Performance Evaluation of Metaheuristics Search Techniques in Resource Allocation Problems

Name: Amit Shah

ID: 0704711

2011

A thesis submitted in partial fulfilment of the

requirements for the degree of

Master of Computer and Information Sciences

Auckland University of Technology

Primary Supervisor: Dr. Andrew Connor

Abstract

Existing research has focused on solving problems in the area of project management using variety of approaches including search based software engineering approach. The main aim of this research is to evaluate the performance of metaheuristics search techniques such as genetic algorithm, simulated annealing and tabu search in resource allocation problem with project management discipline. This will enable the paper to introduce an alternative approach to solve this Resource Constrained Project Scheduling Problem (RCPSP). The nature of this research is both constructive and experimental therefore software development research methodology will be utilised as a guideline.

This study reports a comprehensive set of experiments which evaluate the performance of metaheuristics search techniques. Initial set of experiments were performed over various numerical test function to verify the implementation of search techniques. The next stage of experiments had focused on the scalability of these search techniques. Based on the first two experiments, search techniques were evaluated against a discrete problem to further explore the scalability. Finally, a multi objective test case problem was evaluated which focused around RCPSP. For each of these experiments the parameters were fine-tuned during the design phase of the experiments.

Based on the experiments, it was apparent that the metaheuristics search techniques can be used to solve problems in resource allocation within project management discipline. Finally, a comparison analysis strongly suggests that overall simulated annealing had performed better than genetic algorithm and tabu search.

Key words: Software engineering, search based software engineering, metaheuristics search techniques, resource constrained project scheduling problem, cost allocation, project management.

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 (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning."

Signed:

Amit Shah

28th November 2011

Acknowledgements

First and foremost I offer my sincere gratitude to my supervisor Dr. Andrew Connor, who supported me throughout my research with his patience and knowledge. In addition to this, I would also like to thank all the staff members of Auckland University of Technology for their continued support during the past three years of my studies.

I would also like to extend my gratitude towards my work (PriceWaterhouseCoopers) who supported and allowed me to take regular study leaves.

Special thanks go to my wife for her enormous patience and support; my parents for their guidance and wisdom; my sister and her husband for the inspiration they provide. Their encouragement has helped me through the completion of this research.

Table of Contents

List of Figures

List of Tables

List of Equations

1 Introduction

This chapter presents the motivation and background for this research. The main objective of this research is to evaluate search based approaches when applied to project scheduling, cost estimation and resource allocation problems in software engineering project management.

1.1 Background and Motivation

The discipline of Software Engineering has been in existence since early 1960s and since then, it has evolved tremendously and during which many new subsidiaries of software engineering have been introduced. Search based approaches in Software Engineering date back to the 1970s when researchers started to investigate automated approach to software testing (Miller & Spooner, 1976). Search Based Software Engineering (SBSE) was formalised in 2001 (Harman & Jones, 2001) and presented as sub-discipline software engineering. The underlying principles of SBSE involve the application of metaheuristics search techniques like genetic algorithms, simulated annealing and tabu search to software engineering problems. The approach is inspired by the observation that many activities in software engineering can be formulated as optimisation problems and are therefore tractable to automatic solution.

Since then the majority of SBSE research is still focused on software testing as this is an area that is easily formulated as a search problem (Harman, 2007). However, a growing number of researchers have implemented SBSE to solve problems in more challenging areas of Software Engineering, such as Requirements Engineering, Resource Allocation, Cost Estimation and Project Scheduling (Harman, 2007). This research focuses on the areas of Resource Allocation, Cost Estimation and Project Scheduling in software project planning.

Managing software projects has been a crucial task since the early days of software engineering. Over the years, basic principles of software engineering have evolved and as a result principles of software project planning have evolved as well. In recent years, almost all industries use software and because of which the demand of software development and/or implementation have increased considerably. Most of the development and/or implementation of software require a project management strategy in order to be successful. As software complexity and size increases, the need for both efficient and effective project management approaches will grow. Since the size and the number of software projects have increased considerably, the problems with resource allocation, cost estimation and project scheduling have become more visible than ever. Hence, there is huge demand for new methods and technologies to enable software project

managers to solve the problems in the area of cost estimation, resource allocation and project scheduling more efficiently and effectively. Over the years, the demand for introducing new methods and/or technologies mentioned above have increased and based on that the need for this research can be justified.

1.2 Definitions

1.2.1 Heuristics

Heuristics is a method which has the ability to produce good solution for a given problem within a reasonable amount of computation time. An ideal example of heuristic approach would be to use the rule of thumb when making a decision (Herroelen, 2005).

1.2.2 Metaheuristics

Like heuristics, metaheuristics is also an approach to solve a problem. Metaheuristics approach tries to optimise a problem by continuously trying to find an optimum solution based on the given criteria. Common metaheuristics techniques include but are not limited to genetic algorithm, simulated annealing, and tabu search (Harman & Jones, 2001).

1.2.3 Search Based Software Engineering

In the discipline of software engineering there are several problems and search based software engineering is one of the approaches to solves those problems using metaheuristics search techniques such as genetic algorithm, simulated annealing, tabu search, etc (Harman & Jones, 2001).

1.2.4 Work Breakdown Structure

In project management, when all the deliverables are grouped into smaller deliverable components is called Work Breakdown Structure (WBS) (Herroelen, 2005). Establishing a WBS is a part of the project scheduling exercise and this is one of the main ingredients for establishing requirements for funding and resources.

1.3 Statement of Problem

Most software projects require some sort of project management to govern its resources, cost and scheduling. The biggest challenge for a software project is to complete on time and budget. It has been observed that most software projects need to be considered at least partial failures because so few projects meet all of the cost, schedule, quality or requirement objectives (May, 1998). Such doom and gloom is not uncommon in the literature review, and any discussion of critical success and failure factors in software development would be incomplete without

recognising the essays of the Fred Brooks (1995) that discussed many such factors. According to the widely quoted Chaos Report by the Standish Group, citied by Yeo (2002), declared that software projects are in chaos with only 16.2% of software projects are actually being successful. Whilst some authors have questioned the relevance and integrity of the Chaos report (Glass, 2006), it is clear that developing software is equally challenging today as it was at the time of writing of the Mythical Man Month (Brooks, 1995) though the reason and their relative impact may have changed in the intervening period. The prevalence of failing software is one of the major drivers for this research as there is a clear need for more effective project management practices.

To complete a project successfully, project managers have two main goals. The first is the creation of a realistic project plan, and secondly the management of unforeseen circumstance to ensure that plan is implemented. Part of the creation of the plan involves the allocation of tasks to resources with proper deadlines.

However, there are several reasons that could force a project to fail or not complete within a given time frame and budget. Problems in the management of resources, time frame, budget, risks, and many more are some of the main contributing factors for failure. Such problem falls into two categories that match the above goals. The problems in the first category are caused by creating an unrealistic plan because the availability of a given resource is not fully understood or interactions between tasks are not clearly defined. The problems in the second category are caused as a result of unforeseen circumstances. This research essentially addresses the first category of problems by providing an approach for project managers to better understand the constraints imposed on their projects by the availability of resources. Whilst small project plans for allocating resources and scheduling can be established using heuristics (rule of thumb) approaches. Such approaches may not be suitable for larger projects, and therefore tool support is recommended to plan a project. In most cases managing project (small or big) can encounter problems in the following areas:

1.3.1 Resource Allocation

In planning a project, allocating resources to tasks is one of the most important activities. This part of planning is categorised in two sections, first section focuses on allocating resources to each tasks based on the priority of the task. A simple rule is applied whereby if the priority of the task is higher than more resources are allocated. At times, tasks with low priority have minimum or no resources assigned to it. The second section focuses on planning a contingency for all tasks

so that if there are any available resources they can be allocated based on the priority of the tasks. Examples of resources are human resources, equipment, software licenses, funding, and many more. Depending on the size of the project, resources can either be allocated manually or by the means of a computer program.

1.3.2 Cost Estimation

Cost is one of the most important ingredients to plan a software project. The main purpose of this activity is to estimate cost accurately so that the project can be completed within allocated budget. Inappropriate cost estimation can lead to cost overrun or even failure of a project. During this stage of planning, based on the estimated costs it allows the project managers to secure resources for each tasks. The planning for this stage is usually done in three steps. First, identify all the tasks of the projects and their priority, based on the result generated from the first step, the Second focuses on estimating start and end date for each tasks and how much resources each task would require. Finally, the project managers need to establish total cost for each task based on resources allocated. After allocating each task, project managers can estimate cost for the entire project.

1.3.3 Project Scheduling

The main purpose of scheduling a software project is to determine its start and end date. Typically a start and end date is set for each task so that an overall timeframe is determined. The Project Manager cannot start working on scheduling until they have established a work break structure (WBS), list of tasks, effort required for each tasks, and list of available resources. Furthermore, during the life of the project, project managers are consistently working on schedule to ensure that project's deadline is achieved. Since this task is implemented at the very start of projects, sometimes it may be hard to determine start and end date due to many reasons such as availability resources, cuts in funding from project sponsors, and many more. Inappropriate scheduling of a project indicates that the project may not be completed within the given time frame and budget. Furthermore, it may also mean that third party resources (equipment, human resources, etc) that were allocated based on the initial start and end date of a task may not be available.

As mentioned earlier, most aspects of project planning can have problems in the area of resource allocation, cost estimation and project scheduling. When all the three problems are combined they can be classified as a "Resource Constrained Project Scheduling Problem" (RCPSP). Researchers have explored RCPSP solutions for several decades with the first major review of

RCPSP literature occurring in 1995 (OZdamar & Ulusoy, 1995) with work continuing to the present day (Bianco & Caramia, 2011). The sheer volume of articles published of this topic over this time period indicates the significance of the challenge presented by the problem.

The main goal of RCPSP is to estimate cost and resources required for each tasks so that the duration of the project is minimised without exceeding budget. The RCPSP is very complex problem to solve and visualise. To demonstrate this problem, a simple project with ten activities displayed in Figure 1.1. In this figure each node represents an activity which also displays its precedence constraint using the arcs. Furthermore each node has predefined days and resources to complete a task. Task #1 and Task #10 both acts as a project milestone.

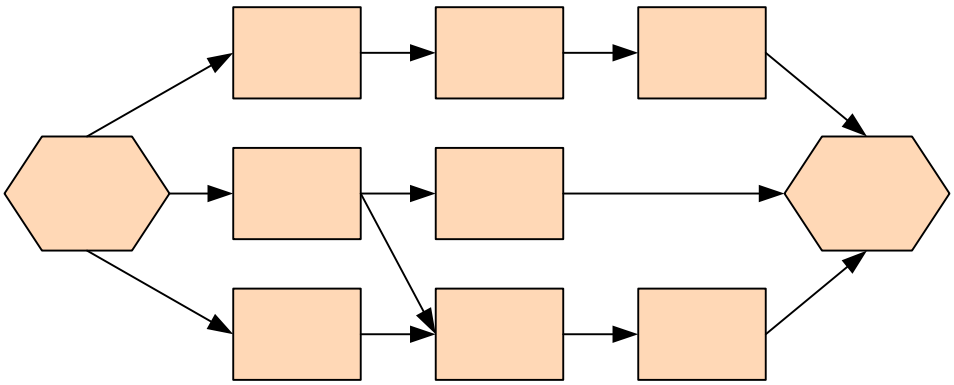

Figure 1.1 Project Network Diagram

Based on the scenario outlined in Figure 1.1, a solution needs to be generated using heuristics approach whereby all the resources are optimally utilised to complete ten tasks. This simple example assumes that all resources are equal and may be assigned to any task. An optimum solution for Figure 1.1 is graphically presented in Figure 1.2. Hence looking at this scenario, it was relatively easy to find an optimum schedule using heuristics (rule of thumb) approach but by heuristics do not gurantee an optimum solution but it has the potential to provide a good solution within a reasonable amount of computation time. In “real-world” projects have hundreds of tasks and it could be very difficult to generate an optimum schedule using heuristic approach because it would be very difficult to remember various combinations of resources, tasks, available time, available cost, etc.

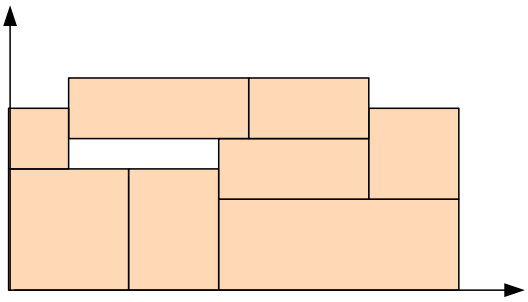

Figure 1.2 Optimal Solution for Figure 1.1

Over the years, several researchers have tried to solve RCPSP using various methods, but the main categories are presented in Figure 1.3. However, this research will try to resolve the RCPSP using "Classical Metaheuristics" which includes implementations of Genetic Algorithm, Tabu Search and Simulated Annealing.

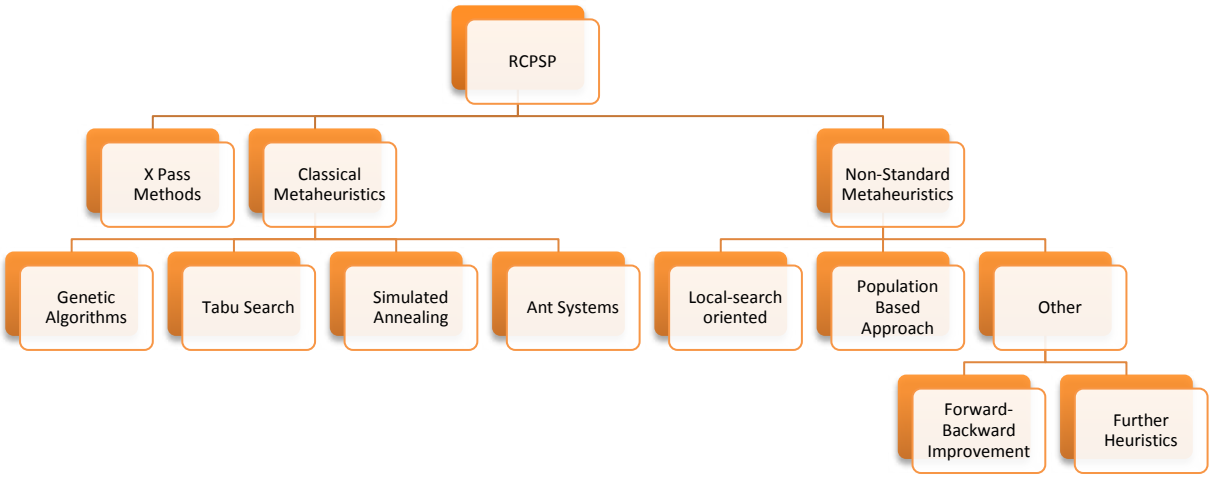

Figure 1.3 RCPSP Solving Techniques (Kolish & Hartman, 2006)

1.4 Research Question

Based on the problems mentioned in section 1.2, it is clear that there is an ongoing issue with project planning (resource allocation, cost estimation and project scheduling) activities. For the purpose of this research, there will be an attempt made to represent these in accordance with the principles underlying search based software engineering and solve the problem using metaheuristic search algorithms. As a result of initial study, following questions have been formulated:

1.4.1 Question 1

Can resource allocation and cost estimation be effectively formulated as a search problem that is tractable to be solved using metaheuristic search algorithms?

1.4.2 Question 2

Which metaheuristic search algorithms demonstrate the most desirable performance when solving these problems?

1.5 Thesis Structure

This thesis is categorised into five main chapters. This first chapter identifies problems in the area of project scheduling, cost estimation, and resource allocation. To illustrate the problem a simple example with an ideal solution is presented. The second chapter provides the context for this research by identifying past and current research in the area of search based software engineering, metaheuristics search techniques, discrete optimisation problems, project management principles, problems with cost estimation, resource allocation and project scheduling. Furthermore, this chapter identifies how various authors have tried to use search based software engineering to solve these problems. The third chapter focuses on the research methodologies used to conduct this research. This chapter describes the framework that was developed to conduct experiments to explore the research questions. The fourth chapter is broadly categorised into four main sections which includes verification of metaheuristics search techniques, the scalability testing using Bump functions, scalability testing using n-Queens problem and lastly solve RCPSP using genetic algorithm, simulated annealing and tabu search. The last chapter of this research will focus on discussions around the data and the analysis generated in previous section and then the paper will be completed with a conclusion of the discussions and identify the direction(s) for future research.

2 Literature Review

The main objective of this chapter is to identify past and current research which has tried to resolve project scheduling problems using search based software engineering approaches and to define the basis of metaheuristic search algorithms. Based on the knowledge gained from literature review, limited number algorithms will be selected for implementation to solve sample project scheduling problems. While identifying the literature, this chapter also defines and explains some of the key concepts used across all the chapters.

To ensure the robustness of this research, the review of literature has been conducted using the most relevant literature. Primarily, the review of literature has been conducted to lay the background knowledge of this research. After that, the review of literature will be further categorised into three problem areas as mentioned in the Chapter 1, Section 1.2. Hence the review of literature will be categorised into:

- **Search and Optimisation Algorithm**: One of the main objectives of this research is to evaluate metaheuristics search techniques. As such there are several techniques for optimising problems, but for this research, literature about few methods will be detailed.
- **Discrete Problems**: As a part of the evaluation process in this study, metaheuristics search techniques will be evaluated against a discrete problem. Literature in this area will describe various discrete problems, but for the purpose of this research n-queen problem has been selected to conduct experiments. Literature in this area will help design the experiments and additionally results from these experiments can be compared against the literature review. Additionally, Metaheuristics search techniques will also be evaluated against variety of numerical test functions and the description for these numerical test functions will be detailed in Experimental Design. This will mainly focus on presenting the solutions space and the formula used to carry out the experiments.
- **Search Based Software Engineering**: This research is categorised under the discipline of search based software engineering, hence this section will give a detailed introduction about the discipline. Furthermore, following sections of literature review will use this as a foundation to explain their arguments and findings.
- **Project Management**: Since this research is partially related to managing projects, it is important that the literature in the discipline of project management has been reviewed. This review will enable the paper to establish basic principles of project management

which can be applied in evaluation process of the data that is generated from the implementation.

- **Software Project Management**: In addition to literature review in project management discipline, this chapter will also conduct a review in software project management a child discipline of project management discipline. Review in this area will lay the foundation knowledge for research in software project planning (cost estimation and resource allocation).
- **Cost Estimation Models**: This review of literature will highlight how various authors have used the principles of search based software engineering to build cost estimation model.
- **Resource Constraint Project Scheduling Problem**: Similar to literature review of cost estimation models, this chapter will also review literature from various authors who have tried to resource constraint project scheduling problem using search based software engineering.

For the purpose of gathering accurate and robust information, various articles have been collected from different disciplines and sources. The main disciplines that were used to gather articles are as follows (but not limited to):

- Software Engineering
- Search based software engineering
- Optimisation
- Numerical test functions
- Discrete problems
- Metaheuristics search techniques
- Project management
- Resource constrained project scheduling problem

In addition to this, to a build strong research background, articles have been selected from a wide range of time line starting as early as 1960s to most contemporary articles dated in 2011. Although some articles are old, but they help develop a good research background. As the research progresses, contemporary articles have been selected for annotations.

2.1 Search and Optimisation Algorithms

The purpose of this section is to compare a variety of optimisation techniques. There are many different methods available for evaluating solutions to a given problem and searching for an optimal solution. These vary from calculus based techniques through to combinatorial methods. There are four main types of optimisation methods.

2.1.1 Classical Optimisation Theory

Classical optimisation theory develops the use of differential calculus to determine maxima and minima for unconstrained and constrained functions. The classification of such techniques varies in two general two classes, direct and indirect. Indirect methods search for optimum points by solving the equations that are obtained by setting the first derivative of the objective function to zero. Direct methods search for optima by taking a point on the objective function surface and moving in a direction determined by the gradient at that point. Both direct and indirect approaches are generally referred to as gradient methods (Gill, Murray & Wright, 1981).

Neither of these general methods is particularly useful outside of the simple domain where the objective function is uni-modal, smooth and continuous. They act primarily as local search and are prone to become trapped on sub-optimal peaks. Classical techniques are not considered sufficiently robust to provide useful solutions to many complex problems.

2.1.2 Linear Programming

Linear programming (Luenberger, 1984) technique is applicable where the objective function and constraints are formed as linear functions of the independent problem variables. Linear programming methods can easily deal with both equality and inequality constraints.

The most general and widely used linear programming techniques, is the Simplex Method (Nelder & Mead, 1965). In 1947, George Dantzig created a simplex algorithm to solve linear programs for planning and decision-making in large-scale enterprises. The algorithm's success led to a vast array of specializations and generalizations that have dominated practical operations research for half a century. This method can deal with large numbers of problem variable and is easily implemented as a numerical computation. For simple linear optimisation problems the simplex method is a powerful tool which finds the optimum point in a multi-variate feasible region. Unfortunately, many optimisation problems are non-linear and other methods are required to find solutions.

2.1.3 Non-Linear Programming

Non-linear programming (Luenberger, 1984) methods fall into both uni-variate and multi-variate categories. Essentially, non-linear programming methods are search techniques where an algorithm directs the search towards an optimal solution. Because the methods are based on searching, as opposed to calculus, the objective functions do not need to be smooth or continuous. Hence, non-linear programming methods have become widely popular. Examples of uni-variate methods include the Golden Section Search (Press, Flannery, Teukolsky & Vettering, 1986), Rosenbrock's method (Rosenbrock, 1960) and Quasi-Newton methods such as BFGS. These simple searches have been extended into the multi-variate domains to give rise to methods such as Powell's Method of Conjugate Directions (Powell, 1977) and the Hooke and Jeeves Pattern Search (Hooke & Jeeves, 1961). Using these methods, constraints are normally dealt with by the use of penalty functions.

A non-linear programming method has advantage of numerical computation, but also suffers in that they are generally local search methods. Without resorting to numerous and complex penalty functions they are not robust enough to search extremely convoluted objective function surfaces to any degree of accuracy. It is necessary to check that the methods are finding truly optimum solutions and this process of reiteration prolongs the time required for an analysis.

2.1.4 Metaheuristics

In recent years, research in the area of solving problems with optimisation techniques has increased considerably and leading to development of new systems and methods. When the performances of new developments are compared to linear programming methods, it is clear that new optimisation techniques are more robust and efficient. Harman & Jones (2001) mentioned that most of these techniques are currently implemented in disciplines like software / mechanical engineering, biotic engineering, software testing, and many more. Additionally, such techniques have also been used to solve problems like Travelling Salesman Problem, N-Queen Problem and many more. The importance of metaheuristics has been increasing over the years and to support that argument, many researchers have attempted to solve "real world" problems.

2.1.4.1 Simulated Annealing

Simulated annealing is a metaheuristics search technique which can be used to solve optimisation problems. The technique has the ability to find solutions in large and small spaces. Unlike few other search techniques, this technique is a direct search method involving a single search trajectory (Kirkpatrick, Gelatt & Vecchi, 1983). The name and inspiration for this search

technique was derived from the process of annealing solid material (e.g.: metal). This annealing process involved heating and gradually cooling the solid material so that the defects are reduced. After the completion of this process, it can be concluded that the solid material has reached a global minimum state.

Simulated annealing operation is very straight forward. When the algorithm is initiated, an initial set of problem is selected on random. Hence a value of x is chosen for the solution and the cost function is minimised. The size of the problem usually determines the cost of the function. Hence, if the size of the problem is large then the cost of the problem is larger and vice versa. After initial value is selected, it changes slightly to generate a new candidate solution from the neighbourhood of the initial solution. After selecting a new candidate solution, value of the cost function is obtained and if the value is better than previous candidate solutions then it is retained. However, if the value is worse than any other candidate solution, then there is small probability that the search will move to the next candidate solution and continue. The calculation of the probability is calculated using the equation mentioned below and the equation is similar to Maxwell-Boltzmann probability function.

$$p = exp\left(\frac{-\Delta \mathrm{E}}{T}\right)$$

Equation 2.1 Maxwell-Boltzmann Probability Function

In equation presented above, T represents the temperature and ΔE represents the change in energy. Apart from the equation, this search technique requires some basic ingredients and they are as follows (Clark, et al., 2003):

- Definition of neighbourhood and its configuration.
- Cooling schedule specification

When there is a change in the value of the cost function, it determines the change in energy. The units of temperature control parameters and cost function are the same. Additionally, temperature control parameter also enables the probability of selection. During the initial stages of the execution process of the algorithm, the temperature is kept steady and this allows the system to gain momentum in searching. Ideally as the temperature drops, the probability of selecting a bad solution reduces. Hence towards the end, this algorithm tends to move towards an optimum solution.

2.1.4.2 Tabu Search

Tabu search has similar search method characteristics to simulated annealing and is generally implemented as a single search trajectory direct search method. The concept was originally coined by Glover (1989, 1990), and since then the application of this search technique has increased considerably. Tabu search has been successfully implemented to solve discrete combinatorial optimisation problems such as graph colouring and Travelling Salesman Problems, and has also been applied to a range of practical problems. In terms of operations, tabu search uses a local or neighbourhood search procedure to iteratively move from one potential solution to an improved solution in the neighbourhood. This iteration stops when terminating criteria has been met. To avoid finding an inappropriate solution and to ensure that the search space is explored by the local search procedures, tabu search thoroughly examines the neighbourhood of each solution as the search progresses. The solutions admitted to the new neighbourhood, are determined through the use of memory structures. These memory structures, the search progresses by iteratively moving from the current solution to an improved solution. These memory structures form what is known as the tabu list, a set of rules and banned solutions used to filter which solutions will be admitted to the neighbourhood to be explored by the search. In its simplest form, a tabu list is a short-term set of the solutions that have been visited in the recent past. To implement this technique efficiently, it is necessary that following ingredients must be considered (Clark, et al., 2003):

- Neighbourhood of a solution
- Aspiring move definition

2.1.4.3 Genetic Algorithms

Unlike simulated annealing and tabu search, genetic algorithm is not a local search method. This search technique classified as an evolutionary search method. This algorithm was built on the principles of Darwinian Evolution (Clarke et al., 2003; Goldberg, 1989) and was first developed in 1975 (Holland, 1975). Since its introduction, this search technique has been used in variety of disciplines and there is substantial research to identify its practical implementations.

Goldberg (1989) further adds that genetic algorithm is a non-derivative based optimisation technique and the outcome of this algorithm is not but the survival of the fittest. When the algorithm is initiated, a candidate solution set is created on random and this is called population. Using the existing population, new generation is created using genetic operators like crossover,

mutation, and reproduction. Ideally as the algorithm progresses, the solutions are improved and optimum solutions can be achieved over time.

- **Crossover**: Is a genetic operator that combines two chromosomes to produce a new chromosome. The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during the evolution according to a user-defined crossover probability. There are several methods of selection of chromosomes for crossover like Roulette wheel selection, Boltzman selection, Tournament selection, Rank selection and many more. There are also varieties of crossover techniques such as one-point crossover, two-point crossover, three parent crossover, crossover for ordered chromosomes, and "cut and splice".
- **Mutation**: This operator alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new genes values, the genetic algorithm may be able to generate a better solution. Mutation is an important part of the genetic search as it helps to prevent the population from stagnation at any local optima. Mutation occurs during evolution according to a user-defined mutation probability.

Genetic Algorithms are a broad and effective search method which has been applied to a wide range of practical problems. The term Genetic Algorithm is particularly broad and covers many variations in implementation ranging from the simple GA presented by Golberg (1989) through to complex multi-objective algorithms such as NSGA-II (Deb, Agrawal, Pratap & Meyarivan, 2000). To efficiently implement genetic algorithm, following ingredients must be considered:

- Probability of crossover and mutation operators
- Selection criteria for crossover, mutation or reproduction

2.1.4.4 Particle Swarm Optimisation

Particle swarm optimisation has been in existence since 1995. The inspiration behind this technique is derived by the social behaviour of bird flocking. This behaviour slightly makes this algorithm like genetic algorithm hence would classify under evolutionary methods of solving discrete optimisation problems (Kennedy & Eberhart, 1995; Shi & Eberhart (1998)). In saying that, this search technique doesn't have operators in the same was as a genetic algorithm (crossover, mutation and reproduction). PSO optimizes a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-

space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position and is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions. An extensive survey of PSO applications is made by Poli (2008) and has verified that this technique has been used for both research and practical implementation.

2.1.4.5 Ant Colony Optimisation

Ant colony optimisation is a general search technique and unlike the previous search techniques. This search technique was introduced Marco Dorigo in 1992. It is ideally suited to graph based problem domains but can also solve discrete combinatorial optimisation problem (Christodoulou, 2010). As the name suggests “ant colony”, the principles behind this technique came from the behaviour of real ant colonies. The main idea is that ants in a colony lay a path and other members of colony follow the path. Eventually the path leads to an optimum solution. Since its introduction, this technique has been used extensively to solve huge range of numerical test problems. The ACO algorithm has been applied to the RCPSP (Merkle, Middendorf & Schmeck, 2000) which can be represented as a graphing problem.

2.1.4.6 Other Algorithms

Apart from the algorithms mentioned earlier, there are varieties of algorithms which can be used and these techniques are (but not limited to) Artificial Immune System (Farmer, Packard & Perelson, 1986), Harmony Search (Geemm, Kim & Loganathan, 2001), the Firefly algorithm (Yang, 2009) and Spiral Optimisation (Tamura & Yasuda, 2011). Many of these algorithms are in essence attempts to hybridise or reinforce concepts that are taken from more established algorithms. Whilst they may offer certain features that make them useful and appealing, they are currently excluded from this study.

