**	5	2	813	21	7	831	39	845	53	0.5
19	809	17	5	**797**	5	1	809	17	816	24	0.45
20	**799**	7	2	834	42	3	812	20	848	56	0.44
Average	802.5	10.5	3.1	819.4	27.4	3.4	814.2	22.2	834.6	42.6	0.46

Again, the re-ordering operation minimised the solutions in 18 cases, and yielded the minimum average cost, DC, and DR.

In the last instance (F-n72-k4), the best-known solution cost 237, and the vehicle capacity was 30000. Table 7-13 shows the obtained results in different road closure events.

Table 7-13. Results of F-n72-k4 instance with different road closures

No	Re-order			Shift			Exchange		New vehicle		Time(s)
	Cost	DC	DR	Cost	DC	DR	Cost	DC	Cost	DC	
1	**238**	1	4	252	15	3	269	32	257	20	0.49
2	**242**	5	6	247	10	1	245	8	251	14	0.53
3	**241**	4	7	247	10	1	245	8	251	14	0.61
4	**238**	1	3	264	27	12	293	56	274	37	0.55
5	**238**	1	4	255	18	3	261	24	261	24	0.52
6	**240**	3	5	250	13	1	270	33	254	17	0.51
7	**238**	1	2	255	18	3	280	43	261	24	0.47
8	**241**	4	4	255	18	5	282	45	262	25	0.5
9	**239**	2	4	255	18	3	261	24	261	24	0.61
10	**239**	2	2	247	10	1	266	29	251	14	0.65
11	**241**	4	3	247	10	4	253	16	256	19	0.62
12	**238**	1	2	255	18	9	267	30	262	25	0.5
13	**238**	1	4	261	24	10	267	30	268	31	0.72
14	**241**	4	5	247	10	4	253	16	256	19	0.48
15	**239**	2	2	247	10	1	245	8	251	14	0.52
16	**239**	2	2	252	15	3	269	32	257	20	0.47
17	**238**	1	3	250	13	1	270	33	254	17	0.48
18	**241**	4	7	251	14	10	266	29	268	31	0.59
19	**240**	3	6	250	13	1	270	33	254	17	0.55
20	**238**	1	4	259	22	9	285	48	270	33	0.6
Average	239.4	2.4	4.0	252.3	15.3	4.3	265.9	28.9	259.0	22.0	0.55

As confirmed in Table 7-13, re-ordering was the best operation in all cases. Reordering also minimised the average cost and DC values.

According to the above analyses, road closures are best overcome by trying to reorder the customers in the affected route. The second-best choice is the shift operation, which serves the affected customer by diverting another nearby vehicle. Furthermore, in all instances, sending a new vehicle to serve the affected customer was better than exchanging two customers between two vehicles. Overall, the results confirm that HCA can solve road-closure problems and provide high-quality solutions. The reorder operation is easily implemented without affecting other routes. Moreover, unlike the shift and exchange operations, it does not violate the vehicle capacity. Therefore, it is more flexible than the other operations, and is applicable to different road closure events. On the other hand, deploying a new vehicle is suitable if the distribution company wishes to maximise the customer satisfaction, as it minimises the DR among the four operations.

7.5 Summary and Conclusion

In this chapter, the HCA was applied to optimisation problems in dynamic environments, namely, to DCVRPs. Dynamic problems are important in both the research and industrial domains, as they describe many real-world applications. One objective of this study was to evaluate the effectiveness of HCA in problems resembling real-life situations. The DCVRP was executed on different types of events, and the HCA performance was evaluated in computational experiments with two sources of DCVRP dynamism (new customers and road closure).

In the new-customer experiments, new orders arrived during the operation time, and the algorithm was required to optimally integrate these orders into the current schedule. These experiments defined the relationship between the number of time slices per day and the degree of dynamism. The relationship between the number of time slices and solution quality was also clarified. The HCA consistently and effectively covered all new customers in terms of vehicle number and total distance. The initial cost of the known customers was unrelated to the final cost.

We also proposed a new technique that controls the arrival of new customer, preventing the accumulation of orders or long delays in processing the orders. This technique offers an alternative to fixing the number of time slices and obtains better results in some instances. The SDM efficiently preserved the properties of the good solutions between consecutive time slices, which is superior to the re-initialisation strategy. In addition, the Pareto-front technique in the condensation process helped to diversify the generated solutions, and optimised the two objectives (cost and number of vehicles) simultaneously.

According to the experimental results, HCA is capable of finding higher quality solutions for DCVRPs than the IWD algorithm. The HCA outperformed the IWD algorithm because it considers stages of the water cycle (i.e., evaporation and condensation) that largely influence the efficiency of problem-solving and because the depth factor and information sharing facilitate the exploration of stronger solutions, thus improving the solution quality.

Next, the algorithm was applied on several static CVRPs instances in which a road was closed at a random time and location ahead of the vehicle locations. These road-closure experiments aimed to find the best approach that minimises the cost and disruption of deviating from the original schedule. Four operations (re-ordering, shifting, exchanging, and deploying a new vehicle) were designed and tested in different scenarios-based cases. By comparing the algorithm results in the different cases, we confirmed that re-ordering provides a recovery schedule at minimum possible cost.

Overall, the preliminary experimental results supported that HCA is applicable to other dynamic problems, such as time-series classification (see Appendix B). Although this chapter provides valuable information and justification for solving DCVRPs by the HCA, future work should consider VRPs in other dynamic environments, such as dynamic pickup and delivery with time windows in VRPs. Additional constraints, such as multiple depots, a waiting strategy, order rejection policies, or heterogeneous vehicles, could also be imposed. Whether a decentralised system is more effective than a centralised system in DCVRPs is also worth investigating. In this chapter, we addressed only two events (new customers and road closure). Thus, adding additional event sources would raise some interesting problems, such as service delays, order cancellations, or vehicle breakdowns. Further objective functions related to DCVRP could also be considered. Although the EIR technique provided good results, it could be extended to handle the locations of vehicles and the new customers. Different techniques to trigger the events manager such as when the number of new customers reaches a specific value is also worth considering in future work. Finally, it can be concluded that to guarantee the performance and robustness of the HCA, more experiments need to be conducted on instances with many customers and that the closure of two or more roads in a single execution should be investigated.

Chapter 8 Conclusions and Future Research

This chapter summarises and explains the research contributions of this thesis. Furthermore, it explains how this research has answered the proposed research questions. Finally, it discusses the limitations of the current work and proposes directions for future work.

8.1 Summary of the Research

Water-based algorithms are a subclass of NIAs inspired by the natural water cycle. Consequently, these water-based algorithms share particular aspects of their conceptual framework. These algorithms have different abstract levels to describe the natural water activities. Hence, the search strategies of these water-based algorithms are different and dependent on different sources of water activities. Obviously, every water-based algorithm has its own advantages and disadvantages as clarified in Chapter 2. However, as revealed in the literature review, previous water-based algorithms only partially simulated the natural water cycle and omit some important factors of the process. The partial simulation of a natural process limits their capability and performance, especially in terms of exploration and exploitation.

The research presented in this thesis resulted in the development of a new nature-inspired optimisation algorithm called the hydrological cycle algorithm (HCA). HCA is based on the continuous movement of water in the full natural hydrological cycle. The properties of water droplets, activities associated with their movements, and the water cycle stages are incorporated into a computational process.

The initial HCA developed was further improved by taking into account the limitations and weaknesses of prior water-based algorithms. These negative aspects were mitigated by including some features and variables that lead to improving the quality of the solution. In addition, the refinement involved enabling information sharing through direct and indirect communication among the water drops. Such information sharing improved the overall performance and solution quality of the algorithm. Indirect communication was achieved in the flow stage by depositing and removing soil on/from paths and using path-depth heuristics. Direct communication was implemented via the condensation stage and was shown to promote the exploitation of good solutions. The addition of the temperature factor helped control the occurrence of the cycle and thus improved the performance of the algorithm in terms of reducing the execution time. The soil deposition and removal along with depth of the paths also helped to promote the exploration process.

In this thesis, the HCA was designed, implemented, evaluated, and validated on different types of academic benchmarked optimisation problems: the travelling salesman problem, continuous optimisation, vehicle routing, and multi-objective dynamic vehicle routing. These problems are important as they describe many practical applications and are useful for evaluating the performances of new algorithms. By solving these problems, the HCA demonstrated its potential for dealing with other types of optimisation problems. The experimental results were competitive with those of other metaheuristic algorithms, validating the effectiveness of the proposed algorithm. The algorithm was shown to readily escape from local optima solutions and converge towards the global best solution. In conclusion, the proposed algorithm is extendible to other optimisation problems and demonstrates the usefulness of exploiting natural processes in the design of new optimisation algorithms.

8.2 Contributions of the Research

The main contributions of this thesis are presented by answering the following research questions:

Q1. How should we design an optimisation algorithm based on the hydrological water cycle in nature?

In this research, through a deep study of the hydrological water cycle foundation in nature, the correlation between the system elements was analysed and how these elements influence each other to ensure the sustainability of the cycle. We found that the activities and behaviours of the hydrological water cycle can potentially be exploited for building an optimisation algorithm. Thus, water droplets on the ground seek the shortest paths towards the ocean. The water stages are complementary and their interactions help to control the water circulation. The condensation process also provides a motivational concept for information sharing through the collision and merging of water droplets.

Buoyed by this motivation, a conceptual framework that simulates the functionalities of the water-cycle stages was developed. Accordingly, a mathematical model that describes the water activities was constructed to be used as a computational process. Each stage of the cycle has been employed to play a role in this process. By systematically reviewing the well-established NIAs, challenges and limitations were identified and outlined in Chapter 2. These issues were corrected in the HCA to improve the overall performance. The HCA was designed to maximally balance the exploration and exploitation processes. The exploration achieved by including the depth of the paths as another heuristic and allowing soil deposition. The exploitation is promoted by the information sharing and emphasising on the best water drop (i.e. collector) in the next cycles.

The dependence of the parameter values on the HCA performance was analysed using a structured experiment in Section 3.4.6.1. The results showed that the values of the parameter change in a proper way that facilitate the working mechanism of the HCA. The convergence and the exploration ability of the HCA were evaluated on the double-bridge experiments in Sections 3.4.6.1 and 3.4.6.2. It was confirmed that the HCA processes and the chosen parameters were correlated and help the HCA to converge towards the global solution and exploring the search space more broadly. The additional factors led to better results compared to IWD algorithm. The results of these experiments suggested that exploiting all the important factors in a natural process may lead to better performance.

These experiments also confirmed that changing certain design aspects can dramatically improve the algorithm's performance. The HCA provided a solution for different problems within a reasonable timeframe or number of iterations, by virtue of the information sharing and solution improvement methods incorporated in the condensation stage. After testing the algorithm and confirming its high-quality results, we can confidently present a new effective nature-inspired algorithm that is applicable to other NP-hard problems.

Q2. How do the additional stages of the hydrological cycle algorithm influence the quality of the solutions?

In Section 4 of Chapter 3, this question was answered by conducting several controlled simulation experiments and comparing the HCA results with those of the IWD algorithm, which represents a single stage of the water cycle. The similarities and differences between HCA and other similar water-based algorithms were identified, and the implications of these differences on the overall performance were discussed. For example, the IWD algorithm suffers from the inability to make a different selection when nodes have similar probabilities (similar amount of soils on their paths). In HCA, adding the depth factor resolved this issue. In IWD, the soil is removed from the edges quickly and may become negative, and this leads to premature convergence. In HCA, the soil can be removed and deposited and that promotes the exploration capability.

The algorithms were tested in double-bridge experiments. The results confirmed that the HCA more comprehensively explores the search space than the IWD algorithm, confirming the notion that adding additional stages and getting closer to nature would provide a better solution.

Thus, the search capability of the HCA was enhanced by including the depth factor, velocity fluctuation, soil removal and deposition processes, and direct communication. The HCA was also found to avoid stagnation in local optima and premature convergence where the IWD did not. The additional stages were found to enhance the solution quality and general performance of the algorithm.

The results of TSP, COPs, and CVRP experiments revealed the ability of the HCA in providing better solutions quality with fewer iteration numbers compared with established water-based algorithms, especially the IWD algorithm.

Q3. To what extent can the hydrological cycle algorithm solve the travelling salesman problem and the capacitated vehicle routing problem?

The TSP and CVRP are static combinatorial NP-hard optimisation problems that differ in their number of objective functions and constraints. These problems have been extensively studied, and are now regarded as benchmarked problems. The TSP and CVRP are easily understood but difficult to optimise. The TSP was chosen because it is considered to be a standard testbed for optimisation algorithms as it is easily understood and has few constraints, these properties enable an in-depth analysis. With its multiple constraints, the CVRP is a more complex problem than TSP but was chosen because it describes many real-life situations. Moreover, the search space of both of these problems, TSP and CRVP, is easily represented as a fully connected graph, a representation that is consistent with the default input of the HCA. Therefore, the HCA is easily applied to these problems. The benchmark instances for TSP and CVRP used were sourced from the literature.

Investigations of the HCA applied to the TSP and CVRP problems are presented in Chapters 4 and 6 respectively. The algorithm was also tested on structural instances of the TSP and CVRP. These structural instances are geometric shapes that were easy to design and evaluate because their outputs are readily validated. The structural instances designed for this research are also considered to be suitable for future evaluation of other algorithms. The condensation stage of HCA also has been utilised to perform local improvement methods for further enhancements of the solutions in subsequent cycles.

The HCA algorithm has provided high-quality solutions within a reasonable computation time for both problems. The HCA arrived at the optimal (i.e., best-known) solution in 20 of 24 TSP benchmark instances (83.3%), and near-optimal solutions in the other four instances. The HCA also found the best-known solution in 37 of 87 CVRP benchmark instances (43%) and provided near-optimal solutions (deviating by less than 1% from the best-known) in 31 (62%) of the remaining instances. The experimental results, showed that the HCA outperformed several state-of-the-art algorithms and was competitive with other algorithms. The good performance of HCA on these problems is considered to prove its success. Therefore, it is reasonable to conclude that the HCA provides a new approach and effective approach for solving variants of the TSP and CVRP.

Q4. Can the performance of the HCA compete with those of other similar algorithms?

To answer this question, we reviewed the literature on NIAs and classified them using a simple schema (Section 2.2). Certain popular algorithms were discussed in a comprehensive literature review (Chapter 2) that highlighted their features, weaknesses, and strengths.

In each experiment, the performance of the HCA was compared with several state-of-the-art algorithms, especially water-based algorithms. The experimental results and statistical analysis confirmed the strong performance of HCA relative to other nature-inspired algorithms, especially the water-based algorithms. Furthermore, the proposed algorithm provided high-quality solutions for different types of optimisation problems. The comparison results showed that HCA is very competitive with existing NIAs and outperformed them in some cases.

Q5. What factors affect the performance of the hydrological cycle algorithm when solving continuous optimisation problems?

The HCA was evaluated on continuous optimisation problems using standard benchmark mathematical functions (Chapter 5). Test functions for validating the reliability, convergence, and efficiency of any optimisation algorithm are available in the literature. These functions are difficult to solve because of their fitness-landscape structures. They also have many local-optimal solutions, which are difficult to escape once found. To find the global best solution for these functions, the local optima must be avoided by a proper mechanism that explores as much of the search space as possible.

The HCA was adapted to these functions by proposing a new problem representation for COPs. The new representation was necessary to facilitate the search process of the water drops and removes the need to alter the HCA's structure. The HCA performance was evaluated by examining its success rate, the number of iterations and accuracy. The high success rate achieved confirmed the robustness of the HCA. Success was attributed to the using of the depth factor and soil deposition/removal process that enhanced the exploring of the search space. This leads to generate different solutions each iteration.

A mutation operation, which occurs at the condensation stage, was incorporated with the HCA to justify whether adding another component that does not belong to water cycle would affect its performance in term of solution quality. Unlike some algorithms, which rely on the mutation operation to escape local optima, the HCA algorithm provided high-quality solutions even without the mutation. Accordingly, mimicking a natural process alone can provide good solutions without the need to introduce artificial elements. But we must take into account all the important factors that influence the natural process in order to obtain better results.

Q6. Can the hydrological cycle algorithm deal with other NP-hard problems?

The HCA is designed with the aim of efficiently and effectively exploring a search space. The algorithm was tested on four different domains of problem: discrete, continuous, static, and dynamic (see Chapters 4, 5, 6, and 7). The HCA was found to be more efficient than the other established algorithms results in some problems. The empirical findings of experiments using HCA to solve some NP-hard problems suggest that the HCA could be used to solve other optimisation problems, such as job scheduling problems, knapsack problems, quadratic assignment problems, network routing, and various engineering design problems. The experiments undertaken demonstrated how the HCA can be readily and simply customised to tackle problems with different structures. This research also showed that, as an alternative to changing the HCA's structure some problems can be transformed and simplified to different types with different representations that can be solved by the HCA. The flexibility of the HCA is likely due to the algorithm being easy to understand and having few adjustable variables.

According to Ho and Pepyne (2002), who clarified the NFL theorem, it is impossible to create a general-purpose optimisation algorithm for all the problems. However, an algorithm can outperform other algorithms if it is specialised to a particular problem within certain considerations. Based on the proposed classification scheme in Section 3 of Chapter 2, the HCA is classified as *trail-based* algorithm. From the experiments results, it is deduced that HCA, by nature, is more easily applied on routing-related or path-searching problems rather than continuous problems. Therefore, HCA may perform better in such problems.

Q7. To what extent can the hydrological cycle algorithm cope with the dynamic vehicle routing problem?

Solving problems in a dynamic environment is difficult because the variable changes affect the objective functions, that is, the objective functions are time-dependent. The HCA performance in problems with a dynamic environment, is discussed in Chapter 7. Chapter 6 focused in the static CVRP and Chapter 7 extended the work in Chapter 6 by applying the HCA to the DCVRP variant of the CVRP.

Among the various sources of dynamism in DCVRP, the arrival of new customers and a road closure event were selected. Dynamic problems are commonly treated by simply dividing the problem into a series of static problems, which are then solved sequentially. In each static problem, the optimisation algorithm should update the current solution to accommodate the new changes. Auxiliary components are needed to control the events and track the problem status. Accordingly, the simulation environment was created by the events manager and generator, and the HCA updated the solution when required. The important questions are when to divide the problem, which mechanism should divide the problem, and how many static problems should be createds. In the literature, most studies have specified a constant number of time slices per day.

In the event of new customers, the HCA was evaluated on three instances with different structures and various degrees of dynamism. Through these experiments, we identified the relationship between the number of time slices and the degree of dynamism. We also proposed a new technique called the *exponential increment rate*, which triggers the events manager based on the number of arrival customers and time. In many instances, this technique achieved better results than fixing the number of time slices.

The SDM mechanism assumes that the path depths are not affected by water evaporation from the paths. Therefore, the SDM component preserves the soil and depth levels of good solutions. When connected to the HCA, the SDM component propagates the good properties of solutions through the time slices. The DCVRP is a multi-objective problem requiring simultaneous optimisation of the cost and number of vehicles. The Pareto-front technique, which optimises conflicting objectives in a problem, was integrated into the condensation stage of HCA. This technique helped to diversify the generated solutions and provided solutions that optimised both the cost and vehicle-number objectives.

Overall, the experimental results clarified the importance of SDM mechanism, Pareto-front technique, and information sharing and their impact on the results, which should be considered when solving dynamic problems. They further revealed that HCA can provide good-quality solutions to DCVRPs compared to IWD algorithm. Therefore, the HCA is an effective alternative approach for solving other dynamic-environment problems.

8.3 Which Algorithm Should We Choose for Solving a Problem?

In this thesis, we categorised the optimisation problems and overviewed their characteristics. Many algorithms have been successfully used in many domains. However, there is no easily discernible guideline on how to select an algorithm. Different problems, even different instances of the same problem, might be solved more effectively using different algorithms.

A suitable choice of an algorithm improves quality of the results and reduces computational effort. However, choosing between different optimisation algorithms is a difficult task. The questions "why does one algorithm work better than another?" or "why does this algorithm work at all?" are also hard to answer. As optimisation algorithms are stochastic, analysing their behaviours or proving their efficacy is non-trivial, but the reasons for choosing an algorithm can be determined. In order to choose the most suitable algorithm for a particular problem, we must fully understand the problem itself and analyse it by answering these questions:

- What are you optimising?
- What are the characteristics of the problem?
- Is a simple brute-force technique good enough?
- Are the problem variables continuous or discrete? Or What is the type of the problem domain?
- How many variables exist? Or what size of search space?
- What is the best way to represent the search space of the problem?
- What are objective functions? Does the problem have multiple objectives?
- Does the problem have constraints?

Quoting the Greek philosopher Socrates: "understanding a question is half an answer." By answering the previous questions, we can extend our understanding of the problem and choose the appropriate and effective representation for the problem domain (i.e., its search space). Hence, understanding the type of design space that is being searched influences the choice of algorithm to solve the problem. However, as defined through literature, some problems are complicated or rigid and can be represented only in a specific way; others are more flexible. Consequently, some algorithms can solve a problem after minor adjustments, some require a major restructuring. Otherwise, the problem search space must be represented in a different way to suit the algorithm working mechanism. Thus, each algorithm operates by a unique mechanism, and a proper representation of the problem will improve the results. Alternatively, it is better to choose a different algorithm that matches with the problem search space. For example, some algorithms work efficiently only on a certain domain space, either continuous or discrete variables but not both. An algorithm that operates on both discrete and continuous domains may perform much better in one domain than the other. For a multi-model problem, where the search space may have many local-optimal solutions, a sophisticated algorithm (i.e., more complex implementation) may be an advantage as it has the ability to escape these local solutions.

In summary, it is not possible to choose the best algorithm for a given problem until the nature of that problem is well understood. The best algorithm to use depends upon the type of search space that has been defined because each algorithm has situations in which it is preferable and more applicable than other algorithms. While the best algorithm for a specific problem can be selected by "trial and error", this is not the best approach. A better approach to consider previous studies of similar problems and through a simple analysis determine the most suitable algorithm that match the unsolved problem. Furthermore, our classification of the optimisation algorithms (i.e., trail-based or point-based algorithms) discussed in Chapter 2 provides useful guidance for researchers to facilitate the algorithm choice for a particular problem. The representations of search space are more correlated with the nature of the algorithm process. Based on these differences, we can understand that trail-based algorithms are suitable for discrete (e.g. combinatorial, or path finding) problems in which the node sequence is important. In contrast, point-based algorithms are suitable for continuous domain problems, in which the values of the points are more important than their ordering.

8.3.1 Why Should We Choose HCA?

The main objective of the HCA proposal was to provide researchers and scholars with a new powerful method for problem-solving. The algorithm's performance was evaluated on a number of different domains. This evaluation determined the validity of the algorithm's concept and whether the algorithm delivered superior performance. To ensure that the HCA was competitive, with other algorithms, it was designed to fulfil certain criteria and metrics intended to guarantee computational complexity, efficiency, accuracy, and generalisation. Although the HCA shares some general characteristics with other swarm algorithms, it differs from other algorithms in the following aspects:

1. HCA performs a number of cycles, each with a number of iterations (i.e., there are multiple iterations per cycle). This is advantageous in certain tasks (e.g., solution

improvement methods), where the update is performed at every cycle instead of every iteration to reduce the computational effort. In addition, the system tends to self-organise under the cyclic nature of the algorithm, as the system converges to the solution through feedback from the environment and internal self-evaluation. The cycle is controlled by the temperature variable, which is updated according to the quality of the solution obtained in every iteration.

2. The flow of the water drops is controlled by path-depth and soil-amount heuristics (i.e., by indirect communication). The path-depth factor affects the movement of the water drops and discourages them from choosing the same paths. Therefore, the movement of the water drops is guided by two heuristics rather than one.
3. The water drops can gain or loss a portion of their velocity, encouraging competition among them. This prevents one water drop sweeps the remaining drops, because the high-velocity water drops can carve more paths (i.e., remove more soils) than the slower drops.
4. Soil can be deposited and removed, which helps to avoid premature convergence and promotes the spread of water drops in different directions (enhanced exploration). In addition, embedding the quality of solution with carried soil in water droplets supports using previous experiences of a water to improve its future movements.
5. The HCA performs information sharing (direct communication) among the water drops in its condensation stage. Moreover, selecting a collector water drop is an elitism technique, and encourages exploitation of good solutions. This stage also improves the quality of the obtained solutions by reducing the number of iterations.
6. The condensation is a problem-dependent stage that can be customised according to the problem specifications and constraints.
7. The evaporation process is a selection technique that helps the HCA escape from local optimal solutions. Evaporation is invoked when the quality of the obtained solutions does not improve over a certain number of iterations.
8. The precipitation stage reinitialises the variables and redistributes the water drops in the search space to generate new solutions.
9. The algorithm is flexible and easily customised to different types of problems.
10. The algorithm is easily understood and has only a few easily determinable variables.
11. The HCA does not depend on the quality of the initial solution, because it iteratively builds a solution from scratch. In contrast, some algorithms require an initial solution which they must optimise as far as possible.

12. The natural water system has no central control system. The water drops in HCA are autonomous and generate solutions by a self-organisation behaviour that automatically converges towards the global best solution. To solve a problem, the water drops interact, collaborate, and coordinate their activities by indirectly communicating through the environmental modifications. The optimal solution emerges from the set of water drop solutions. The HCA is a decentralised system that depends on positive/negative feedback through the built-in cycling concept.
13. Natural systems solve complex real-life problems and usually retain the knowledge gained from previous experience. In the HCA, the previous experiences (i.e., solution quality) of the water drops are embedded in their carried soil amounts. Through this memory technique, the water drops draw on their previous solutions to make their future decisions.

The condensation, evaporation and precipitation stages (points 5, 6 and 7 above) distinguish HCA from other water-based algorithms, including IWD and WCA. The distinct characteristics of HCA are important for proper balancing of exploration and exploitation, and promote the convergence towards the global solution. The water-cycle stages are complementary, and improve the overall performance of the HCA.

8.3.2 How Do I Apply HCA to a Problem?

This thesis constructed the fundamental version of HCA, and applied it to different optimisation problems. For HCA application to other problems, we must fully understand the problem structure and represent the problem appropriately. The HCA can then be implemented based on the pseudocode provided in Chapter 3. The parameters can be tuned to suit the problem search space (domain). Figure 8-1 summarises the problem-solving steps of the HCA.

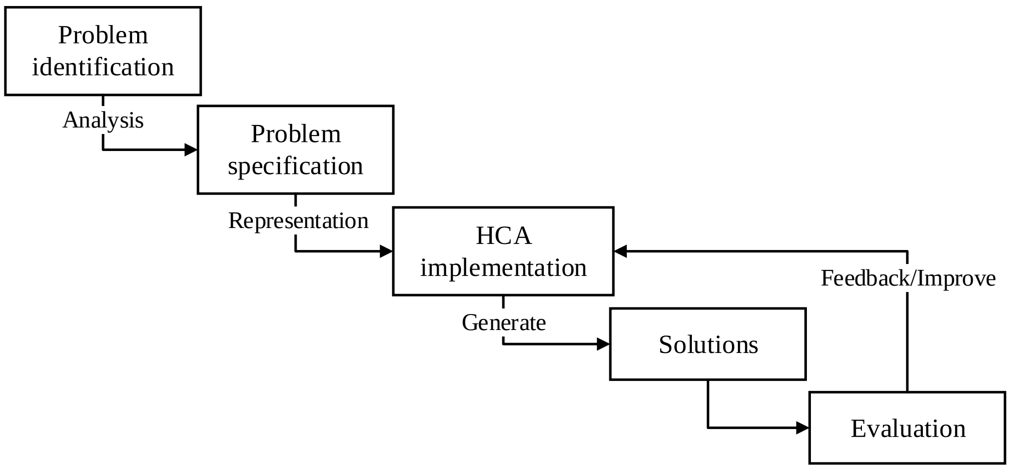


Figure 8-1. Steps for problem-solving

The HCA is improved by proper implementation and appropriate parameter values. The algorithm can be adjusted after evaluating the obtained solutions.

8.4 Thesis Recommendations: Guidelines for Creating a New Algorithm

Nature-inspired algorithms are new computational paradigms inspired by natural processes. When formulating an optimisation algorithm based on a natural process, it seems appropriate to ponder “what is the purpose of the natural process in nature?” and “what is being optimised by this process?” Many natural systems can be potentially adopted in designing new optimisation algorithms. Sophisticated computational algorithms are urgently demanded, especially with the emergence of big data problems. Despite the success of existing NIAs, some crucial issues and challenges must be considered when designing a new algorithm based on a natural process. Ignoring these issues will compromise the algorithm performance.

First, the exploration and exploitation processes must be appropriately controlled. An effective algorithm requires a proper balance between exploration and exploitation. Although finding the balance state for optimal performance is challenging in practise, an approximate balance state can be achieved by adjusting the parameters and properly controlling their values. If the exploration is excessive, the algorithm may miss the global solution or require much computational effort (i.e., converge too slowly). In contrast, excessive exploitation will focus the search on certain areas, reducing the searching efficiency and perhaps leading to premature convergence. A priori knowledge of the problem domain is useful for balancing the exploitation and exploration, because both processes depend on the type of problem.

Second, the diversity of the generated solutions is very important. The diversity can be simply increased by incorporating random components or variables into the algorithm. Randomness aids the escape from local optima, but excessive randomness slows the convergence and wastes the computational efforts. Therefore, imposing the correct amount of randomness is crucial. A deterministic (completely non-random) algorithm has no exploration ability, so generates the same solutions at every iteration. Third, because the algorithm performance depends on the parameter values, the parameters must be tuned and controlled to obtain good results.

Fourth, the performance can be enhanced by allowing interactions among the entities. The interaction guides the entities toward promising solutions and reduces the required number of iterations. In this sense, interaction offers an alternative mechanism for exploitation. After adding some randomness, interaction can also lead to exploration. Furthermore, the algorithm performance can be enhanced by choosing the best problem representation that maximises the working mechanism of the entities. The selection mechanism is crucial for maintaining good entities (i.e., increasing their survival chances) in subsequent iterations.

Finally, an algorithm can be improved by self-adaptation, self-organisation, and feedback. The entities learn from their previous experiences, which enhances their performance in later iterations. Self-adaptation can be achieved in different components of the algorithm, such as generation of the initial entities, the exploration–exploitation balance, setting the parameter values, and the selection mechanism.

8.5 Limitations of the Current Work

Although this research achieved its objectives, there are some unavoidable limitations due to the time constraints. These limitations are listed below.

- The HCA performed well on medium-sized TSP and CVRP problems, but its performance on large-scale instances (with numerous nodes) was not assessed. If completed, this assessment would help us to determne the relationship, if any, between performance and instance size.
- In the TSP and CVRP problems, we considered only Euclidean distances, and rounded the costs to integer values. The algorithm performance could be further evaluated using other distance and cost measures.
- The HCA was not tested on high-dimensional COPs (functions with many variables). This limitation was imposed by the problem representation, in which the search area increases with increasing number of variables. Furthermore, the HCA was tested only on unconstrained single-objective functions.
- As no benchmark dataset is available for DCVRPs, a comparison of HCA with other algorithms in was not possible in these instances. In addition, many factors and assumptions influence the quality of the results, but have not been clarified in previous research.
- To minimise the factors affecting the algorithm's performance, the evaporation rate was controlled only by random values, not by a dedicated technique. Similarly, the selection mechanism of the water drops was implemented only using the roulette wheel technique.
- Unfortunately, exploring and analysing every NIA in detail is infeasible. To identify the strengths and weaknesses of NIAs, we must implement and test them on benchmark problems. This research compared them solely from a conceptual structure-based perspective.