2.2 Discrete Optimisation Problems

As the name suggests these problems are discrete in nature unlike the numerical test functions which are mainly continuous in nature. Usually discrete problems are extremely hard to solve within a reasonable amount of time without the use of some techniques. Ideally these can be solved using metaheuristics search techniques mentioned in section 2.2, but is not limited to that. There are varieties of discrete problems, how for the purpose of this study, the literature will briefly focus on the following:

2.2.1 Travelling Salesman Problem

This problem was first established in 1930s and the main agenda for this problem is to find the shortest route for a travelling salesman in such a way that all the cities are visited and ensure that no two cities are visited twice (Lin & Kernighan, 1973). Authors have also mentioned that underlying principles of this problem has been used to solve variety of optimisation problem in various disciplines like shipping, planning, logistics, DNA sequences, etc. Using this problem there are variety of other problems developed like bottle neck travelling salesman problem, generalised travelling salesman problem and many more.

2.2.2 Knapsack Problem

This problem is also known as the rucksack problem. To explain this problem, let's consider an example where *n* is the number of available items with a weight and a value associated with it. Hence the ideal solution to this problem is to determine the best combination of items where the weight is minimised and the value is maximised (Ross & Tsang, 1989). This problem has some practical implementation when solving RCPSP with financial constraints.

2.2.3 N-Queens Problem

N-Queen problem was first introduced in 1848 and it is a problem of placing chess queen on *n*x*n* chessboard so that no two queen attack each other (Martinjak & Golub, 2007). The main agenda behind this problem is to place maximum number of queens on a chessboard so that no two queens attack each other. On a chessboard a queen can attack in all the cells in its same row or column and either diagonal. The original problem related to allocating eight queens a spot on the 8 x 8 chessboard (Martinjak & Golub, 2007). Hence to enable the implementation of this problem, it has to be generalised to represent *n* number of queens and this generalisation process has occurred in 1975 (Martinjak & Golub, 2007). Authors have also further argued that this problem usually take longer to compute then other continuous problems. Like other discrete problems, the principles behind this problem has been also implemented to in practical scenarios like parallel memory storage, managing traffic, deadlocks, and many more.

2.2.4 Summary

All the problems mentioned above belong to the same category of NP-Hard problem in combinatorial optimisation studied in computer science discipline. At time, metaheuristics search techniques are implemented to solve other problems but to verify the performance and viability of the search techniques, the above mentioned problems can be used. For the purpose of this study, one of the experiments will focus on solving n-queen problem using genetic algorithm, simulated annealing and tabu search.

2.3 Search Based Software Engineering

Search Based Software Engineering (SBSE) was introduced in 2001 by Harman & Jones (2001). The main objective of SBSE is to solve problems in software engineering using metaheuristics search techniques such as genetic algorithm, simulated annealing, Tabu Search, Particle Swarm Optimisation, and many more. Past literature indicates that several researchers tried to solve problems in software engineering, in the area of software testing and software project cost estimation using genetic programming and parallelisation, which were not categorised under SBSE. SBSE topic was consistently researched since 2001, but this topic was revisited in 2007 by Harman (2007) to reassess the current and future of SBSE. In his article Harman concluded that SBSE is still a widely researched topic and is expecting that the research will grow more in this domain. So far, research has been carried out in following areas:

- Software testing
- Test data generation
- Reverse engineering
- Software Estimation
- Resource allocation and cost estimation.

According to authors Harman & Jones (2001) and Harman (2007), to efficiently solve any optimisation problem in software engineering which are not polynomially solvable, it is necessary that it should be reformulated as “Search Problem” or “NP Complete Problem”. To reformulate optimisation problem as a search problem, following are the three main ingredients:

- **Representation**: To represent the most optimum solutions, it is necessary to identify and define the nature of the problem. Based on the requirements, each problem will have its own unique representation.
- **Fitness Function**: When a metaheuristic search technique is implemented, it may go through several iterations and generate possible candidate solutions. To determine if the selected candidate solution is optimum, it is evaluated against a fitness function. Each fitness function provides the ability to generate a landscape which represents all the candidate solutions. Ideal fitness landscapes should not be flat or have sudden spikes in the presentation. Fitness functions are easier to develop when the type of the problem is simple, but it becomes increasingly difficult to develop a fitness functions when the type of the problem is extremely complex.

- **Operators**: There are varieties of search techniques which can be implemented to solve "search problem" and each techniques require different types of operators. As a minimum, each search technique will require an operator which can help generate initial candidate solution.

The factors mentioned above have been well established since the introduction of SBSE but in comparison to that, Clarke et al. (2003) have also shared similar view, but the research is constructed from a different point of view whereby operators have been combined with the "Fitness Function". Their main reason for such an approach is to reduce complexity whilst reformulating software engineering as a search problem.

2.3.1 Validity of Search Based Software Engineering

After reviewing the literature it is clear that not all software engineering problems can be solved using metaheuristics search techniques, this is mainly because some problems may require human intervention. Before metaheuristics search techniques are implemented to solve a software engineering problem, it must answer the following questions:

- Is the solution space big enough?
- If the solution space is big enough then, would it be possible to implement metaheuristics search techniques?
- Is the developed fitness function suitable to the search problem?
- For the given problem, are there any existing solutions?

Based on the answer for the questions mentioned above, we can validate the use of SBSE to solve a search problem. Once validated the next step would be to select and implement a metaheuristics search techniques.

2.3.2 Metaheuristics Search Techniques in SBSE

Before 2001, literature indicates that optimisation problems in Software Engineering can be solved using classical techniques such as linear programming. However, authors Clark et al. (2003) and Harman (2007) have argued that linear programming models are not the best option for solving optimisation problems. This is because there are instances where the problems have multiple characteristics, fitness functions and they could also be multi objective. Clarke et al. (2003) and Harman (2007) have identified three areas where problems could persist when implementing metaheuristics search techniques, but they have also provided potential solution to overcome the problems. A detailed analysis can be viewed in Table 2.1.

Categories	Metaheuristics Search Techniques	
	Limitations	How to overcome limitations
Global Optimum	An algorithms performance can be based on the solution it delivers. In this case, it is highly like that a global optimum may not be achieved.	Search techniques have a threshold limitations and have issues with: 1: search method may not reach threshold 2: how is the threshold set?
Predictability	Algorithms can generate different results at each execution	Metaheuristics are almost all stochastic search techniques so by their very nature they can produce different results each time executed.
Computation Expense	To yield accurate results, large number of candidate solutions must be evaluated before a possible candidate can be selected of good quality.	These techniques may not be fast enough to yield results, but on the other hand these techniques are only applicable when the solution space and problem domain are big enough. Hence in such circumstances, software engineering should have the patience to wait for a good candidate solution.

Table 2.1 Search-Based Techniques (Clarke et al., 2003)

Table 2.1 indicates that metaheuristics search techniques may face certain challenges to solve search problem. However, it has been argued that even though there are limitations, they can be resolved (Clarke et al., 2003) and there is literature in the past which has proved that metaheuristics search techniques have been implemented successfully by the following researchers:

- **McMinn (2004), Harman (2007) and Harman, Sung, Lakhotia, McMinn, & Shin (2010)** have explored the possibility of implementing SBSE in the area of test data generation and additionally, few customised metaheuristics search techniques were developed for their empirical study.

- **Arcuri & Yao (2007) and Afzal, Torkar, & Feldt (2009)** have carried out an in-depth research for generating software tests using SBSE approach. Authors have highlighted the quality, feasibility, usability of search techniques and they have also highlighted few limitations which coincide with the limitations presented in Table 2.1. However, all the authors have concluded their research that metaheuristics search techniques can be successfully implemented in generating software tests.
- **Wong, Aaron, Segall, Lynch, & Mancoridis (2010) and Krogmann, Kuperberg & Reussner (2010)** have successfully implemented metaheuristics search techniques like genetic algorithm to solve problems in the area of reverse engineering. Addition to this, authors have also identified past research and a growing trend for future research.

Based on the examples mentioned above, it is clear that metaheuristics search techniques have been used and the research and implementation increased considerably. Harman & Jones (2001) and Harman (2007) have both specified in their research that to solve problems in software engineering using SBSE, any metaheuristics search techniques (genetic algorithms, simulated annealing, tabu search, hill climbing, and many more) can be used, but having said that Harman (2007) has clearly identified genetic algorithm, simulated annealing and hill climbing are the most common search techniques that have been applied to date in the SBSE community. Furthermore, the metaheuristics search techniques can be categorised in two categories such as local search techniques and evolutionary search techniques and can be viewed in Figure 2.1.

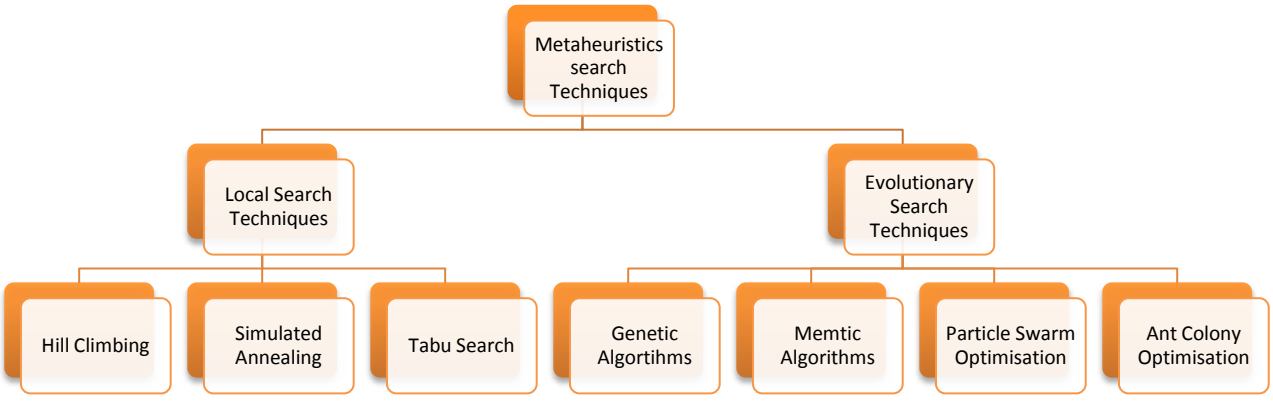

Figure 2.1 Search-Based Techniques (Clarke et al., 2003)

The majority of these metaheuristics search techniques have been discussed in section 2.1.4 and the selection of a set of candidate algorithms is discussed in section 3.2.

2.4 Software Project Planning

The software engineering discipline has been in existence for a long time and since its introduction there have been substantial involvement project management techniques to manage software development / maintenance projects. Over the years, there has been extensive publication in the area of project management / scheduling. Herroelen (2005) has further suggested that there is abundance literature in this area, but for several reasons the theories has not been implemented into the practice. Project management in the discipline of software engineering has always been a problem and there could be several reasons for it. Herroelen (2005) argued that these problems are mainly caused because of the following reasons:

- Poor project management skills
- Poor leadership skills
- Size of the projects
- Lack of resources
- Inappropriate cost estimation and allocation methods

Furthermore, Herroelen (2005) also mentioned that above problems has been identified by literature in the past. To overcome the above mentioned problems, Herroelen has proposed a hierarchical project management model which can be observed in Figure 2.2. In interest of solving the above mentioned problems, author has also suggested the use of heuristics approaches and there has been literature in the past whereby researchers and practitioners have used algorithms to solve project management / scheduling problem.

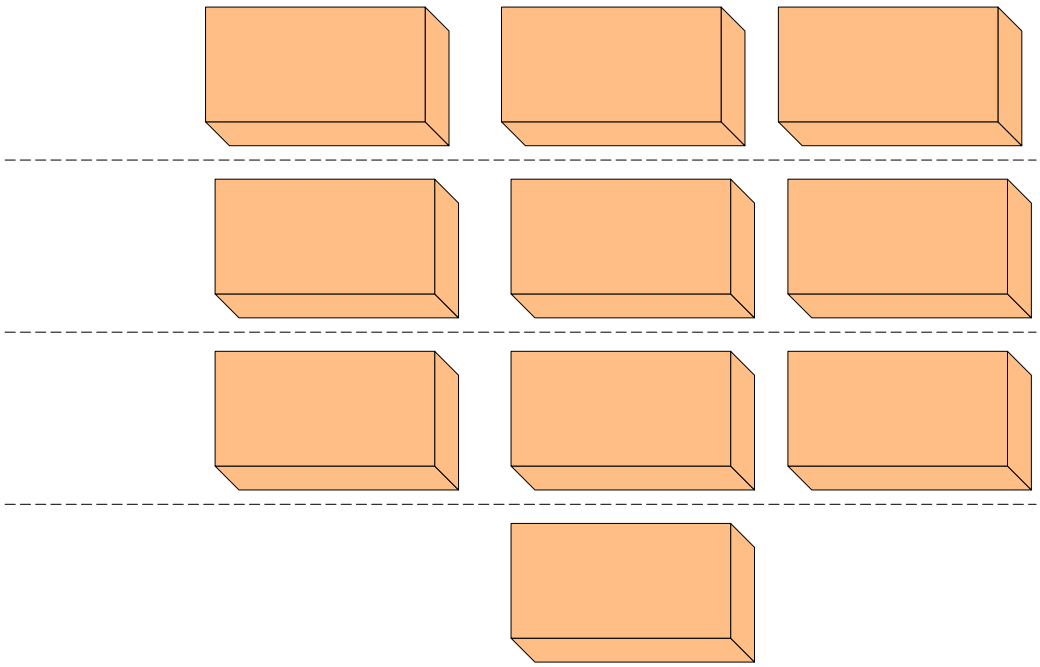

Figure 2.2 Hierarchical Project Management Model (Herroelen, 2005)

The problem with software project planning has been acknowledged by variety for literature in the past. In interest of that, Gueorguiev, Harman and Antoniol (2009), Chang, Jiang, Di, Zhu and Ge (2008) and Killc, Ulusoy and Serifoglu (2008) have also identified similar trends and have tried to solve this problem using search-based software engineering approaches. All the authors mentioned above had one purpose for their research and that was to resolve problems in software project planning, whereby projects are completed on time without being escalated and within budget. Literature also suggests that in the past researchers and practitioners have used following methods to resolve these problems but are not limited to:

- Third party project management software
- Past experience of project planners / managers
- Case based reasoning

2.5 Resource Constrained Project Scheduling Problem (RCPSP)

The main focus of the previous section of the literature review was to highlight general problems associated with software project planning. RCPSP is a subsection of this issue identified with software project planning and literature in this section will make the use of search-based software engineering to resolve those issues. Kolisch & Hartmann (2006) have argued that the problem with software project planning is a high level problem and when the problems are

analysed further, it turns out that in most cases the problems were caused because the resources were scarce. Furthermore, Pinto, Ainbinder & Rabinowitz (2009) have argued that there are three main resources which are usually scarce in a software project and they are as follows:

- Lack of human resource
- Lack of funding
- Lack of available time

The above mentioned categories are similar to Herroelen (2005) whereby he was trying to explain reasons for failure or escalation of a project. Kolisch and Hartmann (2006) have suggested that if a software project is facing RCPSP, then it is very likely that project will either fail or be escalated. This is the main justification stated by Kolisch and Hartmann (2006) in support of their research to solve RCPSP. Authors have argued that literature in the past suggest that researchers and practitioners have used several different methodologies to solve RCPSP, but unfortunately, none of the methodologies have been successfully implemented in the "real-world". Categories can be observed in Figure 2.3.

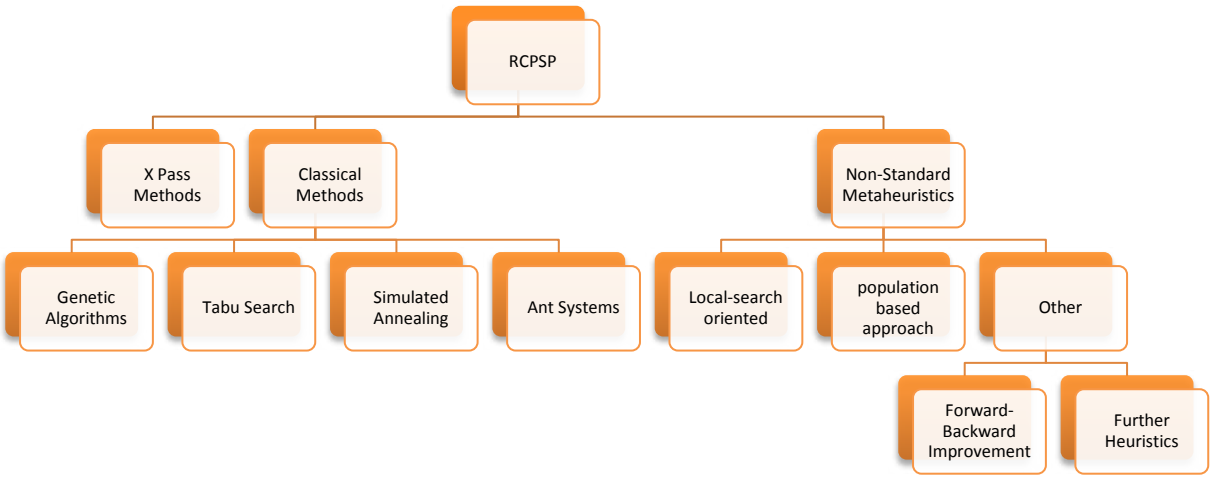

Figure 2.3 Methodologies to Solve Resource Constraint Project Scheduling Problem (Kolisch & Hartmann, 2006)

Comparing Figure 2.1 (Clarke et al., 2003) and Figure 2.3 (Kolisch & Hartmann, 2006), it is clear that there is a different taxonomy of methods to solve search problems. Furthermore, Kolisch & Hartmann (2006) have clearly extended the thoughts of Clarke et al. (2003) by conducting experiments to resolve this problem (i.e. implementing search techniques to solve RCPSP). Having said that, Kolisch & Hartmann (2006) have conducted experiments based on their

assumptions of their own past research conducted in 2001, the results can be biased. On the other hand Gueorguiev, Harman, & Antoniol (2009) have conducted experiments using data from the "real-world" scenario and this could potentially return results which are not biased.

Kolisch & Hartmann (2006) and Gueorguiev, Harman, & Antoniol (2009) all have mainly focused on solving RCPSP using search-based software engineering approaches. The authors have clearly followed the guidelines provided by Harman and Jones (2001) and Clarke et al. (2003) whereby they reformulated the RCPSP as search problem. In the next stage authors have selected a representation of the problem and after that; they have identified their fitness functions to evaluate candidate solutions. Having said that, each research had different criteria for fitness functions and this is mainly because the nature of the experiments was different.

2.6 Cost Estimation in Software Project Planning

In most projects, cost is the most crucial factor which could decide the success rate of project completion. Over the years, there have been abundant studies, whereby researchers and practitioners have tried to reduce the cost of the project (Azar, Harmanani, & Korkmaz, 2009). Many articles also suggest that simply managing the project and resources are not enough; the cost should also be managed efficiently. Furthermore, based on the past literature it is clear that estimating cost in a software project planning has been an ongoing issue and has not been completely resolved as yet. Lokan (2005) has also suggested that since the size of projects are growing considerably, it is necessary that there should be some automations / models to solve the cost estimation problem. To that end, Lokan (2005) has suggested the use of search-based software engineering concept to solve this problem. The concepts discussed by Lokan (2005) are extended by Li, Xie, & Goh (2009), whereby they have argued that if the cost of completing a software project is too high, then it could be possible that the software project was not successfully. On the other hand Li, Xie, & Goh (2009) have argued that in such cases there could be off-the-shelf products which could meet the requirements of the software project. In the past, studies have developed several methodologies for solving the cost estimation problem, classification of these methodologies can be observed in Figure 2.4.

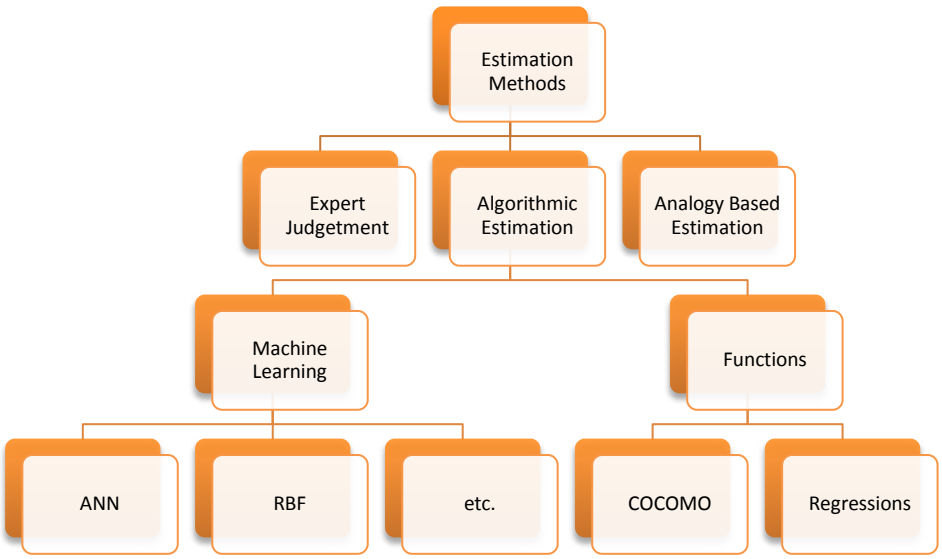

Figure 2.4 Software Cost Estimation Methodologies (Li, Xie, & Goh, 2007)

Figure 2.4, suggests that research has generated several methods to solve cost estimation problem. Lokan (2005) on the other hand, has presented his own view of estimations based on analogy. This can be observed in Figure 2.5.

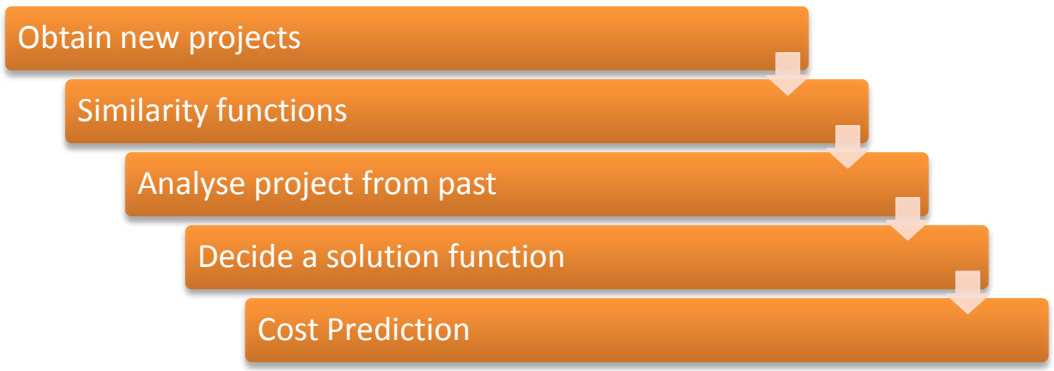

Figure 2.5 Estimation Method Based on Analogy (Lokan, 2005)

Lokan (2005), Li, Xie, & Goh (2007) and Li, Xie, & Goh (2009), all have tried to solve cost estimation problem using the concept of search-based software engineering. The authors have clearly followed the guidelines provided by Harman and Jones (2001) and Clarke et al. (2003) whereby they reformulated the cost estimation as an optimisation problem. In the next stage authors have selected a representation of the problem and identified their fitness functions to evaluate candidate solutions.

3 Research Design

Literature review presented in chapter two has identified various research opportunities. On that basis, this chapter represents the research methodology and the research questions. Thereon, based on research questions this chapter will present an experimental design that is adopted for the purpose of this work.

3.1 Research Methodology

Research in search based software engineering has been extensive since 2001, but they were not categorised under search based software engineering discipline. According to Harman (2007) this discipline has not been researched to its full potential hence there is enough room for growth and improvement. The literature review has identified a trend of research methodologies used to conduct research in the discipline of search-based software engineering. Most of the researchers mentioned in the literature review have used the constructivist approach to conduct their research. Furthermore, each of them has also made the use of experimental research methods where they have conducted experiments to justify their assumptions.

In the literature review, most researchers have applied the combination of constructivist and experimental research methodologies. This research will utilise the framework presented by Nunamaker, Chen, & Purdin (1991) and Hevner, March, Parl, & Ram (2004). The research method presented by Nunamaker, Chen, & Purdin (1991) is the system development research methodology. The research framework presented by Hevner, March, Parl, & Ram (2004) is design-science research guidelines. Both have both argued that system development research methodology and design-science research guidelines have been used as a backbone for conducting research in the area of software engineering.

3.1.1 How does this research fit into System Development Research Methodology?

According to Nunamaker, Chen, & Purdin (1991) any research in the field of engineering or applied science having the characteristics of developmental and formulative behaviour can potentially use the system development research methodology. For this research, one of the main objectives would be to develop genetic algorithm, simulated annealing and tabu search metaheuristics search techniques to solve resource allocation and cost estimation problem. Since this research is constructive in nature, it is an indication that system development research methodology may be suitable.

3.1.2 How does this research fit into Design-Science Research Guidelines?

The concept of design-science has been an integral part of software engineering (Hevner, March, Park & Ram, 2004). Design-science can contribute a model or a pattern to solve software engineering problem. The main objective of design-science would be to find new knowledge, products, ideas, and innovations. Literature suggests that the findings generated from the research are built on existing theories that have already been applied and well researched. Following are the reason as to how this research fit into design-science research guidelines:

- **Discipline**

  The design-science research guideline is mainly utilised in the discipline of software engineering and based on the literature review, search based software engineering is a subsection of software engineering discipline. Hence, it would be the preferred choice of research guideline.

- **Research Implementation**

  There has been substantial research in the areas of search based software engineering, software project planning and resource allocation and cost estimation in software project planning. Furthermore, there is good amount of literature in which researchers and practitioners have tried to solve resource allocation and cost estimation problem using variety of metaheuristics search techniques.

- **Past Research**

  Based on the literature review, it is apparent that since 2001 there has been substantial amount work put in the discipline of search based software engineering. Additionally, findings from the past research in search based software engineering have not only been applied and tested but also the findings have been extended by other researchers.

Hence from the reasons mentioned above it is an indication that this research fits well into the design-science research guideline presented by Hevner, March, Parl, & Ram (2004).

3.1.3 Research Questions

To validate the research topic, the following questions were explored:

3.1.3.1 Question 1

Can resource allocation and cost estimation be effectively formulated as a search problem that is tractable to be solved using metaheuristic search algorithms?

3.1.3.2 Question 2

Which metaheuristic search algorithms demonstrate the most desirable performance when solving these problems?

3.1.4 Research Guidelines

For this research, the guidelines presented by Holz, Applin, Haberman, Joyce, Purchase, & Reed (2006) will be utilised. The main objective of establishing guidelines is to assist in the high level planning of the research. When conducting research using approach as presented by Holz, Applin, Haberman, Joyce, Purchase, & Reed (2006) in the area of computer science the researchers has explored the following four critical points:

3.1.4.1 What do you expect to achieve out of this research?

- To answer this question, the past and the current research will be identified. For the purpose of this paper, a detailed research has been carried out in the area of search based software engineering and how it can be used to solve resource allocation and cost estimation problem in software project planning.
- Identify areas where there is a potential for growth in future. Since the introduction of search-based software engineering in 2001, research has sprouted in several different areas. Hence this research will identify how it has affected the area of software project planning and what affect would it have in future.
- In this research, the aim is to create knowledge using constructivist approach, which means it will identify a different way to solve existing problem (i.e. resource allocation and cost estimation problem in software project planning).

3.1.4.2 Where is the data coming from?

- To collect data a tool is developed. This tool will demonstrate the ability to handle multiple search techniques such as genetic algorithm, simulated annealing and tabu search. In addition to this, all the data collected will be stored in a database which can be accessed at later stages for further analysis. The main objective of this tool will be to

solve continuous and discrete numerical test functions and also will have the ability to solve uni-objective and multi-objective problems. The data required for conducting experiments will be conceptual in nature.

3.1.4.3 What are you going to do with the data?

- Once all the experiments are completed, all the collected data will be used to evaluate the performance of the implemented metaheuristics search techniques. The collected data will also help identify any particular themes or patterns in the behaviour of the algorithms.

3.1.4.4 Has the research achieved the goal?

- The evaluated results will be compared against the goals that were set out to achieve at the beginning of this research.
- Recognise limitations of this research.
- Conclude the research by identifying potential improvements and a recognisable path for other researchers for future.

From above mentioned questions for this research, it is clear that this approach would suit the research. It is absolutely necessary that this research will need to construct a software / tool which will include the implementation of metaheuristics search techniques like genetic algorithm, simulated annealing and tabu search. These search techniques will then be used to solve continuous and discrete problems and will also resource allocation and cost estimation problem in software project planning. This approach is illustrated in Figure 3.1.

Figure 3.1 Research Methodology (Holz, Applin, Haberman, Joyce, Purchase, & Reed, 2006)

3.1.5 System Development Research Methodology and Design Science

To conduct this research in an effective and efficient manner, a research guideline and methodology will be used as a reference. This will enable this research to answer the questions in robust and rigorous way. The guidelines used for this research is presented in section 3.1.4 and as for the research methodology, the nature of this research (exploratory and constructivist) coincides with the research methodology presented by Nunamaker & Chen (1990) and is illustrated in Figure 3.2.

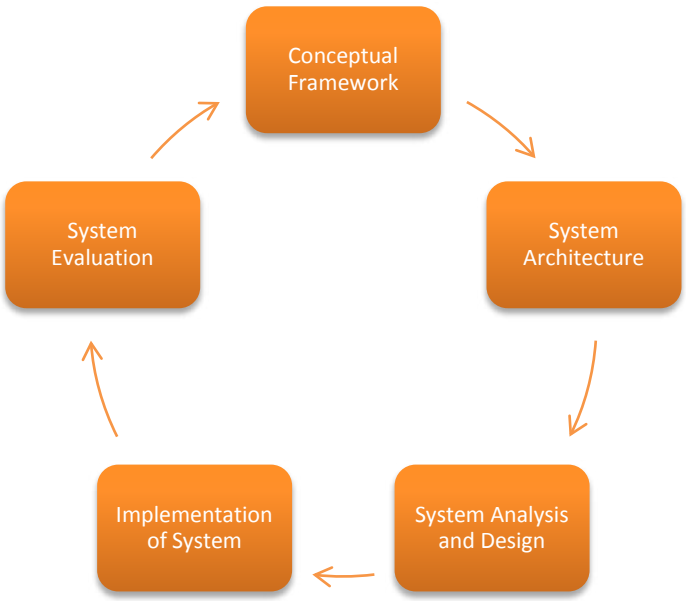

Figure 3.2 System Development Framework (Nunamaker & Chen, 1990)

The framework presented above is also further extended by Holz, Applin, Haberman, Joyce, Purchase, & Reed (2006) and Nunamaker, Chen, & Purdin (1991). Hence for the purpose of this research, the foundation of the implementation is established using the framework presented above. The description of how each element of framework is used to develop the implementation is presented below:

3.1.5.1 Prepare a Conceptual Framework

Establishing a conceptual framework is nothing but laying down a foundation for developing an application to solve the problems stated at the beginning of this research. If the conceptual framework is developed without fully understanding the problem space, the summary of data collected could be inconclusive and because of which end results of the research would not be able to answer the research questions. Conceptual frameworks are usually built on research questions and an in-depth literature review. Based on extensive literature review and research questions, it is clear that there is a need for further research in this area. Once the application has been built using the conceptual framework, the performance of the application can be observed and additionally it would also give a detailed insight on the research problem.

3.1.5.2 Develop a System Architecture

A good conceptualisation of the problem space and framework can lead to good system architecture. Hence, good system architecture will allow its components to be placed in appropriate places and have its own unique functionality. Based on the research definition and research methodology (constructivist approach) various components will be designed and

developed. In addition to this, Nunamaker & Chen (1990) have specified that before defining the architecture, the scope and functionality of the developed system must be established. Before the architecture was implemented, following aspects of the research were considered:

One of the research objectives is to examine if the metaheuristics search techniques like genetic algorithm, simulated annealing and tabu search can be successfully implemented in the area of project management to solve problems like resource allocation, cost estimation and scheduling. Hence there is a need to develop these metaheuristics search techniques within the project management component. In addition to this, each search techniques should be implemented to provide a basis for comparison and evaluation against other implemented search techniques. The framework of the system should also have the capability of implementing additional search techniques in the area of project management and by doing so; this implementation can be extended for future research.

An examination of effectiveness of applying several metaheuristics search techniques to solve problem in the area of project management is another important element of this research. Therefore each implemented search technique should go through an evaluation process. This process should be embedded as a part of the framework. Ideally the evaluation process will include measuring performance of each search techniques against various continuous and discrete numerical test functions. The developed framework supports the flexible inclusion and exclusion of numerical test functions into the evaluation process. This process is developed in such a way that the end users can configure the inclusion or exclusion of numerical test functions depending on the metaheuristics search techniques.

3.1.5.3 Analyse and design the system

Based on the literature review and identification of limitation of prior research to solve resource allocation, cost estimation, and project scheduling problems in the area of project management has led to this implementation. The system will be designed into three main steps. The first step of the design is to establish metaheuristics search techniques like genetic algorithm, simulated annealing and tabu search. This step of the design will act as foundation of the system and will be used in during each evaluation and / or data analysis phases. The second step is to design the "Evaluation Engine" which will determine the viability and feasibility of the implemented search techniques and additionally will also attempt to solve resource allocation, cost estimation and project scheduling problem based on case study. The last part of the system design is to prepare an engine which collects the data from all the evaluations which will be used for answering the

research questions and additionally will help conclude this research with a direction of future research. The entire system design is illustrated in Figure 3.3.

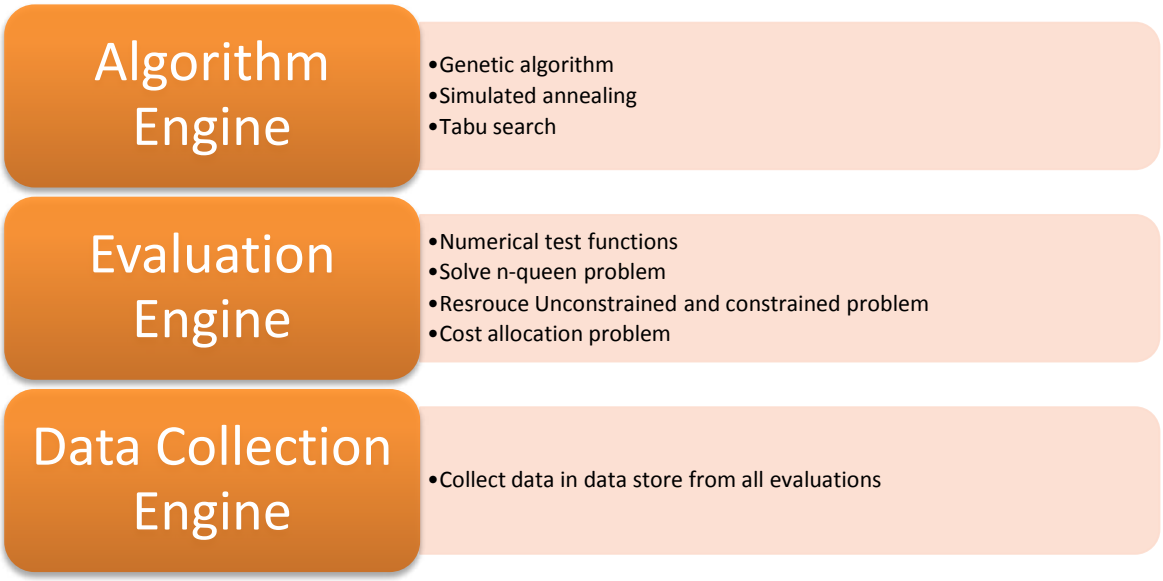

Figure 3.3 Overview of System Design

3.1.5.4 Build the System

In order to develop the system effectively and efficiently the system design mentioned earlier will be used as a blue print. This element of the framework will involve the development of each metaheuristics search techniques mentioned earlier and it would also require the implementation of the following numerical test functions:

- Axis Parallel Hyper Ellipsoid fitness function
- Griewangk fitness function
- Rastrigin fitness function
- Rosenbrock's Banana fitness function
- Schwefel fitness function
- Axis parallel hyper-ellipsoid fitness function
- Griewangk fitness function
- Himmelblaus fitness function
- Bump fitness function with dimensions of 2, 5, 10, 15, 20 and 50.

The next stage is to develop the component which will allow the search techniques to solve n-queens, resource allocation, cost estimation and project scheduling problem. The last step in building the system is to implement a data collection engine which collects data from all the

evaluations and store in a structurally designed data store. Finally all the developed components of the system should support flexible configurations of all the parameters to enable the examination of various scenarios without further development of the system.

3.1.5.5 Evaluation

The research methodology and the guidelines presented in Section 3.1 require the observation and evaluations of the implementation. To facilitate this evaluation process of search based software engineering an experimental methodology is applied to asses both the usability and the performance (Collis & Hussey, 2009). The experimentation methodology will enable this research to confirm a theory as presented in literature review, highlight the relevance of the experiment against research questions and will also validate the implementation. Ideally before any research experiments are carried out, an expected solution to the experiment must be defined and experiments should be flexible enough to be repeated several times if required. The final results are generated once all the experiments are completed and all the data is collected. These results are then compared against expected solution. Above all, the design of the experiments should be aligned with the main research objectives. This will help answer the research questions set out at the beginning of the research.

3.2 Metaheuristics Selection & Implementation

Using the research objectives, guidelines and methodology, this section will describe in detail about the selection, design and implementation of metaheuristics search techniques. As discussed in Section 3.1, the development of these metaheuristics search techniques will go through an iterative process. At each stage of development, the implementation will be validated against literature review, research objectives and the research methodology. Hence each the development of each search techniques will go through the steps of designing of conceptual framework, identify its place in the system architecture, design of algorithm, actual development of the search technique and lastly the evaluation process. By doing so, there is a high probability of achieving results. In this research, metaheuristics search techniques like genetic algorithm, simulated annealing, and tabu search will be constructed. These search techniques are developed on a standard Dell laptop with Intel Core i5 with 4GB RAM and Windows XP as its operating system. All the development work is carried out in Microsoft environment. The development platform is Visual Studio 2010, the programming language is C# and the underlying development framework is .NET Framework 4.0. The architecture, modules and expected behaviour of each search techniques are as follows:

3.2.1 Algorithm Selection

As part of this research a range of traditional and metaheuristic optimisation algorithms were considered. Due to time constraints, only a very small number of the algorithms were selected for implementation and evaluation. These were Genetic Algorithms, Simulated Annealing and Tabu Search.

These were considered to be "reasonably representative" of a range of algorithms that include population-based approaches, trajectory approaches and memory based approaches. The algorithms are also considered to be some of the most popular by the SBSE community (Harman, 2007) and are therefore a good starting point for evaluating algorithms on a particular problem domain.

One of the limitations of this study is the small range of algorithms investigated along with the relatively naive implementations. However, such a limitation does not detract from the quality of the research. If the algorithms are not suitable for solving the RCPSP then further algorithms will be investigated.

3.2.2 Genetic Algorithm

One of the main aims of this research is to identify the feasibility of genetic algorithm in the area of project management. The GA implemented in this research is a simple GA based on the work of Goldberg (1989) consisting of a single population of individuals, roulette wheel selection, single-point crossover and bit-string mutation. Whilst more complex variations on the Genetic Algorithm exist, the use of a simple GA is intended to provide a baseline for future work to allow any performance gain for more complex implementations to be evaluated. Hence, to address this genetic algorithm has been implemented using various classes. The main classes are as follows:

3.2.2.1 clsChromosome

This is the most important part of genetic algorithm implementation. This class will facilitate the crossover and mutation operations. This class has a few publicly available properties like fitness function value, numbers of genes, length and the mutation rate. The idea is that clsGeneticAlgorithm will hold a collection (array) of clsChromosome instances and each chromosome object will be evaluated against the fitness function which is implemented as an interface in clsGeneticAlgorithm. After the evaluation against fitness functions the outcome is recorded in variable _Fitness. The class diagram of this implementation is illustrated in Figure 3.4.

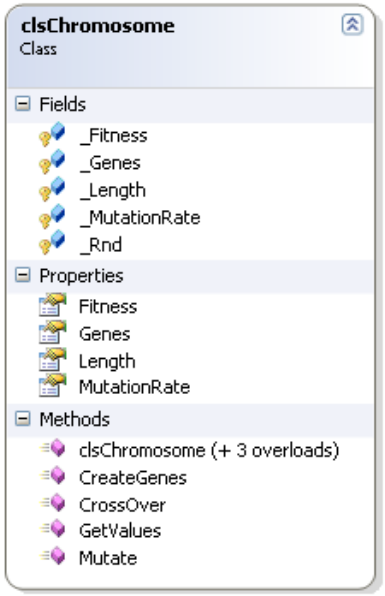

Figure 3.4 UML Diagram of Chromosome Class

3.2.2.2 clsChromosomeCollection

To get some modularisation and simplicity, the collection (array) of clsChromosome is constructed in separate class and illustrated in Figure 3.5. The main objective of this class is to act as a collection of object for clsChromosome.

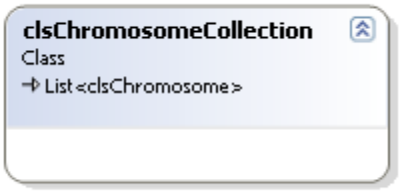

Figure 3.5 UML Diagram of Chromosome Collection Class

3.2.2.3 clsChromosomeComparer

Once the fitness values are calculated for all the objects in clsChromosomeCollection, each individual item in clsChromosomeCollection is compared by its fitness. Figure 3.6 will illustrate the implementation.

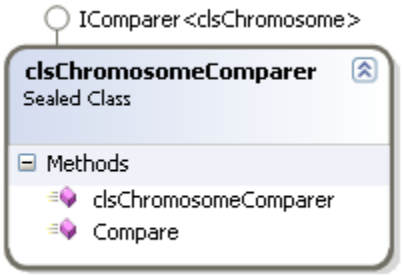

Figure 3.6 UML Diagram of Chromosome Comparer Class

3.2.2.4 clsGeneticAlgorithm

Like clsChromosome, this class is also very crucial for the development of this algorithm. The main objective of this class is to execute the genetic algorithm based on the supplied parameters and function. This class has a number of publicly available properties which are mainly about the configurations of the algorithm. The values are passed on to clsChromosome for mutation or crossover. Additionally, this is the class which calculates the fitness value for each chromosome and sorts it in an appropriate order. The functions engine is designed as an interface hence the type of function can be configurable.

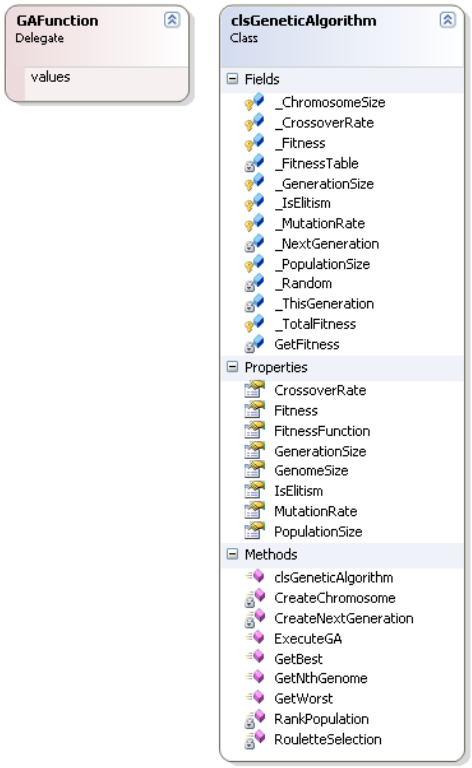

Figure 3.7 UML Diagram of Genetic Algorithm Class

In addition to the implementation of genetic algorithm, there is a process as to how this algorithm is executed. Hence this section will describe the basic schema of genetic algorithm. First step will create an initial population i.e., the first generation of the genetic process. The members of this initial population are known as chromosome. The next step will determine the objective function value for each chromosome in the population and after that; the algorithm applies a selection parameters to randomly partition the generation into pairs of individual. For each resulting pair of parent, the crossover operator is applied and subsequently the mutation operator is applied to newly produced offspring. The implementation of the crossover and mutation operator is illustrated in Figure 3.8 and Figure 3.9 respectively. The flow of genetic algorithm implemented in this research is similar to evaluation carried out by Pinto, Ainbinder & Rabinowitx (2009) where they have also used genetic algorithm to tackle the same problem.

```
/// <summary>
/// Method for creating a cross over.
/// This method determines which chromosomes are carried forward to the next generation.
/// </summary>
public void CrossOver(ref clsChromosome ChromosomeVer2, out clsChromosome ChromosomeChild1, out clsChromosome ChromosomeChild2)
{
    int intPosition = (int)(_Rnd.NextDouble() * (Double)_Length);
    ChromosomeChild1 = new clsChromosome(_Length, false);
    ChromosomeChild2 = new clsChromosome(_Length, false);
    for (int intCounter = 0; intCounter < _Length; intCounter++)
    {
        if (intCounter < intPosition)
        {
            ChromosomeChild1._Genes[intCounter] = _Genes[intCounter];
            ChromosomeChild2._Genes[intCounter] = ChromosomeVer2._Genes[intCounter];
        }
        else
        {
            ChromosomeChild1._Genes[intCounter] = ChromosomeVer2._Genes[intCounter];
            ChromosomeChild2._Genes[intCounter] = _Genes[intCounter];
        }
    }
}
```

Figure 3.8 Implementation of Crossover Operator

```
/// <summary>
/// Method for mutating the current genes for the next generation by using built-in random function of .net framework.
/// </summary>
public void Mutate()
{
    for (int intCounter = 0; intCounter < _Length; intCounter++)
    {
        if (_Rnd.NextDouble() < _MutationRate)
        {
            _Genes[intCounter] = (float)(_Genes[intCounter] + _Rnd.NextDouble()) / 2.0;
        }
    }
}
```

Figure 3.9 Implementation of Mutation Operator

Overall it is fairly complex to visualise the entire implementation of genetic algorithm, hence the pseudo code is presented below:

Begin Genetic Algorithm

```
        Generate the initial generation of chromosome of size (x);
        Evaluate fitness of each chromosomes based on the supplied function;
        g = 1;
        While g < |G| DO
                g += 1;
                Select individuals for reproduction;
                Crossover individuals;
                Mutate offspring;
                Revaluate fitness of each chromosome;
        End While
End
```

Once the algorithm is implemented, the next stage of development is to fine tune the genetic algorithm using the parameters. The ideal parameters for genetic algorithm are mutation rate, crossover rate, population size, generation size, chromosome size, and the iterations. After fine tuning the algorithms the final set of parameters are presented in Table 3.1.

Algorithm	Parameter Name	Parameter Value
Genetic algorithm	Crossover Rate	0.8%
	Mutation Rate	0.06%
	Population Size	300
	Chromosome Size	2

Table 3.1 Parameters for Genetic Algorithm

3.2.3 Simulated Annealing

Like genetic algorithm, simulated annealing was also implemented using similar tools and techniques to tackle the same problem. In this implementation of simulated annealing, three classes are developed. This algorithm will also utilise the interface from the evaluation engine.

3.2.3.1 clsSimulatedAnnealing

This is the main class in implementation of simulated annealing which handles the process of finding an optimum solution for a given function. The class includes a few publicly available properties which are used as configuration for the algorithm. The method ExecuteSA() is executed to find the best possible solution for the given parameters and the fitness function. The SAFunction class implements an interface from the evaluation engine which allows the selection of fitness when executing this algorithm.

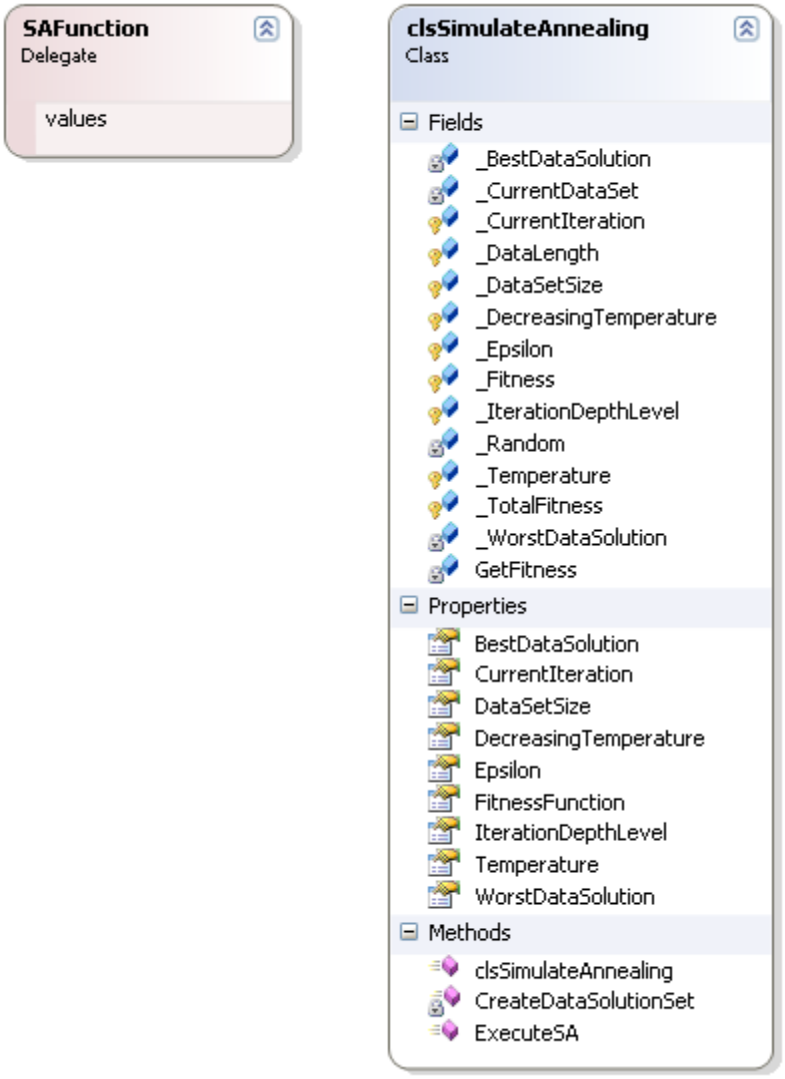

Figure 3.10 UML Diagram of Simulated Annealing Class

3.2.3.2 clsSimulatedAnnealingData

To make this implementation simpler, the data is stored in this class. This is a helper class and can be embedded in clsSimulatedAnnealing. This class is illustrated in Figure 3.11.

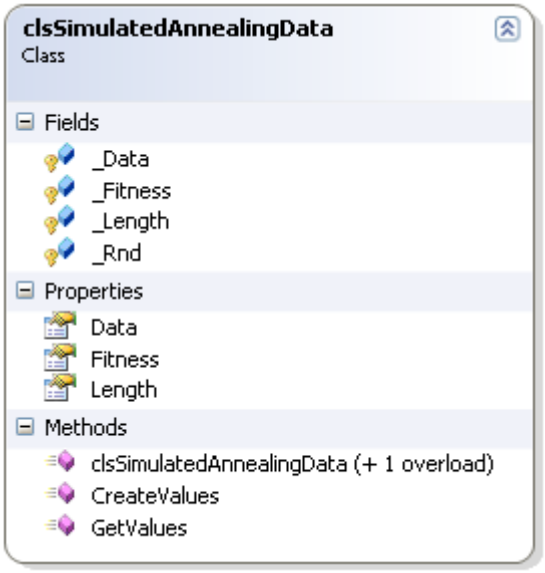

Figure 3.11 UML Diagram for Simulated Annealing Data Class

3.2.3.3 clsSimulatedAnnealingDataCollection

The collection (array) of data objects is constructed in separate class and illustrated in Figure 3.12.

Figure 3.12 UML Diagram for Simulated Annealing Data Collection Class

Simulated annealing is a popular metaheuristics search techniques and its main objective is to find an acceptable solution in a fixed amount of time (Martinjak & Golub, 2007). For this implementation, the first step of the algorithm is to select a “best” solution randomly. The next stage is to start the iteration process based on the supplied parameters. In all the iteration a new candidate for the solution is selected randomly and it is evaluated against the current best solutions. Depending on the outcome of the evaluation process, a new best candidate can be selected. This evaluation process is heavily dependent on the supplied fitness function. The iteration is repeated until the terminating conditions are met and the best solution at that time is returned. Additionally at each stage the temperature is reduced by the cooling ratio which can be configured. So the probability of replacing the current best candidate solution with a better candidate solution is high when the temperature is high. As the temperature reduces, it is an

indication that the algorithm might be closer to find its best candidate solution. The above mentioned flow of algorithm has also been flowed by Martinjak & Golub (2007) but in this instance they have used this metaheuristics search techniques to solve the n-queen problem. Even though the problems are different the underlying concepts of simulated annealing still remains the same. The pseudo code for this algorithm is presented below:

```
Begin Simulated Annealing
        Set T to T0;
        Select initial solution X;
        Px = Evaluate(X);
        Xbest = X;
        Pbest = Px;
        While c < |Cmax| DO
                Y = Select a neighbour on random from solution set;
                If (Evaluate(Y)>Px) then
                        X = Y;
                        Px = Evaluate(Y);
                        If (Px > Pbest) then
                                Xbest = X;
                                Pbest = Px;
                else
                        r = Random(1);
                        if r < exp(e(Evaluate(Y)-Evalute(X))/T) then
                                X = Y;
                                Px = Evalute(Y);
                Set cooling ration to T
        End While
        Return Xbest;
End
```

Like genetic algorithm, simulated annealing will also go through the process of fine tuning the parameters. The ideal parameters for simulated annealing are starting temperature, decreasing temperature, epsilon and the Markov Chain length. After fine tuning the algorithm the final set of parameters are presented in Table 3.2.

Algorithm	Parameter Name	Parameter Value
Simulated Annealing	Starting Temperature	50000
	Decreasing Temperature	0.85
	Epsilon	0.01
	Markov Chain Length	10

Table 3.2 Standard Parameters for Simulated Annealing

3.2.4 Tabu Search

The implementation of tabu search has used the same tools and technologies that were used by genetic algorithm and simulated annealing. For this implementation, three main classes are designed and developed. This implementation also uses the evaluation engine to determine its fitness function. The classes for tabu search are as follows:

3.2.4.1 clsTabuSearch

This is the main class in implementation of tabu search which handles the process of finding an optimum solution for a given function. The class includes a few publicly available properties which are used as configuration for the algorithm. The method ExecuteTS() is executed to find the best possible solution for given parameters and the function. The TSFunction class implements an interface from the evaluation engine which allows the selection of fitness when executing this algorithm.

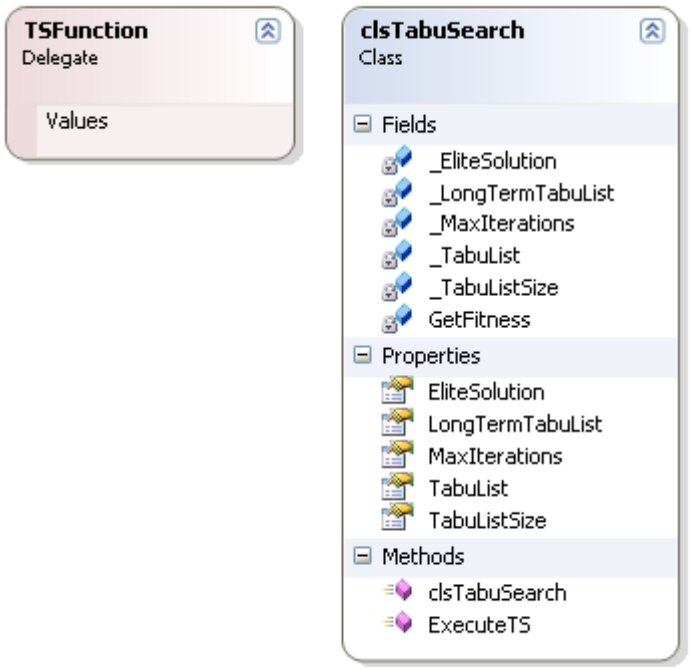

Figure 3.13 UML Diagram for Tabu Search Class

3.2.4.2 clsTabuSearchData

To make this implementation simpler, the data is stored in this class. This is nothing but a helper class and can be embedded in clsTabuSearchData. This class is illustrated in Figure 314.

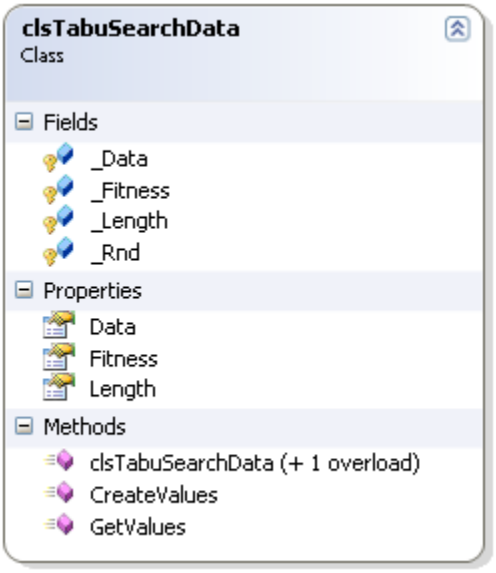

Figure 3.14 UML Diagram for Tabu Search Data Class

3.2.4.3 clsTabuSearchDataCollection

The collection (array) of data objects is constructed in separate class and illustrated in Figure 3.15.

Figure 3.15 UML Diagram for Tabu Search Data Collection Class

Like genetic algorithm and simulated annealing, tabu search is also a popular algorithm when tackling problems in the area of project management. The main objective of this search technique is to solve discrete combinatorial optimisation problem. The implementation in this section mainly focuses on replacing the current solution (X) with another one (Y) with the maximum fitness function value in the whole neighbourhood of X. At times when X is compared with Y, it could be possible that they are the same. In this case they are temporarily stored in a list called “tabu list”. A similar algorithm was also implemented by Matinjak & Golub, 2007 and Kanmani & Maragathavalli, 2010. The pseudo code for this algorithm is presented below.