8.6 Future Research Work

In addition to the future works that been described at the end of the chapters. This section describes further possible directions for future research, built on the contributions and limitations of this thesis.

First, although the HCA successfully solved different problems in this study, it remains in its preliminary stages, and there is scope for further improvements. To reduce the number of parameters and the complexity of the HCA, our initial standard model excluded some details and factors existing in the natural water cycle. The model could be improved by adding other natural details (i.e., embedding other mechanisms and factors) of the cycle such as the width of the path, or by considering an additional indispensable process in the cycle.

The effect of the technique used on the quality of the solution obtained needs to be evaluated in a comparative study of different techniques for selecting the evaporated water drops. Other techniques for avoiding local optimal solutions and stagnation, maintaining the exploration–exploitation balance, or avoiding premature convergence, should also be considered.

The parameter settings and specifications employed in this research were appropriate and yielded good results. However, it is well-known that the performance of an algorithm is highly dependent on the setting of parameters. There have been many studies on parameter setting, but yet there is no clear guideline on how to choose the best values of the parameters. For some problems, it requires performing a series of systematic experiments supported by statistical tests to detect which configuration is best. Therefore, there is a need for flexible guidelines relating to parameter adjustment, which should be appropriate for different types of algorithms. Another future direction of research is to focus on the adaptability of the algorithms; in particular, the parameter values should self-adapt based on the quality of the solution. The number of water drops used was constant in the present execution, but it could be changed each cycle in a future version.

Another avenue for future research is to focus on proposing different approaches to represent the search space for problems with continuous domain. In addition, it is worth investigation to analyse the relationship between the problem representation and the algorithm convergence.

Another avenue for future research is a comprehensive performance evaluation of HCA and other algorithms on large-scale problems. HCA could be applied to problems such as constraint satisfaction, scheduling, network routing, classification, and machine learning. The performance of HCA on asymmetric TSPs and CVRPs or their variants are also worthy of future investigation.

As another potential extension, the HCA could be executed by a parallel computing technique to reduce the computational time. Alternatively, a large-scale problem could be divided into a set of sub-problems, each of which is solved by HCA.

Finally, the research presented in this thesis has raised the following questions:

- How does the number of water drops affect the quality of the solution?
- Does allowing the number of water drop to change (increase or decrease) affect the quality of the solution?
- Does embedding another factor (the path width) into the water movements enhance the HCA performance?

By answering these questions, we could investigate how the number of entities influences the search capability and performance of swarm algorithms.

References

Abbatecola, L., Fanti, M. P., & Ukovich, W. (2016). A review of new approaches for dynamic vehicle routing problem. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)* (pp. 361–366). Fort Worth, TX, USA. https://doi.org/10.1109/coase.2016.7743429

AbdAllah, A. M. F. M., Essam, D. L., & Sarker, R. A. (2017). On solving periodic re-optimization dynamic vehicle routing problems. *Applied Soft Computing*, *55*, 1–12. https://doi.org/10.1016/j.asoc.2017.01.047

Abdulmajeed, N. H., & Ayob, M. (2014). A firework algorithm for solving capacitated vehicle routing problem. *International Journal of Advancements in Computing Technology*, *6*(1), 79.

Agarwal, P., & Mehta, S. (2014). Nature-inspired algorithms: state-of-art, problems and prospects. *International Journal of Computer Applications*, *100*(14), 14–21. https://doi.org/10.5120/17593-8331

Ahmmed, A., Rana, M. A. A., Haque, A. A. M. M., & Mamun, M. Al. (2008). A multiple ant colony system for dynamic vehicle routing problem with time window. In *2008 Third International Conference on Convergence and Hybrid Information Technology* (pp. 182–187). Busan: IEEE. https://doi.org/10.1109/iccit.2008.249

Ahrari, A., & Atai, A. A. (2010). Grenade explosion method—a novel tool for optimization of multimodal functions. *Applied Soft Computing*, *10*(4), 1132–1140. https://doi.org/10.1016/j.asoc.2009.11.032

Ai, T. J., & Kachitvichyanukul, V. (2009). Particle swarm optimization and two solution representations for solving the capacitated vehicle routing problem. *Computers & Industrial Engineering*, *56*(1), 380–387. https://doi.org/10.1016/j.cie.2008.06.012

al-Rifaie, M. M., Bishop, J. M., & Blackwell, T. (2012). Information sharing impact of stochastic diffusion search on differential evolution algorithm. *Memetic Computing*, *4*(4), 327–338. https://doi.org/10.1007/s12293-012-0094-y

Alfa, A. S., Heragu, S. S., & Chen, M. (1991). A 3-OPT based simulated annealing algorithm for vehicle routing problems. *Computers & Industrial Engineering*, *21*(1), 635–639. https://doi.org/10.1016/0360-8352(91)90165-3

Alijla, B. O., Wong, L.-P., Lim, C. P., Khader, A. T., & Al-Betar, M. A. (2014). A modified intelligent water drops algorithm and its application to optimization problems. *Expert Systems with Applications*, *41*(15), 6555–6569. https://doi.org/10.1016/j.eswa.2014.05.010

Allen, M., & Peissel, M. (1993). *Dangerous natural phenomena*. Chelsea House. Retrieved from https://books.google.co.nz/books?id=10cLpApjf70C

Alssager, M., & Othman, Z. A. (2016). Cuckoo search algorithm for capacitated vehicle routing problem. *Journal of Theoretical and Applied Information Technology, 88*(1), 11–19.

Álvarez, E., Díaz, F., & Osaba, E. (2014). A multi–agent approach for dynamic production and distribution scheduling. *International Journal of Engineering Management and Economics*, *4*(3–4), 229–248. https://doi.org/10.1504/ijeme.2014.066943

Asi, M., & Dib, N. I. (2010). Design of multilayer microwave broadband absorbers using central force optimization. *Progress In Electromagnetics Research B, 26*, 101–113. https://doi.org/10.2528/PIERB10090103

Askarzadeh, A., Coelho, L. dos S., Klein, C. E., & Mariani, V. C. (2016). A population-based simulated annealing algorithm for global optimization. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 004626–004633). IEEE. https://doi.org/10.1109/SMC.2016.7844961

Augerat, P., Belenguer, J. M., Benavent, E., Corberán, A., Naddef, D., & Rinaldi, G. (1998). *Computational results with a branch-and-cut code for the capacitated vehicle routing problem*. Citeseer. Retrieved from http://www.iasi.cnr.it/reports/R495/R495.pdf

Avila, C., & Valdez, F. (2015). An Improved Simulated Annealing Algorithm for the Optimization of Mathematical Functions (pp. 241–251). Springer, Cham. https://doi.org/10.1007/978-3-319-17747-2_20

Baker, B. M., & Ayechew, M. A. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, *30*(5), 787–800. https://doi.org/10.1016/S0305-0548(02)00051-5

Barbarosoglu, G., & Ozgur, D. (1999). A tabu search algorithm for the vehicle routing problem. *Computers & Operations Research*, *26*(3), 255–270. https://doi.org/10.1016/S0305-0548(98)00047-1

Bartumeus Ferré, F. (2005). *Lévy processes in animal movement and dispersal*. Universitat de Barcelona. Retrieved from http://hdl.handle.net/2445/35293

Basu, S. (2012). Tabu search implementation on traveling salesman problem and its variations: a literature survey. *American Journal of Operations Research*, *2*(2), 163–173. https://doi.org/10.4236/ajor.2012.22019

Bechikh, S., Chaabani, A., & Said, L. Ben. (2015). An efficient chemical reaction optimization algorithm for multiobjective optimization. *IEEE Transactions on Cybernetics*, *45*(10), 2051–2064. https://doi.org/10.1109/tcyb.2014.2363878

Beheshti, Z., & Shamsuddin, S. M. H. (2013). A review of population-based meta-heuristic algorithms. *Int. J. Adv. Soft Comput. Appl*, *5*(1), 1–35.

Bektas, T., Repoussis, P. P., & Tarantilis, C. D. (2014). Chapter 11: dynamic vehicle routing problems. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition* (pp. 299–347). SIAM. https://doi.org/10.1137/1.9781611973594.ch11

Bell, J. E., & McMullen, P. R. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics, 18*(1), 41–48. https://doi.org/10.1016/j.aei.2004.07.001

Benyus, J. M. (1997). *Biomimicry: innovation inspired by nature*. *The American Biology Teacher*. HarperCollins. Retrieved from https://books.google.co.nz/books?id=4XybQgAACAAJ

Bertsimas, D., & Nohadani, O. (2010). Robust optimization with simulated annealing. *Journal of Global Optimization*, *48*(2), 323–334. https://doi.org/10.1007/s10898-009-9496-x

Bhuvaneswari, M., Hariraman, S., Anantharaj, B., & Balaji, N. (2014). Nature inspired algorithms: a review. *International Journal of Emerging Technology in Computer Science and Electronics*, *12*(1), 21–28.

Biegler, L. T. (2010). *Nonlinear programming*. Pennsylvania: Society for Industrial and

Applied Mathematics. https://doi.org/10.1137/1.9780898719383

Bilchev, G., & Parmee, I. C. (1995). The ant colony metaphor for searching continuous design spaces. In *AISB workshop on evolutionary computing* (pp. 25–39). Springer. https://doi.org/10.1007/3-540-60469-3_22

Bishop, J. M., & Locket, G. H. (2002). *Introduction to chemistry*. Benjamin Cummings San Francisco.

Biswas, A., Mishra, K. K., Tiwari, S., & Misra, A. K. (2013). Physics-inspired optimization algorithms: a survey. *Journal of Optimization, 2013,* 1–16. https://doi.org/10.1155/2013/438152

Blazinskas, A., & Misevicius, A. (2011). Combining 2-opt, 3-opt and 4-Opt with K-swap-kick perturbations for the traveling salesman problem. In *17th International Conference on Information and Software Technologies* (pp. 27–29).

Bock, S. (2010). Real-time control of freight forwarder transportation networks by integrating multimodal transport chains. *European Journal of Operational Research*, *200*(3), 733–746. https://doi.org/10.1016/j.ejor.2009.01.046

Bonyadi, M. R., & Michalewicz, Z. (2017). Particle swarm optimization for single objective continuous space problems: A review. *Evolutionary Computation*, *25*(1), 1–54. https://doi.org/10.1162/evco_r_00180

Bonyadi, M. R., Michalewicz, Z., & Li, X. (2014). An analysis of the velocity updating rule of the particle swarm optimization algorithm. *Journal of Heuristics*, *20*(4), 417–452. https://doi.org/10.1007/s10732-014-9245-2

Bookstaber, D. (1997). *Simulated annealing for traveling salesman problem*. *SAREPORT. nb.*

Booyavi, Z., Teymourian, E., Komaki, G. M., & Sheikh, S. (2014). An improved optimization method based on the intelligent water drops algorithm for the vehicle routing problem. In *2014 IEEE Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS)* (pp. 59–66). https://doi.org/10.1109/cipls.2014.7007162

Brabazon, A., O'Neill, M., & McGarraghy, S. (2015). *Natural computing algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-43631-8

Brajevic, I. (2011). An artificial bee colony algorithm for the capacitated vehicle routing problem. In *Proceedings of the European computing conference (ECC'11)* (pp. 239–244). World Scientific and Engineering Academy and Society. Retrieved from http://dl.acm.org/citation.cfm?id=1991016.1991059

Branke, J., Middendorf, M., Noeth, G., & Dessouky, M. (2005). Waiting strategies for dynamic vehicle routing. *Transportation Science, 39*(3), 298–312. https://doi.org/10.1287/trsc.1040.0095

Braun, H. (1991). On solving travelling salesman problems by genetic algorithms. In *Parallel Problem Solving from Nature* (pp. 129–133). Berlin/Heidelberg: Springer-Verlag. https://doi.org/10.1007/BFb0029743

Bräysy, O. (2001). Genetic algorithms for the vehicle routing problem with time windows. *Arpakannus,(1)*, 33–38.

Bräysy, O., & Gendreau, M. (2002). Tabu search heuristics for the vehicle routing problem with time windows. *Top*, *10*(2), 211–237. https://doi.org/10.1007/BF02579017

Bräysy, O., & Gendreau, M. (2005). Vehicle routing problem with time windows, part I: route construction and local search algorithms. *Transportation Science, 39*(1), 104–118. https://doi.org/10.1287/trsc.1030.0056

Breedam, A. Van. (1995). Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research*, *86*(3), 480–490. https://doi.org/10.1016/0377-2217(94)00064-J

Brownlee, J. (2011). *Clever algorithms: nature-inspired programming recipes*. Lulu.com.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1997). A new rank based version of the ant system. A computational study. *Central European Journal for Operations Research and Economics, 7*, 25–38.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999). Applying the ant system to the vehicle routing problem. In *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization* (pp. 285–296). Boston, MA: Springer US. https://doi.org/10.1007/978-1-4615-5775-3_20

Burtscher, M. (2014). *A high-speed 2-opt tsp solver for large problem sizes*. Retrieved from goo.gl/g29Nw2

Can, U., & Alatas, B. (2015). Physics based metaheuristic algorithms for global optimization. *American Journal of Information Science and Computer Engineering*, *1*(3), 94–106.

Cardoso, M. F., Salcedo, R. L., & Feyo de Azevedo, S. (1996). The simplex-simulated annealing approach to continuous non-linear optimization. *Computers & Chemical Engineering*, *20*(9), 1065–1080. https://doi.org/10.1016/0098-1354(95)00221-9

Caric, T., & Gold, H. (2008). Vehicle routing problem. *RAIRO-Operations Research-Recherche*, 152. https://doi.org/10.5772/62148

Casas, N. (2015). Genetic algorithms for multimodal optimization: a review. *CoRR*, *abs/1508.0*. Retrieved from http://arxiv.org/abs/1508.05342

Chakri, A., Khelif, R., Benouaret, M., & Yang, X.-S. (2017). New directional bat algorithm for continuous optimization problems. *Expert Systems with Applications*, *69*, 159–175. https://doi.org/10.1016/j.eswa.2016.10.050

Chan, S. (2001). Complex adaptive systems. In *ESD.83 Research Seminar in Engineering Systems* (Vol. 31, pp. 1–9). https://doi.org/10.1002/cplx.20316

Chase, N., Redemacher, M., Goodman, E., Averill, R., & Sidhu, R. (2010). A benchmark study of optimization search algorithms. *MI, USA: Red Cedar Technology*.

Chelouah, R., & Siarry, P. (2000). A continuous genetic algorithm designed for the global optimization of multimodal functions. *Journal of Heuristics*, *6*(2), 191–213. https://doi.org/10.1023/A:1009626110229

Chen, A., Yang, G., & Wu, Z. (2006). Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. *Journal of Zhejiang University-Science A*, *7*(4), 607–614. https://doi.org/10.1631/jzus.2006.A0607

Chen, S.-M., & Chien, C.-Y. (2011). Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert Systems with Applications*, *38*(12), 14439–14450. https://doi.org/10.1016/j.eswa.2011.04.163

Chen, W.-N., Zhang, J., Chung, H. S.-H., Zhong, W.-L., Wu, W.-G., & Shi, Y. (2010). A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE*

Transactions on Evolutionary Computation, *14*(2), 278–300. https://doi.org/10.1109/tevc.2009.2030331

Chen, W.-N., Zhang, J., Lin, Y., Chen, N., Zhan, Z.-H., Chung, H. S.-H., … Shi, Y.-H. (2013). Particle swarm optimization with an aging leader and challengers. *IEEE Transactions on Evolutionary Computation*, *17*(2), 241–258. https://doi.org/10.1109/TEVC.2011.2173577

Chen, Z., Zhou, S., & Luo, J. (2017). A robust ant colony optimization for continuous functions. *Expert Systems with Applications*, *81*, 309–320. https://doi.org/10.1016/j.eswa.2017.03.036

Chiong, R. (2009). *Nature-inspired algorithms for optimisation*. (R. Chiong, Ed.) (Vol. 193). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-00267-0

Chong, E. K. P., & Zak, S. H. (2013). *An introduction to optimization* (4th Ed.). New York: John Wiley & Sons.

Choong, S. S., Wong, L.-P., & Lim, C. P. (2017). An artificial bee colony algorithm with a modified choice function for the Traveling Salesman Problem. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 357–362). IEEE. https://doi.org/10.1109/SMC.2017.8122629

Christofides, N., & Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Or*, *20*(3), 309. https://doi.org/10.2307/3008733

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, *12*(4), 568–581. https://doi.org/10.1287/opre.12.4.568

Coello Coello, C. A., Lamont, G. B., & Veldhuizen, D. A. Van. (2007). *Evolutionary algorithms for solving multi-objective problems*. *Design*. Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-36797-2

Colby, B. R. (1961). *Effect of depth of flow on discharge of bed material*. US Geological Survey. Retrieved from https://pubs.er.usgs.gov/publication/wsp1498D

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed optimization by ant colonies. In *Proceedings of ECAL91: The european conference on artificial life* (pp. 134–142).

Corana, A., Marchesi, M., Martini, C., & Ridella, S. (1987). Minimizing multimodal functions of continuous variables with the “simulated annealing” algorithm. *ACM Transactions on Mathematical Software*, *13*(3), 262–280. https://doi.org/10.1145/29380.29864

Cortés, C. E., Núñez, A., & Sáez, D. (2008). Hybrid adaptive predictive control for a dynamic pickup and delivery problem including traffic congestion. *International Journal of Adaptive Control and Signal Processing*, *22*(2), 103–123. https://doi.org/10.1002/acs.1005

Cotofrei, P., & Stoffel, K. (2002). Classification rules + time = temporal rules. *Proceedings of the 2002 International Conference on Computational Science*, *2329*, 572–581. https://doi.org/10.1007/3-540-46043-8_58

Cui, L., Li, G., Wang, X., Lin, Q., Chen, J., Lu, N., & Lu, J. (2017). A ranking-based adaptive artificial bee colony algorithm for global numerical optimization. *Information Sciences*, *417*, 169–185. https://doi.org/10.1016/j.ins.2017.07.011

Cui, L., Li, G., Zhu, Z., Lin, Q., Wen, Z., Lu, N., … Chen, J. (2017). A novel artificial bee colony algorithm with an adaptive population size for numerical function optimization. *Information Sciences*, *414*, 53–67. https://doi.org/10.1016/j.ins.2017.05.044

Dąbkowski, M., Redlarski, G., & Pałkowski, A. (2013). Using river formation dynamics algorithm in mobile robot navigation. In *Mechatronic Systems and Materials IV* (Vol. 198, pp. 138–143). Trans Tech Publications. https://doi.org/10.4028/www.scientific.net/ssp.198.138

Dam, T.-L., Li, K., & Fournier-Viger, P. (2016). Chemical reaction optimization with unified tabu search for the vehicle routing problem. *Soft Computing*. https://doi.org/10.1007/s00500-016-2200-4

Dandu, N. K. (2013). *Optimization of benchmark functions using chemical reaction optimization*. North Dakota State University. Retrieved from http://hdl.handle.net/10365/22782

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science, 6*(1), 80–91. https://doi.org/10.1287/mnsc.6.1.80

Darwin, C. (1968). On the origin of species by means of natural selection. 1859. *London: Murray Google Scholar*.

Davendra, D. (2010). *Traveling salesman problem, theory and applications*. (D. Davendra, Ed.). InTech. https://doi.org/10.5772/547

Dávid, B., & Krész, M. (2017). The dynamic vehicle rescheduling problem. *Central European Journal of Operations Research*, *25*(4), 809–830. https://doi.org/10.1007/s10100-017-0478-7

Davie, T. (2008). *Fundamentals of hydrology*. Taylor & Francis.

de Castro, L. N. (2006). *Fundamentals of natural computing: basic concepts, algorithms, and applications*. CRC Press. Retrieved from https://books.google.co.nz/books?id=2wTOBQAAQBAJ

Deneubourg, J. L., Aron, S., Goss, S., & Pasteels, J. M. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior, 3*(2), 159–168. https://doi.org/10.1007/BF01417909

Digalakis, J. G., & Margaritis, K. G. (2002). An experimental study of benchmarking functions for genetic algorithms. In *SMC 2000 Conference Proceedings. 2000 IEEE International Conference on Systems, Man and Cybernetics.* (Vol. 5, pp. 3810–3815). IEEE. https://doi.org/10.1109/ICSMC.2000.886604

Ding, C., Lu, L., Liu, Y., & Peng, W. (2011). Swarm intelligence optimization and its applications. In G. Shen & X. Huang (Eds.), *Advanced Research on Electronic Commerce, Web Application, and Communication: International Conference, ECWAC 2011, Guangzhou, China, April 16-17, 2011. Proceedings, Part I* (pp. 458–464). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-20367-1_74

Dorigo, M. (1992). Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico Di Milano, Italy*.

Dorigo, M., Birattari, M., & St, T. (2006). Ant Colony Optimization, (November).

Dorigo, M., & Gambardella, L. M. (1995). Ant-Q: A reinforcement learning approach to the traveling salesman problem. In *Proceedings of ML-95, Twelfth Intern. Conf. on Machine Learning* (pp. 252–260). USA: Morgan Kaufmann Publishers Inc. Retrieved from http://dl.acm.org/citation.cfm?id=3091622.3091654

Dorigo, M., & Gambardella, L. M. (1996). A study of some properties of Ant-Q. In *International Conference on Parallel Problem Solving from Nature* (pp. 656–665). Springer. https://doi.org/10.1007/3-540-61723-X_1029

Dorigo, M., & Gambardella, L. M. (1997a). Ant colonies for the travelling salesman problem. *Biosystems*, *43*(2), 73–81. https://doi.org/10.1016/S0303-2647(97)01708-5

Dorigo, M., & Gambardella, L. M. (1997b). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66. https://doi.org/10.1109/4235.585892

Dorigo, M., Maniezzo, V., & Colorni, A. (1991a). *Ant system: An autocatalytic optimizing process*. Italy: Technical report.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991b). *Positive feedback as a search strategy*. Citeseer.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *26*(1), 29–41. https://doi.org/10.1109/3477.484436

Dorigo, M., Stützle, T., & Stutzle, T. (2004). *Ant colony optimization*. *The MIT Press*. Scituate, MA, USA: Bradford Company. Retrieved from https://books.google.co.nz/books?id=%5C_aefcpY8GiEC

Dowlatshahi, M. B., Nezamabadi-pour, H., & Mashinchi, M. (2014). A discrete gravitational search algorithm for solving combinatorial optimization problems. *Information Sciences*, *258*, 94–107. https://doi.org/10.1016/j.ins.2013.09.034

Dréo, J., & Siarry, P. (2004). Continuous interacting ant colony algorithm based on dense heterarchy. *Future Generation Computer Systems*, *20*(5), 841–856. https://doi.org/10.1016/j.future.2003.07.015

Duan, H., Liu, S., & Lei, X. (2008). Air robot path planning based on intelligent water drops optimization. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (pp. 1397–1401). https://doi.org/10.1109/IJCNN.2008.4633980

Duch, W. (2013). Rule discovery. In W. Dubitzky, O. Wolkenhauer, K.-H. Cho, & H. Yokota (Eds.), *Encyclopedia of Systems Biology* (pp. 1879–1883). New York, NY: Springer New York. https://doi.org/10.1007/978-1-4419-9863-7_616

Eglese, R. W. (1990). Simulated annealing: A tool for operational research. *European Journal of Operational Research*, *46*(3), 271–281. https://doi.org/10.1016/0377-2217(90)90001-R

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing* (Vol. 53). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-05094-1

El Rhalibi, a., & Kelleher, G. (2003). An approach to dynamic vehicle routing, rescheduling and disruption metrics. *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483)*, *4*, 3613–3618. https://doi.org/10.1109/ICSMC.2003.1244450

Elhassania, M., Jaouad, B., & Ahmed, E. A. (2014). Solving the dynamic vehicle routing problem using genetic algorithms. In *2014 International Conference on Logistics*

Operations Management (pp. 62–69). IEEE. https://doi.org/10.1109/GOL.2014.6887419

Engelbrecht, A. P. (2007). *Computational intelligence: an introduction*. Wiley. Retrieved from https://books.google.co.nz/books?id=IZosIcgJMjUC

Eskandar, H., Sadollah, A., Bahreininejad, A., & Hamdi, M. (2012). Water cycle algorithm – A novel metaheuristic optimization method for solving constrained engineering optimization problems. *Computers & Structures*, *110–111*, 151–166. https://doi.org/10.1016/j.compstruc.2012.07.010

Esquivel, S. C., & Coello Coello, C. A. (2003). On the use of particle swarm optimization with multimodal functions. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.* (Vol. 2, pp. 1130–1136). IEEE. https://doi.org/10.1109/CEC.2003.1299795

Ezzatneshan, A. (2010). A algorithm for the vehicle problem. *International Journal of Advanced Robotic Systems*, *7*(2), 1. https://doi.org/10.5772/9698

Faulin, J., & del Valle, A. (2008). Solving the capacitated vehicle routing problem using the ALGELECT electrostatic algorithm. *Journal of the Operational Research Society*, *59*(12), 1685–1695. https://doi.org/10.1057/palgrave.jors.2602502

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 37–54. https://doi.org/10.1145/240455.240463

Fekete, S. P., Kroeller, A., Lorek, M., & Pfetsch, M. (2011). Disruption management with rescheduling of trips and vehicle circulations. *ArXiv E-Prints*. https://doi.org/10.1115/JRC2011-56040

Fenton, A. (2016). The bees algorithm for the vehicle routing problem. *CoRR*, *abs/1605.0*. Retrieved from http://arxiv.org/abs/1605.05448

Ferrucci, F. (2013). *Pro-active Dynamic Vehicle Routing*. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-33472-6

Financial and Economic Analysis Team. (2010). *Understanding transport costs and charges*. New Zealand: Ministry of Transport.

Fink, E. (2002). *Changes of problem representation* (Vol. 110). Heidelberg: Physica-Verlag HD. https://doi.org/10.1007/978-3-7908-1774-4

Fisher, M. L. (1994). Optimal solution of vehicle routing problems using minimum K-trees. *Operations Research*, *42*(4), 626–642. https://doi.org/10.1287/opre.42.4.626

Fleischmann, B., Gnutzmann, S., & Sandvoß, E. (2004). Dynamic vehicle routing based on online traffic information. *Transportation Science*, *38*(4), 420–433. https://doi.org/10.1287/trsc.1030.0074

Fogel, D. B. (1995). Phenotypes, genotypes, and operators in evolutionary computation. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation* (Vol. 1, p. 193). IEEE. https://doi.org/10.1109/ICEC.1995.489143

Formato, R. A. (2007). Central force optimization: a new metaheuristic with applications in applied electromagnetics. *Progress In Electromagnetics Research*, *77*, 425–491. https://doi.org/10.2528/PIER07082403

Formato, R. A. (2008). Central force optimization: A new nature inspired computational framework for multidimensional search and optimization. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)* (pp. 221–238). Springer. https://doi.org/10.1007/978-3-540-78987-1_21

Formato, R. A. (2009). Central force optimization: A new deterministic gradient-like optimization metaheuristic. *OPSEARCH*, *46*(1), 25–51. https://doi.org/10.1007/s12597-009-0003-4

Formato, R. A. (2011). Central force optimization with variable initial probes and adaptive decision space. *Applied Mathematics and Computation*, *217*(21), 8866–8872. https://doi.org/10.1016/j.amc.2011.03.151

Gambardella, L. M., & Dorigo, M. (1996). Solving symmetric and asymmetric TSPs by ant colonies. In *Proceedings of IEEE International Conference on Evolutionary Computation* (pp. 622–627). IEEE. https://doi.org/10.1109/ICEC.1996.542672

Gao, S., Vairappan, C., Wang, Y., Cao, Q., & Tang, Z. (2014). Gravitational search algorithm combined with chaos for unconstrained numerical optimization. *Applied Mathematics and Computation*, *231*, 48–62. https://doi.org/10.1016/j.amc.2013.12.175

Garai, G., Debbarman, S., & Biswas, T. (2013). An efficient ant colony optimization algorithm for function optimization. In *2013 IEEE Congress on Evolutionary Computation* (pp. 2345–2351). https://doi.org/10.1109/CEC.2013.6557849

Gath, M. (2016). *Optimizing transport logistics processes with multiagent planning and control*. Wiesbaden: Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-14003-8

Gendreau, M., Guertin, F., Potvin, J.-Y., & Taillard, É. D. (1999). Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, *33*(4), 381–390. https://doi.org/10.1287/trsc.33.4.381

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, *40*(10), 1276–1290. https://doi.org/10.1287/mnsc.40.10.1276

Geng, X., Chen, Z., Yang, W., Shi, D., & Zhao, K. (2011). Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Applied Soft Computing*, *11*(4), 3680–3689. https://doi.org/http://dx.doi.org/10.1016/j.asoc.2011.01.039

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1

Goel, L., & Panchal, V. K. (2013). Feature extraction through information sharing in swarm intelligence techniques. In *Knowledge—Based Processes in Software Development* (pp. 151–175). https://doi.org/10.4018/978-1-4666-4229-4.ch010

Golden, B. L., Raghavan, S., & Wasil, E. A. (2008). *The vehicle routing problem: latest advances and new challenges*. *Information Systems Journal* (Vol. 43). Springer US. https://doi.org/10.1007/978-0-387-77778-8

Goss, S., Aron, S., Deneubourg, J. L., & Pasteels, J. M. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, *76*(12), 579–581. https://doi.org/10.1007/BF00462870

Green, D., Aleti, A., & Garcia, J. (2017). The nature of nature: why nature-inspired algorithms work. In S. Patnaik, X.-S. Yang, & K. Nakamatsu (Eds.), *Nature-Inspired Computing and Optimization: Theory and Applications* (pp. 1–27). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-50920-4_1

Grefenstette, J., Gopal, R., Rosmaita, B., & Van Gucht, D. (1985). Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms* (pp. 160–168). USA: L. Erlbaum Associates Inc. Retrieved from http://dl.acm.org/citation.cfm?id=645511.657094

Gülcü, S. D., Gülcü, Ş., Kahramanli, H., Campus, A. K., & Selçuklu, K. (2013). Solution of travelling salesman problem using intelligent water drops algorithm. In *Proceedings of the 2nd International Conference on Information Technology and Computer Networks (ITCN '13)*. Turkey: WSEAS Press. Retrieved from http://www.wseas.us/e-library/conferences/2013/Antalya/ITCN/ITCN-04.pdf

Guo, P., & Zhu, L. (2012). Ant colony optimization for continuous domains. In *2012 8th International Conference on Natural Computation* (pp. 758–762). https://doi.org/10.1109/ICNC.2012.6234538

Haghani, A., & Yang, S. (2007). Real-time emergency response fleet deployment: concepts, systems, simulation & case studies. In V. Zeimpekis, C. D. Tarantilis, G. M. Giaglis, & I. Minis (Eds.), *Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies* (pp. 133–162). Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-71722-7_7

Han, J. (2010). *Evaluation of optimization algorithms for improvement of a transportation companies (in-house) vehicle routing system*. University of Nebraska-Lincoln. Retrieved from http://digitalcommons.unl.edu/imsediss%5Cnhttp://digitalcommons.unl.edu/imsediss/13

Hanshar, F. T., & Ombuki-Berman, B. M. (2007). Dynamic vehicle routing using genetic algorithms. *Applied Intelligence*, *27*(1), 89–99. https://doi.org/10.1007/s10489-006-0033-z

Harmanani, H. M., Azar, D., Helal, N., & Keirouz, W. (2011). A simulated annealing algorithm for the capacitated vehicle routing problem. In *CATA* (pp. 96–101).