Begin Tabu Search
    Select initial solution X;
    Xbest = X;

```
        Pbest = Evaluate(X);
        While c < |Cmax| DO
                N = N(X) \ TabuList;
                Find Y = N such that Evaluation(Y) is maximum;
                TabuList[c] = Change(Y,X);
                X = Y
                if (Evaluate(X) > Pbest) then
                        Xbest = X;
                        Pbest = Evaluate(X);
        End While
        Return Xbest;
End
```

To ensure the development of this algorithm has been accurate, it will go through a series of fine tuning exercises. In these exercises, the algorithm is executed several times and during each execution different parameters are supplied to see the behaviour of the algorithm. After several tests, final set of parameters for the tabu search algorithm are presented in Table 3.3.

Algorithm	Parameter Name	Parameter Value
Tabu Search	Tabu List Size	10
	Neighbourhood Size	300

Table 3.3 Standard set of parameters for Tabu Search

3.3 Experimental Design

As established in section 3.1, the nature of this research is constructivist and experimental in nature. Hence Nunamaker, Chen & Purdin (1991) have identified that the design of the experiments is very crucial part of the research. An effective design will help address the research questions. The main aim of this section is to describe the technical design and implementation of all the experiments in this research. Additionally, in this empirical study, the principal goal is to solve problems like resource allocation, cost estimation and project scheduling in the area of project management. Before these research problems are tackled, each metaheuristics search techniques will be evaluated against various numerical test functions. To implement an effective experimental design, a class library has been developed using the same tools and technologies used for developing metaheuristic search techniques. The library was developed in such a way that additional functions / problems could be easily added and it would be available for all the metaheuristics search techniques without modifying them. The architecture of the evaluation engine is illustrated in Figure 3.16.

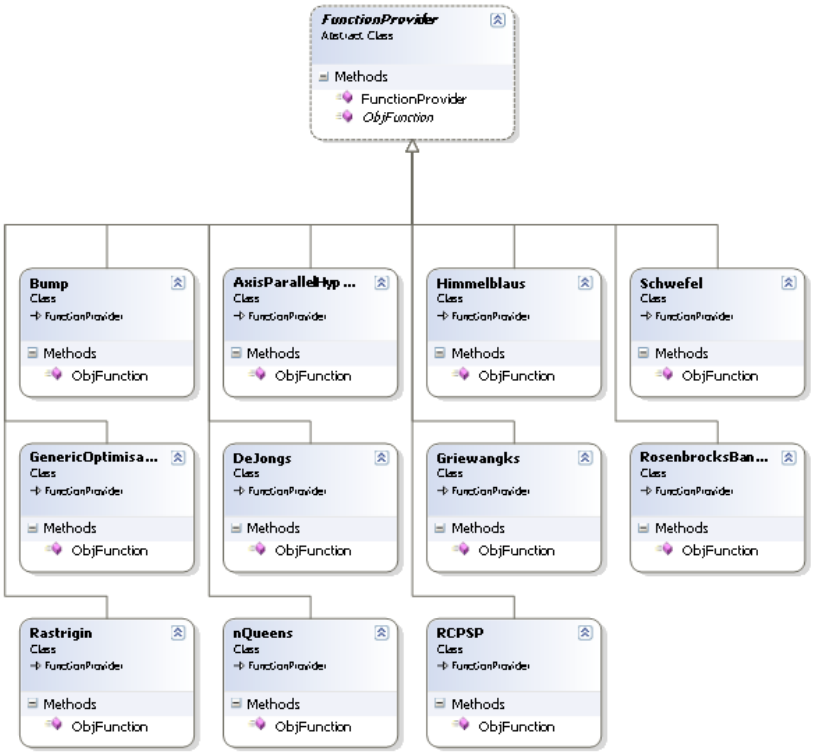

Figure 3.16 UML Diagram for Evaluation Engine

To address the research questions in structured manner, the empirical study will be carried out in four phases and they are as follows:

3.3.1 Phase 1: Metaheuristics Verification

According to the authors Elbeltagi, Hegazy & Grierson (2005), it is a common approach to measure performance of algorithms using numerical test functions. This is mainly because each numerical test function will have its own unique characteristics which can shape the outcome of the algorithm. To judge the performance of each algorithm, at the end of each evaluation following questions will be answered:

- Did the algorithm find an optimum solution?
- How many evaluations does it take to find optimum solution?
- What is average variance of performance between each data point for each algorithm (presented in Table 4.2 to 4.8 and in Table 4.10 to 4.15)?

This experiment will be carried out in six times for each metaheuristics search technique against following numerical test functions:

3.3.1.1 Rastrigin Fitness Function

This will be the first experiment where metaheurisitcs search techniques will be evaluated against this fitness function. The analysis and data collected from this evaluation is presented in section 4.1.1 and the equation used to carry out this evaluation is presented in Equation 3.1. The global minimum for this function is established at 0 (Buche, Schraudolph & Koumoutsakos, 2005) and the solution space (between -5.12 and 5.12) is presented in Figure 3.1. Furthermore, this function is difficult to solve because the solution space is large and there are large number of local minima.

$$f(x) = An + \sum_{i=1}^{n} [x_i^2 - 10\,A\cos(2\pi x_i)]$$

Equation 3.1 Rastrigin Fitness Function

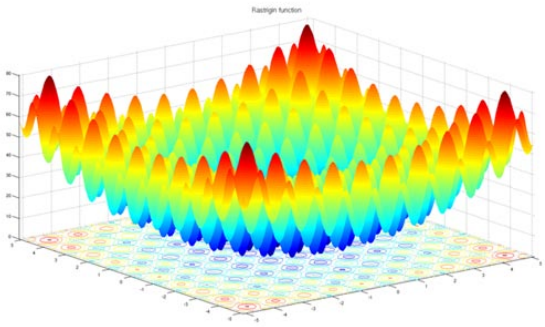

Figure 3.17 Solution Space for Rastrigin Numerical Test Function (Buche, Schraudolph & Koumoutsakos, 2005)

3.3.1.2 Rosenbrock's Fitness Function

Like Rastrigin fitness function, this function also has a global minimum of 0 (Buche, Schraudolph & Koumoutsakos, 2005). The search space for this function is restricted between -10 and 10. The solution space for this function is presented in Figure 3.18. The equation used to generate data is presented in Equation 3.2. Additionally, the analysis and data collected from this evaluation is presented in Section 4.1.2.

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Equation 3.2 Rosenbrock's Banana Fitness Function

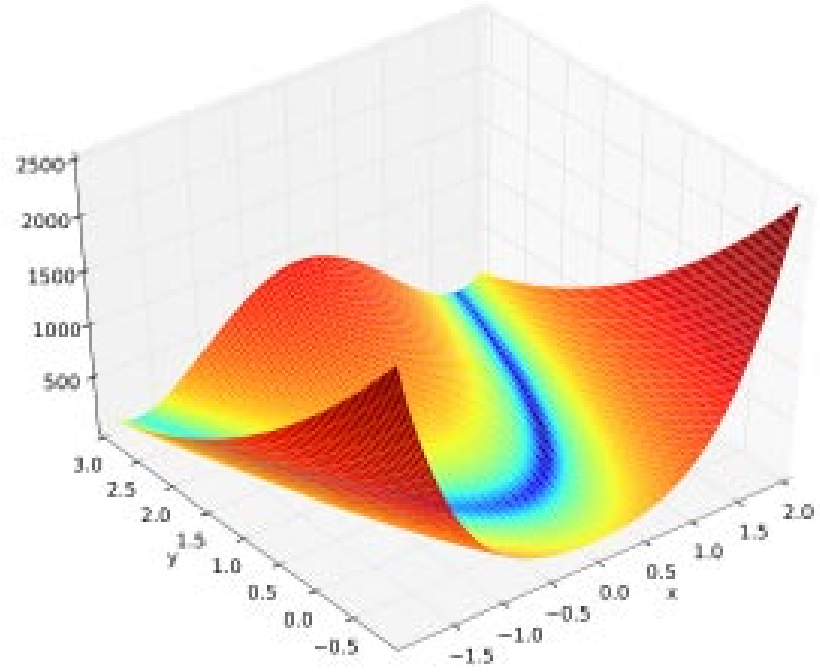

Figure 3.18 Solution Space for Rosenbrock's Banana Fitness Function (Buche, Schraudolph & Koumoutsakos, 2005)

3.3.1.3 Schwefel Fitness Function

Unlike the previous two functions, the minimum value for this function is established at -512 and the solution space is presented in Figure 3.19 (Barchiesi, 2009). The analysis and data collected from this evaluation is presented in Section 4.1.3. This evaluation was carried using the equation presented in Equation 3.3.

$$f(x) = \sum_{i=1}^{n} - x_i \, . \, sin\left(\sqrt{|x_i|}\right)$$

Equation 3.3 Schwefel Fitness Function (Barchiesi, 2009)

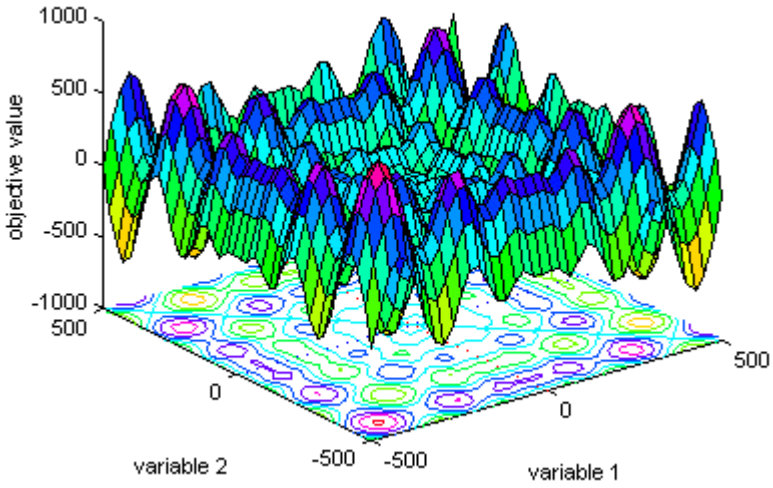

Figure 3.19 Solution Space for Schwefel Fitness Function (Barchiesi, 2009)

3.3.1.4 Axis Parallel Hyper-Ellipsoid Fitness Function

The search space for this fitness function is between -5.12 and 5.12 and the global minimum value is established at 0. The solution space is presented in Figure 3.20 and the equation is presented in Equation 3.4. Based on the equation, the analysis and data collected is presented in Section 4.1.4.

$$f(x) = \sum_{i=1}^{n} i.x_i^2$$

Equation 3.4 Axis Parallel Hyper-Ellipsoid Fitness Function

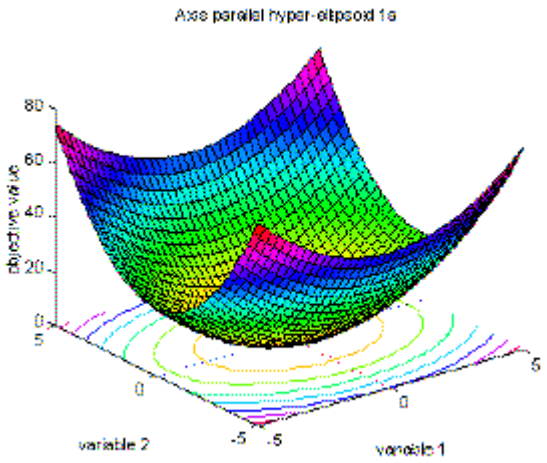

Figure 3.20 Solution Space for Axis Parallel Hyper-Ellipsoid Fitness Function (Peram, Veeramachaneni & Mohan, 2003)

3.3.1.5 Griewangk Fitness Function

In this fitness function, the range of search space is between -600 and 600 and the global minimum value is 0 (Peram, Veeramachaneni & Mohan, 2003). The equation and solution space is presented below in Equation 3.5 and Figure 3.21 respectively. Using the equation mentioned below data is collected and presented in Section 4.1.5.

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Equation 3.5 Griewangk Fitness Function

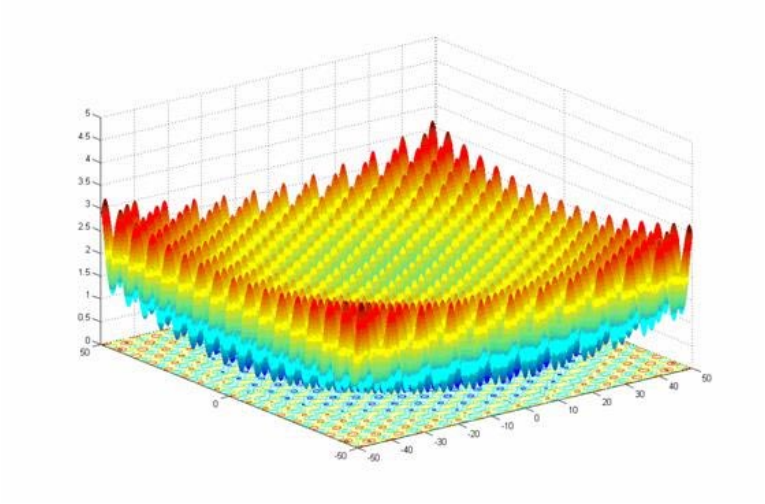

Figure 3.21 Solution Space for Griewangk Fitness Function (Peram, Veeramachaneni & Mohan, 2003)

3.3.1.6 Himmelblaus Fitness Function

This is the last experiment in the first phase of experiments. This function has four local maxima (Takahashi & Kobayashi, 2001):

- 3.584428 and -1.848126 with a local minimum of 0
- -3.779310 and -3.283186 with a local minimum of 0
- -2.805118 and 3.131312 with a local minimum of 0
- 3.0 and 2.0 with a local minimum of 0.

Apart from the four local maxima, this function has a global minimum of 0. The solution space for this function is presented in Figure 3.22 and the equation is presented in Equation 3.6. Analysis and data collected from this experiment is presented in Section 4.1.6.

$$f(x) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

Equation 3.6 Himmelblaus Fitness Function

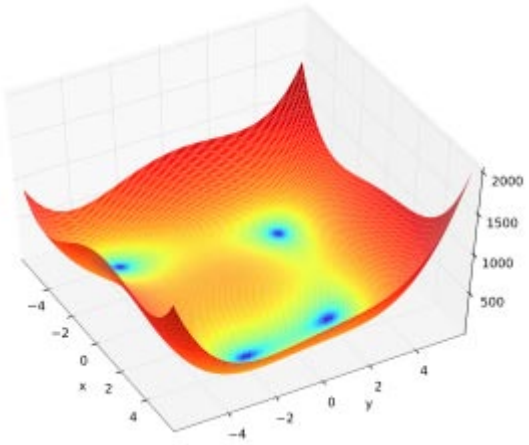

Figure 3.22 Solution Space for Himmelblaus Fitness Function (Takahashi & Kobayashi, 2001)

3.3.2 Phase 2: Scalability Testing

Numerical test functions implemented in the first evaluation are continuous in nature and their input parameters vary continuously between an upper and lower bound. However, the functions have different solution space and as a result it is difficult to conclude the performance of algorithms. This phase utilises the Bump function (Keane, 1994) that has been designed to be easy to code with arbitrary numbers of dimensions but hard to solve. This function gives a highly bumpy surface where the true global optimum is usually defined by the constraints. The Bump function has previously been used as a scalability testing function (Connor, 1999). This phase of experiments was further divided into six sub exercises and in each exercise the bump function was supplied with a different dimension to observe the behaviour of all the metaheuristics search techniques. The base function was flexible enough, so that changes in dimension were configurable and would not require further development. The global minimum for this function is established at 0 and the solution space for this function with a dimension of two is presented in Figure 3.23. In addition to this the equation used to conduct analysis and collect data is presented in Equation 3.7.

$$f(x) = \frac{abs(\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i))}{\sqrt{\sum_{i=1}^{n} ix_i^2}}$$

Equation 3.7 Bump Fitness Function

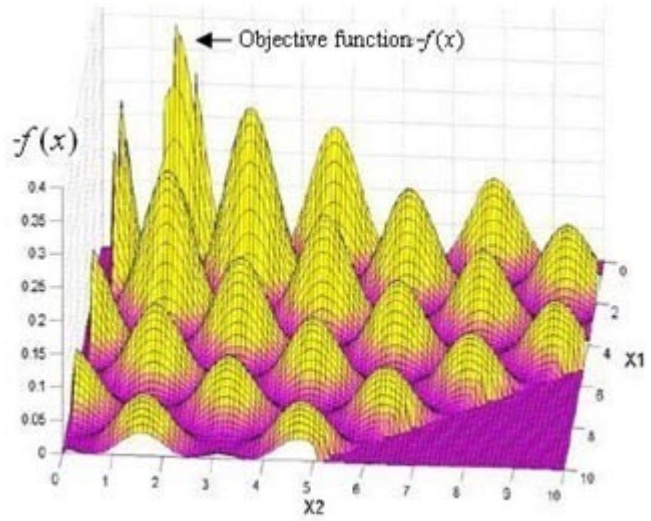

Figure 3.23 Solution Space for Bump Function (n = 2) (Keane, 1994)

This phase of experiment will carry out evaluation of genetic algorithm, simulated annealing and tabu search using the equation mentioned above. Each algorithm will be evaluated against dimension of 2, 5, 10, 15, 20, and 50. The data and analysis for each dimension are presented in Sections 4.2.1 to 4.2.6.

3.3.3 Phase 3: Discrete Scalability Testing

The first two phases of the evaluation mainly focused around the validity and scalability of the metaheuristics search techniques against numerical test functions and continuous bump function with variety of dimension. Although the evaluation provides useful insight into the performance of search techniques, project scheduling problems tends to be discrete in nature – either there are a limited number of values each parameter can take or the parameter values can only be used in certain combinations. Hence the numerical test functions may not provide robust data to support the evaluation of algorithm performance. The main aim of this phase is to use the same metaheuristics search techniques to solve a discrete problem. For this experiment, the n-queens problem has been chosen mainly because this problem falls in the category of NP-hard problems.

In this experiment, genetic algorithm, simulated annealing and tabu search have been adapted to solve n-queen problem, identify efficiencies and achievements. The literature for this problem has been presented in section 2.3.3 and the idea behind the implementation is derived from evaluations carried out by Martinjak & Golub (2007). Based on the idea, algorithms will be developed which allocates each queen a location on the chessboard. The main criterion is to ensure that the algorithm will place each queen uniquely for each row and column. To achieve this functionality a solution representation is developed using Martinjak & Golub (2007) as

reference. The solution representation of this problem is n-tuples (q1, q2, q3, ….,qn) that are a permutation of tuple (1,2, 3,….,n) (Martinjak & Golub, 2007).

For each search techniques, the upper bound complexity is determined as well as the complexity of fitness function. The tool and technology used to develop this component is the same as the previous phases. Since all the three search techniques have the heuristics element in them (Harman, 2007), it seems like there will be a need to develop a specific heuristics function. The implementation of this function is illustrated in Figure3.24. To validate this implementation the data collected from this experiment will compared against the data collected by Martinjak & Golub (2007). The results collected from this experiment are presented in Chapter 4, Section4.3.

```
6  namespace Master.Thesis.EvaluationFunction.Functions
7  {
8      public class nQueens : FunctionProvider
9      {
10
11         #region Methods
12
13         public override Object ObjFunction(Object[] X, int N)
14         {
15             // ---- Declarations ----
16             Random Rnd = new Random();
17             int Position1 = Rnd.Next(N);
18             int Position2 = Rnd.Next(N);
19             Object[] Y;
20
21             // ---- Initialise ----
22             Y = X;
23
24             // ---- Assign Values ----
25             Y[Position1] = X[Position2];
26             Y[Position2] = X[Position1];
27
28             // ---- return the object ----
29             return Y;
30         }
31
32         #endregion
33
34     }
35 }
36
```

Figure 3.24 Heuristics Implementation of n-queen Problem

3.3.4 Phase 4: Resource Allocation, Cost Estimation and Project Scheduling Problem

This is the last and the most crucial phase of the experiment for this research problem in project management are tackled using metaheuristics search techniques. Previous phases of experimentation have been intended to help refine and validate the implementation of the algorithms. Additionally, previous evaluation would also help generate some analysis of performance for each search techniques. Based on the previous experiment design and implemented, this phase will only focus on discrete problems. This experiment is further divided into two categories i.e.: uni-objective and multi-objective.

3.3.4.1 Uni-Objective: Resource Unconstrained and Constrained Project Scheduling

The main aim of this experiment is to schedule resource-unconstrained and constrained project by use of metaheuristics search techniques. The tools and technologies required to implement these search techniques is the same as previous evaluations. Based on the literature review, traditionally most of the projects were schedules using critical path method (Herroelen, 2005). This approach mainly relies on forward and backward pass calculation to solve the critical path in a network. The first step of the evaluation will generate a network diagram which will highlight all the critical paths in the projects. At this stage of evaluation, total duration of the project will be established and will be used as a benchmark to compare the performance of each metaheuristics search techniques at a later stage. The next step of this evaluation is to schedule the same project without any resource constraints and using the same metaheuristics search techniques. Results from this evaluation will be compared against the results generated from critical path method. The last step is almost similar to the previous step except the project and its activities have constrained the resources. The results from this evaluation will be compared against both the previous steps. The project data for this experiment was also used by Christodoulou (2010) where they have evaluated same case study using ant colony optimisation search techniques. The results generated from this experiment are presented in Section 4.4.1.

3.3.4.2 Multi-Objective: Cost Estimation and Project Scheduling

The main objective of this experiment is to tackle the time-cost trade-off problem in the area of project management. Traditionally, there are trade-offs between time and cost when scheduling a project, hence if the resource is less expensive, then the project will take longer to complete. The principal aim of the implementation would be to identify the relationship between time and cost and from there on schedule the project in an effective and efficient manner meaning that the total cost of the project should not be too high or too low. In addition to this, the implementation will primarily focus on cost and duration of the activities. The logic of all the metaheuristics search techniques to generate a trade-off curve will go through a four step process and they are as follows:

- **Step 1**: Activities with longest and shortest duration should be selected and stored in a temporary collection object (array). All the activities in this collection object must have its total cost and duration. This information will be utilised from the critical path method. At this stage, a random solution is selected as best possible solution by default.

- **Step 2**: Once the collection object is initialised, the total cost and duration of the project is established. To generate the time-cost trade-off curve, calculate the distance between each consolidated project element (total cost and duration) with the segment of the convex hull.
- **Step 3**: For each item in the collection object, a fitness value is computed using the formula mentioned in Equation 4.1. After the fitness value is computed, the probability of each item is determined and if the item is selected it is then saved into a temporary collection.
- **Step 4**: Repeat step 2 – 3 until appropriate time-cost trade-off curve is generated.

The main idea behind the above mentioned four steps is to ensure that time-cost trade-off curve is generated for each metaheuristics search techniques. The source data used in this experiment is referenced from Elbeltagi, Hegazy & Grierson (2005) and Feng, Liu & Burns (1997). In addition to this, the results generate from this experiment is presented in chapter 4, section 4.4.2.

3.4 Summary

This chapter described the design and implementation of genetic algorithm, simulated annealing and tabu search in the area of project management. Various numerical test functions were also designed and implemented. The study was designed to ensure that metaheuristics search techniques were validated for their implementation and performance. By doing so, these experiments acted as the foundation of the research before it tackled more advanced problems like resource allocation, cost estimation and project scheduling problem. All the search techniques were fine-tuned during this designing process and because of this exercise, standard set of parameters were established for each search techniques.

4 Results

The main aim of this chapter is to present the data collected from various experiments and analyse on the behaviour and performance of metaheuristics search techniques. The experimental design is described in detail in Chapter 3 Section 3.3. The implementation will follow the established guidelines and the results are also presented in same order and structure. Chapter 3 has already identified research questions and its corresponding experiments. These experiments are carried out in four phases and they are as follows:

4.1 Phase 1: Metaheuristics Verification

The initial data was collected by executing genetic algorithm, simulated annealing and tabu search based on selected numerical test functions. The main aim of this evaluation is to conduct experiments to ensure that all the metaheuristics search techniques have found the global minimum value. These evaluations were carried out for each search techniques to point where there was no further improvement in the solution. For all the evaluations, each implemented search techniques was able to find global minimum value. The summary from these evaluations is presented in Table 4.1

Numerical Test Functions	Metaheuristics Search Techniques								
	Genetic Algorithm			Simulated Annealing			Tabu Search		
	IV	Min	Eval	IV	Min	Eval	IV	Min	Eval
Rastrigin	4.2	0	9	3.5	0	8	3.8	0	9
Rosenbrock's Banana	3.3	0	9	4.1	0	9	3	0	10
Schwefel	512	-512	8	475	-512	10	420	-512	10
Axis Parallel Hyper – Ellipsoid	3.7	0	8	3.1	0	10	3.3	0	7
Griewangk	4.0	0	9	3.9	0	8	2.8	0	10
Himmelblaus	4.9	0	9	3.3	0	8	3	0	9

Table 4.1 Summary of Results

The columns in Table 4.1 represent the following:

- **IV**: This stands for "Initial Value" and represents the first value generated when each search technique is evaluated against each numerical test functions.
- **Min**: This stands for "Minimum" and represents the global minimum achieved by each search technique for each numerical test functions.
- **Eval**: This stands for "Evaluations" and the values are represented in thousands. The value presented in this column indicates the number iterations taken for each algorithm to find global minimum.

Based on the summary of results presented in Table 4.1, the overall performance of each search techniques can be judged. For Rastrigin fitness function, simulated annealing found the global minimum quicker than genetic algorithm and tabu search. However, the initial objective function value is lower than other two search techniques. In Rosenbrock's Banana fitness function, genetic algorithm and simulated annealing took the same number of iterations to find global minimum value, but the initial objective function value is lower than simulated annealing. This indicates that convergence of simulated annealing is much quicker than genetic algorithm. In contrast to this, tabu search took the longest to find the global minimum value. For Schwefel fitness function, the solution space a considerably larger and more complex making it more complicated to solve this problem. In this function genetic algorithm found the global minimum value the quickest but the initial objective function value was higher than other two search techniques which mean that when all the search techniques were initialised, the initial objective function value of genetic algorithm was quite far from global minimum value. For Axis-Parallel Hyper Ellipsoid fitness function, tabu search was the quickest to find global minimum value and unfortunately this was the only function were tabu search was deemed to be the best performing search technique. Simulated annealing had achieved the global minimum value quickest when evaluated against Griewangk fitness function. In the same function, the initial objective function value for tabu search was the least but took the longest to find the global minimum value. Like the previous evaluation, simulated annealing was the quickest to find global minimum value, but in this function, genetic algorithm and tabu search took the same number of iterations to find global minimum value.

Considering the analysis mentioned above, simulated annealing has performed better in most fitness functions when compared against other two search techniques. For analysing the performance of each fitness function, they will be judged on the basis of initial objective function value and how soon the search technique has found the global minimum objective function value. Analysis and implementation for each fitness function is as follows:

4.1.1 Rastrigin Fitness Function

The equation used to implement this function is presented in Equation 3.1. In this function, simulated annealing and tabu search had performed considerably well by achieving 3.5 and 3.8 initial objective function value respectively. In this case, simulated annealing had achieved the least initial objective function value and it also found an optimum solution earlier than genetic algorithm and tabu search. Looking at the performance analysis presented in Table 4.2, it is clear that genetic algorithm was not performing well up to 7000 iterations whereas simulated annealing and tabu search both were performing well against the average objective function value. Ideally if the values are less than average, then the performance of search technique at that data point might be considered better. The data collected from this evaluation has been illustrated in Figure 4.1. Table 4.2 will represent the performance for each algorithm against average results at the interval of 1000 iterations.

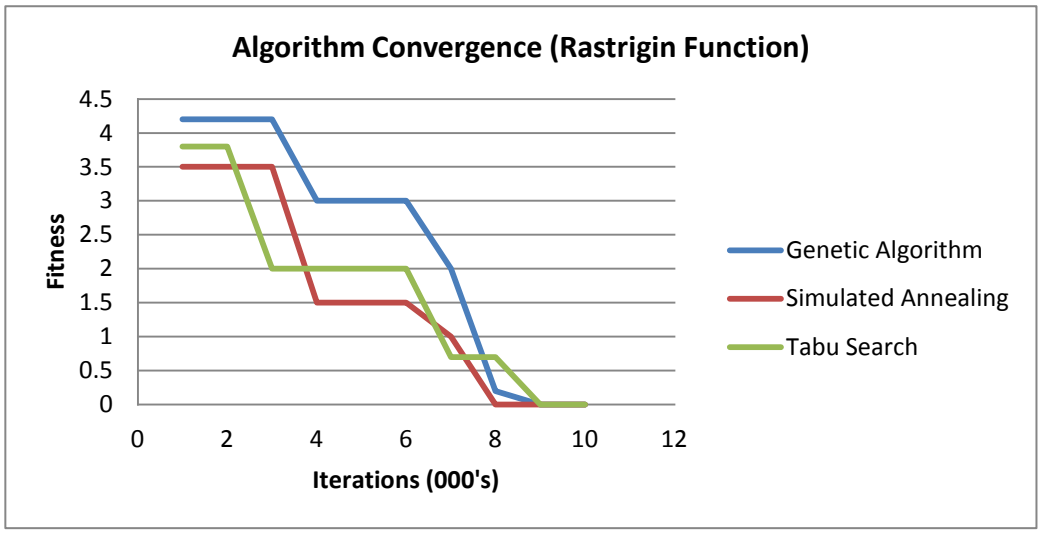

Figure 4.1 Algorithm Convergence (Rastrigin Function)

Iterations (000's)	Average	GA	SA	TS
1	3.8	4.2	3.5	3.8
2	3.8	4.2	3.5	3.8
3	3.2	4.2	3.5	2
4	2.2	3	1.5	2
5	2.2	3	1.5	2
6	2.2	3	1.5	2
7	1.2	2	1	0.7
8	0.3	0.2	0	0.7
9	0	0	0	0
10	0	0	0	0

Table 4.2 Rastrigin Function Performance Analysis

4.1.2 Rosenbrock's Banana Fitness Function

This function was implemented using the equation presented in Equation 3.2 and like Rastrigin function, simulated annealing has performed well but additionally genetic algorithm has also performed well by finding the minimum objective function quicker than tabu search. The initial objective function value for simulated annealing was lot higher than other two, but simulated annealing has dropped fairly quickly to find the minimum objective function value. Hence looking at the performance results in Figure 4.2, it is clear that simulated annealing has converged lot quicker than other two search techniques. Hence for this fitness function, the best performing search techniques based on average results was genetic algorithm and simulated annealing. The visual representation of data collected from this evaluation is presented in Figure 4.2 and Table 4.3 will compare the results for each algorithm at interval of 1000 iterations against the average results.

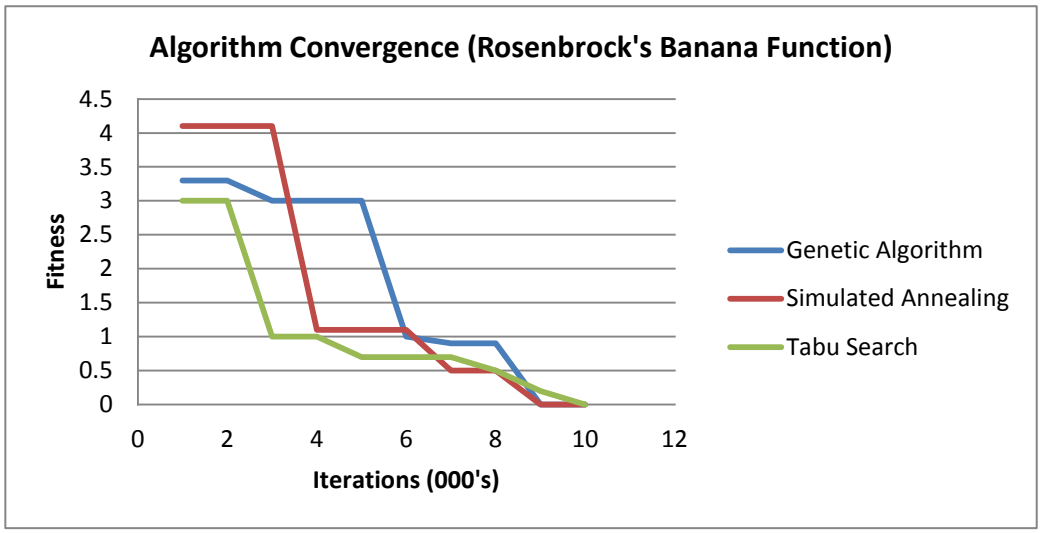

Figure 4.2 Algorithm Convergence (Rosenbrock's Banana Function)

Iterations (000's)	Average	GA	SA	TS
1	3.5	3.3	4.1	3
2	3.5	3.3	4.1	3
3	2.7	3	4.1	1
4	1.7	3	1.1	1
5	1.6	3	1.1	0.7
6	0.9	1	1.1	0.7
7	0.7	0.9	0.5	0.7
8	0.6	0.9	0.5	0.5
9	0.1	0	0	0.2
10	0	0	0	0

Table 4.3 Rosenbrock's Banana Function Performance Analysis

4.1.3 Schwefel Fitness Function

The equation used to implement this function is presented in Equation 3.3. As mentioned earlier, the solution space for this function is different to other functions and the global minimum value for this function is established at -512. All the search techniques are able to find global minimum objective function value, some faster than others. The initial objective function value of tabu search was the least but it took the longest to find the global minimum objective function value. Similarly, simulated annealing took the same number of iterations as tabu search to find the global minimum objective function value but the initial objective function value was much higher than tabu search. Genetic algorithm was the quickest to find the global minimum objective function value, but the initial value was lot higher than other two object function value. When the overall performance of each search technique is compared, genetic algorithm seems to be the best performing search technique. The illustration of data collected from this evaluation is presented in Figure 4.3 and Table 4.4 will present the performance analysis for each algorithm against average results at the interval of 1000 iterations.

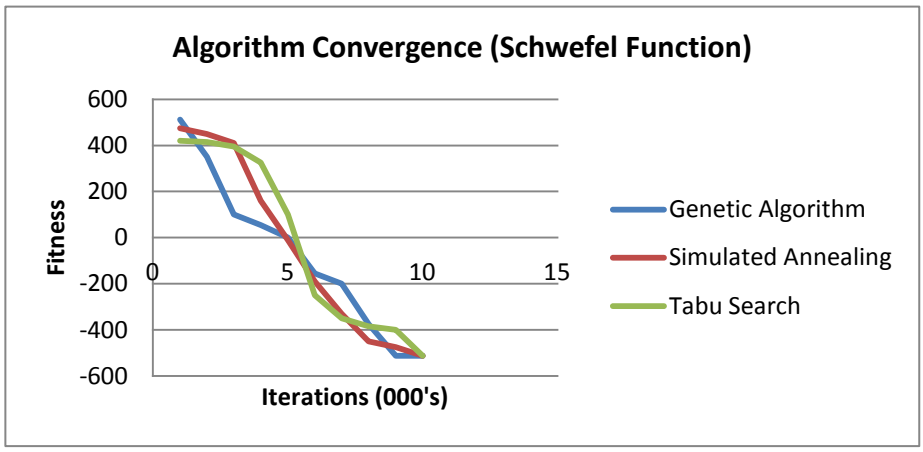

Figure 4.3 Algorithm Convergence (Schwefel Function)

Iterations (000's)	Average	GA	SA	TS
1	469	512	475	420
2	405	350	450	500
3	302	101	410	460
4	180.333	55	160	426
5	30	0	-10	100
6	-198.333	-155	-190	-250
7	-293.333	-200	-330	-350
8	-403.333	-375	-450	-385
9	-462.333	-512	-475	-400
10	-512	-512	-512	-512

Table 4.4 Schwefel Function Performance Analysis

4.1.4 Axis Parallel Hyper-Ellipsoid Fitness Function

The equation used to implement this function is presented in Equation 3.4. In this function the least initial objection function value was found by simulated annealing. However, simulated annealing took the longest to find the minimum global objective function value. In contrast to this, the initial objective function value for tabu search was slightly higher than simulated annealing, but it found the global minimum objective function value lot quicker than simulated annealing and genetic algorithm. To summarise the overall performance of each search techniques based on average results, tabu search has performed better than the other two search techniques. The performance of each algorithm can be measured against average results at the iteration of 1000 in Table 4.5 and the results are illustrated in Figure 4.4.

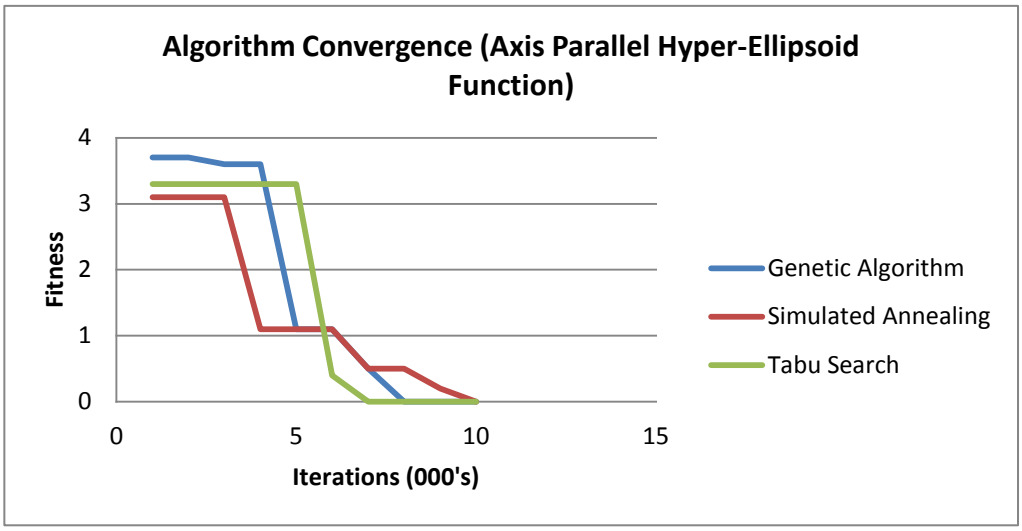

Figure 4.4 Algorithm Convergence (Axis Parallel Hyper-Ellipsoid Function)

Iterations (000's)	Average	GA	SA	TS
1	3.4	3.7	3.1	3.3
2	3.4	3.7	3.1	3.3
3	3.3	3.6	3.1	3.3
4	2.7	3.6	1.1	3.3
5	1.8	1.1	1.1	3.3
6	0.9	1.1	1.1	0.4
7	0.3	0.5	0.5	0
8	0.2	0	0.5	0
9	0.1	0	0.2	0
10	0	0	0	0

Table 4.5 Axis Parallel Hyper-Ellipsoid Function Performance Analysis

4.1.5 Griewangk Fitness Function

The equation used to implement this function is presented in Equation 3.5. In this function, the minimum initial objective function value was achieved by tabu search but it took the longest to find the global minimum objective function value. Simulated annealing was the fastest to find global minimum objective function value. Although, the initial value was high, the convergence progress quickly towards the minimum objective function value. Genetic algorithms initial objective function value was higher than other two search techniques and it found the minimum objective function value just before tabu search. Figure 4.5concludes that the performance of both genetic algorithm and simulated annealing are consistent across all the iterations, but because the optimum solution was found quicker by simulated annealing, it can be determined as better performing search techniques. The performance analysis of the collected data against average results at the interval of 1000 iterations is presented in Table 4.6.

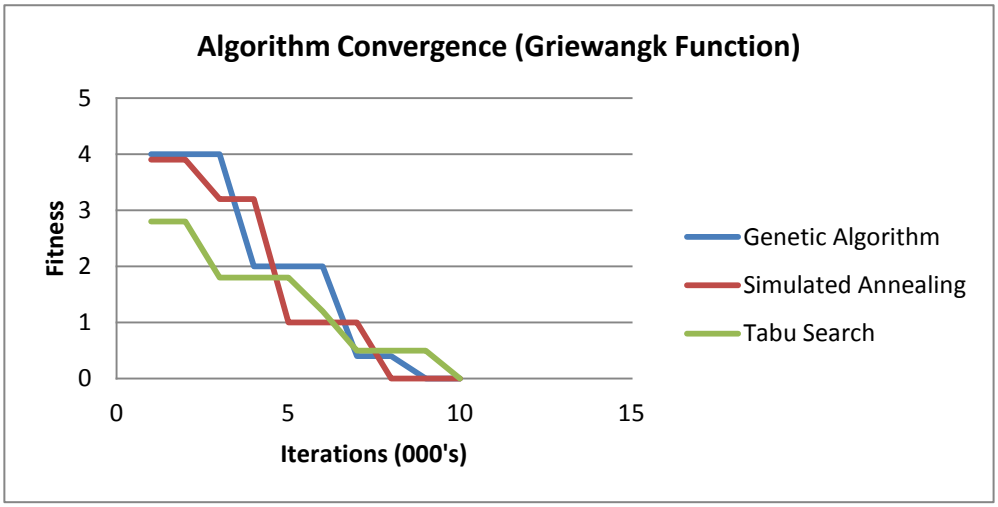

Figure 4.5Algorithm Convergence (Griewangk Function)

Iterations (000's)	Average	GA	SA	TS
1	3.6	4	3.9	2.8
2	3.6	4	3.9	2.8
3	3	4	3.2	1.8
4	2.3	2	3.2	1.8
5	1.6	2	1	1.8
6	1.4	2	1	1.2
7	0.6	0.4	1	0.5
8	0.3	0.4	0	0.5
9	0.2	0	0	0.5
10	0	0	0	0

Table 4.6 Griewangk Function Performance Analysis

4.1.6 Himmelblaus Fitness Function

The function was implemented using the equation presented in Equation 3.6. Like most functions mentioned earlier, simulated annealing found the global minimum objective function value lot quicker than genetic algorithm and tabu search. In this case, the initial objective function value achieved by genetic algorithm was quite high when compared with other two and tabu search found the least initial objective function value. During the initial stages of convergence, it appears as though tabu search was performing well against average results, but towards the end, it was clear that convergence of simulated annealing dropped quickly and found the global minimum objective function value fastest. Additionally, genetic algorithm was the worst performing search techniques, but right through all the iterations, the objective function value was higher than average results. The performance of each algorithm can be measured against average results at the interval of 1000 iterations in Table 4.8 and the results are illustrated in Figure 4.7.

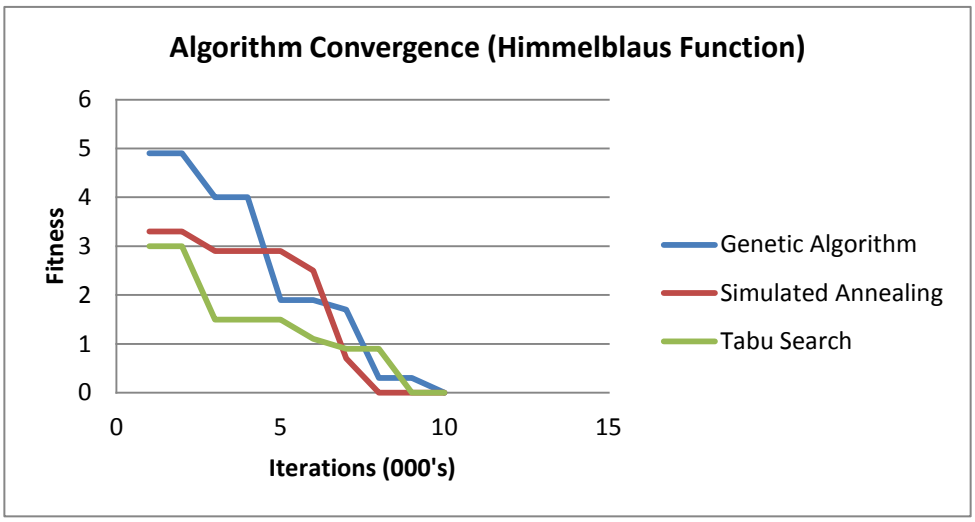

Figure 4.6 Algorithm Convergence (Himmelblaus Function)

Iterations (000's)	Average	GA	SA	TS
1	3.7	4.9	3.3	3
2	3.7	4.9	3.3	3
3	2.8	4	2.9	1.5
4	2.8	4	2.9	1.5
5	2.1	1.9	2.9	1.5
6	1.8	1.9	2.5	1.1
7	1.1	1.7	0.7	0.9
8	0.4	0.3	0	0.9
9	0.1	0.3	0	0
10	0	0	0	0

Table 4.7 Himmelblau Function Performance Analysis

4.1.7 Summary

The first phase of the evaluation indicates that overall performance of simulated annealing has been better then genetic algorithm and tabu search. Not only that, but detailed performance analysis also indicates that simulated annealing has performed better in most evaluations The best performing results are presented in Table 4.9.

Fitness Functions	Best Performing Algorithm
Rastrigin	Simulated Annealing
Rosenbrock's Banana	Genetic Algorithm and Simulated Annealing
Schwefel	Genetic Algorithm
Axis Parallel Hyper – Ellipsoid	Tabu Search
Griewangk	Simulated Annealing
Himmelblaus	Simulated Annealing

Table 4.8 Best Performing Algorithm

To answer the questions that were set out earlier, all the algorithms had found global minimum objective function value some faster than others. In addition to this, the performance variance against the average results for metaheuristics search techniques has been presented. The results achieved from this evaluation are purely based on continuous numerical test functions which were fine tuned by adjusting the input parameters (experimental design, Section 3.3.1); hence these numerical test functions were limited to get an understanding of the discrete problem that the research is addressing. So, the next stage will evaluate how these search techniques perform on the scalable Bump problem.

4.2 Phase 2: Scalability Testing

After the broader evaluations of genetic algorithm simulated annealing and tabu search on continuous numerical test functions, this section will evaluate these search techniques using the bump fitness function with the dimension of 2, 5, 10, 15, 20 and 50. The equation used to implement this function is presented in Equation 3.7 and the data collected from these evaluations has been illustrated in Figure 4.7, 4.8, 4.9, 4.10, 4.11 and 4.12. In addition to that the summary of data collected is presented in Table 4.9 which highlights the initial objective function value and global minimum objective function value and the number of iterations the search techniques took to find an optimum solution.

Bump Fitness Function	Metaheuristics Search Techniques								
	Genetic Algorithm			Simulated Annealing			Tabu Search		
Dimensions	IV	Min	Eval	Max	Min	Eval	Max	Min	Eval
n = 2	5.1	0	9	4.8	0	9	5.3	0	9
n = 5	4.9	0	9	4.8	0	9	5.1	0	10
n = 10	5.3	0	10	5	0.3	10	4.5	0.9	10
n = 15	4.1	0	10	4.3	1.1	10	4.7	1.3	10
n = 20	4.7	0.6	10	4.2	1.3	10	4.5	1.7	10
n = 50	4.8	1	10	4.6	1.5	10	4.8	1.9	10

Table 4.9 Summary of Results from Evaluating Bump Function

The columns in Table 4.1 represent the following:

- **IV**: This stands for “Initial Value” and represents the first value generated when each search technique is evaluated against each dimension of bump function.
- **Min**: This stands for “Minimum” and represents the global minimum achieved by each search technique for each dimension in bump function.
- **Eval**: This stands for “Evaluations” and the values are represented in thousands. The value presented in this column indicates the number iterations taken for each algorithm to find global minimum.

As mentioned in phase 1 of evaluation, the performance of an algorithm can be judged on its initial objective function value and how soon the global minimum objective function value was found. Based on the criteria mentioned above and the summary of results presented in Table 4.9, the overall performance of each search techniques can be judged. When the dimension of the bump function is initialised at 2, all the search techniques can find a global minimum objective function value and each one of them have taken same number iterations to achieve

this results. When the dimension is increased to 5, it takes longer for tabu search to find minimum objective function value whereas for genetic algorithm and simulated annealing it took the same number of iterations as the previous evaluation. In the next stage of evaluation the dimension is increased to 10 and during this evaluation genetic algorithm and simulated annealing can find global minimum objective function value, but tabu search has struggled to find an optimum value. When the dimension is initialised to 15, genetic algorithm can find global minimum object function value whereas simulated annealing and tabu search could not find optimum value. To get more detailed understanding of the performance, the dimension is further increased to 20 and 50. The results from these evaluations indicate that none of the search techniques were able to find a global minimum objective function value. Hence, it is apparent that the ability to search for an optimum solution becomes difficult as the size of the problem is increased. The detailed analysis each implementation is as follows:

4.2.1 Bump (n = 2)

In this instance, all the search techniques have achieved global minimum object function value at the same time, but having said that, the initial objective function value achieved by all the algorithms are not same. The initial objective function value achieved by simulated annealing was lower than other two search techniques and the convergence gradually moved towards the minimum objective function value. In compare to that, the initial object function value was second highest and there is a sudden drop in the convergence between iterations 4000 and 6000. Overall tabu search had the highest initial objective function value and in most iterations when the results are compared against average results this search techniques has not performed well. Hence based on the analysis presented it is apparent that performance of simulated annealing is better than other two search techniques. The data collected from this evaluation has been presented in Figure 4.7 and the performance analysis for each algorithm against average on interval of 1000 iterations is presented in Table 4.10.

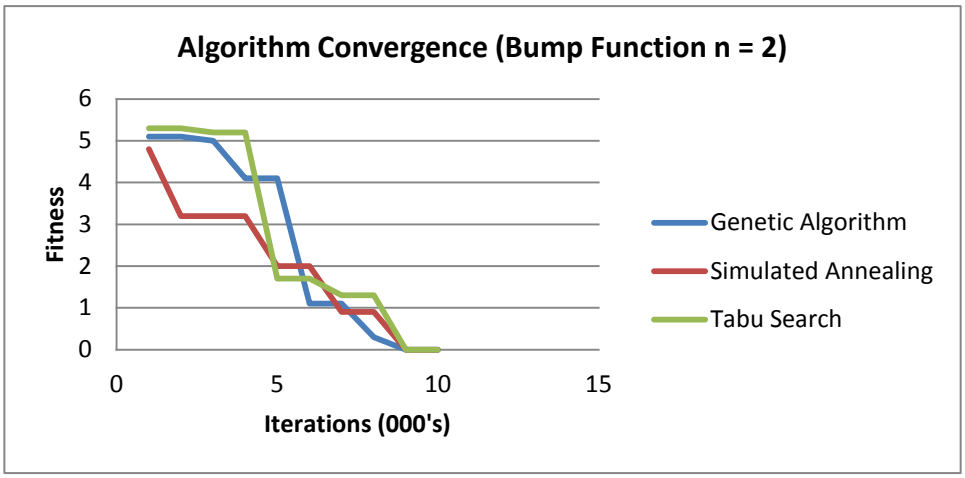

Figure 4.7 Algorithm Convergence (Bump Function n = 2)

Iterations (000's)	Average	GA	SA	TS
1	5.1	5.1	4.8	5.3
2	4.5	5.1	3.2	5.3
3	4.5	5	3.2	5.2
4	4.2	4.1	3.2	5.2
5	2.6	4.1	2	1.7
6	1.6	1.1	2	1.7
7	1.1	1.1	0.9	1.3
8	0.8	0.3	0.9	1.3
9	0	0	0	0
10	0	0	0	0

Table 4.10 Bump Function Performance Analysis (n = 2)

4.2.2 Bump (n = 5)

In this implementation, the dimension of the bump function was increased to 5 to ensure that the algorithm can still find global minimum objective function value when the size of the problem has increased. Hence for this implementation all the algorithms have found an optimum solution but genetic algorithm was the fastest. Unlike previous evaluation, simulated annealing took longer to find the optimum value. When considering the overall performance, genetic algorithm has performed well when compared against the average whereas simulated annealing has performed well only on certain evaluations. Hence for this evaluation the best performing search techniques was genetic algorithm. The data collected from this evaluation is illustrated in Figure 4.8 and performance analysis for each algorithm against average at interval of 1000 iterations is presented in Table 4.11.

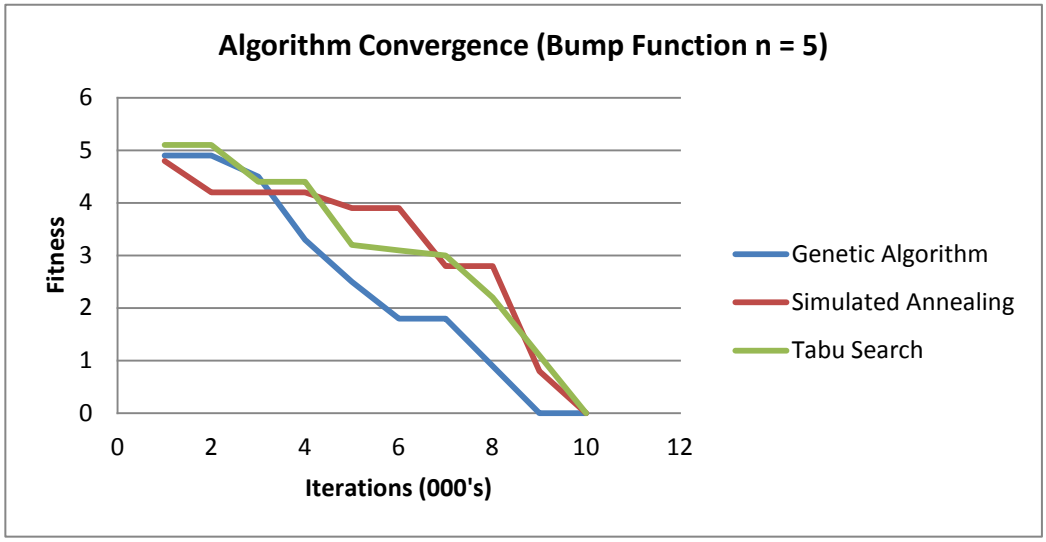

Figure 4.8 Algorithm Convergence (Bump Function n = 5)

Iterations (000's)	Average	GA	SA	TS
1	4.9	4.9	4.8	5.1
2	4.7	4.9	4.2	5.1
3	4.4	4.5	4.2	4.4
4	4	3.3	4.2	4.4
5	3.2	2.5	3.9	3.2
6	2.9	1.8	3.9	3.1
7	2.5	1.8	2.8	3
8	2	0.9	2.8	2.2
9	0.6	0	0.8	1.1
10	0	0	0	0

Table 4.11 Bump Function Performance Analysis (n = 5)

4.2.3 Bump (n = 10)

Looking at the illustration of data collected from this evaluation in Figure 4.9, it is clear that when the dimension of the bump function is increased the search techniques take longer to find the optimum solution. In this case, genetic algorithm found an optimum solution, but simulated annealing and tabu search could not find an optimum solution. Even when the optimum solution was not found evaluations were carried until there was no improvement in the solution. Genetic algorithm has achieved the highest initial objective function value. Even though tabu search did not find its global minimum, it is clear that up to 7000 iterations tabu search was performing better than average. Table 4.12 represents the performance evaluations of all the algorithms against the average at interval of 1000 iterations.

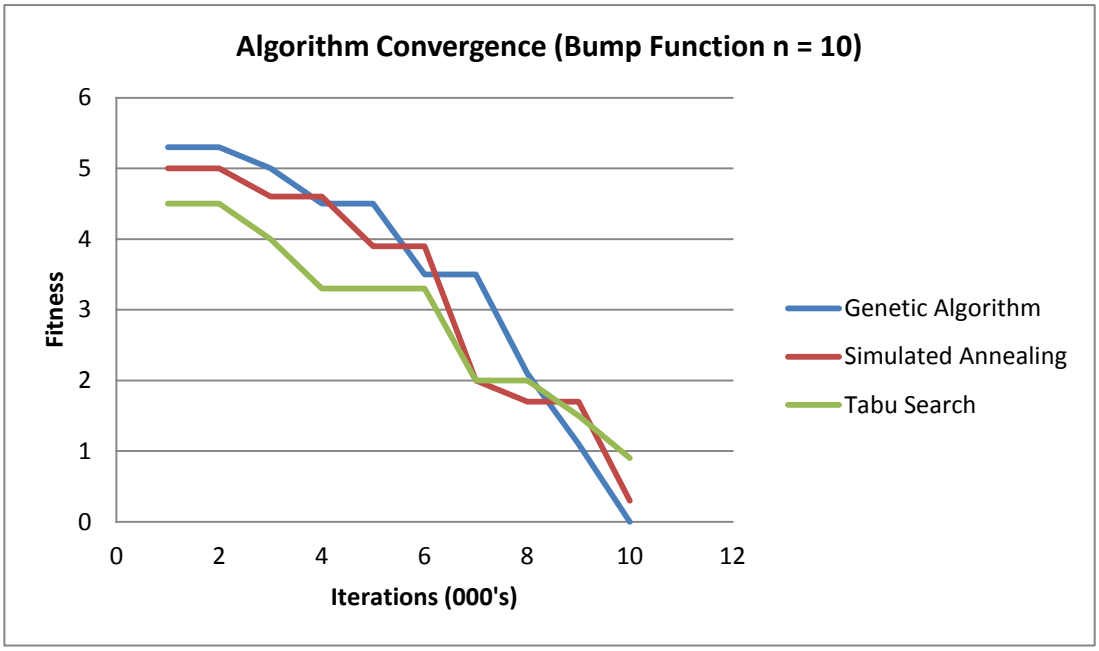

Figure 4.9 Algorithm Convergence (Bump Function n = 10)

Iterations (000's)	Average	GA	SA	TS
1	4.9	5.3	5	4.5
2	4.9	5.3	5	4.5
3	4.5	5	4.6	4
4	4.1	4.5	4.6	3.3
5	3.9	4.5	3.9	3.3
6	3.6	3.5	3.9	3.3
7	2.5	3.5	2	2
8	1.9	2.1	1.7	2
9	1.4	1.1	1.7	1.5
10	0.4	0	0.3	0.9

Table 4.12 Bump Function Performance Analysis (n = 10)

4.2.4 Bump (n = 15)

In this instance simulated annealing and tabu search could not find an optimum solution and this is caused by increasing the dimension of the bump function. When compared these two algorithms with previous implementation of bump function with the dimension of 10, there is a huge difference in the final solution. In contrast to that, so far genetic algorithm has performed consistently well by finding a global minimum objective function value for each implementation. Since genetic algorithm has found the global minimum value, it can be determined that this algorithm has performed better than other two. The data collected from this evaluation is illustrated in Figure 4.10 and Table 4.13 presents the performance analysis of each algorithm against the average results at interval of 1000 iterations.