Hassan, R., Cohanim, B., de Weck, O., & Venter, G. (2005). A comparison of particle swarm optimization and the genetic algorithm. In *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*. Reston, Virigina: American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2005-1897

Heidari, A. A., Ali Abbaspour, R., & Rezaee Jordehi, A. (2017). An efficient chaotic water cycle algorithm for optimization tasks. *Neural Computing and Applications*, *28*(1), 57–85. https://doi.org/10.1007/s00521-015-2037-2

Held, M., Hoffman, A. J., Johnson, E. L., & Wolfe, P. (1984). Aspects of the traveling salesman problem. *IBM Journal of Research and Development*, *28*(4), 476–486. https://doi.org/10.1147/rd.284.0476

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*(1), 106–130. https://doi.org/10.1016/S0377-2217(99)00284-2

Helsgaun, K. (2006). An effective implementation of K-opt moves for the lin-kernighan TSP heuristic. In *Roskilde University, 2007. Case* (p. 109).

Hlaing, Z. C. S. S., & Khine, M. A. (2011). An ant colony optimization algorithm for solving traveling salesman problem. In *International Conference on Information Communication and Management* (Vol. 16, pp. 54–59).

Ho, W., Ho, G. T. S., Ji, P., & Lau, H. C. W. (2008). A hybrid genetic algorithm for the multi-depot vehicle routing problem. *Engineering Applications of Artificial Intelligence*, *21*(4), 548–557. https://doi.org/10.1016/j.engappai.2007.06.001

Ho, Y.-C., & Pepyne, D. L. (2002). Simple explanation of the No-Free-Lunch theorem and its implications. *Journal of Optimization Theory and Applications*, *115*(3), 549–570. https://doi.org/10.1023/A:1021251113462

Hoffman, K. L., & Padberg, M. (2013). Traveling salesman problem. In *Encyclopedia of Operations Research and Management Science* (pp. 849–853). Dordrecht: Kluwer Academic Publishers. https://doi.org/10.1007/1-4020-0611-X_1068

Hogg, T., & Williams, C. P. (1993). Solving the really hard problems with cooperative search. In *Proceedings of the national conference on artificial intelligence* (pp. 231–236). Washington, D.C.: AAAI Press. Retrieved from http://dl.acm.org/citation.cfm?id=1867270.1867305

Holland, J. (1975). Adaptation in natural and artificial systems: An introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press,* 183. Retrieved from https://books.google.co.nz/books?id=JE5RAAAAMAAJ

Holland, J. (1992). Genetic algorithms. *Scientific American*, *267*(1), 66–72.

Housroum, H., Hsu, T., Dupas, R., & Goncalves, G. (2006). A hybrid GA approach for solving the dynamic vehicle routing problem with time windows. In *2006 2nd International Conference on Information Communication Technologies* (Vol. 1, pp. 787–792). https://doi.org/10.1109/ICTTA.2006.1684473

Hu, G., Chu, X., Niu, B., Li, L., Lin, D., & Liu, Y. (2016). An Augmented Artificial Bee Colony with Hybrid Learning for Traveling Salesman Problem (pp. 636–643). https://doi.org/10.1007/978-3-319-42291-6_63

Huang, H., & Hao, Z. (2006). ACO for continuous optimization based on discrete encoding. In *International Workshop on Ant Colony Optimization and Swarm Intelligence* (pp. 504–505). Springer. https://doi.org/10.1007/11839088_53

Huo, J., Zhang, Y., & Zhao, H. (2015). An improved artificial bee colony algorithm for numerical functions. *International Journal of Reasoning-Based Intelligent Systems*, *7*(3/4), 200. https://doi.org/10.1504/IJRIS.2015.072947

Ibrahim, I., Ibrahim, Z., Ahmad, H., & Yusof, Z. M. (2015a). A multi-state gravitational search algorithm for combinatorial optimization problems. In *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks* (pp. 9–14). https://doi.org/10.1109/CICSyN.2015.12

Ibrahim, I., Ibrahim, Z., Ahmad, H., & Yusof, Z. M. (2015b). Rule-based multi-state gravitational search algorithm for discrete optimization problem. In *2015 4th International Conference on Software Engineering and Computer Systems (ICSECS)* (pp. 142–147). https://doi.org/10.1109/ICSECS.2015.7333099

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2007). Planned route optimization for real-time vehicle routing. *Dynamic Fleet Management*, *38*, 1–18. https://doi.org/10.1007/978-0-387-71722-7_1

Incropera, F. P., DeWitt, D. P., Bergman, T. L., & Lavine, A. S. (2007). *Fundamentals of heat and mass transfer* (6th Ed.). USA: John Wiley & Sons.

Jackson, D. E., & Ratnieks, F. L. W. (2006). Communication in ants. *Current Biology*, *16*(15), R570–R574. https://doi.org/10.1016/j.cub.2006.07.015

Jamil, M., & Yang, X.-S. (2013). A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, *4*(2), 150–194. https://doi.org/10.1504/IJMMNO.2013.055204

Jati, G. K., Manurung, H. M., & Suyanto. (2012). Discrete cuckoo search for traveling salesman problem. In *2012 7th International Conference on Computing and Convergence Technology (ICCCT)* (pp. 993–997). Seoul: IEEE.

Jeon, G., Leep, H. R., & Shim, J. Y. (2007). A vehicle routing problem solved by using a hybrid genetic algorithm. *Computers & Industrial Engineering*, *53*(4), 680–692. https://doi.org/10.1016/j.cie.2007.06.031

Jr., I. F., Yang, X.-S., Fister, I., Brest, J., & Fister, D. (2013). A brief review of nature-inspired algorithms for optimization. *CoRR*, *abs/1307.4*. Retrieved from http://arxiv.org/abs/1307.4186

Jun, Q., Wang, J., & Zheng, B. (2008). A hybrid multi-objective algorithm for dynamic vehicle routing problems. *Computational Science – ICCS 2008 Lecture Notes in Computer Science Volume 5103*, *5103 LNCS*, 674–681. https://doi.org/10.1007/978-3-540-69389-5_75

Kadry, S. N., & El Hami, A. (2015). A New Hybrid Simulated Annealing Algorithm for Large Scale Global Optimization. *International Journal of Manufacturing, Materials, and Mechanical Engineering*, *5*(3), 24–36. https://doi.org/10.4018/IJMMME.2015070102

Kang-Ping Wang, Lan Huang, Chun-Guang Zhou, & Wei Pang. (2003). Particle swarm optimization for traveling salesman problem. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)* (Vol. 3, pp. 1583–1585). IEEE. https://doi.org/10.1109/ICMLC.2003.1259748

Kao, Y., Chen, M.-H., & Huang, Y.-T. (2012). A hybrid algorithm based on ACO and PSO for capacitated vehicle routing problems. *Mathematical Problems in Engineering*, *2012*, 17. https://doi.org/10.1155/2012/726564

Karaboga, D. (2005). *An idea based on honey bee swarm for numerical optimization*. Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

Karaboga, D., & Akay, B. (2009). Artificial bee colony (ABC), harmony search and bees algorithms on numerical optimization. In *Innovative production machines and systems virtual conference*. Erciyes University.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, *39*(3), 459–471. https://doi.org/10.1007/s10898-007-9149-x

Karaboga, D., & Gorkemli, B. (2011). A combinatorial artificial bee colony algorithm for traveling salesman problem. In *2011 International Symposium on Innovations in Intelligent Systems and Applications* (pp. 50–53). IEEE. https://doi.org/10.1109/INISTA.2011.5946125

Kari, L., & Rozenberg, G. (2008). The many facets of natural computing. *Communications of the ACM*, *51*(10), 72–83. https://doi.org/10.1145/1400181.1400200

Kaur, E. A., & Kaur, E. M. (2015). A comprehensive survey of test functions for evaluating the performance of particle swarm optimization algorithm. *International Journal of Hybrid Information Technology*, *8*(5), 97–104.

Kennedy, J. (2011). Particle swarm optimization. In *Encyclopedia of machine learning* (pp. 760–766). Springer.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of 1995 IEEE International Conference on Neural Networks* (Vol. 4, pp. 1942–1948). https://doi.org/10.1109/ICNN.1995.488968

Khachaturyan, A. G., Semenovskaya, S. V, & Vainstein, B. (1979). A statistical-thermodynamic approach to determination of structure amplitude phases. *Sov. Phys. Crystallogr, 24*, 519–524.

Khoshbakht, M. Y., & Sedighpour, M. (2011). An optimization algorithm for the capacitated vehicle routing problem based on ant colony system. *Australian Journal of Basic and Applied Sciences*, *5*(12), 2729–2737.

Khouadjia, M. R. (2011). *Solving dynamic vehicle routing problems : from single-solution based metaheuristics to parallel population based metaheuristics*. Lille University of Science and Technology. Retrieved from http://ori.univ-lille1.fr/notice/view/univ-lille1-ori-29586

Khouadjia, M. R., Sarasola, B., Alba, E., Jourdan, L., & Talbi, E.-G. (2012). A comparative study between dynamic adapted PSO and VNS for the vehicle routing problem with dynamic requests. *Applied Soft Computing*, *12*(4), 1426–1439. https://doi.org/10.1016/j.asoc.2011.10.023

Kilby, P., Prosser, P., & Shaw, P. (1998). Dynamic VRPs: a study of scenarios. *Report APES-06-1998*, 1–11.

Kindervater, G. A. P., & Savelsbergh, M. W. P. (1997). Vehicle routing: handling edge exchanges. *Local Search in Combinatorial Optimization*, 337–360.

Kir, S., Yazgan, H. R., & Tüncel, E. (2017). A novel heuristic algorithm for capacitated vehicle routing problem. *Journal of Industrial Engineering International*, *13*(3), 323–330. https://doi.org/10.1007/s40092-017-0187-9

Kiran, M. S., & Babalik, A. (2014). Improved artificial bee colony algorithm for continuous optimization problems. *Journal of Computer and Communications*, *2*(4), 108–116. https://doi.org/10.4236/jcc.2014.24015

Kiran, M. S., Iscan, H., & Gündüz, M. (2013). The analysis of discrete artificial bee colony algorithm with neighborhood operator on traveling salesman problem. *Neural Computing and Applications*, *23*(1), 9–21. https://doi.org/10.1007/s00521-011-0794-0

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

Knox, J. (1994). Tabu search performance on the symmetric traveling salesman problem. *Computers & Operations Research*, *21*(8), 867–876. https://doi.org/10.1016/0305-0548(94)90016-7

Kocer, H. E., & Akca, M. R. (2014). An improved artificial bee colony algorithm with local search for traveling salesman problem. *Cybernetics and Systems*, *45*(8), 635–649. https://doi.org/10.1080/01969722.2014.970396

Kordestani, J. K., Firouzjaee, H. A., & Reza Meybodi, M. (2018). An adaptive bi-flight cuckoo search with variable nests for continuous dynamic optimization problems. *Applied Intelligence*, *48*(1), 97–117. https://doi.org/10.1007/s10489-017-0963-7

Kulović, M. (2004). Freight transport costs model based on truck fleet operational parameters. *PROMET-Traffic &Transportation*, *16*(6), 321–325. https://doi.org/10.7307/ptt.v16i6.608

Kumar, S. N., & Panneerselvam, R. (2012). A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, *4*(3), 66–74. https://doi.org/10.4236/iim.2012.43010

Kumar, V. S., Thansekhar, M., & Saravanan, R. (2014). Multi objective optimization for vehicle routing problem using fitness aggregated genetic algorithm (FAGA). In *Second International Conference on Advances in Industrial Engineering Applications*. Chennai.

Kumbharana, S. N., & Pandey, G. M. (2013). Solving travelling salesman problem using firefly algorithm. *International Journal for Research in Science & Advanced Technologies*, *2*(2), 53–57.

Lackner, A. (2004). MACS-DVRPTW instances data. Retrieved January 5, 2017, from https://github.com/orivalde/dvrptw

Lam, A. Y. S., & Li, V. O. K. (2010). Chemical-reaction-inspired metaheuristic for optimization. *IEEE Transactions on Evolutionary Computation*, *14*(3), 381–399. https://doi.org/10.1109/TEVC.2009.2033580

Lam, A. Y. S., & Li, V. O. K. (2012). Chemical reaction optimization: A tutorial. *Memetic Computing*, *4*(1), 3–17. https://doi.org/10.1007/s12293-012-0075-1

Lampton, C. (2000). *Science of Chaos*. Diane Publishing Company. Retrieved from https://books.google.co.nz/books?id=XigCAAAACAAJ

Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: a review of representations and operators. *Artificial Intelligence Review*, *13*(2), 129–170. https://doi.org/10.1023/A:1006529012972

Larsen, A. (2001). *The dynamic vehicle routing problem*. Denmark Technical University of Denmark (DTU). https://doi.org/10.1016/j.cor.2004.04.013

Larsen, A., Madsen, O. B. G., & Solomon, M. M. (2007). Classification of dynamic vehicle routing systems. In *Dynamic Fleet Management* (Vol. 38, pp. 19–40). https://doi.org/10.1007/978-0-387-71722-7_2

Li, H., Liu, K., & Li, X. (2010). A comparative study of artificial bee colony, bees algorithms and differential evolution on numerical benchmark problems. In Z. Cai, H. Tong, Z. Kang, & Y. Liu (Eds.), *Computational Intelligence and Intelligent Systems: 5th International Symposium, ISICA 2010, Wuhan, China, October 22-24, 2010. Proceedings* (pp. 198–207). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-16388-3_22

Li, J.-Q., Mirchandani, P. B., & Borenstein, D. (2007). The vehicle rescheduling problem: model and algorithms. *Networks*, *50*(3), 211–229. https://doi.org/10.1002/net.20199

Li, Y., Zhan, Z.-H., Lin, S., Zhang, J., & Luo, X. (2015). Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems. *Information Sciences*, *293*, 370–382. https://doi.org/10.1016/j.ins.2014.09.030

Li, Y., Zhou, A., & Zhang, G. (2011). Simulated annealing with probabilistic neighborhood for traveling salesman problems. In *2011 Seventh International Conference on Natural Computation* (pp. 1565–1569). IEEE. https://doi.org/10.1109/ICNC.2011.6022345

Li, Z., Li, Z., Nguyen, T. T., & Chen, S. (2015). Orthogonal chemical reaction optimization algorithm for global numerical optimization problems. *Expert Systems with Applications*, *42*(6), 3242–3252. https://doi.org/10.1016/j.eswa.2014.11.045

Liang, Y., Wan, Z., & Fang, D. (2017). An improved artificial bee colony algorithm for solving constrained optimization problems. *International Journal of Machine Learning and Cybernetics*, *8*(3), 739–754. https://doi.org/10.1007/s13042-015-0357-2

Liao, T., de Oca, M. A., Aydin, D., Stützle, T., & Dorigo, M. (2011). An incremental ant colony algorithm with local search for continuous optimization. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (pp. 125–132). New York, NY, USA: ACM. https://doi.org/10.1145/2001576.2001594

Lieshout, R. van. (2016). *The road-based vehicle rescheduling problem: methods and extensions*. Erasmus University Rotterdam. Retrieved from http://hdl.handle.net/2105/34078

Lin, Q., Zhu, M., Li, G., Wang, W., Cui, L., Chen, J., & Lu, J. (2018). A novel artificial bee colony algorithm with local and global information interaction. *Applied Soft Computing*, *62*, 702–735. https://doi.org/10.1016/j.asoc.2017.11.012

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, *44*(10), 2245–2269. https://doi.org/10.1002/j.1538-7305.1965.tb04146.x

Lin, S. W., Ying, K. C., Lee, Z. J., & Hsi, F. H. (2006). Applying simulated annealing approach for capacitated vehicle routing problems. In *Management Science / Operations Research* (Vol. 1, pp. 639–644). https://doi.org/10.1109/ICSMC.2006.384457

Liu, B., Li, W., & Pan, S. (2016). A Novel Adaptive Cooperative Artificial Bee Colony Algorithm for Solving Numerical Function Optimization. In L. Zhang, X. Song, & Y. Wu (Eds.) (pp. 25–36). Springer, Singapore. https://doi.org/10.1007/978-981-10-2663-8_3

Liu, J., & Wang, Y. (2014). An improved central force optimization algorithm for multimodal optimization. *Journal of Applied Mathematics*, *2014*, 1–12. https://doi.org/10.1155/2014/895629

Liu, L., Dai, Y., & Gao, J. (2014). Ant colony optimization algorithm for continuous domains based on position distribution model of ant colony foraging. *Scientific World Journal*, *2014*. https://doi.org/10.1155/2014/428539

Liu, Y., Ling, X., Shi, Z., Lv, M., Fang, J., & Zhang, L. (2011). A survey on particle swarm optimization algorithms for multimodal function optimization. *JSW*, *6*(12), 2449–2455. https://doi.org/10.4304/jsw.6.12.2449-2455

Liu, Y., & Tian, P. (2015). A multi-start central force optimization for global optimization. *Applied Soft Computing*, *27*, 92–98. https://doi.org/10.1016/j.asoc.2014.10.031

Lund, K., Madsen, O. B. G., & Rygaard, J. M. (1996). *Vehicle routing problems with varying degrees of dynamism* (IMM-REP). IMM Institute of Mathematical Modelling. Retrieved from https://books.google.co.nz/books?id=jwiBYgEACAAJ

Luo, Q., Wen, C., Qiao, S., & Zhou, Y. (2016). Dual-system water cycle algorithm for constrained engineering optimization problems. In D.-S. Huang, V. Bevilacqua, & P. Premaratne (Eds.), *Intelligent Computing Theories and Application: 12th International Conference, ICIC 2016, Lanzhou, China, August 2-5, 2016, Proceedings, Part I* (pp. 730–741). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-42291-6_73

Lysgaard, J. (1997). *Clarke & Wright's Savings algorithm*. Retrieved from goo.gl/z3z2xB

Maaranen, H., Miettinen, K., & Penttinen, A. (2007). On initial populations of a genetic algorithm for continuous optimization problems. *Journal of Global Optimization*, *37*(3), 405–436. https://doi.org/10.1007/s10898-006-9056-6

Maciejewski, M., Bischoff, J., Hörl, S., & Nagel, K. (2017). Towards a testbed for dynamic vehicle routing algorithms. In J. Bajo, Z. Vale, K. Hallenborg, A. P. Rocha, P. Mathieu, P. Pawlewski, … J. Holmgren (Eds.), *Highlights of Practical Applications of Cyber-Physical Multi-Agent Systems: International Workshops of PAAMS 2017, Porto, Portugal, June 21-23, 2017, Proceedings* (pp. 69–79). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-60285-1_6

Maidment, D. R. (1993). *Handbook of hydrology* (Vol. 1). New York: McGraw-Hill.

Maniezzo, V. (1999). Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, *11*(4), 358–369. https://doi.org/10.1287/ijoc.11.4.358

Marler, R. T., & Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, *26*(6), 369–395. https://doi.org/10.1007/s00158-003-0368-6

Martens, D., De Backer, M., Haesen, R., Vanthienen, J., Snoeck, M., & Baesens, B. (2007). Classification with ant colony optimization. *IEEE Transactions on Evolutionary Computation*, *11*(5), 651–665. https://doi.org/10.1109/TEVC.2006.890229

Masum, A. K. M., Faruque, M. F., Shahjalal, M., Iqbal, M., & Sarker, H. (2011). Solving the vehicle routing problem using genetic algorithm. *IJACSA) International Journal of Advanced Computer Science and Applications*, *2*(7), 6. Retrieved from www.ijacsa.thesai.org

Masutti, T. A. S., & de Castro, L. N. (2009). A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem. *Information Sciences*, *179*(10), 1454–1468. https://doi.org/10.1016/j.ins.2008.12.016

Matai, R., Singh, S., & Lal, M. (2010). Traveling salesman problem: An overview of applications, formulations, and solution approaches. In *Traveling Salesman Problem, Theory and Applications*. InTech. https://doi.org/10.5772/12909

Mathur, M., Karale, S. B., Priye, S., Jayaraman, V. K., & Kulkarni, B. D. (2000). Ant colony approach to continuous function optimization. *Industrial & Engineering Chemistry Research*, *39*(10), 3814–3822. https://doi.org/10.1021/ie990700g

Mavrovouniotis, M., & Yang, S. (2010). Ant colony optimization with direct communication for the traveling salesman problem. In *2010 UK Workshop on Computational Intelligence (UKCI)* (pp. 1–6). IEEE. https://doi.org/10.1109/UKCI.2010.5625608

Mazidi, A., Fakhrahmad, M., & Sadreddini, M. (2016). A meta-heuristic approach to CVRP problem: local search optimization based on GA and ant colony. *Journal of Advances in Computer Research*, *7*(1), 1–22. Retrieved from http://jacr.iausari.ac.ir/article_17690.html

Mazzeo, S., & Loiseau, I. (2004). An ant colony algorithm for the capacitated vehicle routing. *Electronic Notes in Discrete Mathematics*, *18*, 181–186. https://doi.org/10.1016/j.endm.2004.06.029

Medina, A. J. R., Pulido, G. T., & Ramirez-Torres, J. G. (2009). A comparative study of neighborhood topologies for particle swarm optimizers. In *Proceedings of the International Joint Conference on Computational Intelligence* (pp. 152–159). SciTePress

- Science and and Technology Publications. https://doi.org/10.5220/0002324801520159

Mirjalili, S., & Hashim, S. Z. M. (2010). A new hybrid PSOGSA algorithm for function optimization. In *2010 International Conference on Computer and Information Application* (pp. 374–377). https://doi.org/10.1109/ICCIA.2010.6141614

Mitchell, M. (1998). *An introduction to genetic algorithms*. Cambridge: MIT press.

Monmarché, N., Venturini, G., & Slimane, M. (2000). On how Pachycondyla apicalis ants suggest a new search algorithm. *Future Generation Computer Systems*, *16*(8), 937–946. https://doi.org/10.1016/S0167-739X(00)00047-9

Montemanni, R., Gambardella, L. M., Rizzoli, A. E., & Donati, A. V. (2002). *A new algorithm for a dynamic vehicle routing problem based on ant colony system*. *IDSIA*. Switzerland. Retrieved from http://www.idsia.ch/~roberto/papers/dvrp.pdf

Montemanni, R., Gambardella, L. M., Rizzoli, A. E., & Donati, A. V. (2005). Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, *10*(4), 327–343. https://doi.org/10.1007/s10878-005-4922-6

Moorthy, S. K. (2012). *Evolving optimal solutions by nature inspired algorithms*. Pune. Retrieved from http://hdl.handle.net/10603/98396

Msallam, M. M., & Hamdan, M. (2011). Improved intelligent water drops algorithm using adaptive schema. *International Journal of Bio-Inspired Computation*, *3*(2), 103. https://doi.org/10.1504/IJBIC.2011.039909

Mu, Q., Fu, Z., Lysgaard, J., & Eglese, R. W. (2011). Disruption management of the vehicle routing problem with vehicle breakdown, *62*(4), 742–749. https://doi.org/10.1057/jors.2010.19

Na, B., Jun, Y., & Kim, B.-I. (2011). Some extensions to the sweep algorithm. *The International Journal of Advanced Manufacturing Technology*, *56*(9), 1057–1067. https://doi.org/10.1007/s00170-011-3240-7

Najera, A. G. (2007). *A genetic algorithm for the capacitated vehicle routing problem*. Birmingham. Retrieved from goo.gl/LnVZAq

Nasrinpour, H., Bavani, A., & Teshnehlab, M. (2017). Grouped bees algorithm: A grouped version of the bees algorithm. *Computers*, *6*(4), 5. https://doi.org/10.3390/computers6010005

Nel Yomtov. (2014). From termite den to office building (p. 32). Cherry Lake Publishing. Retrieved from https://books.google.co.nz/books?id=EPs4ngEACAAJ

NEOS. (2016). Optimization guide. Retrieved January 1, 2016, from https://neos-guide.org/

Ngatchou, P., Zarei, A., & El-Sharkawi, A. (2005). Pareto multi objective optimization. In *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems* (pp. 84–91). https://doi.org/10.1109/ISAP.2005.1599245

Nguyen, T. T., Yang, S., & Branke, J. (2012). Evolutionary dynamic optimization: a survey of the state of the art. *Swarm and Evolutionary Computation*, *6*(Supplement C), 1–24. https://doi.org/10.1016/j.swevo.2012.05.001

Nouioua, M., & Li, Z. (2017). Using differential evolution strategies in chemical reaction optimization for global numerical optimization. *Applied Intelligence*, *47*(3), 935–961. https://doi.org/10.1007/s10489-017-0921-4

Okulewicz, M., & Mańdziuk, J. (2013). Application of particle swarm optimization algorithm to dynamic vehicle routing problem. In L. Rutkowski, M. Korytkowski, R. Scherer Rafałand Tadeusiewicz, L. A. Zadeh, & J. M. Zurada (Eds.), *Artificial Intelligence and Soft Computing: 12th International Conference, ICAISC 2013, Zakopane, Poland, June 9-13, 2013, Proceedings, Part II* (pp. 547–558). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-38610-7_50

Okulewicz, M., & Mańdziuk, J. (2017). The impact of particular components of the PSO-based algorithm solving the dynamic vehicle routing problem. *Applied Soft Computing*, *58*(Supplement C), 586–604. https://doi.org/10.1016/j.asoc.2017.04.070

Oliveira, S. M. De, Souza, R. De, Am, M., & Silva, L. (2008). A solution of dynamic vehicle routing problem with time window via ant colony system metaheuristic, 21–26. https://doi.org/10.1109/SBRN.2008.20

Osaba, E., Yang, X.-S., Diaz, F., López-Garcia, P., & Carballedo, R. (2016). An improved discrete bat algorithm for symmetric and asymmetric traveling salesman problems. *Engineering Applications of Artificial Intelligence*, *48*, 59–71. Retrieved from http://arxiv.org/abs/1604.04138

Ouaarab, A., Ahiod, B., & Yang, X.-S. (2014). Discrete cuckoo search algorithm for the travelling salesman problem. *Neural Computing and Applications*, *24*(7–8), 1659–1669. https://doi.org/10.1007/s00521-013-1402-2

Pang-Ning, T., Steinbach, M., & Kumar, V. (2006). *Introduction to data mining*. *Library of Congress*. https://doi.org/10.1016/0022-4405(81)90007-8

Parpinelli, R. S., Lopes, H. S., & Freitas, A. A. (2002). Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, *6*(4), 321–332. https://doi.org/10.1109/TEVC.2002.802452

Passino, K. M. (2005). *Biomimicry for optimization, control, and automation* (1st Ed.). London: Springer-Verlag. https://doi.org/10.1007/b138169

Paz, A., & Moran, S. (1981). Non deterministic polynomial optimization problems and their approximations. *Theoretical Computer Science*, *15*(3), 251–277. https://doi.org/10.1016/0304-3975(81)90081-5

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Perlman, H. (2015). The water cycle. Retrieved January 1, 2016, from https://water.usgs.gov/edu/watercycle.html

Perretto, M., & Lopes, H. S. (2005). Reconstruction of phylogenetic trees using the ant colony optimization paradigm. *Genetics and Molecular Research*, *4*(3), 581–589.

Pham, D. T., & Castellani, M. (2009). The bees algorithm: modelling foraging behaviour to solve continuous optimization problems. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, *223*(12), 2919–2938. https://doi.org/10.1243/09544062JMES1494

Pham, D. T., & Castellani, M. (2014). Benchmarking and comparison of nature-inspired population-based continuous optimisation algorithms. *Soft Computing*, *18*(5), 871–903. https://doi.org/10.1007/s00500-013-1104-9

Pham, D. T., & Castellani, M. (2015). A comparative study of the bees algorithm as a tool for function optimisation. *Cogent Engineering*, *2*(1), 1091540. https://doi.org/10.1080/23311916.2015.1091540

Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., & Zaidi, M. (2005). *The bees algorithm. Technical note. Manufacturing Engineering Centre (Technical Note MEC 0501)*. Cardiff.

Pham, D. T., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., & Zaidi, M. (2006). The bees algorithm – A novel tool for complex optimisation. In *Intelligent Production Machines and Systems* (pp. 454–459). Elsevier. https://doi.org/10.1016/B978-008045157-2/50081-X

Pillac, V. (2013, December 13). *Dynamic vehicle routing solution methods and computational tools*. *A Quarterly Journal of Operations Research*. Universidad de Los Andes (Colombia). https://doi.org/10.1007/s10288-012-0226-8

Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, *225*(1), 1–11. https://doi.org/10.1016/j.ejor.2012.08.015

Pillac, V., Guéret, C., & Medaglia, A. L. (2010). *Dynamic vehicle routing problems: state of the art and prospects*. Retrieved from http://hal.archives-ouvertes.fr/hal-00623474/%5Cnhttp://hal.archives-ouvertes.fr/docs/00/62/34/74/PDF/Pillac2010_-_Dynamic_vehicle_routing_problems_-_state_of_the_art_and_prospects.pdf

Pillac, V., Guéret, C., & Medaglia, A. L. (2012). An event-driven optimization framework for dynamic vehicle routing. *Decision Support Systems*, *54*(1), 414–423. https://doi.org/10.1016/j.dss.2012.06.007

Potvin, J.-Y. (1996). Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, *63*(3), 337–370. https://doi.org/10.1007/BF02125403

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, *31*(12), 1985–2002. https://doi.org/10.1016/S0305-0548(03)00158-8

Psaraftis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, *14*(2), 130–154. https://doi.org/10.1287/trsc.14.2.130

Psaraftis, H. N. (1995). Dynamic vehicle routing: status and prospects. *Annals of Operations Research, 61*(1), 143–164. https://doi.org/10.1007/BF02098286

Psaraftis, H. N., Wen, M., & Kontovas, C. A. (2016). Dynamic vehicle routing problems: three decades and counting. *Netw.*, *67*(1), 3–31. https://doi.org/10.1002/net.21628

Qi, M., Zhang, Y., Zhang, J., & Miao, L. (2013). A new two-phase hybrid metaheuristic for vehicle routing problem with time windows. *Proceedings of the Eastern Asia Society for Transportation Studies, 9*.