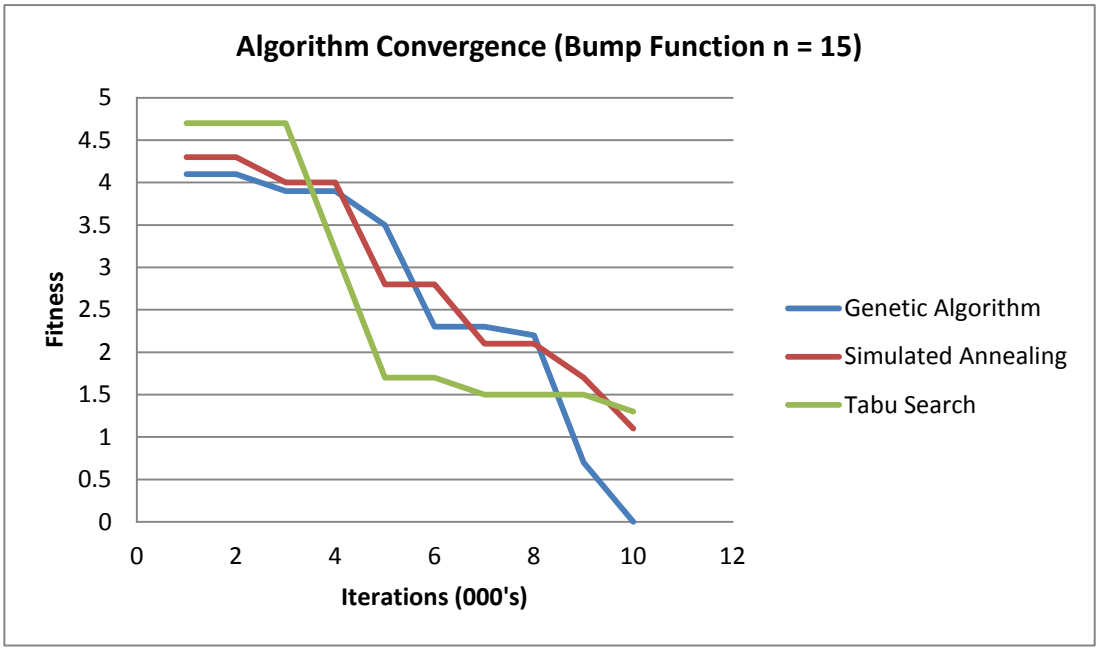

Figure 4.10 Algorithm Convergence (Bump Function n = 15)

Iterations (000's)	Average	GA	SA	TS
1	4.4	4.1	4.3	4.7
2	4.4	4.1	4.3	4.7
3	4.2	3.9	4	4.7
4	3.7	3.9	4	3.2
5	2.7	3.5	2.8	1.7
6	2.3	2.3	2.8	1.7
7	2	2.3	2.1	1.5
8	1.9	2.2	2.1	1.5
9	1.3	0.7	1.7	1.5
10	0.8	0	1.1	1.3

Table 4.13 Bump Function Performance Analysis (n = 15)

4.2.5 Bump (n = 20)

When the dimension of the bump function is increased to 20, it is noticed that all the search techniques were not able to find the global minimum objective function value. Performance of simulated annealing and tabu search have been consistent when compared against the previous two evaluations whereas for this evaluation genetic algorithm struggled to find an optimum solution. For genetic algorithm, the initial objective function value was the highest but between iterations 8000 and 10000 the convergence of dropped considerably and performed better than average. Furthermore, the performance of genetic algorithm in previous iterations was less than average and in contrast to that simulated annealing and tabu search were better or equal to average. The performance of each algorithm can be measured against average results at the interval of 1000 iterations in Table 4.14 and the results are illustrated in Figure 4.11.

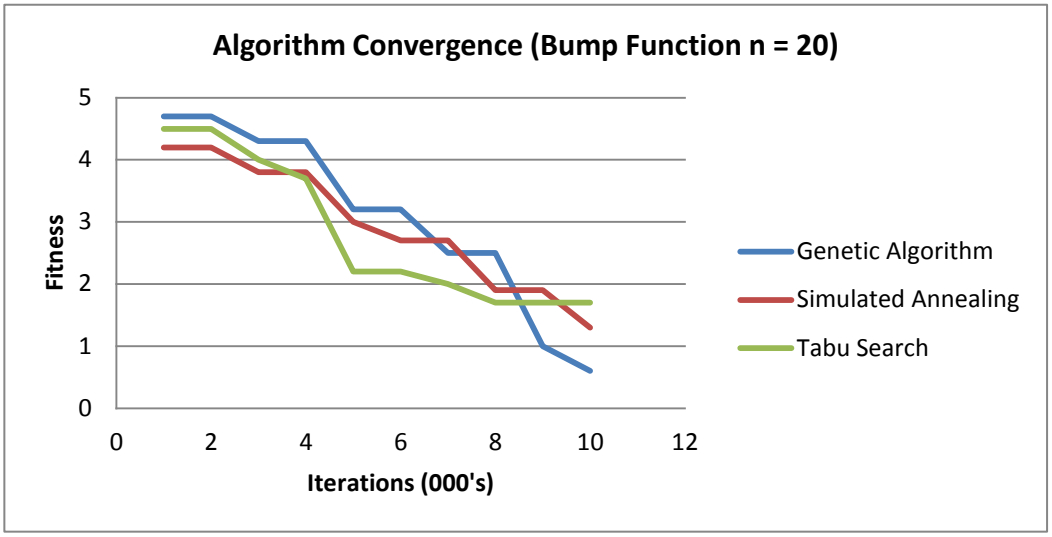

Figure 4.11 Algorithm Convergence (Bump Function n = 20)

Iterations (000's)	Average	GA	SA	TS
1	4.5	4.7	4.2	4.5
2	4.5	4.7	4.2	4.5
3	4	4.3	3.8	4
4	3.9	4.3	3.8	3.7
5	2.8	3.2	3	2.2
6	2.7	3.2	2.7	2.2
7	2.4	2.5	2.7	2
8	2	2.5	1.9	1.7
9	1.5	1	1.9	1.7
10	1.2	0.6	1.3	1.7

Table 4.14 Bump Function Performance Analysis (n = 20)

4.2.6 Bump (n = 50)

To ensure that the performances of all the algorithms are consistent, the dimension of the bump function is increased to 50. Based on the results illustrated in Figure 4.12, it is clear that performance of each search techniques has reduced considerably. Like previous evaluations, genetic algorithm has found the best global minimum objective function value when compared against simulated annealing and tabu search. However, when the performance of genetic algorithm is compared against average results, it is apparent that genetic algorithm did not converge well until the end of the evaluation. In contrast to that, simulated annealing did not find the best global minimum value, but it had consistently converged towards the global minimum. Similar behaviour is also observed by tabu search. The performance of each algorithm can be measured against average results at the interval of 1000 iterations in Table 4.15.

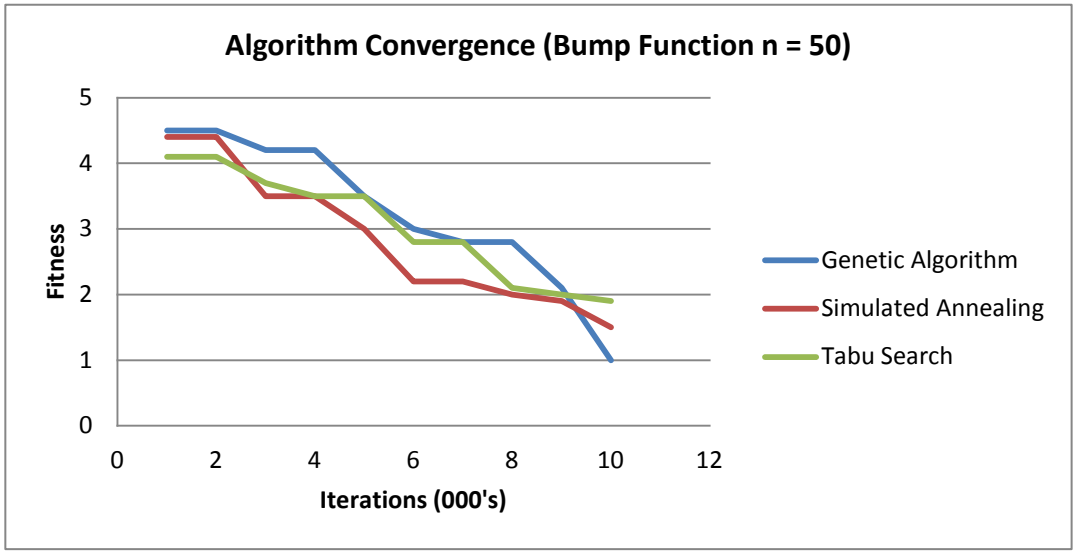

Figure 4.12 Algorithm Convergence (Bump Function n = 50)

Iterations (000's)	Average	GA	SA	TS
1	4.3	4.5	4.4	4.1
2	4.3	4.5	4.4	4.1
3	3.8	4.2	3.5	3.7
4	3.7	4.2	3.5	3.5
5	3.3	3.5	3	3.5
6	2.7	3	2.2	2.8
7	2.6	2.8	2.2	2.8
8	2.3	2.8	2	2.1
9	2	2.1	1.9	2
10	1.5	1	1.5	1.9

Table 4.15 Bump Function Performance Analysis (n = 50)

4.2.7 Summary

Based on the first and second phase of evaluations, the performance of simulated annealing has been better than genetic algorithm and tabu search. When the dimension of the bump function was 5 and 10, the performance of tabu search had been much better than genetic algorithm and simulated annealing. In saying that, as the dimension of the bump function kept on increasing the performance of tabu search and simulated annealing dropped considerably whereas the performance of genetic algorithm was consistent throughout except when the dimension was increased to 20 and 50, genetic algorithm found it hard to find the optimum solution. The performance of the algorithm was based on the parameters which were established in experimental design (Chapter 3, Section 3.3) hence there is a possibility that the performance of the algorithms can be improved by changing the standard parameters. Evaluations from phases one and two will act as the building blocks for phases three and four. In the next phase the study will evaluate a n-queens problem which is discrete in nature.

4.3 Phase 3: Discrete Scalability Testing

The previous exercises focused around validating the performance of genetic algorithm, simulated annealing and tabu search using numerical test functions and emphasised on validating the scalability of the algorithms using bump function. In this exercise the scalability has been explored further by solving n-queen problem using the metaheuristics search techniques mentioned above. According to literature review, it has been established that n-queen problem can be solved using metaheuristics search techniques (Laguna, 1994 and Martinjak & Golub 2007). The data collected from similar evaluations carried out by Martinjak & Golub (2007) will be used as the benchmark to compare the data collected from this evaluation. All the algorithms are executed until the first solution is found; in a series of ten runs for a given number of queens. In addition to this each, evolution ends when a solution is found. The analysis for each algorithm is as follows:

4.3.1 Genetic Algorithm

In this algorithm, mutation rate was 0.06% which helps changing two randomly chosen positions. The crossover rate was established at 0.08% and was implemented in a way that redundant positions of parents were transferred to a child, whilst other positions were selected randomly. The population size used for this evaluation was established at 100 chromosomes. The results generated from this evaluation are presented in Table 4.16.

n (Queen)	Iterations (in 10 runs)					
	Min	Min % Change	Max	Max % Change	Average	Avg % Change
8	3	200%	12	20%	5.2	30%
10	18	13%	220	95%	55.8	14%
30	220	4%	1751	13%	1014.5	11%
50	499	2%	12752	-85%	4237.9	-76%
75	985	-12%	15417	-4%	5149.4	-10%
100	3231	-5%	15323	5%	9473.1	7%
200	12787	-4%	46641	-3%	20786.4	-9%
300	18653	-8%	35347	-7%	22489.3	-19%
500	32389	-11%	150294	-10%	76841.2	-14%

Table 4.16 Genetic algorithm results in n-queen problem

In Table 4.16, column “n-Queen” represents the number of queens that was used to solve the problem. The columns “Min” and “Max” indicates minimum and maximum iterations the algorithm took to find the optimum solution which were executed in ten runs. In addition to this,

the % change column for min, max and average is calculated by dividing the number of iterations the algorithm took to find the best solution (in this evaluation) by the number of iteration the same algorithm took to find the best solution in the evaluation carried out by Martinjak & Golub (2007).

Based on the results presented in Table 4.16, it is evident that the genetic algorithm implemented in this research can solve the n-queen problem. When the results are compared against the results presented by Martinjak & Golub (2007), the performance of this algorithm is consistent, but having said that there are some differences in each scenario (i.e.: solving the problem using different number of queens). Such inconsistencies are common when using a stochastic search technique. When this problem was solved for scenario 8, 10, 30 and 50 queens, genetic algorithm took more iteration to find the most optimum solution when compared against results presented by Martinjak & Golub (2007). In contrast to that, for scenario 50, 75, 100, 200, 300 and 500 queens, genetic algorithm found the best solution faster when compared against Martinjak & Golub (2007). The variance for each scenario is presented in Table 4.16.

This algorithm was executed until the first solution was found; in a series of ten runs. This step is then further repeated for different size of the problem (i.e., different number of queens). As illustrated in Figure 4.13, when the size of the problem was less than 75 queens, genetic algorithm was able to find optimum solution for each run but as the size of the problem was increased the genetic algorithm was not able to find optimum solution for each run. When the size of the problem was 75 queens then genetic algorithm could find optimum solution 9 times out of 10, whereas when the size of the problem was 100 and 200 queens, it could find optimum solution 9 times out of 10. Lastly when the size of the problem was 300 and 500 genetic algorithm did not perform well because it only found the optimum solution 7 times out of 10. Hence as the size of the problem is increased the performance of this algorithm reduces. The reduction in performance behaviour matches previous evaluation where the genetic algorithm was used to solve Bump function with a dimension of 15, 20 and 50.

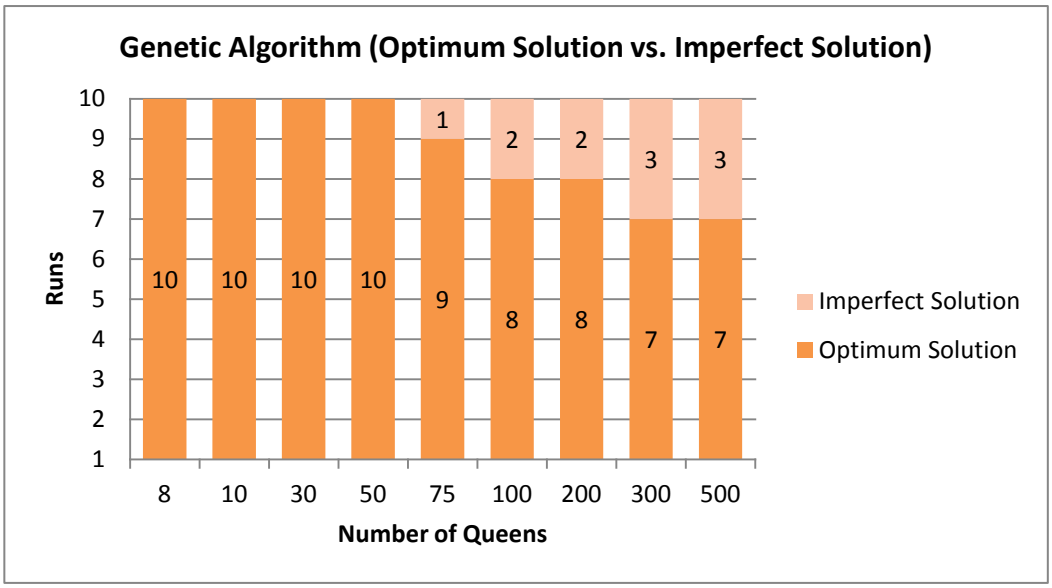

Figure 4.13 Number of Optimum Solutions vs. Imperfect Solutions

4.3.2 Simulated Annealing

Like genetic algorithm, one of the objectives is to change two randomly chosen positions until such point where no better solution can be found. In this algorithm, decreasing temperature was initialised at 0.99% and the starting temperature was established at 1000. Values for these parameters have also been used by Martinjak & Golub (2007). The results generated from this evaluation are presented in Table 4.17 and the definition of this table is the same as Table 4.16.

n (Queen)	Iterations (in 10 runs)					
	Min	Min % Change	Max	Max % Change	Average	Avg % Change
8	79	20%	982	22%	501.4	2%
10	211	4%	2249	12%	1346.2	2313%
30	1524	9%	3951	5%	1125.7	11%
50	1911	5%	5741	6%	4356.1	3%
75	3685	12%	10158	3%	5236.8	2%
100	4265	-5%	10254	-13%	9314.5	-2%
200	8864	-3%	37598	-3%	19745.3	-5%
300	19548	-10%	30215	-5%	20148.6	-10%
500	32146	-11%	80215	-4%	72467.2	-6%

Table 4.17 Simulated Annealing results in n-queen problem

Like genetic algorithm, simulated annealing can also solve the n-queen problem. When the results were compared against the benchmark (i.e.: results presented by Martinjak & Golub (2007)), the performance of simulated annealing is almost similar, but in saying that there were

variances at each scenario and those variances are presented in Table 4.17. When the number of queens were less than or equal to 75, simulated annealing took more iterations to find the most optimum solution, but when the number of queens was greater than 75, then simulated annealing took less iterations to find the most optimum solution.

The performance of simulated annealing has been better than genetic algorithm and tabu search by finding optimum solutions for all the runs for all size of the problem except when the size of the problem is 500 queens, it was not able to find an optimum solution only one time. In compare to that, genetic algorithm and tabu search both could not only find optimum solution seven times out of ten runs. The performance of simulated annealing is in contrast with previous evaluations whereby simulated annealing was the second best performing algorithm. However, when looking at the trend, if the size of the problem is increased it may be possible that performance of simulated annealing could also deteriorate. This reduction in performance behaviour matches the behaviour when simulated annealing solved Bump problem with dimensions of 10, 15, 20 and 50. The variance analysis is presented in the Figure 4.14.

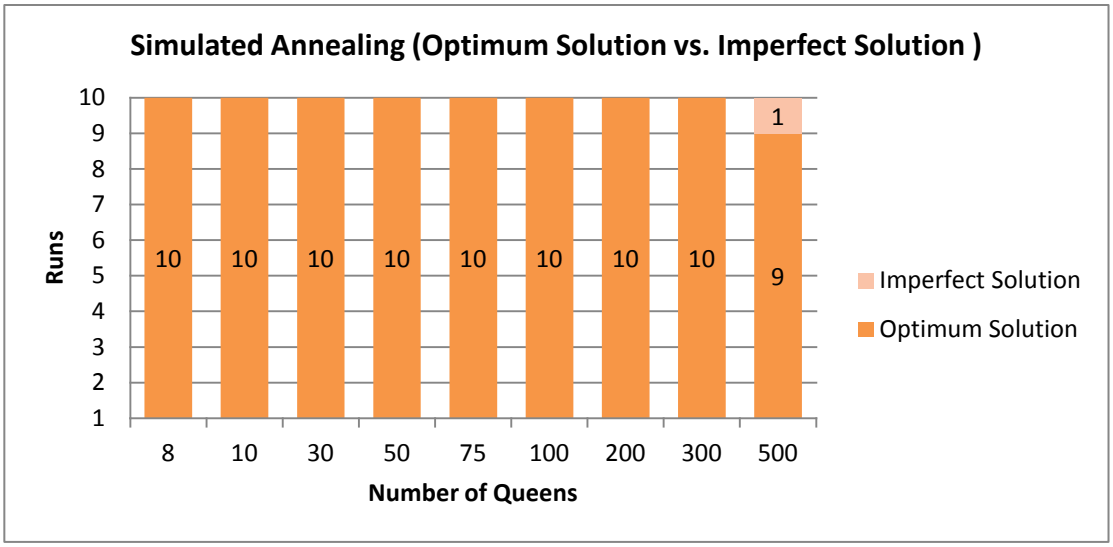

Figure 4.14 Number of Optimum Solutions vs. Imperfect Solutions

4.3.3 Tabu Search

When tabu search algorithm is implemented to solve n-queen problem, the neighbourhood of the current solution X is a set of all n-tuples that are different from X in one exchange of queen places. In all the iterations, this algorithm finds the best solution in the neighbourhood. The tabu list remembers the last L pairs of exchanging positions, in order to avoid searching the same neighbourhood repeatedly. The initial solution with random positions of queens leads to a better

performance of the algorithm than the initial solution of queens. The results generated from this evaluation are presented in Table 4.16.

n (Queen)	Iterations (in 10 runs)					
	Min	Min % Change	Max	Max % Change	Average	Avg % Change
8	3	50%	20	18%	11.3	74%
10	5	25%	33	10%	15.1	44%
30	10	43%	40	167%	22.9	114%
50	11	-8%	48	71%	19.4	5%
75	16	-16%	44	-10%	27.7	-5%
100	21	-19%	71	-8%	39.5	-6%
200	79	-11%	179	-4%	118.6	-2%
300	159	-3%	266	-6%	205.7	-2%
500	344	-1%	410	-13%	380.5	-4%

Table 4.18 Tabu Search results in n-queen problem

The results presented in Table 4.18, indicates that n-queen problem can be successfully solved using a tabu search algorithm. When these results are compared against the results presented by Martinjak & Golub (2007), the performance is consistent with the benchmark results, but there are still some variances at each scenario

Tabu search has been the lowest performing algorithm across all the evaluations when compared against genetic algorithm and simulated annealing. This algorithm was executed until the first solution was found; in a series of ten runs. This step is then further repeated for different size of the problem (i.e.:- different number of queens). The variance analysis is illustrated in Figure 4.15. Tabu search could not find optimum for all the runs if the size of the problem was greater than or equal to 50 queens. The behaviour is similar to genetic algorithm, except genetic algorithm could find optimum solutions for all runs when the size of the problem was 50 queens but for the same problem, tabu search could only find optimum solution 9 times out of 10 runs. In addition to that, as the size of the problem is increased the performance of the algorithm is deteriorated. This behaviour is consistent with tabu search solved Bump problem with a dimension of 10, 15, 20, and 50. As the size of the dimension increased, it became more difficult for tabu search to find optimum solution.

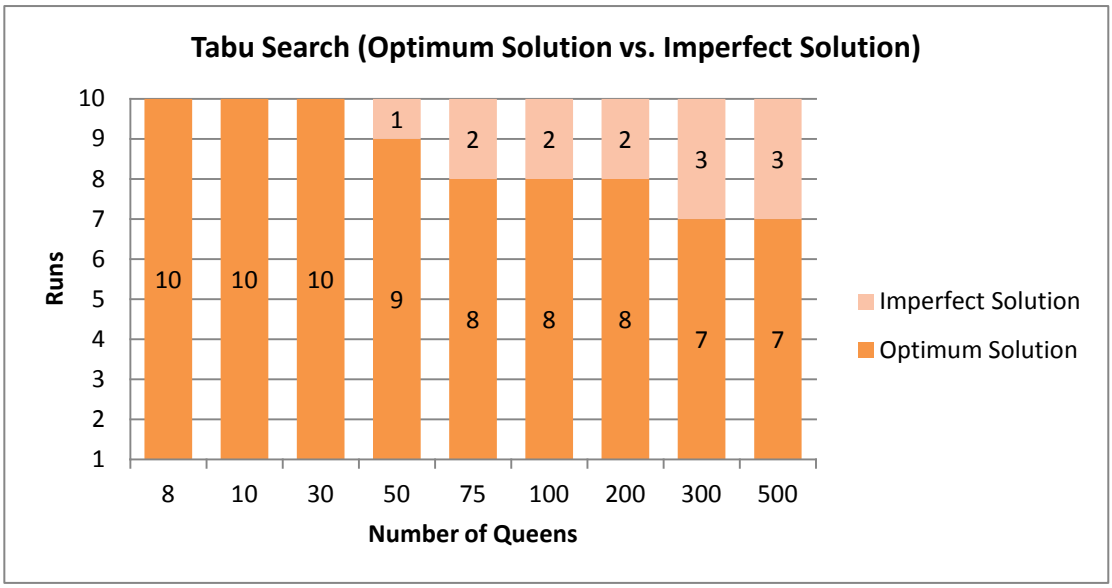

Figure 4.15 Number of Optimum Solutions vs. Imperfect Solutions

4.3.4 Summary

This evaluation showed that n-queen problem can be solved using genetic algorithm, simulated annealing and tabu search. In addition to that, the results generated from this evaluation were also compared against results generated by Martinjak & Golub (2007) and the performance of each algorithm was almost similar. The performance of each algorithm can be judged on the number fitness function computation (Feng, Liu & Burns, 1997). For this evaluation the performance is for each algorithm is presented in Table 4.19 and comparison between each algorithm is illustrated Figure 4.16 on a log scale.

n (Queen)	Genetic Algorithm	Simulated Annealing	Tabu Search
8	455	513	257
10	5,145	1,102	512
30	99,548	2,236	5,021
50	125,648	3,015	23,664
75	65,235	5,526	80,515
100	790,525	7,747	210,357
200	1,954,568	20,790	2,135,659
300	2,277,855	22,154	10,125,463
500	5,490,358	50,157	42,265,835

Table 4.19 Average number of fitness function computation

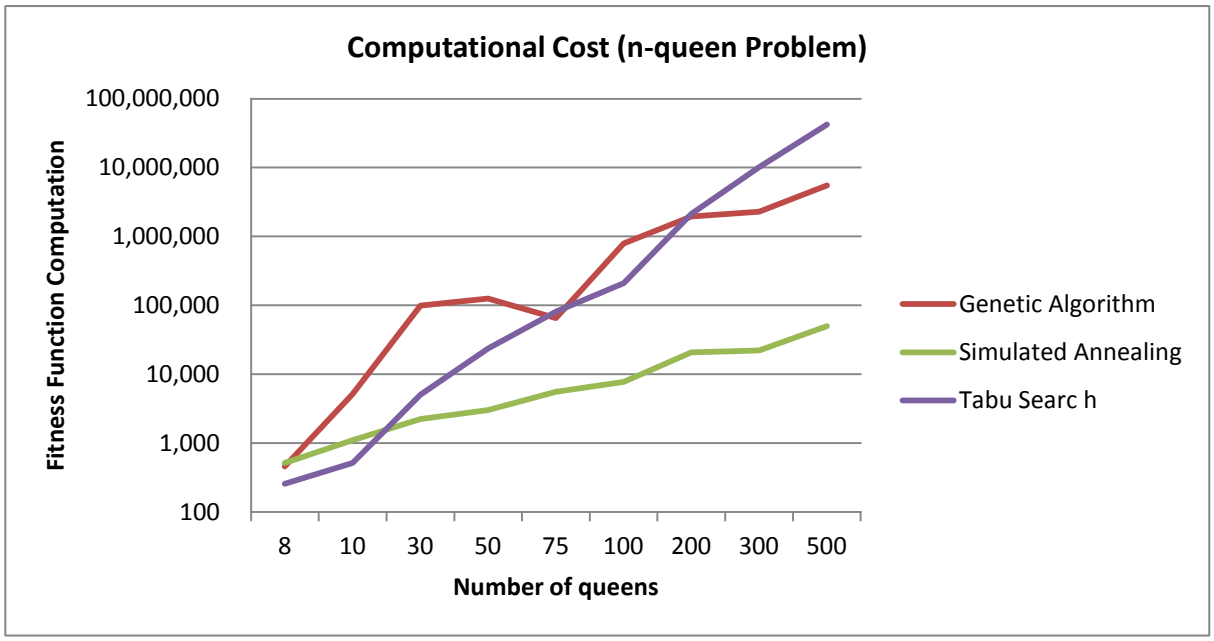

Figure 4.16 Average Fitness Function Computation

Based on the illustration in Figure 4.18, it is clear that overall simulated annealing found the most optimum solution with the least number of fitness function computation for each scenario. When trying to solve the solve the problem with 8 queens, all the algorithms found the most optimum solution with the least number of fitness function computation but having said that tabu search was the best performing algorithm in this scenario by finding the optimum solution using the least number of fitness function computation. As the size of the problem is increased, tabu search and genetic algorithm both required more fitness function computation to find the most optimum solution. Based on the analysis presented above, it is clear that overall the best performing search techniques was simulated annealing followed by genetic algorithm and lastly tabu search which the least performing search technique.

4.4 Phase 4: Resource Allocation, Cost Estimation and Project Scheduling Problem

This is the last phase of the experiments where problems like resource allocation, cost estimation and project scheduling are tackled. This experiment is further compartmentalised in to two main sections:

4.4.1 Scheduling Resource Unconstrained and Constrained Project

To schedule a project effectively, project planners must select appropriate costing and resourcing options. This selection will determine the duration of the project. In most cases, projects have multiple costing and resourcing options which lead to multiple due dates. The main objective in the evaluation is to schedule resource unconstrained and constrained project using metaheuristics search techniques. As with the previous evaluations, the same algorithms were implemented to resolve continuous and discrete problems whereby in most of the cases the algorithms were able to find the optimum solution. Hence the performance of algorithms from previous evaluations will be compared against the performance of the same algorithm in this evaluation.

Traditionally, project schedules can be generated using a critical path method and that project planners can also include resources and activities assigned to those resources. Unfortunately, such schedules have a flip side whereby it is difficult for project planners to identify when the resources were freed from the previous activity. Hence this evaluation will overcome the limitation identified by using critical path method. Before the evaluation process starts, let us consider a small project presented in Table 4.20 by each activity with its early start, early finish, late start, late finish and total float. This data was also used by Christodoulou (2010) to schedule the project using ant colony optimisation algorithm. Figure 4.17 illustrates the critical path for this small project.