Rabanal, P., Rodríguez, I., & Rubio, F. (2007). Using river formation dynamics to design heuristic algorithms. In *Unconventional Computation* (Vol. 4618, pp. 163–177). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-73554-0_16

Rabanal, P., Rodríguez, I., & Rubio, F. (2008a). Finding minimum spanning/distances trees by using river formation dynamics. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, & A. F. T. Winfield (Eds.), *Ant Colony Optimization and Swarm Intelligence: 6th International Conference, ANTS 2008, Brussels, Belgium, September 22-24, 2008.*

Proceedings (pp. 60–71). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-87527-7_6

Rabanal, P., Rodríguez, I., & Rubio, F. (2008b). Solving dynamic TSP by using river formation dynamics. In *Proceedings - 4th International Conference on Natural Computation, ICNC 2008* (Vol. 1, pp. 246–250). https://doi.org/10.1109/ICNC.2008.760

Rabanal, P., Rodríguez, I., & Rubio, F. (2009). Applying river formation dynamics to solve NP-complete problems. In R. Chiong (Ed.), *Nature-Inspired Algorithms for Optimisation* (pp. 333–368). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-00267-0_12

Ralphs, T. K., Kopman, L., Pulleyblank, W. R., & Trotter, L. E. (2003). On the capacitated vehicle routing problem. *Mathematical Programming*, *94*(2–3), 343–359. https://doi.org/10.1007/s10107-002-0323-0

Rao, A., & Hedge, S. (2015). Literature survey on travelling salesman problem using genetic algorithms. *International Journal of Advanced Research in Eduation Technology (IJARET)*, *2*(1), 42.

Rao Venkata, R., & Savsani, V. J. (2012). *Mechanical design optimization using advanced optimization techniques*. Springer London. https://doi.org/10.1007/978-1-4471-2748-2

Rashedi, E., Nezamabadi-pour, H., & Saryazdi, S. (2009). GSA: A gravitational search algorithm. *Information Sciences*, *179*(13), 2232–2248. https://doi.org/10.1016/j.ins.2009.03.004

Rashedi, E., Nezamabadi-pour, H., & Saryazdi, S. (2010). BGSA: binary gravitational search algorithm. *Natural Computing*, *9*(3), 727–745. https://doi.org/10.1007/s11047-009-9175-3

Reinelt, G. (1991). TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing*, *3*(4), 376–384.

Reinelt, G. (1995). TSPLIB. Retrieved from https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Rizzoli, A. E., Montemanni, R., Lucibello, E., & Gambardella, L. M. (2007). Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence*, *1*(2), 135–151. https://doi.org/10.1007/s11721-007-0005-x

Rocki, K., & Suda, R. (2012). Accelerating 2-opt and 3-opt local search using GPU in the travelling salesman problem. *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, 705–706. https://doi.org/10.1109/CCGrid.2012.133

Rodrigue, J., Comtois, C., & Slack, B. (2006). *The geography of transport systems*. Routledge. https://doi.org/10.4324/9780203001110

Rothlauf, F. (2011). Optimization problems. In *Design of Modern Heuristics: Principles and Application* (pp. 7–44). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-72962-4_2

Rozenberg, G., Bck, T., & Kok, J. N. (2012). *Handbook of natural computing*. (G. Rozenberg, T. Bäck, & J. N. Kok, Eds.) (1st ed.). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-92910-9

Sadollah, A., Eskandar, H., Bahreininejad, A., & Kim, J. H. (2015a). Water cycle algorithm for solving multi-objective optimization problems. *Soft Computing*, *19*(9), 2587–2603. https://doi.org/10.1007/s00500-014-1424-4

Sadollah, A., Eskandar, H., Bahreininejad, A., & Kim, J. H. (2015b). Water cycle algorithm with evaporation rate for solving constrained and unconstrained optimization problems. *Applied Soft Computing*, *30*, 58–71. https://doi.org/10.1016/j.asoc.2015.01.050

Saenphon, T., Phimoltares, S., & Lursinsap, C. (2014). Combining new fast opposite gradient search with ant colony optimization for solving travelling salesman problem. *Engineering Applications of Artificial Intelligence*, *35*, 324–334. https://doi.org/10.1016/j.engappai.2014.06.026

Saiyed, A. R. (2012). *The traveling salesman problem*. Terre Haute, USA. Retrieved from http://cs.indstate.edu/~zeeshan/aman.pdf

Saji, Y., Riffi, M. E., & Ahiod, B. (2014). Discrete bat-inspired algorithm for travelling salesman problem. In *2014 Second World Conference on Complex Systems (WCCS)* (pp. 28–31). https://doi.org/10.1109/ICoCS.2014.7060983

Seçkiner, S. U., Eroğlu, Y., Emrullah, M., & Dereli, T. (2013). Ant colony optimization for continuous functions by using novel pheromone updating. *Applied Mathematics and Computation*, *219*(9), 4163–4175. https://doi.org/10.1016/j.amc.2012.10.097

Shah-Hosseini, H. (2007). Problem solving by intelligent water drops. In *IEEE congress on evolutionary computation* (Vol. 1, pp. 3226–3231). IEEE. https://doi.org/10.1109/CEC.2007.4424885

Shah-Hosseini, H. (2008). Intelligent water drops algorithm: A new optimization method for solving the multiple knapsack problem. *International Journal of Intelligent Computing and Cybernetics*, *1*(2), 193–212. https://doi.org/10.1108/17563780810874717

Shah-Hosseini, H. (2009). The intelligent water drops algorithm: A nature-inspired swarm-based optimization algorithm. *International Journal of Bio-Inspired Computation*, *1*(1/2), 71–79. https://doi.org/10.1504/IJBIC.2009.022775

Shah-Hosseini, H. (2012). An approach to continuous optimization by the intelligent water drops algorithm. *Procedia - Social and Behavioral Sciences*, *32*, 224–229. https://doi.org/10.1016/j.sbspro.2012.01.033

Sharma, S., & Gupta, N. (2016). Routing in wireless mesh network using river formation dynamics. *International Conference on Science, Technology and Management*, *2*(7), 30–46.

Shi, X. H., Liang, Y. C., Lee, H. P., Lu, C., & Wang, Q. X. (2007). Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters*, *103*(5), 169–176. https://doi.org/10.1016/j.ipl.2007.03.010

Shin, K., & Han, S. (2011). A centroid-based heuristic algorithm for the capacitated vehicle routing problem. *Computing and Informatics*, *30*(4), 721–732. Retrieved from http://dblp.uni-trier.de/db/journals/cai/cai30.html#ShinH11

Siddique, N., & Adeli, H. (2015). Nature inspired computing: An overview and some future directions. *Cognitive Computation*, *7*(6), 706–714. https://doi.org/10.1007/s12559-015-9370-8

Singh, S., & Singh, K. P. (2015). Cuckoo search optimization for job shop scheduling problem. In K. N. Das, K. Deep, M. Pant, J. C. Bansal, & A. Nagar (Eds.), *Proceedings of Fourth International Conference on Soft Computing for Problem Solving: SocProS 2014, Volume*

1 (pp. 99–111). New Delhi: Springer India. https://doi.org/10.1007/978-81-322-2217-0_9

Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research, 185*(3), 1155–1173. https://doi.org/10.1016/j.ejor.2006.06.046

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research, 35*(2), 254–265. https://doi.org/10.1287/opre.35.2.254

Southard, J. (2006). *An introduction to fluid motions, sediment transport, and current-generated sedimentary structures*. Ph. D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Spencer, H. (1864). *The principles of biology* (Vol. 1). Williams and Norgate. Retrieved from https://books.google.co.nz/books?id=Ye4GAAAAcAAJ

Spliet, R., Gabor, A. F., & Dekker, R. (2014). The vehicle rescheduling problem. *Computers & Operations Research, 43*(Supplement C), 129–136. https://doi.org/10.1016/j.cor.2013.09.009

Splung. (2015). The phases of matter. Retrieved from http://www.splung.com/content/sid/6/page/phasesofmatter

Srour, A., Othman, Z. A., & Hamdan, A. R. (2014). A water flow-like algorithm for the travelling salesman problem. *Advances in Computer Engineering, 2014*, 1–14. https://doi.org/10.1155/2014/436312

Stadler, W. (1988). Fundamentals of multicriteria optimization. In W. Stadler (Ed.), *Multicriteria Optimization in Engineering and in the Sciences* (pp. 1–25). Boston, MA: Springer US. https://doi.org/10.1007/978-1-4899-3734-6_1

Stattenberger, G., Dankesreiter, M., Baumgartner, F., & Schneider, J. J. (2007). On the neighborhood structure of the traveling salesman problem generated by local search moves. *Journal of Statistical Physics, 129*(4), 623–648. https://doi.org/10.1007/s10955-007-9382-1

Steer, K. C. B., Wirth, A., & Halgamuge, S. K. (2009). The rationale behind seeking inspiration from nature. In *Nature-Inspired Algorithms for Optimisation* (Vol. 193, pp. 51–76). Springer. https://doi.org/10.1007/978-3-642-00267-0_2

Stutzle, T., & Hoos, H. H. (1997). MAX-MIN ant system and local search for the traveling salesman problem. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)* (pp. 309–314). IEEE. https://doi.org/10.1109/ICEC.1997.592327

Stützle, T., & Hoos, H. H. (1996). *Improving the ant system: A detailed report on the MAX--MIN ant system. Technical Report AIDA 96-12*. Germany.

Stützle, T., & Hoos, H. H. (2000). MAX –MIN ant system. *Future Generation Computer Systems, 16*, 889–914. https://doi.org/10.1016/S0167-739X(00)00043-1

Sun, J., Wang, Y., Li, J., & Gao, K. (2011). Hybrid algorithm based on chemical reaction optimization and lin-kernighan local search for the traveling salesman problem. In *2011 Seventh International Conference on Natural Computation* (Vol. 3, pp. 1518–1521). https://doi.org/10.1109/ICNC.2011.6022378

Surjanovic, S., & Bingham, D. (2013). Virtual library of simulation experiments: test functions and datasets. Retrieved January 1, 2016, from http://www.sfu.ca/~ssurjano/index.html

Surlykke, A., Boel Pedersen, S., & Jakobsen, L. (2009). Echolocating bats emit a highly directional sonar sound beam in the field. *Proceedings of the Royal Society B: Biological Sciences*, *276*(1658), 853–860. https://doi.org/10.1098/rspb.2008.1505

Szeto, W. Y., Wu, Y., & Ho, S. C. (2011). An artificial bee colony algorithm for the capacitated vehicle routing problem. *European Journal of Operational Research*, *215*(1), 126–135. https://doi.org/10.1016/j.ejor.2011.06.006

Taha, A., Hachimi, M., & Moudden, A. (2015). Adapted bat algorithm for capacitated vehicle routing problem. *International Review on Computers and Software (IRECOS)*, *10*(6), 610. https://doi.org/10.15866/irecos.v10i6.6512

Takahashi, R. (2005). Solving the traveling salesman problem through genetic algorithms with changing crossover operators. In *Fourth International Conference on Machine Learning and Applications (ICMLA'05)* (pp. 319–324). IEEE. https://doi.org/10.1109/ICMLA.2005.58

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. Wiley Publishing. Retrieved from https://books.google.co.nz/books?id=SIsa6zi5XV8C

Tan, W. F., Lee, L. S., Majid, Z. a, & Seow, H. V. (2012). Ant colony optimization for capacitated vehicle routing problem. *Journal of Computer Science*, *8*(6), 846–852. https://doi.org/10.3844/jcssp.2012.846.852

Tan, Y. (2015). Discrete firework algorithm for combinatorial optimization problem. In *Fireworks Algorithm: A Novel Swarm Intelligence Optimization Method* (pp. 209–226). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-46353-6_13

Tan, Y., & Zhu, Y. (2010). Fireworks algorithm for optimization. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 6145 LNCS, pp. 355–364). https://doi.org/10.1007/978-3-642-13495-1_44

Tian, P., & Yang, Z. (1993). An improved simulated annealing algorithm with genetic characteristics and the traveling salesman problem. *Journal of Information and Optimization Sciences*, *14*(3), 241–255. https://doi.org/10.1080/02522667.1993.10699153

Tian, Y., Song, J., Yao, D., & Hu, J. (2003). Dynamic vehicle routing problem using hybrid ant system. *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, *2*, 970–974. https://doi.org/10.1109/ITSC.2003.1252630

Toth, P., & Daniele, V. (2002). *The vehicle routing problem*. (P. Toth & D. Vigo, Eds.). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718515

Toth, P., & Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, *123*(1–3), 487–512. https://doi.org/10.1016/S0166-218X(01)00351-1

Tran, T. H. (2011). *A water flow algorithm for optimization problems*. National University of Singapore. Retrieved from http://scholarbank.nus.edu.sg/handle/10635/29823

Tsai, C.-F., Tsai, C.-W., & Tseng, C.-C. (2004). A new hybrid heuristic approach for solving large traveling salesman problem. *Information Sciences*, *166*(1), 67–81. https://doi.org/10.1016/j.ins.2003.11.008

Tsai, P. W., Pan, J.-S., Liao, B. Y., Tsai, M. J., & Istanda, V. (2011). Bat algorithm inspired algorithm for solving numerical optimization problems. *Applied Mechanics and Materials*, *148–149*, 134–137. https://doi.org/10.4028/www.scientific.net/AMM.148-149.134

Tuba, M., Subotic, M., & Stanarevic, N. (2011). Modified cuckoo search algorithm for unconstrained optimization problems. In *Proceedings of the 5th European conference on European computing conference* (pp. 263–268). Retrieved from http://dl.acm.org/citation.cfm?id=1991016.1991063

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., & Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, *257*(3), 845–858. https://doi.org/10.1016/j.ejor.2016.08.012

Vaishnav, P., Choudhary, N., & Jain, K. (2017). Traveling salesman problem using genetic algorithm: A survey. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *2*(3), 105–108.

van Laarhoven, P. J. M., & Aarts, E. H. L. (1987). *Simulated annealing: Theory and applications*. *Mathematics and Its Applications* (1st ed., Vol. 37). Dordrecht: Springer Netherlands. https://doi.org/10.1007/978-94-015-7744-1

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, *231*(1), 1–21. https://doi.org/10.1016/j.ejor.2013.02.053

Vinogradov, Y. B. (2009). *Hydrological cycle -III: Surface water runoff*. *Hydrological cycle* (Vol. 3). Eolss Publishers Co Ltd. Retrieved from https://books.google.co.nz/books?id=e33uAAAAMAAJ

Visentini, M. S., Borenstein, D., Li, J.-Q., & Mirchandani, P. B. (2014). Review of real-time vehicle schedule recovery methods in transportation services. *Journal of Scheduling*, *17*(6), 541–567. https://doi.org/10.1007/s10951-013-0339-8

Wallace, J. M., & Hobbs, P. V. (2006). *Atmospheric science: An introductory survey* (Vol. 92). Academic press.

Wang, C., Lin, M., Zhong, Y., & Zhang, H. (2015). Solving travelling salesman problem using multiagent simulated annealing algorithm with instance-based sampling. *International Journal of Computing Science and Mathematics*, *6*(4), 336–353. https://doi.org/10.1504/IJCSM.2015.071818

Wang, J.-S., & Song, J.-D. (2015). Function optimization and parameter performance analysis based on gravitation search algorithm. *Algorithms*, *9*(1), 3. https://doi.org/10.3390/a9010003

Wang, M., Fu, Q., Tong, N., Li, M., & Zhao, Y. (2016). An improved firefly algorithm for traveling salesman problems. In *Proceedings of the 2015 4th National Conference on Electrical, Electronics and Computer Engineering*. Paris, France: Atlantis Press. https://doi.org/10.2991/nceece-15.2016.193

Wang, X., Wu, X., & Hu, X. (2010). A study of urgency vehicle routing disruption management problem. *Journal of Networks*, *5*(12), 1426–1433. https://doi.org/10.4304/jnw.5.12.1426-1433

Wang, Z., Geng, X., & Shao, Z. (2009). An effective simulated annealing algorithm for solving the traveling salesman problem. *Journal of Computational and Theoretical Nanoscience*, *6*(7), 1680–1686. https://doi.org/10.1166/jctn.2009.1230

Warren, R. M., & Warren, R. P. (1958). Basis for judgments of relative brightness. *Journal of the Optical Society of America*, *48*(7), 445. https://doi.org/10.1364/JOSA.48.000445

Wedyan, A. F., & Narayanan, A. (2014). Solving capacitated vehicle routing problem using intelligent water drops algorithm. In *Natural Computation (ICNC), 2014 10th International Conference on* (pp. 469–474). IEEE. https://doi.org/10.1109/ICNC.2014.6975880

Wedyan, A. F., Whalley, J., & Narayanan, A. (2017). Hydrological Cycle Algorithm for Continuous Optimization Problems. *Journal of Optimization*, *2017*, 25. https://doi.org/10.1155/2017/3828420

Wedyan, A. F., Whalley, J., & Narayanan, A. (2018). Solving the Traveling Salesman Problem using Hydrological Cycle Algorithm. *American Journal of Operations Research, 8*(3).

Wilson, N. H. M., & Colvin, N. J. (1977). *Computer control of the Rochester dial-a-ride system*. Massachusetts Institute of Technology, Center for Transportation Studies. Retrieved from https://books.google.co.nz/books?id=j7ceAQAAMAAJ

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. https://doi.org/10.1109/4235.585893

Wong, K.-C. (2015). Evolutionary algorithms: Concepts, designs, and applications in bioinformatics. In *Nature-Inspired Computing* (Vol. 1508.00468, pp. 111–137). IGI Global. https://doi.org/10.4018/978-1-5225-0788-8.ch006

Wu, X.-B., Liao, J., & Wang, Z.-C. (2015). Water wave optimization for the traveling salesman problem. In D.-S. Huang, V. Bevilacqua, & P. Premaratne (Eds.), *Intelligent Computing Theories and Methodologies: 11th International Conference, ICIC 2015, Fuzhou, China, August 20-23, 2015, Proceedings, Part I* (pp. 137–146). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-22180-9_14

Wu, X., & Gao, D. (2017). A Study on Greedy Search to Improve Simulated Annealing for Large-Scale Traveling Salesman Problem. In Y. Tan, H. Takagi, Y. Shi, & B. Niu (Eds.) (pp. 250–257). Springer International Publishing. https://doi.org/10.1007/978-3-319-61833-3_26

Wu, X., Zhou, Y., & Lu, Y. (2017). Elite opposition-based water wave optimization algorithm for global optimization. *Mathematical Problems in Engineering*, *2017*, 1–25. https://doi.org/10.1155/2017/3498363

Xiao, J., & Li, L. (2011). A hybrid ant colony optimization for continuous domains. *Expert Systems with Applications*, *38*(9), 11072–11077. https://doi.org/10.1016/j.eswa.2011.02.151

Xiao, Y., Zhao, Q., Kaku, I., & Mladenovic, N. (2014). Variable neighbourhood simulated annealing algorithm for capacitated vehicle routing problems. *Engineering Optimization*, *46*(4), 562–579. https://doi.org/10.1080/0305215X.2013.791813

Xing, B., & Gao, W.-J. (2013). *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms* (Vol. 62). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-03404-1

Yaghoobi, T., & Esmaeili Toudeshki, E. (2016). An improved artificial bee colony algorithm for global numerical optimisation. *International Journal of Bio-Inspired Computation*, *1*(1), 1. https://doi.org/10.1504/IJBIC.2016.10004305

Yang, F.-C., & Ni, B. (2014). Water flow-like optimization algorithm for multi-objective continuous optimization problem. In E. Qi, J. Shen, & R. Dou (Eds.), *Proceedings of 2013 4th International Asia Conference on Industrial Engineering and Management Innovation (IEMI2013)* (pp. 471–487). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-40060-5_46

Yang, F.-C., & Wang, Y.-P. (2007). Water flow-like algorithm for object grouping problems. *Journal of the Chinese Institute of Industrial Engineers*, *24*(6), 475–488. https://doi.org/10.1080/10170660709509062

Yang, J., Jaillet, P., & Mahmassani, H. (2004). Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, *38*(2), 135–148. https://doi.org/10.1287/trsc.1030.0068

Yang, N., Li, P., & Li, M. (2007). An angle-based crossover tabu search for vehicle routing problem. In D.-S. Huang, L. Heutte, & M. Loog (Eds.), *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence: Third International Conference on Intelligent Computing, ICIC 2007, Qingdao, China, August 21-24, 2007. Proceedings* (pp. 1215–1222). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-74205-0_125

Yang, X.-S. (2005). Biology-derived algorithms in engineering optimization. *Handbook of Bioinspired Algorithms and Applications*, *1003.1888*, 589–600.

Yang, X.-S. (2008). *Nature-inspired metaheuristic algorithms*. Luniver Press. Retrieved from https://books.google.co.nz/books?id=6w0xh6V4sscC

Yang, X.-S. (2009). Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms* (Vol. 5792, pp. 169–178). https://doi.org/10.1007/978-3-642-04944-6_14

Yang, X.-S. (2010). A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)* (Vol. 284, pp. 65–74). Springer. https://doi.org/10.1007/978-3-642-12538-6_6

Yang, X.-S. (2014a). *Nature-inspired optimization algorithms* (1st ed.). Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V.

Yang, X.-S. (2014b). Swarm intelligence based algorithms: A critical analysis. *Evolutionary Intelligence*, *7*(1), 17–28. https://doi.org/10.1007/s12065-013-0102-2

Yang, X.-S., Cui, Z., Xiao, R., Gandomi, A. H., & Karamanoglu, M. (2013). *Swarm intelligence and bio-inspired computation: Theory and applications*. Elsevier. Retrieved from https://books.google.co.nz/books?id=nEDjmAEACAAJ

Yang, X.-S., & Deb, S. (2009). Cuckoo search via Lévy flights. In *2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009 - Proceedings* (pp. 210–214). https://doi.org/10.1109/NABIC.2009.5393690

Yang, X.-S., & Deb, S. (2013). Multiobjective cuckoo search for design optimization. *Computers and Operations Research*, *40*(6), 1616–1624. https://doi.org/10.1016/j.cor.2011.09.026

Yang, X.-S., Deb, S., Fong, S., He, X., & Zhao, Y.-X. (2016). From swarm intelligence to metaheuristics: Nature-inspired optimization algorithms. *Computer*, *49*(9), 52–59. https://doi.org/10.1109/MC.2016.292

Yang, X.-S., & Hossein Gandomi, A. (2012). Bat algorithm: A novel approach for global engineering optimization. *Engineering Computations*, *29*(5), 464–483. https://doi.org/10.1108/02644401211235834

Yang, X.-S., & Karamanoglu, M. (2013). Swarm intelligence and bio-inspired computation: An overview. In *Swarm Intelligence and Bio-Inspired Computation* (pp. 3–23). Elsevier. https://doi.org/10.1016/B978-0-12-405163-8.00001-6

Yavuz, G., Aydin, D., & Stutzle, T. (2016). Self-adaptive search equation-based artificial bee colony algorithm on the CEC 2014 benchmark functions. In *2016 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1173–1180). IEEE. https://doi.org/10.1109/CEC.2016.7743920

Yazdani, S., Nezamabadi-pour, H., & Kamyab, S. (2014). A gravitational search algorithm for multimodal optimization. *Swarm and Evolutionary Computation*, *14*, 1–14. https://doi.org/10.1016/j.swevo.2013.08.001

Yilmaz, S., & Küçüksille, E. U. (2015). A new modification approach on bat algorithm for solving optimization problems. *Appl. Soft Comput.*, *28*(C), 259–275. https://doi.org/10.1016/j.asoc.2014.11.029

Yin, B., Guo, Z., Liang, Z., & Yue, X. (2017). Improved gravitational search algorithm with crossover. *Computers & Electrical Engineering*. https://doi.org/10.1016/j.compeleceng.2017.06.001

Yu, G. (2016). A new multi-population-based artificial bee colony for numerical optimisation. *International Journal of Computing Science and Mathematics*, *7*(6), 509. https://doi.org/10.1504/IJCSM.2016.081695

Yu, G., & Qi, X. (2004). *Disruption management: Framework, models and applications*. World Scientific. Retrieved from https://books.google.co.nz/books?id=pb9pDQAAQBAJ

Yu, J. J. Q., Lam, A. Y. S., & Li, V. O. K. (2015). Adaptive chemical reaction optimization for global numerical optimization. In *2015 IEEE Congress on Evolutionary Computation (CEC)* (pp. 3192–3199). IEEE. https://doi.org/10.1109/CEC.2015.7257288

Zainudin, S., Kerwad, M. M., & Othman, Z. A. (2015). A water flow-like algorithm for capacitated vehicle routing problem. *Journal of Theoretical & Applied Information Technology*, *77*(1), 125–135.

Zang, H., Zhang, S., & Hapeshi, K. (2010). A review of nature-inspired algorithms. *Journal of Bionic Engineering*, *7*, S232–S237. https://doi.org/10.1016/S1672-6529(09)60240-7

Zhan, S., Lin, J., Zhang, Z., & Zhong, Y. (2016). List-based simulated annealing algorithm for traveling salesman problem. *Computational Intelligence and Neuroscience*, *2016*, 1–12. https://doi.org/10.1155/2016/1712630

Zhang, A., Sun, G., Ren, J., Li, X., Wang, Z., & Jia, X. (2018). A Dynamic Neighborhood Learning-Based Gravitational Search Algorithm. *IEEE Transactions on Cybernetics*, *48*(1), 436–447. https://doi.org/10.1109/TCYB.2016.2641986

Zhang, B., Liu, T., Zhang, C., & Wang, P. (2017). Artificial bee colony algorithm with strategy and parameter adaptation for global optimization. *Neural Computing and Applications*, *28*(S1), 349–364. https://doi.org/10.1007/s00521-016-2348-y

Zhang, L., Yang, F., & Elsherbeni, A. Z. (2009). On the use of random variables in particle swarm optimizations: a comparative study of Gaussian and uniform distributions. *Journal of Electromagnetic Waves and Applications*, *23*(5–6), 711–721. https://doi.org/10.1163/156939309788019787

Zhang, S. Z., & Lee, C. K. M. (2015). An improved artificial bee colony algorithm for the capacitated vehicle routing problem. In *2015 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 2124–2128). IEEE. https://doi.org/10.1109/SMC.2015.371

Zhang, X., Zhang, H., & Gao, M. (2010). Continuous ant colony optimization algorithm based on crossover and mutation. In *Natural Computation (ICNC), 2010 Sixth International Conference on* (Vol. 5, pp. 2605–2608). https://doi.org/10.1109/ICNC.2010.5583072

Zhang, Y., Wang, S., & Ji, G. (2015). A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical Problems in Engineering*, *2015*, 38. https://doi.org/10.1155/2015/931256

Zheng, Y. J. (2015). Water wave optimization: A new nature-inspired metaheuristic. *Computers and Operations Research*, *55*, 1–11. https://doi.org/10.1016/j.cor.2014.10.008

Zhong, W., Zhang, J., & Chen, W. (2007). A novel discrete particle swarm optimization to solve traveling salesman problem. In *2007 IEEE Congress on Evolutionary Computation* (pp. 3283–3287). https://doi.org/10.1109/CEC.2007.4424894

Appendix A: Mathematical Formulation of Functions

Table A-1 shows the mathematical formulations for the used test functions, which have been taken from (Pham et al., 2006; Surjanovic & Bingham, 2013).

Table A-1. The mathematical formulations

Function name	Mathematical formulations
Ackley	$f(x) = -a\, exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$ $a = 20, b = 0.2\ and\ c = 2\pi$
Cross-In-Tray	$f(x) = -0.0001\left(\left\|\sin(x_1)\sin(x_2)\exp\left(\left\|100 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi}\right\|\right)\right\| + 1\right)^{0.1}$
Drop-Wave	$f(x) = -\frac{1 + \cos\left(12\sqrt{x_1^2 + x_2^2}\right)}{0.5\,(x_1^2 + x_2^2) + 2}$
Gramacy & Lee (2012)	$f(x) = \frac{\sin(10\pi x)}{2x} + (x-1)^4$
Griewank	$f(x) = \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Holder Table	$f(x) = -\left\|\sin(x_1)\cos(x_2)\exp\left(\left\|1 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi}\right\|\right)\right\|$
Levy	$f(x) = \sin^2(3\pi w_1) + \sum_{i=1}^{d-1}(w_i - 1)^2\,[1 + 10\sin^2(\pi w_i + 1)]$ $+ (w_d - 1)^2\,[1 + \sin^2(2\pi w_d)]$ $where,\ w_i = 1 + \frac{x_i - 1}{4}, for\ all\ i = 1, \dots, d$
Levy N. 13	$f(x) = \sin^2(3\pi x_1) + (x_1 - 1)^2\,[1 + \sin^2(3\pi x_2)] + (x_2 - 1)^2\,[1 + \sin^2(3\pi x_2)]$
Rastrigin	$f(x) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i)]$
Schaffer N.2	$f(x) = 0.5 + \frac{\sin^2(x_1^2 + x_2^2) - 0.5}{[1 + 0.001\,(x_1^2 + x_2^2)]^2}$
Schaffer N.4	$f(x) = 0.5 + \frac{\cos(\sin(\|x_1^2 - x_2^2\|)) - 0.5}{[1 + 0.001\,(x_1^2 + x_2^2)]^2}$

Function name	Mathematical formulations
Schwefel	$f(x) = 418.9829d - \sum_{i=1}^{d} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$
Shubert	$f(x) = \left(\sum_{i=1}^{5} i\ \cos\big((i+1)\,x_1 + i\big)\right)\left(\sum_{i=1}^{5} i\ \cos\big((i+1)\,x_2 + i\big)\right)$
Bohachevsky	$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$
Rotated Hyper-Ellipsoid	$f(x) = \sum_{i=1}^{d}\sum_{j=1}^{i} x_j^2$
Sphere (Hyper)	$f(x) = \sum_{i=1}^{d} x_i^2$
Sum Squares	$f(x) = \sum_{i=1}^{d} i x_i^2$
Booth	$f(x) = (x_1 + 2x_2 - 7)^2 - (2x_1 + x_2 - 5)^2$
Matyas	$f(x) = 0.26(x_1^2 + \ x_2^2) - 0.48x_1x_2$
McCormick	$f(x) = \sin(x_1 + x_2) + \ (x_1 + x_2)^2 - \ 1.5x_1 + 2.5x_2 + 1$
Zakharov	$f(x) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^4$
Three-Hump Camel	$f(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$
Six-Hump Camel	$f(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$
Dixon-Price	$f(x) = (\mathrm{x}_1 - 1)^2 + \sum_{i=1}^{d} i(2x_i^2 - x_{i-1})^2$
Rosenbrock	$f(x) = \sum_{i=1}^{d-1}[100(x_{i+1} - x_i^2)^2 + (\mathrm{x}_i - 1)^2]$
Shekel's Foxholes (De Jong N.5)	$f(x) = \left(0.002 + \sum_{i=1}^{25}\frac{1}{i + (x_1 - a_{1i})^6 + (x_2 - a_{2i})^6}\right)^{-1}, where$ $a = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \dots & 32 & 32 & 32 \end{pmatrix}$
Easom	$f(x) = -\cos(x_1)\cos(x_2)\ exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$

Function name	Mathematical formulations
Michalewicz	$f(x) = -\sum_{i=1}^{d} \sin(x_i)\ \sin^{2m}\left(\frac{ix_i^2}{\pi}\right), where\ m = 10$
Beale	$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$
Branin	$f(x) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos(x_1) + s$ $a = 1, b = \frac{5.1}{(4\pi^2)}, c = \frac{5}{\pi}, r = 6, s = 10, and\ t = \frac{1}{8\pi}$
Goldstein-Price I	$f(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $\times[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$
Goldstein-Price II	$f(x) = \exp\left\{\frac{1}{2}(x_1^2 + x_2^2 - 25)\right\}^2 + \sin^4(4x_1 - 3x_2) + \frac{1}{2}(2x_1 + x_2 - 10)^2$
Styblinski-Tang	$f(x) = \frac{1}{2}\sum_{i=1}^{d}\left(x_i^4 - 16x_i^2 + 5x_i\right)$
Gramacy & Lee (2008)	$f(x) = x_1 \exp(-x_1^2 - x_2^2)$
Martin & Gaddy	$f(x) = (x_1 - x_2)^2 + \left((x_1 + x_2 - 10)/3\right)^2$
Easton and Fenton	$f(x) = \left(12 + x_1^2 + \frac{1 + x_2^2}{x_1^2} + \frac{x_1^2x_2^2 + 100}{(x_1x_2)^4}\right).\left(\frac{1}{10}\right)$
Hartmann 3-D	$f(x) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{3} A_{ij}\left(x_j - p_{ij}\right)^2\right), where$ $\alpha = (1.0, 1.2, 3.0, 3.2)^T$ $A = \begin{pmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}$ $P = 10^{-4}\begin{pmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{pmatrix}$
Powell	$f(x) = \sum_{i=1}^{d/4}[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2x_{4i-1})^4$ $+ 10(x_{4i-3} - x_{4i})^4]$
Wood	$f(x_1, x_2, x_3, x_4) = 100(x_1 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2$ $+ 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$
DeJong	$\max f(x) = (3905.93) - 100(x_1^2 - x_2)^2 - (1 - x_1)^2$

Appendix B: HCA for Classification and Knowledge Discovery

Introduction

In recent years, *classification* and *association-rule discovery* from complex and big data have attracted increasing interest (Fayyad, Piatetsky-Shapiro, & Smyth, 1996). Classification is a data mining procedure that assigns a set of records to a particular class label using a classifier (Pang-Ning, Steinbach, & Kumar, 2006). Association-rule discovery finds the interesting and hidden relations between the data variables that are helpful for understanding the data structure (Duch, 2013). Data are collections of univariate/multivariate variables with different values (Pang-Ning et al., 2006). The classifier establishes a set of rules by processing a training set, by whch it predicts the category (class label) of a new unclassified (unseen) dataset. The training set contains a set of attributes (variables), one of which represents the class label. Each attribute also has a set of values (Martens et al., 2007). A typical classifier uses rules of the form *IF–THEN–CONCLUSION*, the simplest and most comprehensive way of expressing knowledge (Parpinelli, Lopes, & Freitas, 2002), as follows:

$$Rule = IF\ Condition_1(x)\ AND\ Condition_2(x)\ AND\ \ldots.Condition_n(x)\ THEN\ Conclusion\ (x)$$

In the past years, classification and data mining have focused on data occurring in the same time period or with no reliance on the temporal dimension. However, many real-world applications produce data that continually change over time, known as *temporal data* or *time-series data*. Classification of such datasets, called *temporal classification,* allows each attribute value in the dataset to change with every time period. The classification aims to find a time-associated relationship among the variables (Cotofrei & Stoffel, 2002). Temporal datasets are generated by many real-world applications, such as the stock market, currency exchange rates, clinical sensors in medical treatments, and sensors for observing weather or atmospheric phenomena.

The inputs to a temporal classifier are the feature values, their classes, and their histories, that is, the previous feature values up to a certain time point. However, some of the many techniques developed for temporal classification suffer from weaknesses or limitations. For example, some techniques generate a large number of attributes in the dataset or even a large number of values for each attribute. Finding the most important classification features (feature extraction) and generating the temporal rules in such circumstances can be challenging. Therefore, researchers now focus on metaheuristic algorithms for this type of classification.

Much research has been devoted to classification and knowledge discovery. Some of this research is based on traditional or normal data analysis techniques such as the decision tree, nearest-neighbour methods, and dynamic time warping. Data mining has added evolutionary and swarm algorithms to its arsenal. This section proposes the possible application of HCA as a classifier in temporal classification, finds the temporal association rules between the variables, and discovers the likely patterns in the data. The rules are discovered directly from the training set, then used to classify or identify the data categories. The purpose of this classifier is to deal with a non-static dataset to predict the class of other unclassified records as shown in Figure B-1.

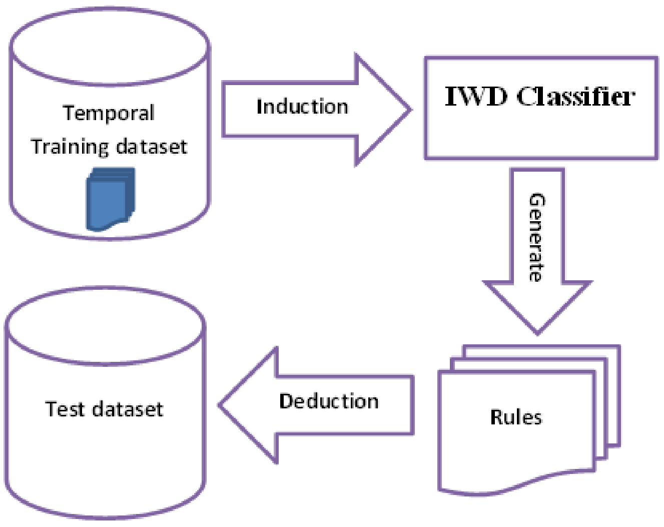


Figure B-1. Schematic of the classification procedure

Initially, the classifier can be developed and evaluated on a synthetic temporal dataset in which temporal relationships are known to exist, and then applied to some benchmark datasets. Moreover, the prediction accuracy rate of the temporal HCA can be evaluated and compared with those of other temporal techniques.

HCA classification approach

Data analysis is rendered complex by the many-valued relationships. The difficulty is enhanced when there is a huge amount of collected data. Therefore, to analyse data by the HCA and find the association rules and any hidden patterns, we must proceed through the following stages.

Stage 1: Data segmentation

The first stage is the segmentation process, a very important step that discretises and identifies the structural changes in the data. A data series can be segmented in many ways. One technique is the sliding-windows technique, which divides (partitions) the data based on some time interval. The time window slides along the data until the end of the series. Sometimes the time window is varied to collect information about the short- and long-term data series. In some data series, the segmentation process assigns each data interval a specific label, and analyses the sequence of labels to obtain new knowledge about the time-series.

Stage 2: Data representation:

Data representation facilitates the way of finding association rules, and improves the efficiency and effectiveness of generating the rules. To analyse the data and generate the rules using the HCA algorithm, we can use the graph representation of COPs. The graph consists of several layers, each representing one variable. Moreover, each layer contains a set of disconnected nodes representing the possible variable values. Therefore, the numbers of layers and nodes depend on the number of variables and number of possible values of each variable, respectively. The last layer contains the names of the classes. Figure B-2 shows a proposed graph representation of normal data classification, where *V* represents a possible variable value, and *C* represents a class name.

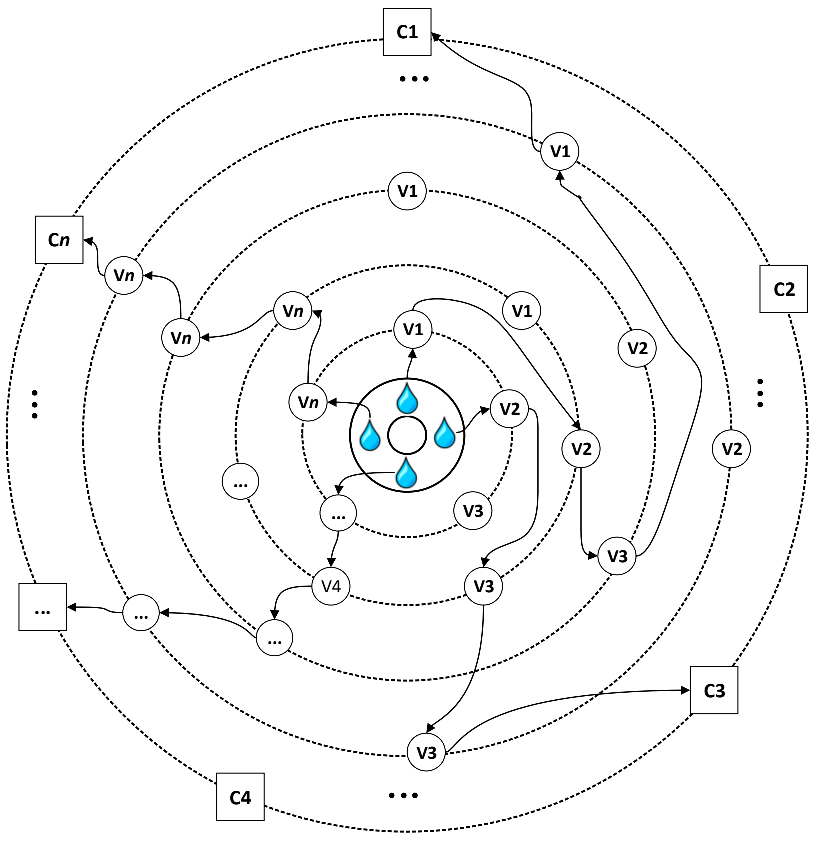


Figure B-2. Data representation of normal data classification

In Figure B-2, a water drop moves between the nodes by selecting a single node from each layer until it reaches the last layer (the class labels). The possible rules are generated as follows:

Rule1 = IF (variable$_1$ = V$_1$) AND (variable$_2$ = V$_1$) AND…. AND (variable $_n$ = V$_2$) THEN Class = C$_4$.

Rule2 = IF (variable$_1$ = V$_3$) AND (variable$_2$ = V$_n$) AND…. AND (variable $_n$ = V$_3$) THEN Class = C$_2$.

Representation of temporal data

To represent temporal data, we can apply the previously proposed graph with slight modifications. In this graph, each layer represents the variable values at a specific time. The values can be binary, discrete numbers, or text labels. In each layer, the nodes are connected by edges, and each node represents the value of one variable at a specific time. The nodes in layer T_n layer are directly connected to those in layer T_{n+1}. Figure B-3 shows a possible way of representing temporal data.

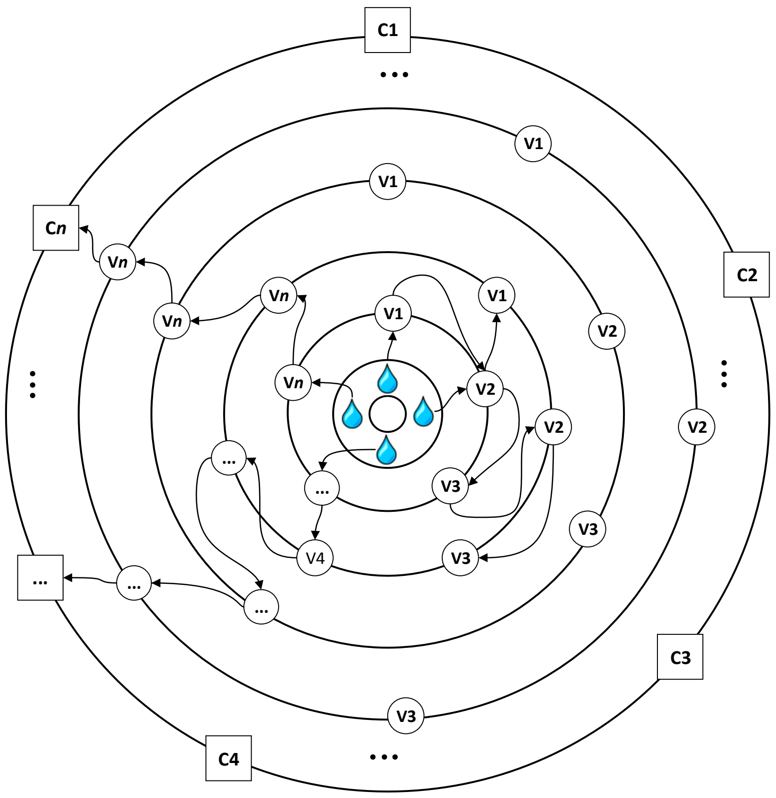


Figure B-3. Data representation in temporal data classification

In Figure B-3, a water drop can move between nodes in the same layer before moving to the next layer.

Stage 3: Rule Generation

The data are divided into portions based on the sliding-window width. The HCA is then continuously applied to each portion of the data until the end of the series. The rules generated in each portion depend on the movement of the water drops between the nodes in that portion. When moving to the next node, a water drop chooses its path under some probability transition rule that depends on the amount of soil deposited on each edge and the depth-value heuristics. The generated rules are formulated as follows:

IF (variable$_1$ = V_3) at T_1 AND…. AND (variable $_n$ = V_n) at T_1 ➔ *THEN (variable$_2$ = V_3) at T_1.*

IF (variable$_2$ = V_3) at T_1 AND…. AND (variable$_2$ = V_1) at T_2 ➔ *THEN (variable $_n$ = V_3) at T_3.*

IF (variable$_3$ = V_2) at T_1 AND…. AND (variable$_1$ = V_3) at T_2 ➔ *THEN Class = C_2 at T_3.*

In these expressions, *V* and *T* represent the variable value and time, respectively. As an example of an association rule, if variable *A* rises and variable *B* falls, then variable *C* rises in the next time period. As another example, if variable *A* rises at a specific time and declines over the next time period while variable *B* remains level over the same period, then variable *C* remains level over that period.

Stage 4: Testing and evaluation

The HCA can be initially tested on a synthetic database of binary variables, where each variable is randomly assigned a value of 0 or 1 in each time period. The generated rules are stored for justifying their accuracy later. For classification purposes, a specific number of the highly accurate rules are then selected and tested on unseen data. After applying HCA to the synthetic data and tuning its parameters, the HCA can be tested on a benchmark dataset. The rule accuracy can be measured by repeating the rule over each period of the sliding window, or by computing the support and confidence measures of each rule.

The possible outcomes of this study are listed below:

- A natural-inspired algorithm (HCA) can be implemented and modified to deal with temporal (time-series) data. The association rules (if/then statements) between multivariate variables in large databases can be found.
- A new way of representing temporal data in a graph format.
- A tool can identify the hidden patterns in the multivariate variables.

Appendix C: DCVRP Instances

The following tables listed the used instances, where *D* represents the demands of customers and *AT* represents available time.

Table C-1. Data for C101-10-90 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	40	50	0	0	42	35	32	10	104
6	42	65	10	0	55	42	10	40	105
16	20	80	40	0	74	92	30	10	106
21	30	50	10	0	28	23	52	10	110
33	10	40	30	0	35	8	45	20	113
44	33	35	10	0	81	85	25	10	118
58	40	15	40	0	47	30	32	30	132
68	47	40	10	0	45	32	30	10	139
89	65	60	30	0	101	55	85	20	140
91	60	55	10	0	41	35	30	10	158
99	58	75	20	0	49	28	30	10	171
8	40	66	20	2	80	87	30	10	174
85	70	58	20	2	75	53	35	50	194
84	72	55	10	5	63	50	35	20	198
97	60	80	10	5	20	15	80	10	214
82	85	35	30	6	78	88	30	10	218
11	35	66	10	7	10	38	70	10	223
18	18	75	20	7	76	45	65	20	230
14	22	75	30	8	59	38	5	30	243
30	20	50	10	8	27	25	55	10	245
73	53	30	10	8	95	65	82	10	245
70	45	35	10	13	83	75	55	20	250
88	65	55	20	13	61	35	5	20	262
9	38	68	20	14	38	2	40	20	297
64	50	40	50	16	86	68	60	30	298
25	25	50	10	23	39	0	40	30	299
77	90	35	10	24	29	23	55	20	328
43	33	32	20	26	98	60	85	30	329
52	25	30	10	27	57	40	5	30	344
37	5	45	10	28	2	45	68	10	367
66	48	40	10	37	71	95	30	30	379
94	65	85	40	38	50	28	35	10	388
56	42	15	10	41	12	35	69	10	405
51	26	32	10	47	15	22	85	10	430
4	42	66	10	54	65	48	30	10	437
31	20	55	10	57	17	20	85	40	464
96	62	80	30	57	5	42	68	10	467
26	25	52	40	62	23	28	52	20	468
34	8	40	40	69	62	50	30	10	476
7	40	69	20	73	92	60	60	10	479
36	5	35	10	73	46	30	30	10	497
72	95	35	20	75	60	38	15	10	511
93	67	85	20	76	40	0	45	20	553
32	10	35	20	78	67	47	35	10	597
54	44	5	20	79	13	25	85	20	628
53	25	35	10	80	3	45	70	30	696
79	88	35	20	89	90	63	58	10	702
19	15	75	20	100	69	45	30	10	705
24	28	55	10	100	48	30	35	10	751
87	66	55	10	101	22	30	52	20	797
100	55	80	10	102					

Table C-2. Data for C101-70-30 instance

ID	**X**	**Y**	**D**	**AT**	**ID**	**X**	**Y**	**D**	**AT**
1	40	50	0	0	26	25	52	40	62
6	42	65	10	0	34	8	40	40	69
7	40	69	20	0	93	67	85	20	76
10	38	70	10	0	32	10	35	20	78
11	35	66	10	0	54	44	5	20	79
15	22	85	10	0	53	25	35	10	80
16	20	80	40	0	79	88	35	20	89
21	30	50	10	0	19	15	75	20	100
28	23	52	10	0	24	28	55	10	100
33	10	40	30	0	87	66	55	10	101
35	8	45	20	0	100	55	80	10	102
36	5	35	10	0	42	35	32	10	104
38	2	40	20	0	55	42	10	40	105
41	35	30	10	0	81	85	25	10	118
44	33	35	10	0	47	30	32	30	132
56	42	15	10	0	45	32	30	10	139
58	40	15	40	0	49	28	30	10	171
68	47	40	10	0	80	87	30	10	174
71	95	30	30	0	75	53	35	50	194
72	95	35	20	0	63	50	35	20	198
74	92	30	10	0	20	15	80	10	214
77	90	35	10	0	76	45	65	20	230
78	88	30	10	0	59	38	5	30	243
84	72	55	10	0	27	25	55	10	245
85	70	58	20	0	95	65	82	10	245
89	65	60	30	0	83	75	55	20	250
90	63	58	10	0	61	35	5	20	262
91	60	55	10	0	86	68	60	30	298
97	60	80	10	0	39	0	40	30	299
99	58	75	20	0	29	23	55	20	328
101	55	85	20	0	98	60	85	30	329
8	40	66	20	2	57	40	5	30	344
82	85	35	30	6	2	45	68	10	367
18	18	75	20	7	50	28	35	10	388
14	22	75	30	8	12	35	69	10	405
30	20	50	10	8	65	48	30	10	437
73	53	30	10	8	17	20	85	40	464
70	45	35	10	13	5	42	68	10	467
88	65	55	20	13	23	28	52	20	468
9	38	68	20	14	62	50	30	10	476
64	50	40	50	16	92	60	60	10	479
25	25	50	10	23	46	30	30	10	497
43	33	32	20	26	60	38	15	10	511
52	25	30	10	27	40	0	45	20	553
37	5	45	10	28	67	47	35	10	597
66	48	40	10	37	13	25	85	20	628
94	65	85	40	38	3	45	70	30	696
51	26	32	10	47	69	45	30	10	705
4	42	66	10	54	48	30	35	10	751
31	20	55	10	57	22	30	52	20	797
96	62	80	30	57					

Table C-3. Data for C101-50-50 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	40	50	0	0	8	40	66	20	2
2	45	68	10	0	82	85	35	30	6
4	42	66	10	0	18	18	75	20	7
6	42	65	10	0	70	45	35	10	13
7	40	69	20	0	9	38	68	20	14
10	38	70	10	0	64	50	40	50	16
11	35	66	10	0	43	33	32	20	26
13	25	85	20	0	52	25	30	10	27
14	22	75	30	0	37	5	45	10	28
15	22	85	10	0	66	48	40	10	37
16	20	80	40	0	51	26	32	10	47
19	15	75	20	0	96	62	80	30	57
21	30	50	10	0	26	25	52	40	62
24	28	55	10	0	34	8	40	40	69
25	25	50	10	0	93	67	85	20	76
28	23	52	10	0	32	10	35	20	78
29	23	55	20	0	54	44	5	20	79
30	20	50	10	0	53	25	35	10	80
31	20	55	10	0	87	66	55	10	101
33	10	40	30	0	100	55	80	10	102
35	8	45	20	0	55	42	10	40	105
36	5	35	10	0	81	85	25	10	118
38	2	40	20	0	49	28	30	10	171
41	35	30	10	0	75	53	35	50	194
42	35	32	10	0	63	50	35	20	198
44	33	35	10	0	20	15	80	10	214
45	32	30	10	0	76	45	65	20	230
47	30	32	30	0	59	38	5	30	243
56	42	15	10	0	27	25	55	10	245
57	40	5	30	0	95	65	82	10	245
58	40	15	40	0	83	75	55	20	250
67	47	35	10	0	61	35	5	20	262
68	47	40	10	0	86	68	60	30	298
71	95	30	30	0	39	0	40	30	299
72	95	35	20	0	98	60	85	30	329
73	53	30	10	0	50	28	35	10	388
74	92	30	10	0	12	35	69	10	405
77	90	35	10	0	65	48	30	10	437
78	88	30	10	0	17	20	85	40	464
79	88	35	20	0	5	42	68	10	467
80	87	30	10	0	23	28	52	20	468
84	72	55	10	0	62	50	30	10	476
85	70	58	20	0	92	60	60	10	479
88	65	55	20	0	46	30	30	10	497
89	65	60	30	0	60	38	15	10	511
90	63	58	10	0	40	0	45	20	553
91	60	55	10	0	3	45	70	30	696
94	65	85	40	0	69	45	30	10	705
97	60	80	10	0	48	30	35	10	751
99	58	75	20	0	22	30	52	20	797
101	55	85	20	0					

Table C-4. Data for C101-70-30 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	40	50	0	0	73	53	30	10	0
2	45	68	10	0	74	92	30	10	0
4	42	66	10	0	77	90	35	10	0
6	42	65	10	0	78	88	30	10	0
7	40	69	20	0	79	88	35	20	0
10	38	70	10	0	80	87	30	10	0
11	35	66	10	0	84	72	55	10	0
13	25	85	20	0	85	70	58	20	0
14	22	75	30	0	88	65	55	20	0
15	22	85	10	0	89	65	60	30	0
16	20	80	40	0	90	63	58	10	0
18	18	75	20	0	91	60	55	10	0
19	15	75	20	0	92	60	60	10	0
21	30	50	10	0	93	67	85	20	0
24	28	55	10	0	94	65	85	40	0
25	25	50	10	0	95	65	82	10	0
26	25	52	40	0	97	60	80	10	0
27	25	55	10	0	98	60	85	30	0
28	23	52	10	0	99	58	75	20	0
29	23	55	20	0	101	55	85	20	0
30	20	50	10	0	8	40	66	20	2
31	20	55	10	0	82	85	35	30	6
33	10	40	30	0	9	38	68	20	14
35	8	45	20	0	43	33	32	20	26
36	5	35	10	0	52	25	30	10	27
37	5	45	10	0	66	48	40	10	37
38	2	40	20	0	96	62	80	30	57
41	35	30	10	0	34	8	40	40	69
42	35	32	10	0	32	10	35	20	78
44	33	35	10	0	53	25	35	10	80
45	32	30	10	0	87	66	55	10	101
47	30	32	30	0	100	55	80	10	102
48	30	35	10	0	81	85	25	10	118
49	28	30	10	0	75	53	35	50	194
50	28	35	10	0	63	50	35	20	198
51	26	32	10	0	20	15	80	10	214
54	44	5	20	0	76	45	65	20	230
55	42	10	40	0	59	38	5	30	243
56	42	15	10	0	83	75	55	20	250
57	40	5	30	0	61	35	5	20	262
58	40	15	40	0	86	68	60	30	298
60	38	15	10	0	39	0	40	30	299
62	50	30	10	0	12	35	69	10	405
64	50	40	50	0	17	20	85	40	464
65	48	30	10	0	5	42	68	10	467
67	47	35	10	0	23	28	52	20	468
68	47	40	10	0	46	30	30	10	497
69	45	30	10	0	40	0	45	20	553
70	45	35	10	0	3	45	70	30	696
71	95	30	30	0	22	30	52	20	797
72	95	35	20	0					

Table C-5. Data for C101-90-10 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	40	50	0	0	57	40	5	30	0
2	45	68	10	0	58	40	15	40	0
4	42	66	10	0	60	38	15	10	0
5	42	68	10	0	61	35	5	20	0
6	42	65	10	0	62	50	30	10	0
7	40	69	20	0	63	50	35	20	0
8	40	66	20	0	64	50	40	50	0
9	38	68	20	0	65	48	30	10	0
10	38	70	10	0	67	47	35	10	0
11	35	66	10	0	68	47	40	10	0
12	35	69	10	0	69	45	30	10	0
13	25	85	20	0	70	45	35	10	0
14	22	75	30	0	71	95	30	30	0
15	22	85	10	0	72	95	35	20	0
16	20	80	40	0	73	53	30	10	0
18	18	75	20	0	74	92	30	10	0
19	15	75	20	0	75	53	35	50	0
20	15	80	10	0	77	90	35	10	0
21	30	50	10	0	78	88	30	10	0
22	30	52	20	0	79	88	35	20	0
23	28	52	20	0	80	87	30	10	0
24	28	55	10	0	81	85	25	10	0
25	25	50	10	0	84	72	55	10	0
26	25	52	40	0	85	70	58	20	0
27	25	55	10	0	86	68	60	30	0
28	23	52	10	0	87	66	55	10	0
29	23	55	20	0	88	65	55	20	0
30	20	50	10	0	89	65	60	30	0
31	20	55	10	0	90	63	58	10	0
32	10	35	20	0	91	60	55	10	0
33	10	40	30	0	92	60	60	10	0
35	8	45	20	0	93	67	85	20	0
36	5	35	10	0	94	65	85	40	0
37	5	45	10	0	95	65	82	10	0
38	2	40	20	0	96	62	80	30	0
41	35	30	10	0	97	60	80	10	0
42	35	32	10	0	98	60	85	30	0
43	33	32	20	0	99	58	75	20	0
44	33	35	10	0	100	55	80	10	0
45	32	30	10	0	101	55	85	20	0
46	30	30	10	0	82	85	35	30	6
47	30	32	30	0	66	48	40	10	37
48	30	35	10	0	34	8	40	40	69
49	28	30	10	0	76	45	65	20	230
50	28	35	10	0	59	38	5	30	243
51	26	32	10	0	83	75	55	20	250
52	25	30	10	0	39	0	40	30	299
53	25	35	10	0	17	20	85	40	464
54	44	5	20	0	40	0	45	20	553
55	42	10	40	0	3	45	70	30	696
56	42	15	10	0					

Table C-6. Data for R101-10-90 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	35	35	0	0	11	30	60	16	39
6	15	30	26	0	39	5	5	16	39
10	55	60	16	0	3	35	17	7	40
15	15	10	20	0	27	45	30	17	40
18	5	30	2	0	48	8	56	27	41
28	35	40	16	0	13	50	35	19	43
43	24	12	5	0	82	55	54	26	45
60	21	24	28	0	86	16	22	41	45
64	27	69	10	0	99	19	21	10	45
73	47	16	25	0	61	17	34	3	46
93	22	22	2	0	17	10	20	19	47
20	15	60	17	1	68	67	5	25	48
26	65	20	6	1	89	26	52	9	48
58	32	12	7	2	92	15	19	1	48
16	30	5	8	4	70	37	47	6	49
32	31	52	27	7	83	15	47	16	49
84	14	37	11	7	63	24	58	19	51
5	55	20	19	9	30	64	42	9	54
25	65	35	3	9	59	36	26	18	55
46	6	38	16	9	94	18	24	22	55
77	49	42	13	10	54	37	31	14	62
79	61	52	3	10	49	13	52	36	63
8	20	50	5	11	62	12	24	13	63
53	27	43	9	11	31	40	60	21	64
7	25	30	3	12	35	65	55	14	64
36	63	65	8	12	4	55	45	13	65
37	2	60	5	12	12	20	65	12	65
24	55	5	29	13	19	20	40	12	68
23	45	10	18	14	52	49	58	10	68
81	56	37	6	15	67	49	73	25	69
96	25	24	20	15	101	18	18	17	69
22	45	20	11	16	21	45	65	9	73
45	11	14	18	16	72	57	68	15	74
14	30	25	23	17	56	63	23	2	76
57	53	12	6	21	98	25	21	12	77
87	4	18	35	21	50	6	68	30	83
88	28	18	26	21	44	23	3	7	87
41	40	25	9	24	95	26	27	27	88
40	60	12	31	25	75	46	13	8	91
34	53	52	11	26	55	57	29	18	92
42	42	7	5	28	85	11	31	7	92
91	31	67	3	29	2	41	49	10	94
66	62	77	20	30	47	2	48	1	96
76	49	11	18	30	71	37	56	5	106
29	41	37	16	32	51	47	47	13	108
74	44	17	9	32	38	20	20	8	123
80	57	48	23	32	78	53	43	14	134
97	22	27	11	32	69	56	39	36	136
9	10	43	9	37	33	35	69	23	137
100	20	26	9	37	90	26	35	15	170
65	15	77	9	38					

Table C-7. Data for R101-70-30 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	35	35	0	0	40	60	12	31	25
6	15	30	26	0	34	53	52	11	26
10	55	60	16	0	42	42	7	5	28
15	15	10	20	0	91	31	67	3	29
16	30	5	8	0	76	49	11	18	30
18	5	30	2	0	29	41	37	16	32
23	45	10	18	0	74	44	17	9	32
24	55	5	29	0	80	57	48	23	32
28	35	40	16	0	97	22	27	11	32
31	40	60	21	0	9	10	43	9	37
32	31	52	27	0	100	20	26	9	37
35	65	55	14	0	65	15	77	9	38
36	63	65	8	0	11	30	60	16	39
39	5	5	16	0	3	35	17	7	40
43	24	12	5	0	27	45	30	17	40
44	23	3	7	0	48	8	56	27	41
45	11	14	18	0	13	50	35	19	43
52	49	58	10	0	82	55	54	26	45
55	57	29	18	0	86	16	22	41	45
60	21	24	28	0	99	19	21	10	45
62	12	24	13	0	61	17	34	3	46
64	27	69	10	0	17	10	20	19	47
66	62	77	20	0	89	26	52	9	48
68	67	5	25	0	92	15	19	1	48
69	56	39	36	0	83	15	47	16	49
70	37	47	6	0	63	24	58	19	51
72	57	68	15	0	30	64	42	9	54
73	47	16	25	0	59	36	26	18	55
78	53	43	14	0	94	18	24	22	55
87	4	18	35	0	54	37	31	14	62
93	22	22	2	0	49	13	52	36	63
20	15	60	17	1	4	55	45	13	65
26	65	20	6	1	12	20	65	12	65
58	32	12	7	2	19	20	40	12	68
84	14	37	11	7	67	49	73	25	69
5	55	20	19	9	101	18	18	17	69
25	65	35	3	9	21	45	65	9	73
46	6	38	16	9	56	63	23	2	76
77	49	42	13	10	98	25	21	12	77
79	61	52	3	10	50	6	68	30	83
8	20	50	5	11	95	26	27	27	88
53	27	43	9	11	75	46	13	8	91
7	25	30	3	12	85	11	31	7	92
37	2	60	5	12	2	41	49	10	94
81	56	37	6	15	47	2	48	1	96
96	25	24	20	15	71	37	56	5	106
22	45	20	11	16	51	47	47	13	108
14	30	25	23	17	38	20	20	8	123
57	53	12	6	21	33	35	69	23	137
88	28	18	26	21	90	26	35	15	170
41	40	25	9	24					