Activity no.	Start Node	End Node	Successor	Early Start	Early Finish	Late Start	Late Finish	Total Float
1	0	2	7,8,9	0	20	15	35	15
2	0	5	7	0	33	45	78	45
3	0	8	9	0	70	24	94	24
4	1	3	8,9	0	40	0	40	0
5	1	5	7	0	37	41	78	41
6	1	6	9	0	56	41	97	41
7	2	7	9	20	87	48	115	28
8	2	8	9	20	79	35	94	15
9	2	9	-	20	98	48	126	28
10	3	8	9	40	94	40	94	0
11	3	9	-	40	94	72	126	32

12	4	5	7	0	29	49	78	49
13	4	6	9	0	43	54	97	54
14	5	7	9	37	74	78	115	41
15	6	9	-	56	85	97	126	41
16	7	9	-	87	98	115	126	28
17	8	9	-	94	126	94	126	0

Table 4.20 Test Problem for discrete optimisation (Christodoulou, 2010)

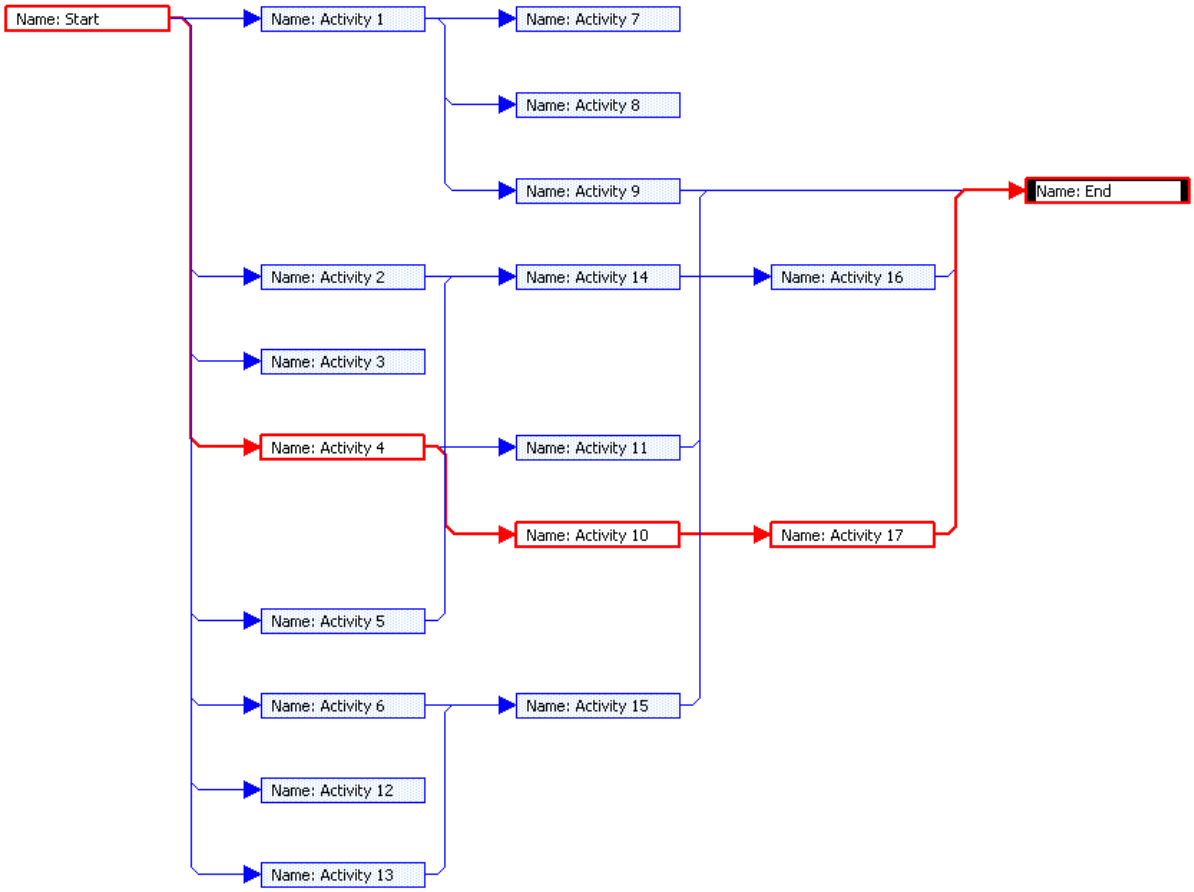

Figure 4.17 Critical Path Method for Data in Table 4.20

The critical path calculations on the above mentioned case study topology is based on the results of early start, early finish, late start, late finish and total float by applying traditional critical path planning methods. Based on the critical path method calculation and activities 4, 10 and 17 have been identified as critical and the total duration of the project is 126 time units.

Christodoulou (2010) has also solved the above mentioned case study using critical path method and the results generated from his evaluation is the same as the results generated from this evaluation. Additionally, he also solved the same study in resource unconstrained and resource constrained environment. Hence the following sections will describe the scheduling of the case study in unconstrained and constrained environment.

4.4.1.1 Evaluation of Resource-Unconstrained Scheduling

Resource unconstrained scheduling is fairly straight forward and in most cases can be solved using critical path methods. Since the case study is relatively simple, all search techniques were able to find an optimum solution. In this case the optimum solution is 126 time units for the project duration. When this solution is compared against the solution presented using critical path method it is the same.

Although in this case the solution is the same as critical path method, it may always not be the same. If the size of the project would be extensively large then finding optimum project duration would take longer and may not be correct because of human intervention. In this case study, the critical path method calculation required ten conditional statements and 17 additions / subtraction for each forward or backward pass in the network. In contrast to that genetic algorithm, simulated annealing and tabu search are lot more efficient in finding the optimum solution. The advantage of this might not be so obvious in this evaluation mainly because of the size of the data, but it is likely that for larger dataset these algorithms would generate results significantly faster and efficiently.

Algorithm	Duration	Critical Path Activities	Iterations
Ant Colony Optimisation (Christodoulou, 2010)	126	4, 10, 17	<= 50
Genetic Algorithm	126	4, 10, 17	<= 39
Simulated Annealing	126	4, 10, 17	<= 49
Tabu Search	126	4, 10, 17	<= 55

Table 4.21 Solution for Resource-Unconstrained Case Study

The results presented above for each algorithm are the same and that is mainly because there are no constraints on the project. However, some search techniques have found the optimum solution sooner than other search techniques. In this case, genetic algorithm was the quickest to find the best solution. In the evaluations carried out by Christodoulou (2010), he has also achieved the same results as genetic algorithm, simulated annealing and tabu search. Although the size of the case is study is fairly small the overall process for calculating the total duration and identifying critical activities was very straight forward. The main idea behind this evaluation is to schedule the project as soon as possible ignoring constraints. However, if we were to assign resources constraint to each task and still want the same due date, there would be some over allocated resources. The resource histogram generated from this evaluation is presented in Figure 4.18.

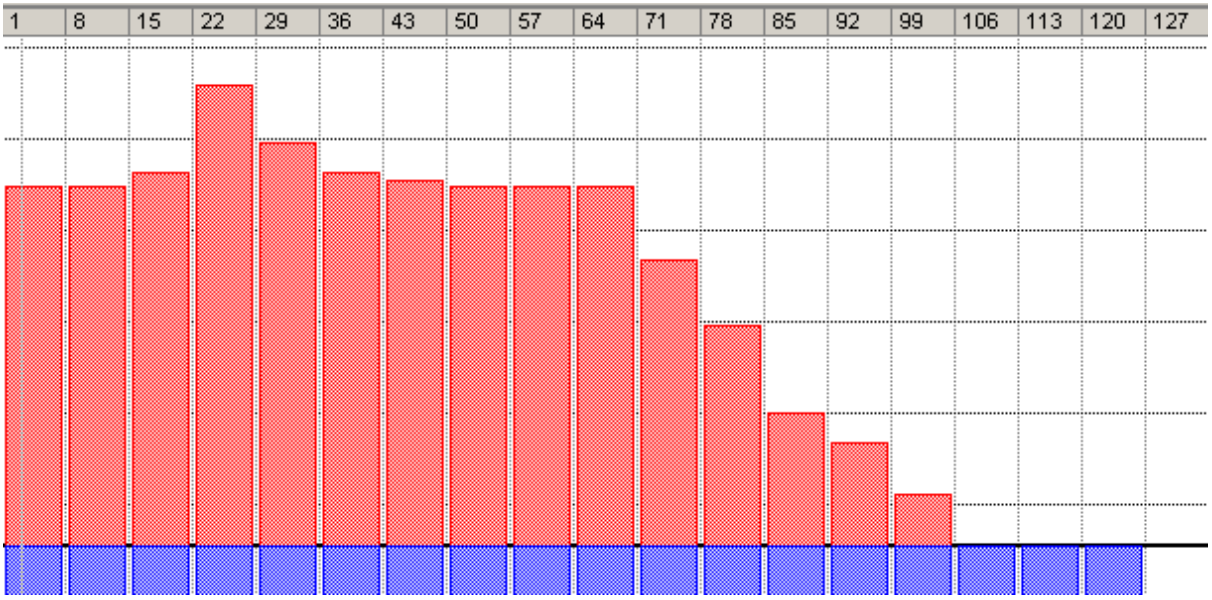

Figure 4.18 Unconstrained Resource-Flow

The histogram presented above indicates that the project will still be completed in the 126 time units. For this experiment, it is assumed that resources are limited and each resource will only be available for a certain hours during the day. Based on Figure 4.18 the blue bars indicate that resources are optimally allocated whereas bars in red means that resources are over allocated. The dark black line in the histogram indicates maximum allocation of resources

4.4.1.2 Evaluation of Resource-Constrained Scheduling

As soon as there is a constraint on resources for the project, the scheduling becomes very complicated and critical path method may not be sufficient to achieve an optimised project schedule. The lack of resources needed to start and complete an activity makes certain critical paths unfeasible and hence some of the activities in a project can be put on hold which in turn can impact the entire project schedule. In critical path method the importance of activities are determined by its total float value. The importance of activity increases as the value of total float drops. Therefore, when scheduling project activities with fewer totals float value gets preference in allocating resources.

It is assumed that each activity presented in Figure 4.19, 4.20 and 4.21 utilises one unit of resources for each day and based on that a resource histogram can be generated. However, for this evaluation it is assumed that the availability of resource is constrained to seven units. Similar case was also implemented in the previous evaluations. The resource histogram which is illustrated in Figure 4.18 which exhibits for certain activities, the need for resources has

exceeded the available resource threshold. When the constraints are implemented following results are generated, shown in Table 4.22.

Algorithm	Duration	Critical Path Activities	Iterations
Ant Colony Optimisation (Christodoulou, 2010)	142	3, 13, 15	<= 50
Genetic Algorithm	139	4, 7, 17	<= 58
Simulated Annealing	147	5, 9, 17	<= 55
Tabu Search	143	2, 9, 17	<= 62

Table 4.22 Solution for Resource-Constrained Case Study

This table represents time taken in the duration column, and also highlights the critical activity. The first results are derived from Christodoulou (2010) experiments. In his experiments, ant colony optimisation finds a solution that takes 142 time units to complete a project and in comparison that genetic algorithm implemented in this research will take 139 time units and the critical activities are 4, 7 and 17. While genetic algorithm has found the solution by projecting to complete the project in 139 time units, it tool more iterations than ant colony optimisation and simulated annealing. Ant colony optimisation has outperformed simulated annealing and tabu search. Although in previous evaluation simulated annealing has performed better in this case tabu search has achieved better results by finding a schedule which can complete the projects in 143 time units with 2, 9 and 17 as critical activities.

The data collection from implementing genetic algorithm, simulated annealing and tabu search was populated in Microsoft project 2007 to generate a resource flow histogram as illustrated in Figure 4.19, 4.20 and 4.21 respectively.

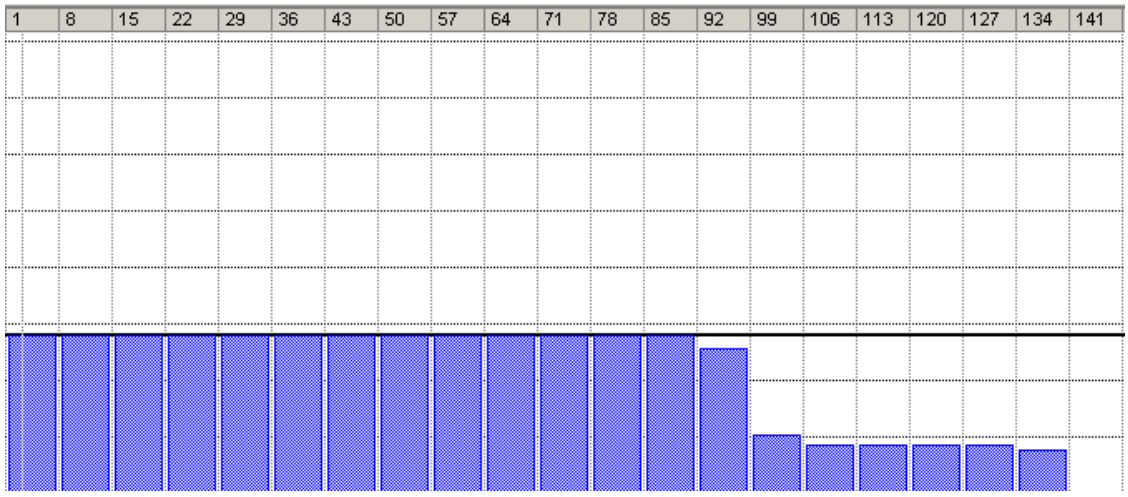

Figure 4.19Genetic Algorithm Constrained Resource-Flow

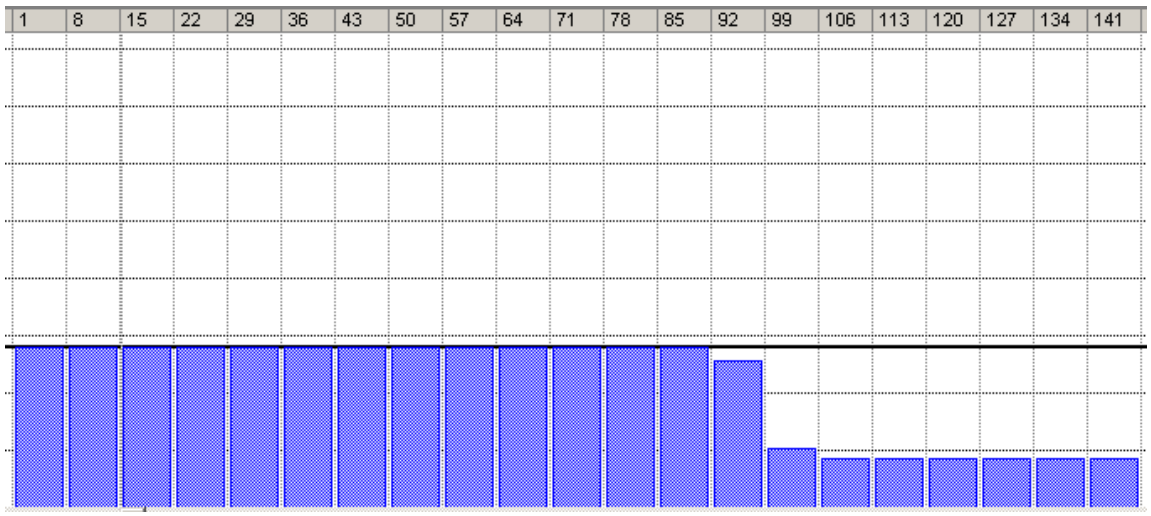

Figure 4.20 Simulated Annealing Constrained Resource-Flow

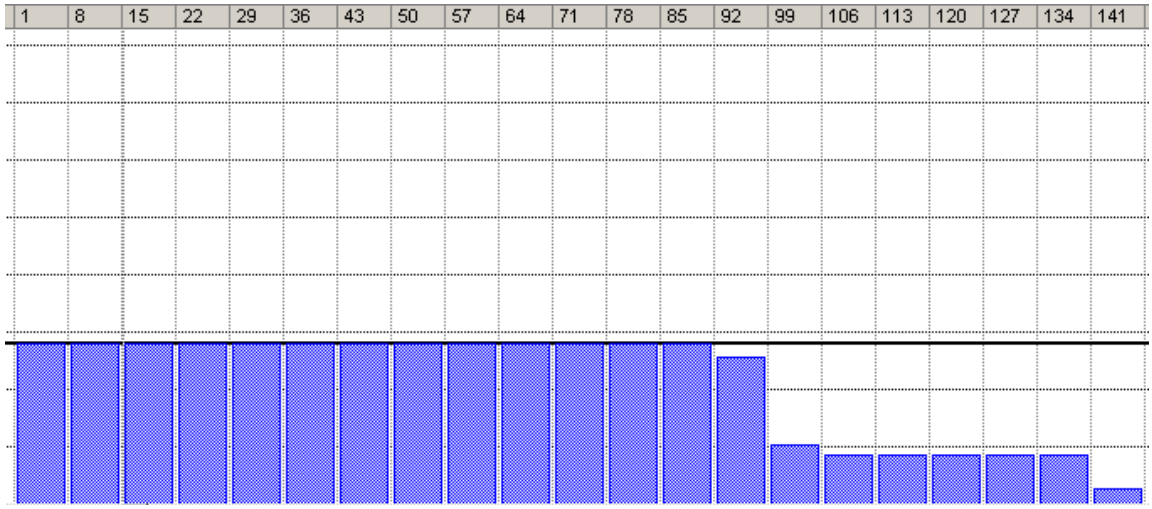

Figure 4.21 Tabu Search Constrained Resource-Flow

4.4.2 Cost Estimation in Project

This section evaluates the performance of the metaheuristics search techniques with a time-cost trade-off construction project scheduling problem which is discrete in nature. Hence to evaluate the algorithm, researcher has used data presented in Table 4.23. This data was also used by Elbeltagi, Hegazy & Grierson (2005) and Feng, Liu & Burns (1997) to solve discrete optimisation problem by implementing a number of different algorithms.

Activity	Depends	Option 1		Option 2		Option 3		Option 4		Option 5	
no.	on	Duration	Cost	Duration	Cost	Duration	Cost	Duration	Cost	Duration	Cost
1		14	2,400	15	2,150	16	1,900	21	1,500	24	1,200
2		15	3,000	18	2,400	20	1,800	23	1,500	25	1,000
3		15	4,500	22	4,000	33	3,200	33	3,200	33	3,200
4		12	45,000	16	35,000	20	30,000	20	30,000	20	30,000
5	1	22	20,000	24	17,500	28	15,000	30	10,000	30	10,000
6	1	14	40,000	18	32,000	24	18,000	24	18,000	24	18,000
7	5	9	30,000	15	24,000	18	22,000	18	22,000	18	22,000
8	6	14	220	15	215	16	200	21	208	24	120
9	6	15	300	18	240	20	180	23	150	25	100
10	2,6	15	450	22	400	33	320	33	320	33	320
11	7,8	12	450	16	350	20	300	20	300	20	300
12	5,9,10	22	2,000	24	1,750	28	1,500	30	1,000	30	1,000
13	3	14	4,000	18	3,200	24	1,800	24	1,800	24	1,800
14	4,10	9	3,000	15	2,400	18	2,200	18	2,200	18	2,200
15	12	12	4,500	16	3,500	16	3,500	16	3,500	16	3,500
16	13,14	20	3,000	22	2,000	24	1,750	28	1,500	30	1,000
17	11,14,15	14	4,000	18	3,200	24	1,800	24	1,800	24	1,800
18	16,17	9	3,000	15	2,400	18	2,200	18	2,200	18	2,200
Total		**100**	**169,820**	**131**	**136,705**	**159**	**107,650**	**166**	**101,178**	**169**	**99,740**

Table 4.23 Test Problem for discrete optimisation (Elbeltagi, Hegazy & Grierson, 2005)

The data presented above relates to a project which constitutes 18 activities and has been presented with five options of different cost and duration. In each case, the first option is the most expensive option but it will take the least number of days to complete the project and the fifth option is the cheapest option and it will take the longest to complete. For each task the project managers would have to choose from five options and this could traditionally be done using heuristics approaches, but to get most optimised solution, one of the five options will be selected for each task using genetic algorithm, simulated annealing and tabu search. As mentioned earlier, this data is related to time-cost trade-off problem and in real world the ideal set of solutions for this problem should look similar to Figure 4.22.

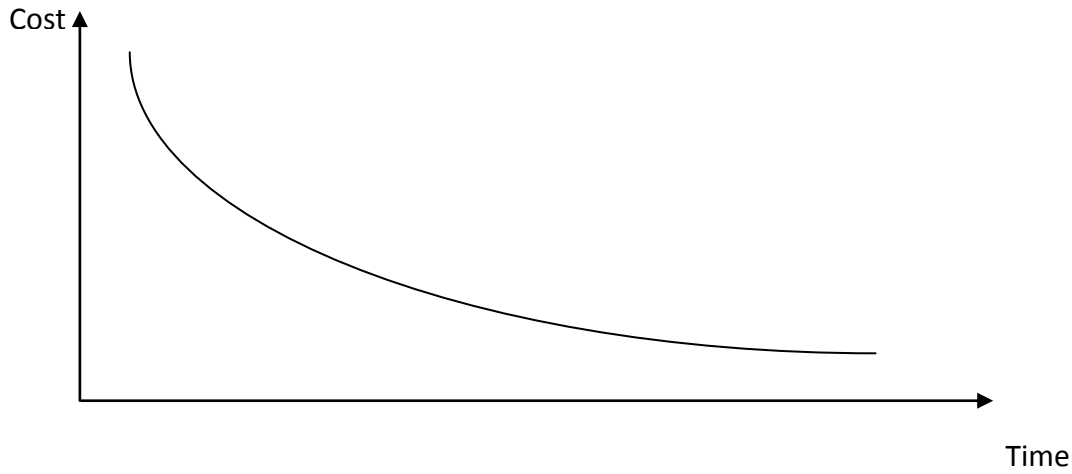

Figure 4.22 Relationship between Time and Cost of Activity (Trade off Curve)

This graph represents the pareto-optimal front of solutions. It is a unique line through the total set of solutions that represents what is considered to be non-dominated solutions. Each solution along the pareto-optimal front is equally valid in terms of how it trades off cost and time.

Before the evaluations are carried out, a critical path must be established for the 18 tasks mentioned in Table 4.23. The critical path is illustrated in Figure 4.23. In this critical path, information about cost and time is not displayed. However, the researcher has used this critical path as a template to input results generated from the evaluations. The critical path can be established using the values provided in column "Depends on" and the tasks which does not have values, indicates that they do not have any precedence.

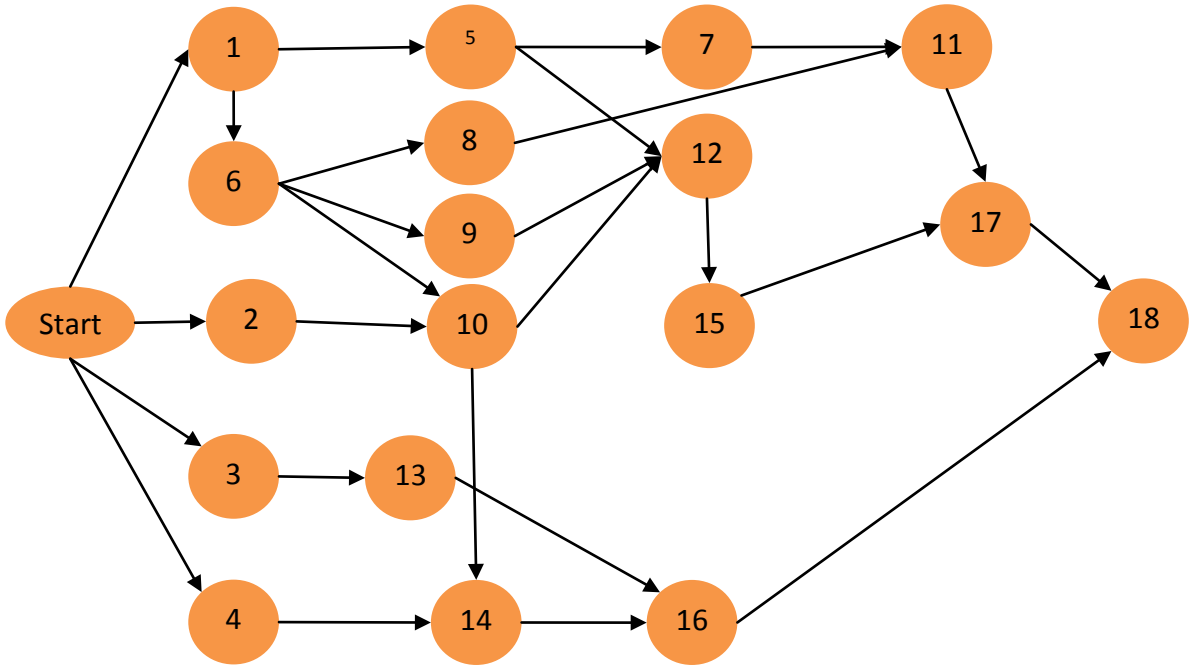

Figure 4.23 Critical Path Method for Data in Table 4.21

All the evaluations carried out must consider the critical path mentioned in Figure 4.23. In addition to this the main objective of this evaluation is to minimise the total cost of the project and to do this a fitness function has been used against each algorithm:

$$f(x) = MIN\left(T\ X\ I + \sum_{i=1}^{n} C_{ij}\right)$$

Equation 4.1 Cost Estimation Fitness Function

The variables mentioned in the above mentioned fitness function represents the following:

- n = number of activities
- C_{ij} = direct cost of activity I using its method of construction j
- T = total project duration
- I = daily indirect cost

Search techniques like genetic algorithm, simulated annealing and tabu search will be used to minimise the total cost of the project using the fitness function mentioned above. The underlying application and parameters for each algorithm is similar to the previous evaluations. The results generated from these evaluations will be compared against the results from evaluations carried out by Elbeltagi, Hegazy & Grierson (2005) and Feng, Liu & Burns (1997).

4.4.2.1 Results and Discussion

The summary of results generated from this evaluation is presented in Table 4.24. This table represents the minimum and average of project cost and duration. It also presents the percentage of success against other algorithm. The percentage of success is calculated based on numbers of days and total cost of the project. Hence, lower the total cost of project and duration, higher the success rate of the algorithm.

Algorithms	Minimum			Average			
	Duration	Cost	Iterations	Duration	Cost	Iterations	% Success
Genetic Algorithm	104	139,320	64	111	152,010	68	50
Simulated Annealing	110	145,820	77	118	156,310	80	30
Tabu Search	108	156,720	71	113	156,910	75	20

Table 4.24 Summary of Results

Genetic algorithm was the best performing algorithm by finding an option for project managers to complete the project in 104 days with total cost of 139,320 units with a success rate of 50%. Whereas best combination found by simulated annealing was to complete the project in 110 days with total cost of 145,820 units and a success rate of 30%. The best combination found by tabu search was to complete the project in 108 days with total cost of 156,720 units and a success rate of 20%. Although the combination found by tabu search enables the project to complete faster than simulated annealing. The cost of proposed combination from tabu search is more than simulated annealing. Hence, simulated annealing has a greater success rate than tabu search. These results can be compared with those of Feng, Liu & Burns (1997) who utilised a Genetic Algorithm to solve this problem and they discovered two non-dominated solutions, 100 days/$133,320 and 101 days/$129,320. The best solution found in the current research is very close to the pareto-optimal front for this problem. The above solutions appear to be an improvement when compared with the results of Elbeltagi, Hegazy & Grierson (2005). Their comparative study indicated that the Particle Swarm Optimisation algorithm was best at solving the problem; however the best solution it found was 110 days/ $161,270. The overall results achieved for each algorithm is presented in a time-cost trade-off curve as illustrated in Figure 4.24.