Table C-8. Data for R101-50-50 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	35	35	0	0	20	15	60	17	1
4	55	45	13	0	26	65	20	6	1
6	15	30	26	0	58	32	12	7	2
10	55	60	16	0	84	14	37	11	7
12	20	65	12	0	5	55	20	19	9
14	30	25	23	0	46	6	38	16	9
15	15	10	20	0	8	20	50	5	11
16	30	5	8	0	53	27	43	9	11
18	5	30	2	0	7	25	30	3	12
21	45	65	9	0	37	2	60	5	12
23	45	10	18	0	96	25	24	20	15
24	55	5	29	0	22	45	20	11	16
25	65	35	3	0	88	28	18	26	21
28	35	40	16	0	41	40	25	9	24
29	41	37	16	0	40	60	12	31	25
31	40	60	21	0	34	53	52	11	26
32	31	52	27	0	42	42	7	5	28
33	35	69	23	0	74	44	17	9	32
35	65	55	14	0	80	57	48	23	32
36	63	65	8	0	97	22	27	11	32
38	20	20	8	0	9	10	43	9	37
39	5	5	16	0	100	20	26	9	37
43	24	12	5	0	11	30	60	16	39
44	23	3	7	0	3	35	17	7	40
45	11	14	18	0	27	45	30	17	40
49	13	52	36	0	48	8	56	27	41
52	49	58	10	0	13	50	35	19	43
55	57	29	18	0	82	55	54	26	45
57	53	12	6	0	86	16	22	41	45
59	36	26	18	0	61	17	34	3	46
60	21	24	28	0	17	10	20	19	47
62	12	24	13	0	89	26	52	9	48
64	27	69	10	0	92	15	19	1	48
65	15	77	9	0	83	15	47	16	49
66	62	77	20	0	63	24	58	19	51
68	67	5	25	0	30	64	42	9	54
69	56	39	36	0	94	18	24	22	55
70	37	47	6	0	54	37	31	14	62
72	57	68	15	0	19	20	40	12	68
73	47	16	25	0	67	49	73	25	69
76	49	11	18	0	56	63	23	2	76
77	49	42	13	0	98	25	21	12	77
78	53	43	14	0	50	6	68	30	83
79	61	52	3	0	95	26	27	27	88
81	56	37	6	0	75	46	13	8	91
85	11	31	7	0	2	41	49	10	94
87	4	18	35	0	47	2	48	1	96
91	31	67	3	0	71	37	56	5	106
93	22	22	2	0	51	47	47	13	108
99	19	21	10	0	90	26	35	15	170
101	18	18	17	0					

Table C-9. Data for R101-70-30 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	35	35	0	0	72	57	68	15	0
3	35	17	7	0	73	47	16	25	0
4	55	45	13	0	75	46	13	8	0
6	15	30	26	0	76	49	11	18	0
9	10	43	9	0	77	49	42	13	0
10	55	60	16	0	78	53	43	14	0
12	20	65	12	0	79	61	52	3	0
14	30	25	23	0	80	57	48	23	0
15	15	10	20	0	81	56	37	6	0
16	30	5	8	0	83	15	47	16	0
17	10	20	19	0	85	11	31	7	0
18	5	30	2	0	86	16	22	41	0
19	20	40	12	0	87	4	18	35	0
20	15	60	17	0	89	26	52	9	0
21	45	65	9	0	91	31	67	3	0
22	45	20	11	0	93	22	22	2	0
23	45	10	18	0	95	26	27	27	0
24	55	5	29	0	96	25	24	20	0
25	65	35	3	0	99	19	21	10	0
27	45	30	17	0	101	18	18	17	0
28	35	40	16	0	26	65	20	6	1
29	41	37	16	0	84	14	37	11	7
31	40	60	21	0	5	55	20	19	9
32	31	52	27	0	46	6	38	16	9
33	35	69	23	0	8	20	50	5	11
34	53	52	11	0	53	27	43	9	11
35	65	55	14	0	7	25	30	3	12
36	63	65	8	0	37	2	60	5	12
38	20	20	8	0	88	28	18	26	21
39	5	5	16	0	41	40	25	9	24
43	24	12	5	0	40	60	12	31	25
44	23	3	7	0	42	42	7	5	28
45	11	14	18	0	74	44	17	9	32
49	13	52	36	0	97	22	27	11	32
50	6	68	30	0	100	20	26	9	37
52	49	58	10	0	11	30	60	16	39
54	37	31	14	0	48	8	56	27	41
55	57	29	18	0	13	50	35	19	43
57	53	12	6	0	82	55	54	26	45
58	32	12	7	0	61	17	34	3	46
59	36	26	18	0	92	15	19	1	48
60	21	24	28	0	30	64	42	9	54
62	12	24	13	0	94	18	24	22	55
63	24	58	19	0	67	49	73	25	69
64	27	69	10	0	56	63	23	2	76
65	15	77	9	0	98	25	21	12	77
66	62	77	20	0	2	41	49	10	94
68	67	5	25	0	47	2	48	1	96
69	56	39	36	0	51	47	47	13	108
70	37	47	6	0	90	26	35	15	170
71	37	56	5	0					

Table C-10. Data for R101-90-10 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	35	35	0	0	59	36	26	18	0
3	35	17	7	0	60	21	24	28	0
4	55	45	13	0	61	17	34	3	0
5	55	20	19	0	62	12	24	13	0
6	15	30	26	0	63	24	58	19	0
8	20	50	5	0	64	27	69	10	0
9	10	43	9	0	65	15	77	9	0
10	55	60	16	0	66	62	77	20	0
11	30	60	16	0	67	49	73	25	0
12	20	65	12	0	68	67	5	25	0
13	50	35	19	0	69	56	39	36	0
14	30	25	23	0	70	37	47	6	0
15	15	10	20	0	71	37	56	5	0
16	30	5	8	0	72	57	68	15	0
17	10	20	19	0	73	47	16	25	0
18	5	30	2	0	74	44	17	9	0
19	20	40	12	0	75	46	13	8	0
20	15	60	17	0	76	49	11	18	0
21	45	65	9	0	77	49	42	13	0
22	45	20	11	0	78	53	43	14	0
23	45	10	18	0	79	61	52	3	0
24	55	5	29	0	80	57	48	23	0
25	65	35	3	0	81	56	37	6	0
26	65	20	6	0	82	55	54	26	0
27	45	30	17	0	83	15	47	16	0
28	35	40	16	0	84	14	37	11	0
29	41	37	16	0	85	11	31	7	0
31	40	60	21	0	86	16	22	41	0
32	31	52	27	0	87	4	18	35	0
33	35	69	23	0	89	26	52	9	0
34	53	52	11	0	90	26	35	15	0
35	65	55	14	0	91	31	67	3	0
36	63	65	8	0	93	22	22	2	0
38	20	20	8	0	94	18	24	22	0
39	5	5	16	0	95	26	27	27	0
40	60	12	31	0	96	25	24	20	0
41	40	25	9	0	97	22	27	11	0
43	24	12	5	0	99	19	21	10	0
44	23	3	7	0	100	20	26	9	0
45	11	14	18	0	101	18	18	17	0
47	2	48	1	0	46	6	38	16	9
49	13	52	36	0	7	25	30	3	12
50	6	68	30	0	37	2	60	5	12
51	47	47	13	0	88	28	18	26	21
52	49	58	10	0	42	42	7	5	28
53	27	43	9	0	48	8	56	27	41
54	37	31	14	0	92	15	19	1	48
55	57	29	18	0	30	64	42	9	54
56	63	23	2	0	98	25	21	12	77
57	53	12	6	0	2	41	49	10	94
58	32	12	7	0					

Table C-11. Data for RC101-10-90 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	40	50	0	0	16	2	40	20	34
15	5	45	10	0	57	50	35	19	34
36	67	85	20	0	63	65	35	3	34
43	55	80	10	0	70	31	52	27	34
46	20	82	10	0	68	64	42	9	35
73	63	65	8	0	59	15	10	20	36
74	2	60	5	0	67	41	37	16	36
84	37	31	14	0	41	60	85	30	38
93	53	43	14	0	23	40	15	40	40
96	56	37	6	0	95	57	48	23	42
97	55	54	26	0	89	24	58	19	43
35	85	35	30	2	62	45	65	9	44
30	90	35	10	3	39	62	80	30	48
66	35	40	16	3	75	20	20	8	49
3	22	75	30	4	28	95	35	20	51
6	20	85	20	5	42	58	75	20	51
38	65	82	10	5	65	45	30	17	51
88	12	24	13	5	86	63	23	2	51
34	85	25	10	8	82	49	58	10	52
31	88	30	10	9	100	26	35	15	52
83	27	43	9	9	44	55	85	20	54
32	88	35	20	10	58	30	25	23	55
54	20	50	5	10	76	5	5	16	59
22	40	5	10	11	13	8	45	20	60
99	26	52	9	11	10	10	35	20	61
33	87	30	10	12	85	57	29	18	62
17	0	40	20	13	4	22	85	10	63
60	10	20	19	13	79	8	56	27	64
78	23	3	7	14	9	15	80	10	65
2	25	85	20	16	19	44	5	20	71
12	8	40	40	17	26	35	5	20	72
53	25	30	3	20	87	21	24	28	76
27	95	30	30	22	55	55	60	16	77
48	2	45	10	22	7	18	75	20	82
64	65	20	6	22	56	30	60	16	83
72	65	55	14	22	80	6	68	30	86
37	65	85	40	23	51	72	35	30	88
94	61	52	3	23	5	20	80	40	89
18	0	45	20	24	81	47	47	13	90
29	92	30	10	24	47	18	80	10	92
20	42	10	40	25	98	4	18	35	92
50	42	12	10	26	101	31	67	3	93
71	35	69	23	26	90	67	5	25	97
8	15	75	20	27	61	15	60	17	106
91	37	47	6	27	14	5	35	10	108
52	55	20	19	28	11	10	40	30	111
77	60	12	31	30	25	38	15	10	121
40	60	80	10	32	69	40	60	21	136
21	42	15	10	33	49	42	5	10	145
24	38	5	30	33	92	49	42	13	145
45	55	82	10	33					

Table C-12. Data for RC101-30-70 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	40	50	0	0	37	65	85	40	23
6	20	85	20	0	94	61	52	3	23
9	15	80	10	0	29	92	30	10	24
10	10	35	20	0	20	42	10	40	25
11	10	40	30	0	8	15	75	20	27
15	5	45	10	0	52	55	20	19	28
18	0	45	20	0	24	38	5	30	33
19	44	5	20	0	16	2	40	20	34
21	42	15	10	0	57	50	35	19	34
27	95	30	30	0	63	65	35	3	34
35	85	35	30	0	70	31	52	27	34
36	67	85	20	0	68	64	42	9	35
40	60	80	10	0	59	15	10	20	36
43	55	80	10	0	67	41	37	16	36
45	55	82	10	0	41	60	85	30	38
46	20	82	10	0	23	40	15	40	40
50	42	12	10	0	89	24	58	19	43
54	20	50	5	0	62	45	65	9	44
66	35	40	16	0	39	62	80	30	48
71	35	69	23	0	75	20	20	8	49
73	63	65	8	0	28	95	35	20	51
74	2	60	5	0	42	58	75	20	51
77	60	12	31	0	65	45	30	17	51
81	47	47	13	0	86	63	23	2	51
84	37	31	14	0	82	49	58	10	52
91	37	47	6	0	100	26	35	15	52
93	53	43	14	0	44	55	85	20	54
95	57	48	23	0	58	30	25	23	55
96	56	37	6	0	76	5	5	16	59
97	55	54	26	0	13	8	45	20	60
98	4	18	35	0	85	57	29	18	62
30	90	35	10	3	4	22	85	10	63
3	22	75	30	4	79	8	56	27	64
38	65	82	10	5	26	35	5	20	72
88	12	24	13	5	87	21	24	28	76
34	85	25	10	8	55	55	60	16	77
31	88	30	10	9	7	18	75	20	82
83	27	43	9	9	56	30	60	16	83
32	88	35	20	10	80	6	68	30	86
22	40	5	10	11	51	72	35	30	88
99	26	52	9	11	5	20	80	40	89
33	87	30	10	12	47	18	80	10	92
17	0	40	20	13	101	31	67	3	93
60	10	20	19	13	90	67	5	25	97
78	23	3	7	14	61	15	60	17	106
2	25	85	20	16	14	5	35	10	108
12	8	40	40	17	25	38	15	10	121
53	25	30	3	20	69	40	60	21	136
48	2	45	10	22	49	42	5	10	145
64	65	20	6	22	92	49	42	13	145
72	65	55	14	22					

Table C-13. Data for RC101-50-50 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	40	50	0	0	30	90	35	10	3
2	25	85	20	0	38	65	82	10	5
3	22	75	30	0	88	12	24	13	5
4	22	85	10	0	34	85	25	10	8
6	20	85	20	0	83	27	43	9	9
8	15	75	20	0	22	40	5	10	11
9	15	80	10	0	33	87	30	10	12
10	10	35	20	0	17	0	40	20	13
11	10	40	30	0	60	10	20	19	13
13	8	45	20	0	78	23	3	7	14
14	5	35	10	0	12	8	40	40	17
15	5	45	10	0	53	25	30	3	20
18	0	45	20	0	48	2	45	10	22
19	44	5	20	0	72	65	55	14	22
21	42	15	10	0	37	65	85	40	23
25	38	15	10	0	94	61	52	3	23
27	95	30	30	0	29	92	30	10	24
31	88	30	10	0	20	42	10	40	25
32	88	35	20	0	52	55	20	19	28
35	85	35	30	0	24	38	5	30	33
36	67	85	20	0	16	2	40	20	34
40	60	80	10	0	57	50	35	19	34
41	60	85	30	0	63	65	35	3	34
42	58	75	20	0	70	31	52	27	34
43	55	80	10	0	59	15	10	20	36
45	55	82	10	0	67	41	37	16	36
46	20	82	10	0	23	40	15	40	40
50	42	12	10	0	89	24	58	19	43
54	20	50	5	0	39	62	80	30	48
56	30	60	16	0	75	20	20	8	49
58	30	25	23	0	28	95	35	20	51
62	45	65	9	0	65	45	30	17	51
64	65	20	6	0	82	49	58	10	52
66	35	40	16	0	100	26	35	15	52
68	64	42	9	0	44	55	85	20	54
69	40	60	21	0	76	5	5	16	59
71	35	69	23	0	85	57	29	18	62
73	63	65	8	0	79	8	56	27	64
74	2	60	5	0	26	35	5	20	72
77	60	12	31	0	87	21	24	28	76
81	47	47	13	0	55	55	60	16	77
84	37	31	14	0	7	18	75	20	82
86	63	23	2	0	80	6	68	30	86
90	67	5	25	0	51	72	35	30	88
91	37	47	6	0	5	20	80	40	89
93	53	43	14	0	47	18	80	10	92
95	57	48	23	0	101	31	67	3	93
96	56	37	6	0	61	15	60	17	106
97	55	54	26	0	49	42	5	10	145
98	4	18	35	0	92	49	42	13	145
99	26	52	9	0					

Table C-14. Data for RC101-70-30 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	40	50	0	0	71	35	69	23	0
2	25	85	20	0	73	63	65	8	0
3	22	75	30	0	74	2	60	5	0
4	22	85	10	0	76	5	5	16	0
6	20	85	20	0	77	60	12	31	0
7	18	75	20	0	81	47	47	13	0
8	15	75	20	0	83	27	43	9	0
9	15	80	10	0	84	37	31	14	0
10	10	35	20	0	86	63	23	2	0
11	10	40	30	0	89	24	58	19	0
13	8	45	20	0	90	67	5	25	0
14	5	35	10	0	91	37	47	6	0
15	5	45	10	0	93	53	43	14	0
16	2	40	20	0	95	57	48	23	0
18	0	45	20	0	96	56	37	6	0
19	44	5	20	0	97	55	54	26	0
21	42	15	10	0	98	4	18	35	0
22	40	5	10	0	99	26	52	9	0
24	38	5	30	0	100	26	35	15	0
25	38	15	10	0	101	31	67	3	0
27	95	30	30	0	30	90	35	10	3
29	92	30	10	0	88	12	24	13	5
31	88	30	10	0	33	87	30	10	12
32	88	35	20	0	17	0	40	20	13
34	85	25	10	0	60	10	20	19	13
35	85	35	30	0	78	23	3	7	14
36	67	85	20	0	12	8	40	40	17
37	65	85	40	0	53	25	30	3	20
38	65	82	10	0	48	2	45	10	22
39	62	80	30	0	72	65	55	14	22
40	60	80	10	0	94	61	52	3	23
41	60	85	30	0	20	42	10	40	25
42	58	75	20	0	52	55	20	19	28
43	55	80	10	0	57	50	35	19	34
45	55	82	10	0	70	31	52	27	34
46	20	82	10	0	23	40	15	40	40
47	18	80	10	0	75	20	20	8	49
50	42	12	10	0	28	95	35	20	51
54	20	50	5	0	65	45	30	17	51
55	55	60	16	0	82	49	58	10	52
56	30	60	16	0	44	55	85	20	54
58	30	25	23	0	85	57	29	18	62
59	15	10	20	0	79	8	56	27	64
61	15	60	17	0	26	35	5	20	72
62	45	65	9	0	87	21	24	28	76
63	65	35	3	0	80	6	68	30	86
64	65	20	6	0	51	72	35	30	88
66	35	40	16	0	5	20	80	40	89
67	41	37	16	0	49	42	5	10	145
68	64	42	9	0	92	49	42	13	145
69	40	60	21	0					

Table C-15. Data for RC101-90-10 instance

ID	X	Y	D	AT	ID	X	Y	D	AT
1	40	50	0	0	58	30	25	23	0
2	25	85	20	0	59	15	10	20	0
3	22	75	30	0	60	10	20	19	0
4	22	85	10	0	61	15	60	17	0
5	20	80	40	0	62	45	65	9	0
6	20	85	20	0	63	65	35	3	0
7	18	75	20	0	64	65	20	6	0
8	15	75	20	0	65	45	30	17	0
9	15	80	10	0	66	35	40	16	0
10	10	35	20	0	67	41	37	16	0
11	10	40	30	0	68	64	42	9	0
12	8	40	40	0	69	40	60	21	0
13	8	45	20	0	70	31	52	27	0
14	5	35	10	0	71	35	69	23	0
15	5	45	10	0	72	65	55	14	0
16	2	40	20	0	73	63	65	8	0
17	0	40	20	0	74	2	60	5	0
18	0	45	20	0	75	20	20	8	0
19	44	5	20	0	76	5	5	16	0
20	42	10	40	0	77	60	12	31	0
21	42	15	10	0	78	23	3	7	0
22	40	5	10	0	81	47	47	13	0
23	40	15	40	0	82	49	58	10	0
24	38	5	30	0	83	27	43	9	0
25	38	15	10	0	84	37	31	14	0
27	95	30	30	0	85	57	29	18	0
29	92	30	10	0	86	63	23	2	0
30	90	35	10	0	88	12	24	13	0
31	88	30	10	0	89	24	58	19	0
32	88	35	20	0	90	67	5	25	0
33	87	30	10	0	91	37	47	6	0
34	85	25	10	0	92	49	42	13	0
35	85	35	30	0	93	53	43	14	0
36	67	85	20	0	95	57	48	23	0
37	65	85	40	0	96	56	37	6	0
38	65	82	10	0	97	55	54	26	0
39	62	80	30	0	98	4	18	35	0
40	60	80	10	0	99	26	52	9	0
41	60	85	30	0	100	26	35	15	0
42	58	75	20	0	101	31	67	3	0
43	55	80	10	0	48	2	45	10	22
44	55	85	20	0	94	61	52	3	23
45	55	82	10	0	52	55	20	19	28
46	20	82	10	0	57	50	35	19	34
47	18	80	10	0	28	95	35	20	51
49	42	5	10	0	79	8	56	27	64
50	42	12	10	0	26	35	5	20	72
53	25	30	3	0	87	21	24	28	76
54	20	50	5	0	80	6	68	30	86
55	55	60	16	0	51	72	35	30	88
56	30	60	16	0					

Appendix D: HCA-DCVRP Results

Figure D-1 shows a solution for C101-10-90 instance.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Instance name: C101-10-90
Day length: 1000
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The initial solution for known customers
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The cost for the route 1 is 215.000
The load of the vehicle 1 is 200
The cost for the route 2 is 20.000
The load of the vehicle 2 is 10
-----------------------------------------------
The total cost for the all routes is 235.000
-----------------------------------------------
The customers will be visited as follows:
     1    68    58    44    33    16     6    99    89    91     1
     1    21     1

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The beginning of the day, current time is: 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicle 2 has served customer 21 at: 10, and his demands: 10
Vehicle 1 has served customer 68 at: 12, and his demands: 10
Vehicle 2 has returned to the depot at: 20, and the remaining load:190
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 15 New customers are available at time: 23 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    58
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 173.000
The load of the vehicle 1 is 160
The cost for the route 2 is 120.000
The load of the vehicle 2 is 200
The cost for the route 3 is 94.000
The load of the vehicle 3 is 110
-----------------------------------------------
The total cost for the all routes is 387.000
-----------------------------------------------
The customers will be visited as follows:
     1    58    44    70    64    73    82    84     1
     1     6     8     9    11    14    16    18    33    30    25     1
     1    91    88    85    89    97    99     1
==============================================================
End updating the current solution
==============================================================
Vehicle 3 has served customer 6 at: 37, and his demands: 10
Vehicle 1 has served customer 58 at: 38, and his demands: 40
Vehicle 3 has served customer 8 at: 39, and his demands: 20
Vehicle 3 has served customer 9 at: 42, and his demands: 20
Vehicle 4 has served customer 91 at: 43, and his demands: 10
Vehicle 3 has served customer 11 at: 46, and his demands: 10
Vehicle 4 has served customer 88 at: 48, and his demands: 20
Vehicle 4 has served customer 85 at: 54, and his demands: 20
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 11 New customers are available at time: 57 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    44    14    89
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 102.000
The load of the vehicle 1 is 140
The cost for the route 2 is 86.000
The load of the vehicle 2 is 120
The cost for the route 3 is 185.000
The load of the vehicle 3 is 120
The cost for the route 4 is 89.000
```

```
The load of the vehicle 4 is 110
----------------------------------------------
The total cost for the all routes is 462.000
----------------------------------------------
The customers will be visited as follows:
     1    44    43    51    52    56    73    70    64    66     1
     1    14    16    18    31    30    25     1
     1    89    84    77    82    33    37     1
     1    96    94    97    99     4     1
==============================================================
End updating the current solution
==============================================================
Vehicle 1 has served customer 44 at: 59, and his demands: 10
Vehicle 4 has served customer 89 at: 59, and his demands: 30
Vehicle 1 has served customer 43 at: 62, and his demands: 20
Vehicle 3 has served customer 14 at: 62, and his demands: 30
Vehicle 3 has served customer 16 at: 67, and his demands: 40
Vehicle 4 has served customer 84 at: 68, and his demands: 10
Vehicle 1 has served customer 51 at: 69, and his demands: 10
Vehicle 1 has served customer 52 at: 71, and his demands: 10
Vehicle 3 has served customer 18 at: 72, and his demands: 20
Vehicle 3 has served customer 31 at: 92, and his demands: 10
Vehicle 5 has served customer 96 at: 93, and his demands: 30
Vehicle 1 has served customer 56 at: 94, and his demands: 10
Vehicle 4 has served customer 77 at: 95, and his demands: 10
Vehicle 3 has served customer 30 at: 97, and his demands: 10
Vehicle 5 has served customer 94 at: 99, and his demands: 40
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 12 New customers are available at time: 100 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    73    25    82    97
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 55.000
The load of the vehicle 1 is 80
The cost for the route 2 is 30.000
The load of the vehicle 2 is 10
The cost for the route 3 is 161.000
The load of the vehicle 3 is 90
The cost for the route 4 is 143.000
The load of the vehicle 4 is 110
The cost for the route 5 is 92.000
The load of the vehicle 5 is 160
----------------------------------------------
The total cost for the all routes is 481.000
----------------------------------------------
The customers will be visited as follows:
     1    73    70    66    64     1
     1    25     1
     1    82    79    72    54     1
     1    97    93    99     4     7    19    24     1
     1    53    32    36    33    34    37    26     1
==============================================================
End updating the current solution
==============================================================
Vehicle 4 has served customer 82 at: 100, and his demands: 30
Vehicle 3 has served customer 25 at: 102, and his demands: 10
Vehicle 4 has served customer 79 at: 103, and his demands: 20
Vehicle 5 has served customer 97 at: 106, and his demands: 10
Vehicle 4 has served customer 72 at: 110, and his demands: 20
Vehicle 1 has served customer 73 at: 113, and his demands: 10
Vehicle 5 has served customer 93 at: 115, and his demands: 20
Vehicle 3 has returned to the depot at: 117, and the remaining load:20
Vehicle 6 has served customer 53 at: 120, and his demands: 10
Vehicle 1 has served customer 70 at: 122, and his demands: 10
Vehicle 1 has served customer 66 at: 128, and his demands: 10
Vehicle 5 has served customer 99 at: 128, and his demands: 20
Vehicle 1 has served customer 64 at: 130, and his demands: 50
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 9 New customers are available at time: 132 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    54     4    32
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 90.000
```

```
The load of the vehicle 1 is 20
The cost for the route 2 is 39.000
The load of the vehicle 2 is 30
The cost for the route 3 is 91.000
The load of the vehicle 3 is 180
The cost for the route 4 is 244.000
The load of the vehicle 4 is 150
----------------------------------------------
The total cost for the all routes is 464.000
----------------------------------------------
The customers will be visited as follows:
     1    54     1
     1     4     7     1
     1    32    36    34    33    37    35    28    26     1
     1    47    42    55    81    74    87   100    19    24     1
==============================================================
End updating the current solution
==============================================================
Vehicle 6 has served customer 32 at: 135, and his demands: 20
Vehicle 6 has served customer 36 at: 140, and his demands: 10
Vehicle 1 has returned to the depot at: 144, and the remaining load:10
Vehicle 5 has served customer 4 at: 146, and his demands: 10
Vehicle 6 has served customer 34 at: 146, and his demands: 40
Vehicle 6 has served customer 33 at: 148, and his demands: 30
Vehicle 5 has served customer 7 at: 150, and his demands: 20
Vehicle 7 has served customer 47 at: 152, and his demands: 30
Vehicle 6 has served customer 37 at: 155, and his demands: 10
Vehicle 7 has served customer 42 at: 157, and his demands: 10
Vehicle 6 has served customer 35 at: 158, and his demands: 20
Vehicle 4 has served customer 54 at: 169, and his demands: 20
Vehicle 5 has returned to the depot at: 169, and the remaining load:50
Vehicle 6 has served customer 28 at: 175, and his demands: 10
Vehicle 6 has served customer 26 at: 177, and his demands: 40
Vehicle 7 has served customer 55 at: 180, and his demands: 40
Vehicle 6 has returned to the depot at: 192, and the remaining load:10
Vehicle 4 has returned to the depot at: 214, and the remaining load:10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 9 New customers are available at time: 218 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    81
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 211.000
The load of the vehicle 1 is 120
The cost for the route 2 is 70.000
The load of the vehicle 2 is 100
----------------------------------------------
The total cost for the all routes is 281.000
----------------------------------------------
The customers will be visited as follows:
     1    81    80    78    74    87   100   101    20    19    24     1
     1    63    75    41    45    49     1
==============================================================
End updating the current solution
==============================================================
Vehicle 7 has served customer 81 at: 226, and his demands: 10
Vehicle 7 has served customer 80 at: 231, and his demands: 10
Vehicle 7 has served customer 78 at: 232, and his demands: 10
Vehicle 8 has served customer 63 at: 235, and his demands: 20
Vehicle 7 has served customer 74 at: 236, and his demands: 10
Vehicle 8 has served customer 75 at: 238, and his demands: 50
Vehicle 8 has served customer 41 at: 257, and his demands: 10
Vehicle 8 has served customer 45 at: 260, and his demands: 10
Vehicle 8 has served customer 49 at: 264, and his demands: 10
Vehicle 7 has served customer 87 at: 272, and his demands: 10
Vehicle 8 has returned to the depot at: 287, and the remaining load:100
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 9 New customers are available at time: 298 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
   100
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 102.000
The load of the vehicle 1 is 70
The cost for the route 2 is 237.000
```

```
The load of the vehicle 2 is 170
---------------------------------------------
The total cost for the all routes is 339.000
---------------------------------------------
The customers will be visited as follows:
     1   100    95   101    10    76     1
     1    24    27    20    19    38    61    59    83    86     1
==============================================================
End updating the current solution
==============================================================
Vehicle 7 has served customer 100 at: 299, and his demands: 10
Vehicle 7 has served customer 95 at: 309, and his demands: 10
Vehicle 9 has served customer 24 at: 310, and his demands: 10
Vehicle 9 has served customer 27 at: 313, and his demands: 10
Vehicle 7 has served customer 101 at: 319, and his demands: 20
Vehicle 9 has served customer 20 at: 340, and his demands: 10
Vehicle 7 has served customer 10 at: 342, and his demands: 10
Vehicle 9 has served customer 19 at: 345, and his demands: 20
Vehicle 7 has served customer 76 at: 351, and his demands: 20
Vehicle 7 has returned to the depot at: 367, and the remaining load: 0
Vehicle 9 has served customer 38 at: 382, and his demands: 20
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 8 New customers are available at time: 405 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    61
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 150.000
The load of the vehicle 1 is 120
The cost for the route 2 is 195.000
The load of the vehicle 2 is 150
---------------------------------------------
The total cost for the all routes is 345.000
---------------------------------------------
The customers will be visited as follows:
     1    61    59    57    39    50     1
     1    29    12     2    98    86    83    71     1
==============================================================
End updating the current solution
==============================================================
Vehicle 10 has served customer 29 at: 422, and his demands: 20
Vehicle 9 has served customer 61 at: 430, and his demands: 20
Vehicle 9 has served customer 59 at: 433, and his demands: 30
Vehicle 9 has served customer 57 at: 435, and his demands: 30
Vehicle 10 has served customer 12 at: 440, and his demands: 10
Vehicle 10 has served customer 2 at: 450, and his demands: 10
Vehicle 10 has served customer 98 at: 473, and his demands: 30
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 7 New customers are available at time: 479 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    39    86
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 88.000
The load of the vehicle 1 is 40
The cost for the route 2 is 155.000
The load of the vehicle 2 is 110
The cost for the route 3 is 92.000
The load of the vehicle 3 is 80
---------------------------------------------
The total cost for the all routes is 335.000
---------------------------------------------
The customers will be visited as follows:
     1    39    50     1
     1    86    92    83    71    62    65     1
     1     5    15    17    23     1
==============================================================
End updating the current solution
==============================================================
Vehicle 9 has served customer 39 at: 488, and his demands: 30
Vehicle 11 has served customer 5 at: 496, and his demands: 10
Vehicle 10 has served customer 86 at: 499, and his demands: 30
Vehicle 10 has served customer 92 at: 507, and his demands: 10
Vehicle 9 has served customer 50 at: 516, and his demands: 10
Vehicle 11 has served customer 15 at: 522, and his demands: 10
```

```
Vehicle 10 has served customer 83 at: 523, and his demands: 20
Vehicle 11 has served customer 17 at: 524, and his demands: 40
Vehicle 9 has returned to the depot at: 535, and the remaining load:10
Vehicle 10 has served customer 71 at: 555, and his demands: 30
Vehicle 11 has served customer 23 at: 558, and his demands: 20
Vehicle 11 has returned to the depot at: 570, and the remaining load:120
Vehicle 10 has served customer 62 at: 600, and his demands: 10
Vehicle 10 has served customer 65 at: 602, and his demands: 10
Vehicle 10 has returned to the depot at: 624, and the remaining load:20
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 7 New customers are available at time: 702 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 208.000
The load of the vehicle 1 is 110
-----------------------------------------------
The total cost for the all routes is 208.000
-----------------------------------------------
The customers will be visited as follows:
     1    90     3    13    40    46    60    67     1
==============================================================
End updating the current solution
==============================================================
Vehicle 12 has served customer 90 at: 725, and his demands: 10
Vehicle 12 has served customer 3 at: 747, and his demands: 30
Vehicle 12 has served customer 13 at: 772, and his demands: 20
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 3 New customers are available at time: 797 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    40
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 149.000
The load of the vehicle 1 is 90
-----------------------------------------------
The total cost for the all routes is 149.000
-----------------------------------------------
The customers will be visited as follows:
     1    40    22    48    46    60    69    67     1
==============================================================
End updating the current solution
==============================================================
Vehicle 12 has served customer 40 at: 819, and his demands: 20
Vehicle 12 has served customer 22 at: 850, and his demands: 20
Vehicle 12 has served customer 48 at: 867, and his demands: 10
Vehicle 12 has served customer 46 at: 872, and his demands: 10
Vehicle 12 has served customer 60 at: 889, and his demands: 10
Vehicle 12 has served customer 69 at: 906, and his demands: 10
Vehicle 12 has served customer 67 at: 911, and his demands: 10
Vehicle 12 has returned to the depot at: 928, and the remaining load:50


********************************************************
        1740

+++++++++++++++++++++++++++++
END OF PROGRAM
+++++++++++++++++++++++++++++
```

Figure D-1. A solution C101-10-90 instance

The final solution of C101-10-90 instance is summarised in Table D-1.

Table D-1.The final solution of C101-10-90

Solution													**Cost**	**load**
1	68	58	44	43	51	52	56	73	70	66	64	1	144	190
1	21	1											20	10
1	6	8	9	11	14	16	18	31	30	25	1		95	180
1	91	88	85	89	84	77	82	79	72	54	1		192	190
1	96	94	97	93	99	4	7	1					113	150

1	53	32	36	34	33	37	35	28	26	1					93	190
1	47	42	55	81	80	78	74	87	100	95	101	10	76	1	236	200
1	63	75	41	45	49	1									70	100
1	24	27	20	19	38	61	59	57	39	50	1				238	190
1	29	12	2	98	86	92	83	71	62	65	1				220	180
1	5	15	17	23	1										92	80
1	90	3	13	40	22	48	46	60	69	67	1				227	150

Figure D-2 shows a solution for C101-30-70 instance.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Instance name: C101-30-70
Day length: 1000
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The initial solution for known customers
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The cost for the route 1 is 135.000
The load of the vehicle 1 is 170
The cost for the 2 route is 134.000
The load of the 2 vehicle is 170
The cost for the 3 route is 130.000
The load of the 3 vehicle is 150
-----------------------------------------------
The total cost for the all routes is 399.000
-----------------------------------------------
The customers will be visited as follows:
     1    91    90    89    85    84    77    72    71    74    78    68     1
     1    21    28    35    33    38    36    58    56    41    44     1
     1    11    16    15   101    97    99     7    10     6     1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The beginning of the day, current time is: 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Vehicle 2 has served customer 21 at: 10, and his demands: 10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 10 New customers are available at time: 16 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    91    28    11

==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 136.000
The load of the vehicle 1 is 180
The cost for the 2 route is 118.000
The load of the 2 vehicle is 180
The cost for the 3 route is 136.000
The load of the 3 vehicle is 190
The cost for the 4 route is 90.000
The load of the 4 vehicle is 150
-----------------------------------------------
The total cost for the all routes is 480.000
-----------------------------------------------
The customers will be visited as follows:
     1    91    90    88    85    84    77    72    71    74    78    82     1
     1    28    30    33    36    38    35     9    10     7     8     6     1
     1    11    14    18    16    15   101    97    99    89     1
     1    64    68    70    73    56    58    41    44     1
==============================================================
End updating the current solution
==============================================================
Vehicle 2 has served customer 28 at: 17, and his demands: 10
Vehicle 3 has served customer 11 at: 17, and his demands: 10
Vehicle 1 has served customer 91 at: 21, and his demands: 10
Vehicle 2 has served customer 30 at: 21, and his demands: 10
Vehicle 1 has served customer 90 at: 25, and his demands: 10
```

```
Vehicle 1 has served customer 88 at: 29, and his demands: 20
Vehicle 4 has served customer 64 at: 29, and his demands: 50
Vehicle 4 has served customer 68 at: 32, and his demands: 10
Vehicle 3 has served customer 14 at: 33, and his demands: 30
Vehicle 1 has served customer 85 at: 35, and his demands: 20
Vehicle 2 has served customer 33 at: 35, and his demands: 30
Vehicle 3 has served customer 18 at: 37, and his demands: 20
Vehicle 4 has served customer 70 at: 37, and his demands: 10
Vehicle 1 has served customer 84 at: 39, and his demands: 10
Vehicle 2 has served customer 36 at: 42, and his demands: 10
Vehicle 3 has served customer 16 at: 42, and his demands: 40
Vehicle 4 has served customer 73 at: 46, and his demands: 10
Vehicle 3 has served customer 15 at: 47, and his demands: 10
Vehicle 2 has served customer 38 at: 48, and his demands: 20
Vehicle 2 has served customer 35 at: 56, and his demands: 20
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 11 New customers are available at time: 62 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    77     9   101    56
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 125.000
The load of the vehicle 1 is 120
The cost for the 2 route is 39.000
The load of the 2 vehicle is 60
The cost for the 3 route is 91.000
The load of the 3 vehicle is 70
The cost for the 4 route is 91.000
The load of the 4 vehicle is 110
The cost for the 5 route is 153.000
The load of the 5 vehicle is 180
-----------------------------------------------
The total cost for the all routes is 499.000
-----------------------------------------------
The customers will be visited as follows:
     1    77    72    71    74    78    82    66     1
     1     9     8     4     6     1
     1   101    94    97     1
     1    56    58    41    43    51    52    44     1
     1    25    26    37    31    10     7    99    96    89     1
==============================================================
End updating the current solution
==============================================================
Vehicle 4 has served customer 56 at: 65, and his demands: 10
Vehicle 1 has served customer 77 at: 66, and his demands: 10
Vehicle 4 has served customer 58 at: 67, and his demands: 40
Vehicle 1 has served customer 72 at: 71, and his demands: 20
Vehicle 1 has served customer 71 at: 76, and his demands: 30
Vehicle 5 has served customer 25 at: 76, and his demands: 10
Vehicle 5 has served customer 26 at: 78, and his demands: 40
Vehicle 1 has served customer 74 at: 79, and his demands: 10
Vehicle 3 has served customer 101 at: 80, and his demands: 20
Vehicle 1 has served customer 78 at: 83, and his demands: 10
Vehicle 4 has served customer 41 at: 83, and his demands: 10
Vehicle 4 has served customer 43 at: 86, and his demands: 20
Vehicle 1 has served customer 82 at: 89, and his demands: 30
Vehicle 3 has served customer 94 at: 90, and his demands: 40
Vehicle 4 has served customer 51 at: 93, and his demands: 10
Vehicle 2 has served customer 9 at: 94, and his demands: 20
Vehicle 4 has served customer 52 at: 95, and his demands: 10
Vehicle 2 has served customer 8 at: 97, and his demands: 20
Vehicle 3 has served customer 97 at: 97, and his demands: 10
Vehicle 2 has served customer 4 at: 99, and his demands: 10
Vehicle 5 has served customer 37 at: 99, and his demands: 10
Vehicle 2 has served customer 6 at: 100, and his demands: 10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 11 New customers are available at time: 104 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Vehicles are committed to the following customers:
    66    44    31
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 26.000
The load of the vehicle 1 is 10
The cost for the 2 route is 34.000
The load of the 2 vehicle is 10
The cost for the 3 route is 122.000
The load of the 3 vehicle is 120
The cost for the 4 route is 246.000
The load of the 4 vehicle is 190
-----------------------------------------------
The total cost for the all routes is 428.000
-----------------------------------------------
The customers will be visited as follows:
     1    66     1
     1    44     1
     1    31    10     7   100    93    96    99     1
     1    89    87    79    54    42    53    32    34    19    24     1
==============================================================
End updating the current solution
==============================================================
Vehicle 4 has served customer 44 at: 104, and his demands: 10
Vehicle 2 has returned to the depot at: 115, and the remaining load:30
Vehicle 5 has served customer 31 at: 117, and his demands: 10
Vehicle 4 has returned to the depot at: 121, and the remaining load:10
Vehicle 1 has served customer 66 at: 126, and his demands: 10
Vehicle 6 has served customer 89 at: 130, and his demands: 30
Vehicle 3 has returned to the depot at: 133, and the remaining load:20
Vehicle 6 has served customer 87 at: 135, and his demands: 10
Vehicle 1 has returned to the depot at: 139, and the remaining load:10
Vehicle 5 has served customer 10 at: 140, and his demands: 10
Vehicle 5 has served customer 7 at: 142, and his demands: 20
Vehicle 5 has served customer 100 at: 161, and his demands: 10
Vehicle 6 has served customer 79 at: 165, and his demands: 20
Vehicle 5 has served customer 93 at: 174, and his demands: 20
Vehicle 5 has served customer 96 at: 181, and his demands: 30
Vehicle 5 has served customer 99 at: 187, and his demands: 20
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 9 New customers are available at time: 214 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    54
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 98.000
The load of the vehicle 1 is 130
The cost for the 2 route is 224.000
The load of the 2 vehicle is 190
-----------------------------------------------
The total cost for the all routes is 322.000
-----------------------------------------------
The customers will be visited as follows:
     1    54    55    75    63     1
     1    24    20    19    34    32    53    49    47    45    42    81    80
1
==============================================================
End updating the current solution
==============================================================
Vehicle 5 has returned to the depot at: 218, and the remaining load:20
Vehicle 6 has served customer 54 at: 218, and his demands: 20
Vehicle 6 has served customer 55 at: 223, and his demands: 40
Vehicle 7 has served customer 24 at: 226, and his demands: 10
Vehicle 6 has served customer 75 at: 250, and his demands: 50
Vehicle 6 has served customer 63 at: 253, and his demands: 20
Vehicle 7 has served customer 20 at: 254, and his demands: 10
Vehicle 7 has served customer 19 at: 259, and his demands: 20
```

```
Vehicle 6 has returned to the depot at: 271, and the remaining load:10
Vehicle 7 has served customer 34 at: 295, and his demands: 40
Vehicle 7 has served customer 32 at: 300, and his demands: 20
Vehicle 7 has served customer 53 at: 315, and his demands: 10
Vehicle 7 has served customer 49 at: 321, and his demands: 10
Vehicle 7 has served customer 47 at: 324, and his demands: 30
Vehicle 7 has served customer 45 at: 327, and his demands: 10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 9 New customers are available at time: 328 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    42
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 63.000
The load of the vehicle 1 is 40
The cost for the 2 route is 250.000
The load of the 2 vehicle is 180
-----------------------------------------------
The total cost for the all routes is 313.000
-----------------------------------------------
The customers will be visited as follows:
     1    42    29    27     1
     1    76    95    86    83    80    81    59    61    39     1
==============================================================
End updating the current solution
==============================================================
Vehicle 7 has served customer 42 at: 331, and his demands: 10
Vehicle 8 has served customer 76 at: 343, and his demands: 20
Vehicle 7 has served customer 29 at: 357, and his demands: 20
Vehicle 7 has served customer 27 at: 359, and his demands: 10
Vehicle 8 has served customer 95 at: 369, and his demands: 10
Vehicle 7 has returned to the depot at: 375, and the remaining load: 0
Vehicle 8 has served customer 86 at: 391, and his demands: 30
Vehicle 8 has served customer 83 at: 400, and his demands: 20
Vehicle 8 has served customer 80 at: 428, and his demands: 10
Vehicle 8 has served customer 81 at: 433, and his demands: 10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 8 New customers are available at time: 467 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    59
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 101.000
The load of the vehicle 1 is 90
The cost for the 2 route is 192.000
The load of the 2 vehicle is 140
-----------------------------------------------
The total cost for the all routes is 293.000
-----------------------------------------------
The customers will be visited as follows:
     1    59    61    57    65     1
     1     5     2    98    12    17    39    50     1
==============================================================
End updating the current solution
==============================================================
Vehicle 8 has served customer 59 at: 484, and his demands: 30
Vehicle 9 has served customer 5 at: 484, and his demands: 10
Vehicle 8 has served customer 61 at: 487, and his demands: 20
Vehicle 9 has served customer 2 at: 487, and his demands: 10
Vehicle 8 has served customer 57 at: 492, and his demands: 30
Vehicle 9 has served customer 98 at: 510, and his demands: 30
Vehicle 8 has served customer 65 at: 518, and his demands: 10
Vehicle 8 has returned to the depot at: 540, and the remaining load:10
Vehicle 9 has served customer 12 at: 540, and his demands: 10
Vehicle 9 has served customer 17 at: 562, and his demands: 40
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
*** 7 New customers are available at time: 597 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    39
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 140.000
The load of the vehicle 1 is 100
The cost for the 2 route is 67.000
The load of the 2 vehicle is 30
-----------------------------------------------
The total cost for the all routes is 207.000
-----------------------------------------------
The customers will be visited as follows:
     1    39    40    50    46    60    62    67     1
     1    23    92     1
==============================================================
End updating the current solution
==============================================================
Vehicle 10 has served customer 23 at: 608, and his demands: 20
Vehicle 9 has served customer 39 at: 611, and his demands: 30
Vehicle 9 has served customer 40 at: 616, and his demands: 20
Vehicle 10 has served customer 92 at: 641, and his demands: 10
Vehicle 9 has served customer 50 at: 646, and his demands: 10
Vehicle 9 has served customer 46 at: 651, and his demands: 10
Vehicle 10 has returned to the depot at: 663, and the remaining load:170
Vehicle 9 has served customer 60 at: 668, and his demands: 10
Vehicle 9 has served customer 62 at: 687, and his demands: 10
Vehicle 9 has served customer 67 at: 693, and his demands: 10
Vehicle 9 has returned to the depot at: 710, and the remaining load: 0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 5 New customers are available at time: 797 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 133.000
The load of the vehicle 1 is 90
-----------------------------------------------
The total cost for the all routes is 133.000
-----------------------------------------------
The customers will be visited as follows:
     1     3    13    22    48    69     1

==============================================================
End updating the current solution
==============================================================
Vehicle 11 has served customer 3 at: 817, and his demands: 30
Vehicle 11 has served customer 13 at: 842, and his demands: 20
Vehicle 11 has served customer 22 at: 875, and his demands: 20
Vehicle 11 has served customer 48 at: 892, and his demands: 10
Vehicle 11 has served customer 69 at: 908, and his demands: 10
Vehicle 11 has returned to the depot at: 929, and the remaining load:110


********************************************************
        1637

+++++++++++++++++++++++++++++
END OF PROGRAM
+++++++++++++++++++++++++++++
```

Figure D-2. A solution C101-30-70 instance

The final solution of C101-30-70 instance is summarised in Table D-2.

Table D-2. The final solution of C101-30-70

Solution														Cost	Load
1	91	90	88	85	84	77	72	71	74	78	82	66	1	139	190
1	21	28	30	33	36	38	35	9	8	4	6	1		115	170
1	11	14	18	16	15	101	94	97	1					133	180
1	64	68	70	73	56	58	41	43	51	52	44	1		106	190
1	25	26	37	31	10	7	100	93	96	99	1			157	180
1	89	87	79	54	55	75	63	1						168	190
1	24	20	19	34	32	53	49	47	45	42	29	27	1	162	200
1	76	95	86	83	80	81	59	61	57	65	1			213	190
1	5	2	98	12	17	39	40	50	46	60	62	67	1	244	200
1	23	92	1											67	30
1	3	13	22	48	69	1								133	90

Figure D-3 shows a solution for C101-50-50 instance.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Instance name: C101-50-50
Day length: 1000
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The initial solution for known customers
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The cost for the route 1 is 155.000
The load of the vehicle 1 is 190
The cost for the 2 route is 94.000
The load of the 2 vehicle is 160
The cost for the 3 route is 102.000
The load of the 3 vehicle is 190
The cost for the 4 route is 123.000
The load of the 4 vehicle is 190
The cost for the 5 route is 93.000
The load of the 5 vehicle is 100
-----------------------------------------------
The total cost for the all routes is 567.000
-----------------------------------------------
The customers will be visited as follows:
     1    68    79    77    72    71    74    78    80    73    67    42    45
47     1
     1    21    25    28    30    33    36    38    35    31    29    24     1
     1     6     4     2     7    10    13    15    16    19    14    11     1
     1    91    90    88    84    85    89    94   101    97    99     1
     1    56    57    58    41    44     1

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The beginning of the day, current time is: 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Vehicle 2 has served customer 21 at: 10, and his demands: 10
Vehicle 1 has served customer 68 at: 12, and his demands: 10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 5 New customers are available at time: 14 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    79    25     6    91    56
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 135.000
The load of the vehicle 1 is 190
The cost for the 2 route is 110.000
The load of the 2 vehicle is 180
The cost for the 3 route is 96.000
The load of the 3 vehicle is 180
The cost for the 4 route is 153.000
The load of the 4 vehicle is 190
```

```
The cost for the 5 route is 142.000
The load of the 5 vehicle is 170
-----------------------------------------------
The total cost for the all routes is 636.000
-----------------------------------------------
The customers will be visited as follows:
     1    79    77    72    71    74    78    80    82    88    89     1
     1    25    28    31    35    38    36    33    47    45    41    42    44
1
     1     6     7    13    15    16    19    18    14    11     1
     1    91    90    84    85    99    97    94   101     2    24    29    30
1
     1    56    57    58    73    67    70     4     8    10     9     1
===============================================================
End updating the current solution
===============================================================
Vehicle 2 has served customer 25 at: 15, and his demands: 10
Vehicle 3 has served customer 6 at: 15, and his demands: 10
Vehicle 2 has served customer 28 at: 18, and his demands: 10
Vehicle 3 has served customer 7 at: 19, and his demands: 20
Vehicle 4 has served customer 91 at: 21, and his demands: 10
Vehicle 2 has served customer 31 at: 22, and his demands: 10
Vehicle 4 has served customer 90 at: 25, and his demands: 10
Vehicle 4 has served customer 84 at: 34, and his demands: 10
Vehicle 5 has served customer 56 at: 35, and his demands: 10
Vehicle 2 has served customer 35 at: 38, and his demands: 20
Vehicle 4 has served customer 85 at: 38, and his demands: 20
Vehicle 3 has served customer 13 at: 41, and his demands: 20
Vehicle 3 has served customer 15 at: 44, and his demands: 10
Vehicle 5 has served customer 57 at: 45, and his demands: 30
Vehicle 2 has served customer 38 at: 46, and his demands: 20
Vehicle 3 has served customer 16 at: 49, and his demands: 40
Vehicle 2 has served customer 36 at: 52, and his demands: 10
Vehicle 1 has served customer 79 at: 53, and his demands: 20
Vehicle 1 has served customer 77 at: 55, and his demands: 10
Vehicle 5 has served customer 58 at: 55, and his demands: 40
Vehicle 3 has served customer 19 at: 56, and his demands: 20
Vehicle 2 has served customer 33 at: 59, and his demands: 30
Vehicle 3 has served customer 18 at: 59, and his demands: 20
Vehicle 4 has served customer 99 at: 59, and his demands: 20
Vehicle 1 has served customer 72 at: 60, and his demands: 20
Vehicle 3 has served customer 14 at: 63, and his demands: 30
Vehicle 4 has served customer 97 at: 64, and his demands: 10
Vehicle 1 has served customer 71 at: 65, and his demands: 30
Vehicle 1 has served customer 74 at: 68, and his demands: 10
Vehicle 4 has served customer 94 at: 71, and his demands: 40
Vehicle 1 has served customer 78 at: 72, and his demands: 10
Vehicle 1 has served customer 80 at: 73, and his demands: 10
Vehicle 5 has served customer 73 at: 75, and his demands: 10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 10 New customers are available at time: 76 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    82    47    11   101    67

===============================================================
Updating the current solution
===============================================================
The cost for the route 1 is 107.000
The load of the vehicle 1 is 80
The cost for the 2 route is 50.000
The load of the 2 vehicle is 80
The cost for the 3 route is 34.000
The load of the 3 vehicle is 10
The cost for the 4 route is 97.000
The load of the 4 vehicle is 80
The cost for the 5 route is 77.000
The load of the 5 vehicle is 110
The cost for the 6 route is 109.000
The load of the 6 vehicle is 190
```

```
-----------------------------------------------
The total cost for the all routes is 474.000
-----------------------------------------------
The customers will be visited as follows:
     1    82    88    89     1
     1    47    45    41    42    43     1
     1    11     1
     1   101    93    96     2     1
     1    67    64    66    70    44    52    51     1
     1    24    26    30    37    34    29     9    10     8     4     1
===============================================================
End updating the current solution
===============================================================
Vehicle 1 has served customer 82 at: 78, and his demands: 30
Vehicle 3 has served customer 11 at: 79, and his demands: 10
Vehicle 2 has served customer 47 at: 81, and his demands: 30
Vehicle 4 has served customer 101 at: 81, and his demands: 20
Vehicle 5 has served customer 67 at: 83, and his demands: 10
Vehicle 2 has served customer 45 at: 84, and his demands: 10
Vehicle 2 has served customer 41 at: 87, and his demands: 10
Vehicle 6 has served customer 24 at: 88, and his demands: 10
Vehicle 2 has served customer 42 at: 89, and his demands: 10
Vehicle 5 has served customer 64 at: 89, and his demands: 50
Vehicle 2 has served customer 43 at: 91, and his demands: 20
Vehicle 5 has served customer 66 at: 91, and his demands: 10
Vehicle 6 has served customer 26 at: 92, and his demands: 40
Vehicle 4 has served customer 93 at: 93, and his demands: 20
Vehicle 3 has returned to the depot at: 96, and the remaining load:20
Vehicle 5 has served customer 70 at: 97, and his demands: 10
Vehicle 6 has served customer 30 at: 97, and his demands: 10
Vehicle 4 has served customer 96 at: 100, and his demands: 30
Vehicle 1 has served customer 88 at: 106, and his demands: 20
Vehicle 5 has served customer 44 at: 109, and his demands: 10
Vehicle 2 has returned to the depot at: 110, and the remaining load: 0
Vehicle 1 has served customer 89 at: 111, and his demands: 30
Vehicle 6 has served customer 37 at: 113, and his demands: 10
Vehicle 5 has served customer 52 at: 118, and his demands: 10
Vehicle 6 has served customer 34 at: 119, and his demands: 40
Vehicle 5 has served customer 51 at: 120, and his demands: 10
Vehicle 4 has served customer 2 at: 121, and his demands: 10
Vehicle 1 has returned to the depot at: 138, and the remaining load: 0
Vehicle 4 has returned to the depot at: 140, and the remaining load: 0
Vehicle 6 has served customer 29 at: 140, and his demands: 20
Vehicle 5 has returned to the depot at: 143, and the remaining load: 0
Vehicle 6 has served customer 9 at: 160, and his demands: 20
Vehicle 6 has served customer 10 at: 162, and his demands: 10
Vehicle 6 has served customer 8 at: 166, and his demands: 20
Vehicle 6 has served customer 4 at: 168, and his demands: 10
Vehicle 6 has returned to the depot at: 184, and the remaining load:10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 9 New customers are available at time: 194 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
===============================================================
Updating the current solution
===============================================================
The cost for the route 1 is 246.000
The load of the vehicle 1 is 180
-----------------------------------------------
The total cost for the all routes is 246.000
-----------------------------------------------
The customers will be visited as follows:
     1    53    32    49    55    54    75    81    87   100     1

===============================================================
End updating the current solution
===============================================================
Vehicle 7 has served customer 53 at: 214, and his demands: 10
Vehicle 7 has served customer 32 at: 229, and his demands: 20
Vehicle 7 has served customer 49 at: 248, and his demands: 10
```

```
Vehicle 7 has served customer 55 at: 272, and his demands: 40
Vehicle 7 has served customer 54 at: 277, and his demands: 20
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 9 New customers are available at time: 298 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    75

==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 41.000
The load of the vehicle 1 is 70
The cost for the 2 route is 283.000
The load of the 2 vehicle is 180
-----------------------------------------------
The total cost for the all routes is 324.000
-----------------------------------------------
The customers will be visited as follows:
     1    75    63     1
     1    76   100    95    86    87    83    81    59    61    27    20     1
==============================================================
End updating the current solution
==============================================================
Vehicle 7 has served customer 75 at: 308, and his demands: 50
Vehicle 7 has served customer 63 at: 311, and his demands: 20
Vehicle 8 has served customer 76 at: 313, and his demands: 20
Vehicle 7 has returned to the depot at: 329, and the remaining load:30
Vehicle 8 has served customer 100 at: 331, and his demands: 10
Vehicle 8 has served customer 95 at: 341, and his demands: 10
Vehicle 8 has served customer 86 at: 363, and his demands: 30
Vehicle 8 has served customer 87 at: 368, and his demands: 10
Vehicle 8 has served customer 83 at: 377, and his demands: 20
Vehicle 8 has served customer 81 at: 409, and his demands: 10
Vehicle 8 has served customer 59 at: 460, and his demands: 30
Vehicle 8 has served customer 61 at: 463, and his demands: 20
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 8 New customers are available at time: 468 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    27

==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 32.000
The load of the vehicle 1 is 30
The cost for the 2 route is 213.000
The load of the 2 vehicle is 150
-----------------------------------------------
The total cost for the all routes is 245.000
-----------------------------------------------
The customers will be visited as follows:
     1    27    23     1

     1    12     5    98    17    20    39    50    65     1

==============================================================
End updating the current solution
==============================================================
Vehicle 9 has served customer 12 at: 487, and his demands: 10
Vehicle 9 has served customer 5 at: 494, and his demands: 10
Vehicle 8 has served customer 27 at: 514, and his demands: 10
Vehicle 8 has served customer 23 at: 518, and his demands: 20
Vehicle 9 has served customer 98 at: 519, and his demands: 30
Vehicle 8 has returned to the depot at: 530, and the remaining load:10
Vehicle 9 has served customer 17 at: 559, and his demands: 40
Vehicle 9 has served customer 20 at: 566, and his demands: 10
Vehicle 9 has served customer 39 at: 609, and his demands: 30
Vehicle 9 has served customer 50 at: 637, and his demands: 10
```

```
Vehicle 9 has served customer 65 at: 658, and his demands: 10
Vehicle 9 has returned to the depot at: 680, and the remaining load:50
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 7 New customers are available at time: 705 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 184.000
The load of the vehicle 1 is 100
-----------------------------------------------
The total cost for the all routes is 184.000
-----------------------------------------------
The customers will be visited as follows:
     1     3    92    62    69    60    46    40     1

==============================================================
End updating the current solution
==============================================================
Vehicle 10 has served customer 3 at: 725, and his demands: 30
Vehicle 10 has served customer 92 at: 743, and his demands: 10
Vehicle 10 has served customer 62 at: 775, and his demands: 10
Vehicle 10 has served customer 69 at: 780, and his demands: 10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 2 New customers are available at time: 797 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    60

==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 130.000
The load of the vehicle 1 is 70
-----------------------------------------------
The total cost for the all routes is 130.000
-----------------------------------------------
The customers will be visited as follows:
     1    60    46    48    40    22     1

==============================================================
End updating the current solution
==============================================================
Vehicle 10 has served customer 60 at: 797, and his demands: 10
Vehicle 10 has served customer 46 at: 814, and his demands: 10
Vehicle 10 has served customer 48 at: 819, and his demands: 10
Vehicle 10 has served customer 40 at: 851, and his demands: 20
Vehicle 10 has served customer 22 at: 882, and his demands: 20
Vehicle 10 has returned to the depot at: 892, and the remaining load:70