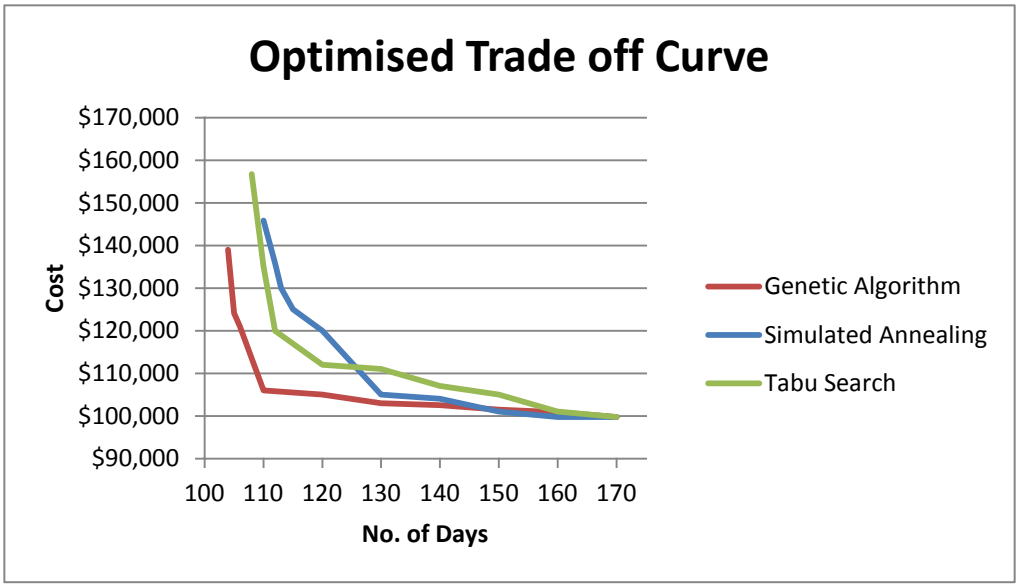

Figure 4.24 Optimised Trade-Off Curve

Figure 4.22, depicts the relationship between time and cost for a project and represents an ideal solution to a time-cost trade-off problem. Based on that, we can conclude that genetic algorithm, simulated annealing and tabu search all have been able to achieve an ideal relationship between

time and cost for a project management scenario. The final solution for each algorithm is illustrated in Figure 4.24 using time-cost trade-off curve. During the initial stages of evaluation, trade-off curve are generated, but they are scattered all over the solution space and does not gather into one region. As the evaluation progresses the trade-off curve takes shape like the relationship described in Figure 4.22. Genetic algorithm took 64 iterations to achieve final generation which produced the trade-off curve whereas simulated annealing took 77 iterations and tabu search took 71 iterations. An effective way to judge a performance of the algorithm is to ensure that the trade-off curve is closest to the axis (Feng, Liu & Burns, 1997). Hence looking at Figure 4.24, it is evident that trade-off curve for genetic algorithm has performed better than that of simulated annealing and tabu search. Performance between simulated annealing and tabu search is almost contradicting to itself whereby for results up to 120 days, trade-off curve for tabu search was closer to axis when compared against simulated annealing, but results after 120 days trade-off curve for simulated annealing was much closer to axis. So far, genetic algorithm has performed consistently well when compared against the previous evaluations, but in saying that genetic algorithm has not performed well when it was used to solve numerical test functions and n-queen problem. Simulated annealing was the second most favourable for this evaluation, and that is consistent with the previous evaluations except it was the most favourable choice of algorithm to solve n-queen problem. Based on the overall results, it is evident that tabu search is the least favourable choice of algorithm to solve the problems mentioned in this and previous exercises. In contrast to that Elbeltagi, Hegazy & Grierson (2005) have mentioned that tabu search has been used widely by many researchers to solve not only time-cost trade-off problem, but many other NP-hard problems. After finding the trade-off curve, project managers can determine the total cost of the project by summing up the estimated indirect cost and direct cost from trade-off curve. Using trade-off curve as the objective function allows for much more efficient evaluation of various other indirect costs.

5 Conclusions

This research has carried out experiments using variety of search techniques like genetic algorithm, simulated annealing and tabu search. These experiments focused on evaluating metaheuristics search techniques against continuous and discrete problems. Majority of the evaluations mainly focused around solving RCPSP using metaheuristics search techniques. The framework and design of the research was detailed in Chapter 3 which also included the architecture of the formulated tool to help collect data from various evaluations. The collected data was then used to compare performance against each search techniques. Based on the data collected over several evaluations it has increased the relevance of the results, and has assisted in answering the research questions that were set out at the beginning of the research.

Question 1:- Can resource allocation and cost estimation be effectively formulated as a search problem that is tractable to be solved using metaheuristic search algorithm?

Answer: According to discussions in section 4.4, metaheuristics search algorithms were used to solve problems with resource allocation and cost estimation in project management. To solve these problems, two separate project management scenario were re-formulated as a search problem. All the implemented search techniques were able to solve the problem. However, some search techniques found the solution faster than other search techniques.

Question2: Which metaheuristic search algorithms demonstrate the most desirable performance when solving the problems?

Answer: Based on the collected data it is clear that all implemented metaheuristics search techniques were able to solve the problem. Overall simulated annealing frequently achieved favourable results for more complex problems when compared against genetic algorithm and tabu search. The only exception to this is the bump problem whereby genetic algorithm out performed simulated annealing and tabu search. The overall performance of each algorithm against each evaluation is presented in Table 5.1 and it is clear that performance of simulated annealing has been concluded as best for six times whereas performance of genetic algorithm has been concluded as best for four times. The difference between performances is not huge between simulated annealing and genetic algorithm. In contrast to that, performance of tabu search has been concluded as best only for one time, a lot less than simulated annealing and genetic algorithm.

Evaluation Type	Best Performing Algorithm
Rastrigin	Simulated Annealing
Rosenbrock's Banana	Genetic Algorithm and Simulated Annealing
Schwefel	Genetic Algorithm
Axis Parallel Hyper-Ellipsoid	Tabu Search
Griewangk	Simulated Annealing
Himmelblaus	Simulated Annealing
Bump	Genetic Algorithm
N-Queen	Simulated Annealing
RCPSP	Genetic Algorithm and Simulated Annealing

Table 5.1 Metaheuristics Search Techniques Favourable Score against each Evaluation

It is possible to conclude that the complexity of the problem domain is clearly a contributing factor towards different algorithms performance. Of the algorithms tested, the Genetic Algorithm is the only population-based search method. For relatively straight forward problems, the population provides a computational overhead that is not justified. However, for more complex problems where the single-trajectory direct search methods become limited in their ability to identify a globally optimum solution, the use of a population-based method becomes advantageous.

To conclude the researcher draws the conclusion that overall performance of simulated annealing is better than the tabu search and genetic algorithm. The work implemented in this research was able to address the research questions and add to the body of knowledge regarding the solution of RCSCP using metaheuristic search algorithms. However, in "real-world" the project management scenarios are complicated and fairly dynamic hence a more deeps research is required to increase the confidence for using search base software engineering approach.

5.1 Limitations

This section describes the limitation of the research. The data and conclusion drawn from this research were carried out with some sample data mainly referenced from Christodoulou, 2010 and Elbeltagi, Hegazy & Grierson (2005). Although the underlying data was different for solving resource constraint project scheduling problem (section 4.4.1) and cost estimation in project planning (Section 4.4.2), the size of the data was not big enough to draw a general conclusion for this problem. Hence, it would be reasonable to state that project management practitioners would not have absolute confidence in the approach presented in this research. In addition to this, the application developed for this research may not be feasible for other project management software to implement. Additional restriction is based on the fact that the application was developed in a .NET Environment and will need modifications to make the implementation more generic so that wider audience can use it.

In addition to the size of the data and environment limitation, the resource constrained project scheduling problem and cost estimation problem were only solved using two main constraints like labour and costing options. But in the “real world” scenario, there are many more constraints which need sincere considerations while scheduling a project. Furthermore, all the experiments were carried out using three search techniques (genetic algorithm, simulated annealing, and tabu search) hence there were three different results presented for each experiments. This is a limitation because he project planners will have to choose the best option and because of the parameters supplied to the algorithms vary, the results may not always be consistent.

Finally, the implemented search techniques were specifically designed to exclude enhanced features that have been known to improve search performance. This was to allow a baseline set of results to be achieved against which future implementations can be compared to show the relative improvement for each enhancement.

5.2 Guidance for Future Research

This section of the paper describes the next step in the development of this research and will also highlight the possibility of new work which could add another facet to this research

Considering the aims established at the beginning of this research and the results obtained from the evaluation, it would be reasonable to state that there is still some need to develop the application in this research to the next level. So far, in the research the implemented application focused on optimising metaheuristics search techniques for various numerical test functions and to solve RCRSP. Hence the next step would be to enhance the scope of this research by implementing a natural evolution which can act as a new method to solve RCPSP. So for the research has only focused on constraints like human and cost, but for future it would be desirable to include additional constraints in project scheduling problem. In project scheduling most common constraints can exists whilst establishing the follows:

- Project team and sub teams
- Skills of team members
- Availability of third party resources
- Effort required from each team members

In project scheduling, the list of constraint is not limited to the list mentioned above, but they would be the most desirable to enhance the scope of this research. When project planners are scheduling a project, the above mentioned constraints have a negative impact on the due date of the project. Hence new methods should have the ability to identify critical elements of the project well in advance so that the project planners can take appropriate action to maintain the schedule of the project.

In addition, future work should also address the simplistic implementation of the algorithms used. In particular, enhanced multi-objective algorithms should be implemented and the performance gain measured relative to the baseline results. What is clear from the current research is that the performance of a given algorithm is closely tied to the complexity of the problem. Whilst multi-objective implementations of the algorithms, for example NSGA-II (Deb, Agrawal, Pratap & Meyarivan, 2000) may lead to further improvements in results, it remains to be proven that the use of such an algorithm is justifiable for RCSCP. The results in these thesis will allow the performance of such algorithms in terms of solution quality be evaluated in light of the additional computation cost of the implementation.

6 References

Afzal, W., Torkar, R., & Feldt, R. (2009). A systematic review of search-based testing for non-functional system properties. *Information and Software Technology, 51*(6), 957-976. doi:10.1016/j.infsof.2008.12.005

Alba, E., & Francisco Chicano, J. (2007). Software project management with GAs. *Information Sciences, 177*(11), 2380-2401. doi:10.1016/j.ins.2006.12.020

Arcuri, A., & Yao, X. (2008). Search based software testing of object-oriented containers. *Information Sciences, 178*(15), 3075-3095. doi:10.1016/j.ins.2007.11.024

Azar, D., Harmanani, H., & Korkmaz, R. (2009). A hybrid heuristic approach to optimize rule-based software quality estimation models. *Information and Software Technology, 51*(9), 1365-1376. doi:10.1016/j.infsof.2009.05.003

Barchiesi, D. (2009, 6-9 July 2009). Adaptive non-uniform, hyper-elitist evolutionary method for the optimization of plasmonic biosensors. Symposium conducted at the meeting of the Computers & Industrial Engineering, 2009. CIE 2009.

Barreto, A., Barros, M. d. O., & Werner, C. M. L. (2008). Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers & Operations Research, 35*(10), 3073-3089. doi:10.1016/j.cor.2007.01.010

Bianco, L., & Caramia, M. (2011). A new lower bound for the resource-constrained project scheduling problem with generalized precedence relations. *Computers & Operations Research, 38*(1), 14-20. doi:10.1016/j.cor.2009.07.003

Brooks, F. P. (1995). *The mythical man-month*. Addison-Wesley Longman Publishing Co., Inc.

Buche, D., Schraudolph, N. N., & Koumoutsakos, P. (2005). Accelerating evolutionary algorithms with Gaussian process fitness function models. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 35*(2), 183-194.

Chang, C., Christensen, M., & Zhang, T. (2001). Genetic Algorithms for Project Management. *Annals of Software Engineering, 11*(1), 107-139. doi:10.1023/a:1012543203763

Chang, C. K., Jiang, H.-y., Di, Y., Zhu, D., & Ge, Y. (2008). Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology, 50*(11), 1142-1154. doi:10.1016/j.infsof.2008.03.002

Christodoulou, S. (2010). Scheduling Resource-Constrained Projects with Ant Colony Optimization Artificial Agents. *Journal of Computing in Civil Engineering, 24*(1), 45-55. doi:10.1061/(asce)0887-3801(2010)24:1(45)

Clarke, J., Dolado, J. J., Harman, M., Hierons, R., Jones, B., Lumkin, M., ... Shepperd, M. (2003). Reformulating software engineering as a search problem. *Software, IEE Proceedings -, 150*(3), 161-175.

Collis, J., & Hussey, R. (2009). *Business research: a practical guide for undergraduate & post-graduate students* (3 ed.): Palgrave Macmillan, London.

Connor, A. M. (1999). "A multi-thread tabu search algorithm" Design Optimization. *International Journal of Product and Process Improvement, 1*(3), 293-304.

Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000). A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II Parallel Problem Solving from Nature PPSN VI. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, & H.-P. Schwefel (Eds.), (Vol.

1917, pp. 849-858): Springer Berlin / Heidelberg. Retrieved from http://dx.doi.org/10.1007/3-540-45356-3_83. doi:10.1007/3-540-45356-3_83

Farmer, J. D., Packard, N. H., & Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena, 22*(1-3), 187-204. doi:10.1016/0167-2789(86)90240-x

Feng, C.-W., Liu, L., & Burns, S. A. (1997). Using Genetic Algorithms to Solve Construction Time-Cost Trade-Off Problems. *Journal of Computing in Civil Engineering, 11*(3), 184-189. doi:10.1061/(asce)0887-3801(1997)11:3(184)

Frolkovič, P. (1990). Numerical recipes: The art of scientific computing. *Acta Applicandae Mathematicae, 19*(3), 297-299. doi:10.1007/bf01321860

Gill, P. E., Murray, W., & Wright, M. H. (1981). Practical optimization. Academic Press. doi:citeulike-article-id:1055426.

Glover, F. (1989). Tabu Search (Part I). *ORSA Journal on Computing, 1*(3), 190-206.

Glover, F. (1990). Tabu Search (Part II). *ORSA Journal on Computing, 2*(1), 4-32.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, Boston, MA.

Gueorguiev, S., Harman, M., & Antoniol, G. (2009. *Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering*. Presented at the meeting of the Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montreal, Canada. doi:10.1145/1569901.1570125

Harman, M. (2007, 20-26 May 2007). Automated Test Data Generation using Search Based Software Engineering. Symposium conducted at the meeting of the Automation of Software Test , 2007. AST '07. Second International Workshop on

Harman, M. (2007. *The Current State and Future of Search Based Software Engineering*. presented at the meeting of the 2007 Future of Software Engineering, doi:10.1109/fose.2007.29

Harman, M., & Jones, B. F. (2001). Search-based software engineering. *Information and Software Technology, 43*(14), 833-839. doi:10.1016/s0950-5849(01)00189-6

Harman, M., Sung Gon, K., Lakhotia, K., McMinn, P., & Shin, Y. (2010, 6-10 April 2010). Optimizing for the Number of Tests Generated in Search Based Test Data Generation with an Application to the Oracle Cost Problem. Symposium conducted at the meeting of the Software Testing, Verification, and Validation Workshops (ICSTW).

Herroelen, W. (2005). Project Scheduling—Theory and Practice. *Production and Operations Management, 14*(4), 413-432. doi:10.1111/j.1937-5956.2005.tb00230.x

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly, 28*(1), 75-105.

Holz, H. J., Applin, A., Haberman, B., Joyce, D., Purchase, H., & Reed, C. (2006a). Research methods in computing: what are they, and how should we teach them? *SIGCSE Bull., 38*(4), 96-114. doi:10.1145/1189136.1189180

Holz, H. J., Applin, A., Haberman, B., Joyce, D., Purchase, H., & Reed, C. (2006b. *Research methods in computing: what are they, and how should we teach them?* Presented at the meeting of the Working group reports on ITiCSE on Innovation and technology in computer science education, Bologna, Italy. doi:10.1145/1189215.1189180

Hooke, R., & Jeeves, T. A. (1961). "Direct Search" Solution of Numerical and Statistical Problems. *J. ACM, 8*(2), 212-229. doi:10.1145/321062.321069

Huang, S.-J., Chiu, N.-H., & Chen, L.-W. (2008). Integration of the grey relational analysis with genetic algorithm for software effort estimation. *European Journal of Operational Research, 188*(3), 898-909. doi:10.1016/j.ejor.2007.07.002

Jae-kwan Lee and Yeong-Dae Kim (1996) “Search Heuristics for Resource Constrainted Project Scheduling”, JORS.

Kanmani, S., & Maragathavalli, P. (2010). Search Based Software Test Data Generation Using Evolutionary Testing Techniques. *International Journal of Software Engineering (IJSE), 1*(5).

Keane, A. J. (1994). Experiences With Optimizers In Structural Design. *Adaptive Computing in Engineering Design and Control - Pymouth, U.K.*

Kennedy, J., & Eberhart, R. (1995, Nov/Dec 1995). Particle swarm optimization. Symposium conducted at the meeting of the Neural Networks.

Kılıç, M., Ulusoy, G., & Şerifoğlu, F. S. (2008). A bi-objective genetic algorithm approach to risk mitigation in project scheduling. *International Journal of Production Economics, 112*(1), 202-216. doi:10.1016/j.ijpe.2006.08.027

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, 220*(4598), 671-680. doi:10.1126/science.220.4598.671

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling. An update. *European Journal of Operational Research, 174*(1), 23-37. doi:10.1016/j.ejor.2005.01.065

Krogmann, K., Kuperberg, M., & Reussner, R. (2010). Using Genetic Search for Reverse Engineering of Parametric Behavior Models for Performance Prediction. *Software Engineering, IEEE Transactions on, 36*(6), 865-877.

Laguna, M. (1994). Clustering for the Design of SONET Rings in Interoffice Telecommunications. *Management Science, 40*(11), 1533-1541.

Li, Y. F., Xie, M., & Goh, T. N. (2007, 2-4 Dec. 2007). A study of genetic algorithm for project selection for analogy based software cost estimation. Symposium conducted at the meeting of the Industrial Engineering and Engineering Management, 2007 IEEE.

Li, Y. F., Xie, M., & Goh, T. N. (2009). A study of mutual information based feature selection for case based reasoning in software cost estimation. *Expert Systems with Applications, 36*(3, Part 2), 5921-5931. doi:10.1016/j.eswa.2008.07.062

Lin, S., & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research, 21*(2), 498-516.

Lokan, C. (2005, 1-1 Sept. 2005). What should you optimize when building an estimation model? Symposium conducted at the meeting of the Software Metrics, 2005. 11th IEEE International Symposium

Luenberger, D. (1984). *Linear and Nonlinear Programming.* (2 ed.) Addison-Wesley.

Martinjak, I., & Golub, M. (2007, 25-28 June). Comparison of Heuristic Algorithms for the N-Queen Problem. Symposium conducted at the meeting of the Information Technology Interfaces, 2007. ITI 2007.

May, L. J. (1998). Major Causes of Software Project Failures. Retrieved from http://paul-hadrien.info/backup/LSE/IS%20470/litterature%20review/MajorCausesofSoftwareProjectFailures.pdf

McMinn, P. (2004). Search-based software test data generation: a survey. *Software Testing, Verification and Reliability, 14*(2), 105-156. doi:10.1002/stvr.294

Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *Evolutionary Computation, IEEE Transactions on, 6*(4), 333-346.

Miller, W., & Spooner, D. L. (1976). Automatic Generation of Floating-Point Test Data. *Software Engineering, IEEE Transactions on, SE-2*(3), 223-226.

Nelder, J. A., & Mead, R. (1965). A Simplex Method for Function Minimization. *The Computer Journal, 7*(4), 308-313. doi:10.1093/comjnl/7.4.308

Nunamaker, J. F., Jr., & Chen, M. (1990, 2-5 Jan 1990). Systems development in information systems research. Symposium conducted at the meeting of the System Sciences, 1990., Proceedings of the Twenty-Third Annual Hawaii International Conference.

Nunamaker, J. F., Jr., Chen, M., & Purdin, T. P. M. (1991). Systems development in information systems research. *Journal of Managment Information Systems, 7*(3), 89-106.

ÖZdamar, L., & Ulusoy, G. (1995). A survey on the resource-constrained project scheduling problem. *IIE Transactions, 27*(5), 574-586. doi:10.1080/07408179508936773

Peram, T., Veeramachaneni, K., & Mohan, C. K. (2003, 24-26 April 2003). Fitness-distance-ratio based particle swarm optimization. Symposium conducted at the meeting of the Swarm Intelligence Symposium, 2003. SIS '03.

Pinto, G., Ainbinder, I., & Rabinowitz, G. (2009). A genetic algorithm-based approach for solving the resource-sharing and scheduling problem. *Computers & Industrial Engineering, 57*(3), 1131-1143. doi:10.1016/j.cie.2009.05.003

Poli, R. (2008). Analysis of the publications on the applications of particle swarm optimisation. *J. Artif. Evol. App., 2008*, 1-10. doi:10.1155/2008/685175

Powell, M. J. D. (1977). Restart procedures for the conjugate gradient method. *Mathematical Programming, 12*(1), 241-254. doi:10.1007/bf01593790

Rosenbrock, H. H. (1960). An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal, 3*(3), 175-184. doi:10.1093/comjnl/3.3.175

Ross, K. W., & Tsang, D. H. K. (1989). The stochastic knapsack problem. *Communications, IEEE Transactions on, 37*(7), 740-747.

Sagarna, R., Arcuri, A., & Xin, Y. (2007, 25-28 Sept. 2007). Estimation of distribution algorithms for testing object oriented software. Symposium conducted at the meeting of the Evolutionary Computation, 2007. CEC 2007. IEEE Congress on

Shi, Y., & Eberhart, R. (1998, 4-9 May 1998). A modified particle swarm optimizer. Symposium conducted at the meeting of the Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence.

Takahashi, O., & Kobayashi, S. (2001, 2001). An adaptive neighboring search using crossover-like mutation for multi modal function optimization. Symposium conducted at the meeting of the Systems, Man, and Cybernetics.

Tamura, K., & Yasuda, K. (2011). Primary study of spiral dynamics inspired optimization. *IEEJ Transactions on Electrical and Electronic Engineering, 6*(S1), S98-S100. doi:10.1002/tee.20628

Trautmann, N., & Baumann, P. (2009, 6-9 July 2009). Resource-allocation capabilities of commercial project management software: An experimental analysis. Symposium conducted at the meeting of the Computers & Industrial Engineering, 2009. CIE 2009.

Wirth, N. (2008). A Brief History of Software Engineering. *Annals of the History of Computing, IEEE, 30*(3), 32-39.

Wong, S., Aaron, M., Segall, J., Lynch, K., & Mancoridis, S. (2010, 13-16 Oct. 2010). Reverse Engineering Utility Functions Using Genetic Programming to Detect Anomalous Behavior in Software. Symposium conducted at the meeting of the Reverse Engineering (WCRE), 2010 17th Working Conference on

Yang, X.-S. (2009). Firefly Algorithms for Multimodal Optimization Stochastic Algorithms: Foundations and Applications. In O. Watanabe & T. Zeugmann (Eds.), (Vol. 5792, pp. 169-178): Springer Berlin / Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-04944-6_14. doi:10.1007/978-3-642-04944-6_14

Yeo, R. (2002). The tangibles and intangibles of organisational performance. Retrieved from http://www.emeraldinsight.com.ezproxy.aut.ac.nz/journals.htm?articleid=882884&show=abstract

Yong-Hua, S., & Irving, M. R. (2001). Optimisation techniques for electrical power systems. II. Heuristic optimisation methods. *Power Engineering Journal, 15*(3), 151-160.

Zong Woo Geem, Joong Hoon Kim, & Loganathan, G. V. (2001). A New Heuristic Optimization Algorithm: Harmony Search. *SIMULATION, 76*(2), 60-68. doi:10.1177/003754970107600201

7 Appendix

7.1 Implementation of Rastrigin Numerical Test Function

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Master.Thesis.EvaluationFunction.Functions
{
    public class Rastrigin : FunctionProvider
    {

        #region Methods

        public override double ObjFunction(double[] Values)
        {
            // ---- Declarations ----
            double ReturnValue;

            // ---- Throw exception if its greater then 2 ----
            if (Values.Count() > 2)
            {
                throw new Exception("Incorrect number of parameters, Function
cancelled.");
            }

            // ---- Set the function value ----
            ReturnValue = (Values[0] * Values[0]) + (Values[1] * Values[1]) - (Math.Cos(18
* Values[0])) - ((float)Math.Cos(18 * Values[1]));

            // ---- Return the value ----
            return ReturnValue;
        }

        #endregion

    }
}
```

7.2 Implementation of Rosenbrock's Banana Numerical Test Function

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Master.Thesis.EvaluationFunction.Functions
{
    public class RosenbrocksBanana : FunctionProvider
    {

        #region Methods

        public override double ObjFunction(double[] Values)
        {
            // ---- Declarations ----
            double ReturnValue;

            // ---- Throw exception if its greater then 2 ----
            if (Values.Count() > 2)
```

```
            {
                throw new Exception("Incorrect number of parameters, Function
cancelled.");
            }

            // ---- Set the function value ----
            ReturnValue = (0.01 * (100 * ((Values[1] - (Values[0] * Values[0])) *
(Values[1] - (Values[0] * Values[0]))) + ((1 - Values[0]) * (1 - Values[0]))));

            // ---- Return the value ----
            return ReturnValue;
        }


        #endregion

    }
}
```

7.3 Implementation of Schwefel Numerical Test Function

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Master.Thesis.EvaluationFunction.Functions
{
    public class Schwefel : FunctionProvider
    {

        #region Methods

        public override double ObjFunction(double[] Values)
        {
            // ---- Declarations ----
            double ReturnValue = 0;

            // ---- Throw exception if its greater then 2 ----
            if (Values.Count() > 2)
            {
                throw new Exception("Incorrect number of parameters, Function
cancelled.");
            }
            // ---- Set the function value ----
            // ---- Loop i through n incrementing by 1. Get the subtotal based on the
formula. ----
            for (int i = 0; i <= Values.Count()-1; i++)
            {
                ReturnValue += Values[i] * (Math.Sin(Math.Sqrt(Math.Abs(Values[i]))));
            }

            // ---- Return the value ----
            return ReturnValue*(-1.0);
        }

        #endregion

    }
}
```

7.4 Implementation of Axis Parallel Hyper-Ellipsoid Numerical Test Function

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Master.Thesis.EvaluationFunction.Functions
{
    public class AxisParallelHyperEllipsoid: FunctionProvider
    {

        #region Methods

        public override double ObjFunction(double[] Values)
        {
            // ---- Declarations ----
            double ReturnValue;

            // ---- Throw exception if its greater then 2 ----
            if (Values.Count() > 2)
            {
                throw new Exception("Incorrect number of parameters, Function
cancelled.");
            }

            // ---- Set the function value ----
            ReturnValue = ((Values[0] * Values[0]) + (2* (Values[1] * Values[1])));

            // ---- Return the value ----
            return ReturnValue;
        }

        #endregion

    }
}
```

7.5 Implementation of Griewangk Numerical Test Function

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Master.Thesis.EvaluationFunction.Functions
{
    public class Griewangks: FunctionProvider
    {

        #region Methods

        public override double ObjFunction(double[] Values)
        {
            // ---- Declarations ----
            double ReturnValuePart1;
            double ReturnValuePart2;
            double ReturnValue;

            // ---- Throw exception if its greater then 2 ----
            if (Values.Count() > 2)
            {
                throw new Exception("Incorrect number of parameters, Function
cancelled.");
            }

            // ---- Set the function value ----
```

```
            ReturnValuePart1 = ((Values[0] * Values[0]) + (Values[1] * Values[1]))/4000;
            ReturnValuePart2 = ((Math.Cos(Values[0] / Math.Sqrt(1))) * (Math.Cos(Values[1]
/ Math.Sqrt(2)))) +1;
            ReturnValue = ReturnValuePart1 + ReturnValuePart2;

            // ---- Return the value ----
            return ReturnValue;
        }

        #endregion

    }
}
```

7.6 Implementation of Himmelblaus Numerical Test Function

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Master.Thesis.EvaluationFunction.Functions
{
    public class Himmelblaus: FunctionProvider
    {

        #region Methods

        public override double ObjFunction(double[] Values)
        {
            // ---- Declarations ----
            double ReturnValuePart1;
            double ReturnValuePart2;
            double ReturnValue;

            // ---- Throw exception if its greater then 2 ----
            if (Values.Count() > 2)
            {
                throw new Exception("Incorrect number of parameters, Function
cancelled.");
            }

            // ---- Set the function value ----
            ReturnValuePart1 = Math.Cos((Values[0] * Values[0]) + Values[1] - 11) *
((Values[0] * Values[0]) + Values[1] - 11);
            ReturnValuePart2 = Math.Cos(Values[0] * (Values[1] * Values[1]) - 7) *
(Values[0] * (Values[1] * Values[1]) - 7);
            ReturnValue = ReturnValuePart1 + ReturnValuePart2;

            // ---- Return the value ----
            return ReturnValue;
        }

        #endregion

    }
}
```

7.7 Implementation of Bump Numerical Test Function

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace Master.Thesis.EvaluationFunction.Functions
{
    public class Bump: FunctionProvider
    {

        #region Methods

        public override double ObjFunction(double[] Values)
        {
            // ---- Declarations ----
            int intCounter;
            double ReturnValuePart1 = 0, ReturnValuePart2 = 0, ReturnValuePart3 = 0,
ReturnValue;

            // ---- Loop through each value ----
            for (intCounter = 0; intCounter < Values.Length; intCounter++)
            {
                ReturnValuePart1 += Math.Abs(Math.Cos(Values[intCounter]) *
Math.Cos(Values[intCounter]) * Math.Cos(Values[intCounter]) *
Math.Cos(Values[intCounter]));
                ReturnValuePart2 = ReturnValuePart2 * Math.Cos(Values[intCounter]) *
Math.Cos(Values[intCounter]);
                ReturnValuePart3 += (intCounter + 1) * Values[intCounter] *
Values[intCounter];
            }

            // ---- Get the square root for ReturnValuePart3 ----
            ReturnValuePart3 = Math.Sqrt(ReturnValuePart3);

            // ---- Set the return value ----
            ReturnValue = ((ReturnValuePart1 - 2 * ReturnValuePart2) / ReturnValuePart3);

            // ---- Return the function value. Multiply by -1 if the value is negative. --
--
            if (ReturnValue < 0) { return ReturnValue * -1; }
            else { return ReturnValue; }
        }

        #endregion

    }
}
```