********************************************************
        1506

+++++++++++++++++++++++++++++
END OF PROGRAM
+++++++++++++++++++++++++++++
```

Figure D-3. A solution C101-50-50 instance

The final solution of C101-50-50 instance is summarised in Table D-3.

Table D-3. The final solution of C101-50-50

Solution															Cost	Load
1	68	79	77	72	71	74	78	80	82	88	89	1			138	200
1	21	25	28	31	35	38	36	33	47	45	41	42	43	1	110	200
1	6	7	13	15	16	19	18	14	11	1					96	180
1	91	90	84	85	99	97	94	101	93	96	2	1			140	200
1	56	57	58	73	67	64	66	70	44	52	51	1			143	200
1	24	26	30	37	34	29	9	10	8	4	1				109	190
1	53	32	49	55	54	75	63	1							136	170
1	76	100	95	86	87	83	81	59	61	27	23	1			233	190
1	12	5	98	17	20	39	50	65	1						213	150
1	3	92	62	69	60	46	48	40	22	1					188	130

Figure D-4 shows a solution for C101-70-30 instance.

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Instance name: C101-70-30
Day length: 1000
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The initial solution for known customers
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The cost for the route 1 is 97.000
The load of the vehicle 1 is 200
The cost for the 2 route is 103.000
The load of the 2 vehicle is 200
The cost for the 3 route is 85.000
The load of the 3 vehicle is 200
The cost for the 4 route is 108.000
The load of the 4 vehicle is 190
The cost for the 5 route is 144.000
The load of the 5 vehicle is 190
The cost for the 6 route is 137.000
The load of the 6 vehicle is 200
-----------------------------------------------
The total cost for the all routes is 674.000
-----------------------------------------------
The customers will be visited as follows:
     1    26    27    29    31    35    37    38    36    33    30    28    25
1
     1    21    24    14    18    19    16    15    13    10     7    11     1
     1    68    64    70    67    73    62    65    69    42    47    51    50
48    44     1
     1     6     4     2    99    97   101    98    94    93    95    92     1
     1    41    45    49    60    58    56    55    57    54    90     1
     1    91    88    89    85    84    79    77    72    71    74    78    80
1

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The beginning of the day, current time is: 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Vehicle 2 has served customer 21 at: 10, and his demands: 10
Vehicle 3 has served customer 68 at: 12, and his demands: 10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 3 New customers are available at time: 14 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    26    24    64     6    41    91

===============================================================
Updating the current solution
```

```
==============================================================
The cost for the route 1 is 97.000
The load of the vehicle 1 is 200
The cost for the 2 route is 105.000
The load of the 2 vehicle is 180
The cost for the 3 route is 80.000
The load of the 3 vehicle is 190
The cost for the 4 route is 96.000
The load of the 4 vehicle is 170
The cost for the 5 route is 106.000
The load of the 5 vehicle is 180
The cost for the 6 route is 128.000
The load of the 6 vehicle is 170
The cost for the 7 route is 92.000
The load of the 7 vehicle is 140
-----------------------------------------------
The total cost for the all routes is 704.000
-----------------------------------------------
The customers will be visited as follows:
     1    26    27    29    31    35    37    38    36    33    30    28    25
1
     1    24    11     9    10     7     8     4    92    89    85    84    90
1
     1    64    67    62    65    69    42    45    47    49    51    50    48
44     1
     1     6     2   101    98    94    93    95    97    99     1
     1    41    60    57    54    55    58    56    73    70     1
     1    91    88    82    79    77    72    71    74    78    80     1
     1    18    19    16    15    13    14     1
==============================================================
End updating the current solution
==============================================================
Vehicle 1 has served customer 26 at: 15, and his demands: 40
Vehicle 2 has served customer 24 at: 15, and his demands: 10
Vehicle 3 has served customer 64 at: 15, and his demands: 50
Vehicle 4 has served customer 6 at: 15, and his demands: 10
Vehicle 1 has served customer 27 at: 18, and his demands: 10
Vehicle 4 has served customer 2 at: 19, and his demands: 10
Vehicle 1 has served customer 29 at: 20, and his demands: 20
Vehicle 3 has served customer 67 at: 21, and his demands: 10
Vehicle 5 has served customer 41 at: 21, and his demands: 10
Vehicle 6 has served customer 91 at: 21, and his demands: 10
Vehicle 1 has served customer 31 at: 23, and his demands: 10
Vehicle 6 has served customer 88 at: 26, and his demands: 20
Vehicle 3 has served customer 62 at: 27, and his demands: 10
Vehicle 2 has served customer 11 at: 28, and his demands: 10
Vehicle 3 has served customer 65 at: 29, and his demands: 10
Vehicle 2 has served customer 9 at: 32, and his demands: 20
Vehicle 3 has served customer 69 at: 32, and his demands: 10
Vehicle 2 has served customer 10 at: 34, and his demands: 10
Vehicle 2 has served customer 7 at: 36, and his demands: 20
Vehicle 5 has served customer 60 at: 36, and his demands: 10
Vehicle 1 has served customer 35 at: 39, and his demands: 20
Vehicle 2 has served customer 8 at: 39, and his demands: 20
Vehicle 4 has served customer 101 at: 39, and his demands: 20
Vehicle 2 has served customer 4 at: 41, and his demands: 10
Vehicle 1 has served customer 37 at: 42, and his demands: 10
Vehicle 3 has served customer 42 at: 42, and his demands: 10
Vehicle 4 has served customer 98 at: 44, and his demands: 30
Vehicle 3 has served customer 45 at: 46, and his demands: 10
Vehicle 5 has served customer 57 at: 46, and his demands: 30
Vehicle 7 has served customer 18 at: 46, and his demands: 20
Vehicle 1 has served customer 38 at: 48, and his demands: 20
Vehicle 3 has served customer 47 at: 49, and his demands: 30
Vehicle 4 has served customer 94 at: 49, and his demands: 40
Vehicle 7 has served customer 19 at: 49, and his demands: 20
Vehicle 5 has served customer 54 at: 50, and his demands: 20
Vehicle 4 has served customer 93 at: 51, and his demands: 20
Vehicle 3 has served customer 49 at: 52, and his demands: 10
Vehicle 1 has served customer 36 at: 54, and his demands: 10
Vehicle 6 has served customer 82 at: 54, and his demands: 30
Vehicle 3 has served customer 51 at: 55, and his demands: 10
Vehicle 4 has served customer 95 at: 55, and his demands: 10
Vehicle 5 has served customer 55 at: 55, and his demands: 40
Vehicle 7 has served customer 16 at: 56, and his demands: 40
Vehicle 6 has served customer 79 at: 57, and his demands: 20
Vehicle 3 has served customer 50 at: 59, and his demands: 10
Vehicle 6 has served customer 77 at: 59, and his demands: 10
Vehicle 2 has served customer 92 at: 60, and his demands: 10
Vehicle 4 has served customer 97 at: 60, and his demands: 10
Vehicle 5 has served customer 58 at: 60, and his demands: 40
Vehicle 1 has served customer 33 at: 61, and his demands: 30
Vehicle 3 has served customer 48 at: 61, and his demands: 10
Vehicle 7 has served customer 15 at: 61, and his demands: 10
Vehicle 5 has served customer 56 at: 62, and his demands: 10
Vehicle 3 has served customer 44 at: 64, and his demands: 10
Vehicle 6 has served customer 72 at: 64, and his demands: 20
Vehicle 7 has served customer 13 at: 64, and his demands: 20
Vehicle 2 has served customer 89 at: 65, and his demands: 30
Vehicle 4 has served customer 99 at: 65, and his demands: 20
Vehicle 6 has served customer 71 at: 69, and his demands: 30
Vehicle 2 has served customer 85 at: 70, and his demands: 20
Vehicle 6 has served customer 74 at: 72, and his demands: 10
Vehicle 2 has served customer 84 at: 74, and his demands: 10
Vehicle 7 has served customer 14 at: 74, and his demands: 30
Vehicle 1 has served customer 30 at: 75, and his demands: 10
Vehicle 6 has served customer 78 at: 76, and his demands: 10
Vehicle 6 has served customer 80 at: 77, and his demands: 10
Vehicle 1 has served customer 28 at: 79, and his demands: 10
Vehicle 3 has returned to the depot at: 81, and the remaining load: 0
Vehicle 5 has served customer 73 at: 81, and his demands: 10
Vehicle 1 has served customer 25 at: 82, and his demands: 10
Vehicle 2 has served customer 90 at: 83, and his demands: 10
Vehicle 5 has served customer 70 at: 90, and his demands: 10
Vehicle 4 has returned to the depot at: 96, and the remaining load:30
Vehicle 1 has returned to the depot at: 97, and the remaining load: 0
Vehicle 7 has returned to the depot at: 105, and the remaining load:60
Vehicle 5 has returned to the depot at: 106, and the remaining load:20
Vehicle 2 has returned to the depot at: 107, and the remaining load:10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 10 New customers are available at time: 118 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
===============================================================
Updating the current solution
===============================================================
The cost for the route 1 is 226.000
The load of the vehicle 1 is 170
-----------------------------------------------
The total cost for the all routes is 226.000
-----------------------------------------------
The customers will be visited as follows:
     1   100    96    87    81    66    43    52    53    32    34     1

===============================================================
End updating the current solution
===============================================================
Vehicle 6 has returned to the depot at: 128, and the remaining load:30
Vehicle 8 has served customer 100 at: 151, and his demands: 10
Vehicle 8 has served customer 96 at: 158, and his demands: 30
Vehicle 8 has served customer 87 at: 183, and his demands: 10
Vehicle 8 has served customer 81 at: 219, and his demands: 10
Vehicle 8 has served customer 66 at: 259, and his demands: 10
Vehicle 8 has served customer 43 at: 276, and his demands: 20
Vehicle 8 has served customer 52 at: 284, and his demands: 10
Vehicle 8 has served customer 53 at: 289, and his demands: 10
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 9 New customers are available at time: 299 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
```

```
    32

==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 87.000
The load of the vehicle 1 is 90
The cost for the 2 route is 229.000
The load of the 2 vehicle is 200
-----------------------------------------------
The total cost for the all routes is 316.000
-----------------------------------------------
The customers will be visited as follows:
     1    32    39    34     1
     1    76    20    86    83    75    63    59    61     1
==============================================================
End updating the current solution
==============================================================
Vehicle 8 has served customer 32 at: 304, and his demands: 20
Vehicle 9 has served customer 76 at: 314, and his demands: 20
Vehicle 8 has served customer 39 at: 315, and his demands: 30
Vehicle 8 has served customer 34 at: 323, and his demands: 40
Vehicle 9 has served customer 20 at: 348, and his demands: 10
Vehicle 8 has returned to the depot at: 357, and the remaining load: 0
Vehicle 9 has served customer 86 at: 405, and his demands: 30
Vehicle 9 has served customer 83 at: 414, and his demands: 20
Vehicle 9 has served customer 75 at: 444, and his demands: 50
Vehicle 9 has served customer 63 at: 447, and his demands: 20
Vehicle 9 has served customer 59 at: 479, and his demands: 30
Vehicle 9 has served customer 61 at: 482, and his demands: 20
Vehicle 9 has returned to the depot at: 527, and the remaining load: 0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 7 New customers are available at time: 696 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 167.000
The load of the vehicle 1 is 140
-----------------------------------------------
The total cost for the all routes is 167.000
-----------------------------------------------
The customers will be visited as follows:
     1     5     3    12    17    40    46    23     1

==============================================================
End updating the current solution
==============================================================
Vehicle 10 has served customer 5 at: 713, and his demands: 10
Vehicle 10 has served customer 3 at: 717, and his demands: 30
Vehicle 10 has served customer 12 at: 727, and his demands: 10
Vehicle 10 has served customer 17 at: 749, and his demands: 40
Vehicle 10 has served customer 40 at: 794, and his demands: 20
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 1 New customers are available at time: 797 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    46

==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 56.000
The load of the vehicle 1 is 50
-----------------------------------------------
The total cost for the all routes is 56.000
-----------------------------------------------
The customers will be visited as follows:
     1    46    23    22     1
```

```
==============================================================
End updating the current solution
==============================================================
Vehicle 10 has served customer 46 at: 828, and his demands: 10
Vehicle 10 has served customer 23 at: 850, and his demands: 20
Vehicle 10 has served customer 22 at: 852, and his demands: 20
Vehicle 10 has returned to the depot at: 862, and the remaining load:40


********************************************************
        1343

+++++++++++++++++++++++++++++
END OF PROGRAM
+++++++++++++++++++++++++++++
```
Figure D-4. A solution C101-70-30 instance

The final solution of C101-70-30 instance is summarised in Table D-4.

Table D-4. The final solution of C101-70-30

Solution																**Cost**	**load**
1	26	27	29	31	35	37	38	36	33	30	28	25	1			97	200
1	21	24	11	9	10	7	8	4	92	89	85	84	90	1		107	190
1	68	64	67	62	65	69	42	45	47	49	51	50	48	44	1	81	200
1	6	2	101	98	94	93	95	97	99	1						96	170
1	41	60	57	54	55	58	56	73	70	1						106	180
1	91	88	82	79	77	72	71	74	78	80	1					128	170
1	18	19	16	15	13	14	1									92	140
1	100	96	87	81	66	43	52	53	32	39	34	1				240	200
1	76	20	86	83	75	63	59	61	1							229	200
1	5	3	12	17	40	46	23	22	1							167	160

Figure D-5 shows a solution for C101-90-10 instance.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Instance name: C101-90-10
Day length: 1000
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The initial solution for known customers
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The cost for the route 1 is 161.000
The load of the vehicle 1 is 200
The cost for the 2 route is 74.000
The load of the 2 vehicle is 200
The cost for the 3 route is 59.000
The load of the 3 vehicle is 190
The cost for the 4 route is 103.000
The load of the 4 vehicle is 200
The cost for the 5 route is 100.000
The load of the 5 vehicle is 200
The cost for the 6 route is 93.000
The load of the 6 vehicle is 170
The cost for the 7 route is 108.000
The load of the 7 vehicle is 190
The cost for the 8 route is 150.000
The load of the 8 vehicle is 190
-----------------------------------------------
The total cost for the all routes is 848.000
-----------------------------------------------
```

```
The customers will be visited as follows:
     1    79    77    72    71    74    78    80    81    55    58     1
     1    21    22    23    25    26    28    30    31    29    27    24    11
9     1
     1    68    64    63    75    73    62    65    69    67    70     1
     1     6     8    10   100   101    98    94    93    95    96     1
     1     4     5     7    12    13    15    16    20    19    18    14     1
     1    44    43    42    41    60    45    47    46    49    52    51    53
50    48     1
     1    91    88    87    84    85    86    89    90    92    99    97     2
1
     1    33    35    37    38    36    32    61    57    54    56     1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The beginning of the day, current time is: 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 1 New customers are available at time: 6 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    79    21    68     6     4    44    91    33

==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 144.000
The load of the vehicle 1 is 180
The cost for the 2 route is 80.000
The load of the 2 vehicle is 200
The cost for the 3 route is 72.000
The load of the 3 vehicle is 200
The cost for the 4 route is 101.000
The load of the 4 vehicle is 200
The cost for the 5 route is 98.000
The load of the 5 vehicle is 200
The cost for the 6 route is 95.000
The load of the 6 vehicle is 200
The cost for the 7 route is 129.000
The load of the 7 vehicle is 190
The cost for the 8 route is 144.000
The load of the 8 vehicle is 200
-----------------------------------------------
The total cost for the all routes is 863.000
-----------------------------------------------
The customers will be visited as follows:
     1    79    77    72    71    74    78    80    81    82    84    85     1
     1    21    25    26    28    30    35    37    31    29    27    24    23
22     1
     1    68    64    63    75    73    62    65    69    67    70    42     1
     1     6     8     2     5     7    10     9    12    14    20    19    18
11     1
     1     4   100   101    98    94    93    95    96    97    99     1
     1    44    43    41    60    58    45    47    46    49    52    51    53
50    48     1
     1    91    88    87    86    89    90    92    13    15    16     1
     1    33    38    36    32    61    57    54    55    56     1
==============================================================
End updating the current solution
==============================================================
Vehicle 2 has served customer 21 at: 10, and his demands: 10
Vehicle 3 has served customer 68 at: 12, and his demands: 10
Vehicle 2 has served customer 25 at: 15, and his demands: 10
Vehicle 3 has served customer 64 at: 15, and his demands: 50
Vehicle 4 has served customer 6 at: 15, and his demands: 10
Vehicle 5 has served customer 4 at: 16, and his demands: 10
Vehicle 2 has served customer 26 at: 17, and his demands: 40
Vehicle 4 has served customer 8 at: 17, and his demands: 20
Vehicle 6 has served customer 44 at: 17, and his demands: 10
Vehicle 2 has served customer 28 at: 19, and his demands: 10
Vehicle 3 has served customer 63 at: 20, and his demands: 20
```

```
Vehicle 6 has served customer 43 at: 20, and his demands: 20
Vehicle 7 has served customer 91 at: 21, and his demands: 10
Vehicle 4 has served customer 2 at: 22, and his demands: 10
Vehicle 2 has served customer 30 at: 23, and his demands: 10
Vehicle 3 has served customer 75 at: 23, and his demands: 50
Vehicle 6 has served customer 41 at: 23, and his demands: 10
Vehicle 4 has served customer 5 at: 25, and his demands: 10
Vehicle 7 has served customer 88 at: 26, and his demands: 20
Vehicle 4 has served customer 7 at: 27, and his demands: 20
Vehicle 7 has served customer 87 at: 27, and his demands: 10
Vehicle 3 has served customer 73 at: 28, and his demands: 10
Vehicle 4 has served customer 10 at: 29, and his demands: 10
Vehicle 3 has served customer 62 at: 31, and his demands: 10
Vehicle 4 has served customer 9 at: 31, and his demands: 20
Vehicle 7 has served customer 86 at: 32, and his demands: 30
Vehicle 8 has served customer 33 at: 32, and his demands: 30
Vehicle 3 has served customer 65 at: 33, and his demands: 10
Vehicle 4 has served customer 12 at: 34, and his demands: 10
Vehicle 5 has served customer 100 at: 35, and his demands: 10
Vehicle 7 has served customer 89 at: 35, and his demands: 30
Vehicle 2 has served customer 35 at: 36, and his demands: 20
Vehicle 3 has served customer 69 at: 36, and his demands: 10
Vehicle 6 has served customer 60 at: 38, and his demands: 10
Vehicle 7 has served customer 90 at: 38, and his demands: 10
Vehicle 2 has served customer 37 at: 39, and his demands: 10
Vehicle 5 has served customer 101 at: 40, and his demands: 20
Vehicle 6 has served customer 58 at: 40, and his demands: 40
Vehicle 8 has served customer 38 at: 40, and his demands: 20
Vehicle 3 has served customer 67 at: 41, and his demands: 10
Vehicle 7 has served customer 92 at: 42, and his demands: 10
Vehicle 3 has served customer 70 at: 43, and his demands: 10
Vehicle 5 has served customer 98 at: 45, and his demands: 30
Vehicle 8 has served customer 36 at: 46, and his demands: 10
Vehicle 4 has served customer 14 at: 48, and his demands: 30
Vehicle 1 has served customer 79 at: 50, and his demands: 20
Vehicle 5 has served customer 94 at: 50, and his demands: 40
Vehicle 8 has served customer 32 at: 51, and his demands: 20
Vehicle 1 has served customer 77 at: 52, and his demands: 10
Vehicle 5 has served customer 93 at: 52, and his demands: 20
Vehicle 3 has served customer 42 at: 53, and his demands: 10
Vehicle 5 has served customer 95 at: 56, and his demands: 10
Vehicle 1 has served customer 72 at: 57, and his demands: 20
Vehicle 2 has served customer 31 at: 57, and his demands: 10
Vehicle 4 has served customer 20 at: 57, and his demands: 10
Vehicle 6 has served customer 45 at: 57, and his demands: 10
Vehicle 2 has served customer 29 at: 60, and his demands: 20
Vehicle 5 has served customer 96 at: 60, and his demands: 30
Vehicle 6 has served customer 47 at: 60, and his demands: 30
Vehicle 1 has served customer 71 at: 62, and his demands: 30
Vehicle 2 has served customer 27 at: 62, and his demands: 10
Vehicle 4 has served customer 19 at: 62, and his demands: 20
Vehicle 5 has served customer 97 at: 62, and his demands: 10
Vehicle 6 has served customer 46 at: 62, and his demands: 10
Vehicle 6 has served customer 49 at: 64, and his demands: 10
Vehicle 1 has served customer 74 at: 65, and his demands: 10
Vehicle 2 has served customer 24 at: 65, and his demands: 10
Vehicle 4 has served customer 18 at: 65, and his demands: 20
Vehicle 5 has served customer 99 at: 67, and his demands: 20
Vehicle 6 has served customer 52 at: 67, and his demands: 10
Vehicle 2 has served customer 23 at: 68, and his demands: 20
Vehicle 1 has served customer 78 at: 69, and his demands: 10
Vehicle 6 has served customer 51 at: 69, and his demands: 10
Vehicle 1 has served customer 80 at: 70, and his demands: 10
Vehicle 2 has served customer 22 at: 70, and his demands: 20
Vehicle 3 has returned to the depot at: 72, and the remaining load:0
Vehicle 6 has served customer 53 at: 72, and his demands: 10
Vehicle 1 has served customer 81 at: 75, and his demands: 10
Vehicle 6 has served customer 50 at: 75, and his demands: 10
Vehicle 6 has served customer 48 at: 77, and his demands: 10
Vehicle 2 has returned to the depot at: 80, and the remaining load: 0
Vehicle 4 has served customer 11 at: 84, and his demands: 10
Vehicle 1 has served customer 82 at: 85, and his demands: 30
Vehicle 7 has served customer 13 at: 85, and his demands: 20
Vehicle 7 has served customer 15 at: 88, and his demands: 10
Vehicle 8 has served customer 61 at: 90, and his demands: 20
Vehicle 7 has served customer 16 at: 93, and his demands: 40
Vehicle 6 has returned to the depot at: 95, and the remaining load: 0
Vehicle 8 has served customer 57 at: 95, and his demands: 30
Vehicle 5 has returned to the depot at: 98, and the remaining load: 0
Vehicle 8 has served customer 54 at: 99, and his demands: 20
Vehicle 4 has returned to the depot at: 101, and the remaining load: 0
Vehicle 8 has served customer 55 at: 104, and his demands: 40
Vehicle 1 has served customer 84 at: 109, and his demands: 10
Vehicle 8 has served customer 56 at: 109, and his demands: 10
Vehicle 1 has served customer 85 at: 113, and his demands: 20
Vehicle 7 has returned to the depot at: 129, and the remaining load:10
Vehicle 1 has returned to the depot at: 144, and the remaining load:20
Vehicle 8 has returned to the depot at: 144, and the remaining load: 0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 8 New customers are available at time: 553 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 201.000
The load of the vehicle 1 is 190
The cost for the 2 route is 70.000
The load of the 2 vehicle is 20
-----------------------------------------------
The total cost for the all routes is 271.000
-----------------------------------------------
The customers will be visited as follows:
     1    76    17    40    39    34    59    66     1
     1    83     1
==============================================================
End updating the current solution
==============================================================
Vehicle 9 has served customer 76 at: 568, and his demands: 20
Vehicle 10 has served customer 83 at: 587, and his demands: 20
Vehicle 9 has served customer 17 at: 600, and his demands: 40
Vehicle 10 has returned to the depot at: 622, and the remaining load:180
Vehicle 9 has served customer 40 at: 645, and his demands: 20
Vehicle 9 has served customer 39 at: 650, and his demands: 30
Vehicle 9 has served customer 34 at: 658, and his demands: 40
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*** 1 New customers are available at time: 696 ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Vehicles are committed to the following customers:
    59

==============================================================
Updating the current solution
==============================================================
The cost for the route 1 is 94.000
The load of the vehicle 1 is 40
The cost for the 2 route is 42.000
The load of the 2 vehicle is 30
-----------------------------------------------
The total cost for the all routes is 136.000
-----------------------------------------------
The customers will be visited as follows:
     1    59    66     1
     1     3     1
==============================================================
End updating the current solution
==============================================================
Vehicle 9 has served customer 59 at: 704, and his demands: 30
Vehicle 11 has served customer 3 at: 716, and his demands: 30
Vehicle 11 has returned to the depot at: 737, and the remaining load:170
```

```
Vehicle 9 has served customer 66 at: 740, and his demands: 10
Vehicle 9 has returned to the depot at: 753, and the remaining load:10

****************************************************************
         1176

+++++++++++++++++++++++++++++++++
END OF PROGRAM
+++++++++++++++++++++++++++++++++
```

Figure D-5. A solution C101-90-10 instance

The final solution of C101-10-90 instance is summarised in Table D-5.

Table D-5. The final solution of C101-90-10

							Solution									Cost	load
1	79	77	72	71	74	78	80	81	82	84	85	1				144	180
1	21	25	26	28	30	35	37	31	29	27	24	23	22	1		80	200
1	68	64	63	75	73	62	65	69	67	70	42	1				72	200
1	6	8	2	5	7	10	9	12	14	20	19	18	11	1		101	200
1	4	100	101	98	94	93	95	96	97	99	1					98	200
1	44	43	41	60	58	45	47	46	49	52	51	53	50	48	1	95	200
1	91	88	87	86	89	90	92	13	15	16	1					129	190
1	33	38	36	32	61	57	54	55	56	1						144	200
1	76	17	40	39	34	59	66	1								201	190
1	83	1														70	20
1	3	1														42	30

Road Closure Examples

Figure D-6 shows a road closure occurred in P-n65-k10 instance at time 16.

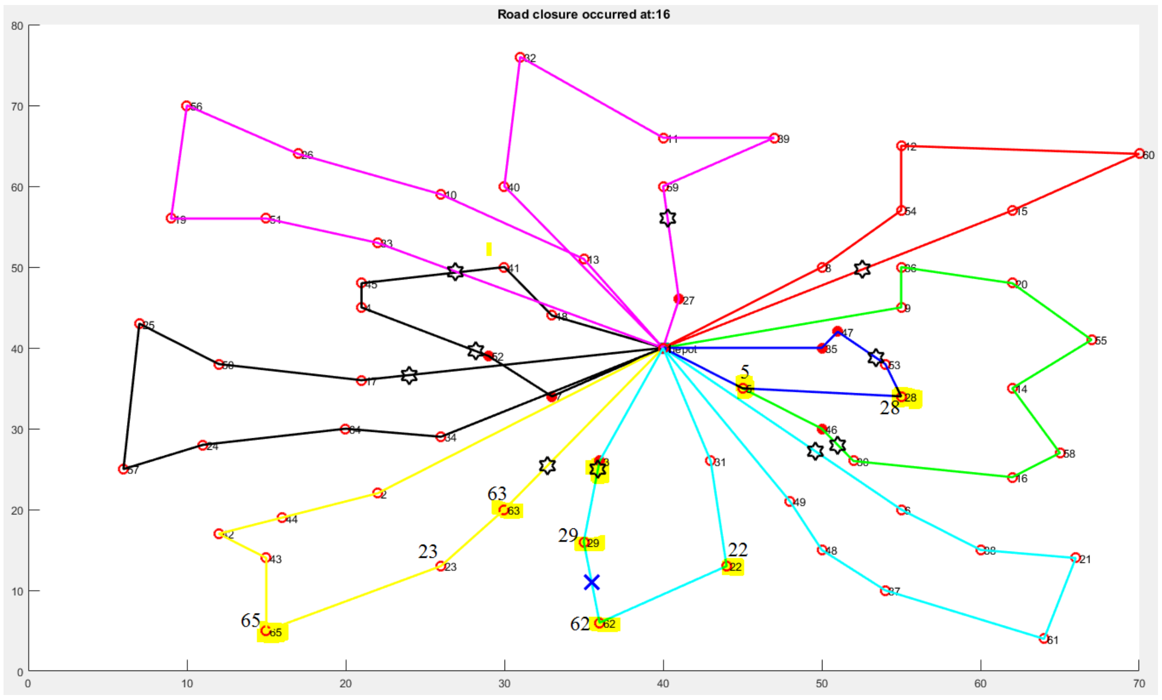


Figure D-6. A road closure in P-n65-k10 instance

Figure D-7 illustrates a possible solution for the previous road closure.

```
---------- The day is Started ----------
Event occurred at time: 16

vehicle 1 on the edge number: 3, between customer: 52  and customer: 4
vehicle 2 on the edge number: 2, between customer: 27  and customer: 59
vehicle 3 on the edge number: 1, between customer: 1  and customer: 6
vehicle 4 on the edge number: 1, between customer: 1  and customer: 15
vehicle 5 on the edge number: 2, between customer: 46  and customer: 30
vehicle 6 on the edge number: 3, between customer: 47  and customer: 53
vehicle 7 on the edge number: 1, between customer: 1  and customer: 63
vehicle 8 on the edge number: 1, between customer: 1  and customer: 17
vehicle 9 on the edge number: 1, between customer: 1  and customer: 33
vehicle 10 on the edge number: 2, between customer: 3  and customer: 29
**************************************************
Road Closure at route 10
The affected route is     1     3    29    62    22    31     1
Road Closure at place 3
Road Closed between node 29 and node 62

Calling HCA: (asymmetric TSP: re-ordering)
The new solution is:
    29    22    62    31     1
The cost after performing re-order operation: 799

--------------------------------------------------
Calling HCA: (shift operator)
The new solution is:
     1    53    28    62     5     1

The cost after performing the shift operation: 834

--------------------------------------------------
Calling HCA: (exchange operator)
The new solution is:
     1    63    62    65    43    42    44     2     1

     1    29    23    22    31     1

The cost after performing the exchange operation: 812

--------------------------------------------------
The cost after adding new vehicle: 848

++++++++++++++ The day is finished ++++++++++++++
```

Figure D-7. A solution for the road closure in P-n65-k10 instance

Figure D-8 shows a road closure occurred in E-n51-k5 instance at time 37.

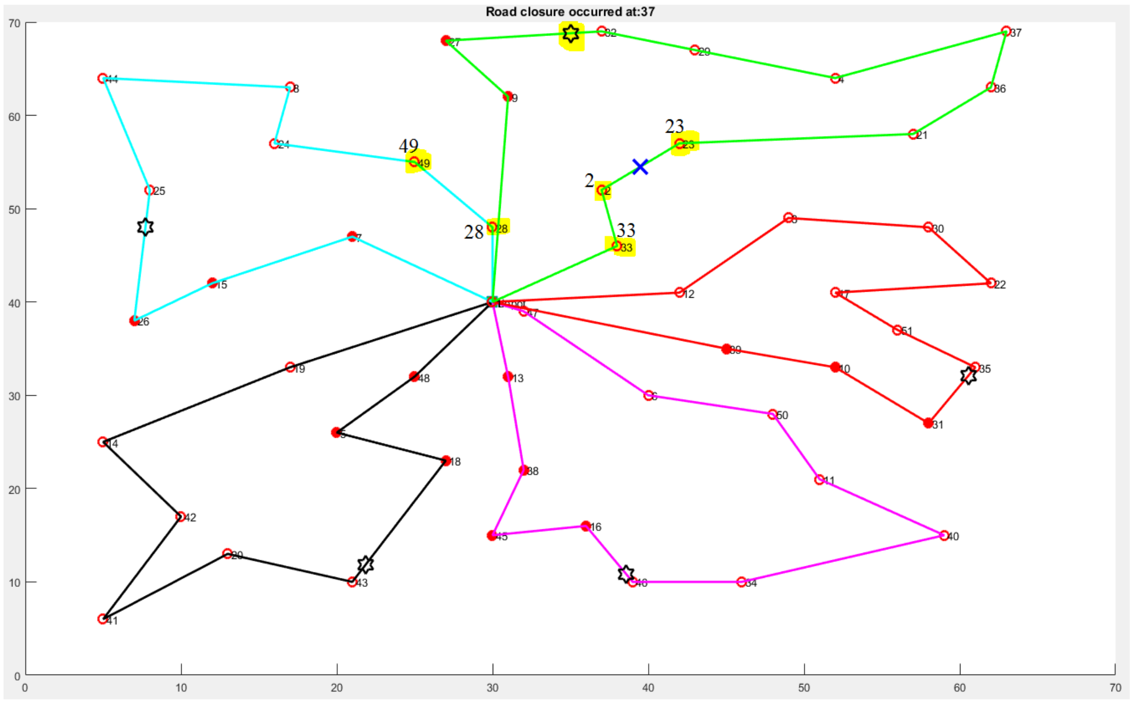


Figure D-8. A road closure in E-n51-k5 instance

Figure D-9 illustrates a possible solution for the previous road closure.

```
---------- The day is Started ----------
Event occurred at time: 37
vehicle 1 on the edge number: 4, between customer: 18  and customer: 43
vehicle 2 on the edge number: 5, between customer: 16  and customer: 46
vehicle 3 on the edge number: 4, between customer: 26  and customer: 25
vehicle 4 on the edge number: 4, between customer: 31  and customer: 35
vehicle 5 on the edge number: 3, between customer: 27  and customer: 32
**************************************************
Road Closure at route 5
The route is      1     9    27    32    29     4    37    36    21    23
2    33     1
Road Closure at place 10
Road Closed between node 23 and node 2

Calling HCA: (asymmetric TSP: re-ordering)
The new solution is:
    32    29     4    21    36    37    23    33     2     1

The cost after performing re-order operation: 535
--------------------------------------------------
Calling HCA: (shift operator)
The new solution is:
     1    25    44     8    24    49     2    28     1

The cost after performing the shift operation: 531
--------------------------------------------------
Calling HCA: (exchange operator)
The new solution is:
     1    25    44     8    24    49     2     1
     1    32    29     4    37    36    21    23    28    33     1
The cost after performing the exchange operation: 540
--------------------------------------------------
The cost after adding new vehicle: 548
++++++++++++++ The day is finished +++++++++++++++
```

Figure D-9. A solution for the road closure in E-n51-k5 instance