Dynamic Scheduling and Planning parallel observations on large Radio Telescope Arrays with the Square Kilometre Array in mind

Johannes Buchner

A thesis submitted to
Auckland University of Technology
in partial fulfilment of the requirements for the degree of
Master of Computer and Information Sciences (MCIS)

2011

School of Computing and Mathematical Sciences
Auckland University of Technology, New Zealand

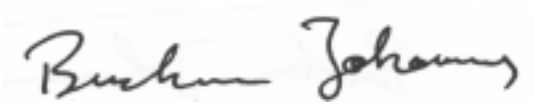
Contents

List of Figures

List of Tables

List of Algorithms

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.



Johannes Buchner

Acknowledgements

Foremost, I am grateful to my parents for supporting my studies. I would not be where and who I am without you.

I would like to thank my supervisor, Sergei Gulyaev, for his support, criticism and patience. I heartily thank Tim Natusch for introducing me to the nuts and bolts of Radio Astronomy. I would like to thank the institute and the university for supporting my research and travel financially.

This research would not have been possible without the great support of Colby Gutierrez-Kraybill from ATA, Phillip Edwards from ATCA/CSIRO, and the many responders to my survey: Ruud Overeem, Arjo de Vries associated with the LOFAR project, Alexander Neidhardt and Martin Ettl (Wettzell), Robert Hawkins (HST), Stefan Schlenker (LHC).

Abstract

Scheduling, the task of producing a time table for resources and tasks, is well-known to be a difficult problem the more resources are involved (a NP-hard problem). This is about to become an issue in Radio astronomy as observatories consisting of hundreds to thousands of telescopes are planned and operated.

The Square Kilometre Array (SKA), which Australia and New Zealand bid to host, is aiming for scales where current approaches – in construction, operation but also scheduling – are insufficient. Although manual scheduling is common today, the problem is becoming complicated by the demand for (1) independent sub-arrays doing simultaneous observations, which requires the scheduler to plan parallel observations and (2) dynamic re-scheduling on changed conditions. Both of these requirements apply to the SKA, especially in the construction phase.

We review the scheduling approaches taken in the astronomy literature, as well as investigate techniques from human schedulers and today's observatories. The scheduling problem is specified in general for scientific observations and in particular on radio telescope arrays. Also taken into account is the fact that the observatory may be oversubscribed, requiring the scheduling problem to be integrated with a planning process.

We solve this long-term scheduling problem using a time-based encoding that works in the very general case of observation scheduling. This research then compares algorithms from various approaches, including fast heuristics from CPU scheduling, Linear Integer Programming and Genetic algorithms, Branch-and-Bound enumeration schemes. Measures include not only goodness of the solution, but also scalability and re-scheduling capabilities.

In conclusion, we have identified a fast and good scheduling approach that allows (re-)scheduling difficult and changing problems by combining heuristics with a Genetic algorithm using block-wise mutation operations. We are able to explain and eradicate two problems in the literature: The inability of a GA to properly improve schedules and the generation of schedules with frequent interruptions.

Finally, we demonstrate the scheduling framework for several operating telescopes: (1) Dynamic re-scheduling with the AUT Warkworth 12m telescope, (2) Scheduling for the Australian Mopra 22m telescope and scheduling for the Allen Telescope Array. Furthermore, we discuss the applicability of the presented scheduling framework to the Atacama Large Millimeter/submillimeter Array (ALMA, in construction) and the SKA. In particular, during the development phase of the SKA, this dynamic, scalable scheduling framework can accommodate changing conditions.

Part I

Introduction

1 Overview

1.1 Radio astronomy

Radio astronomy is the field of astronomy that observes radio waves – electromagnetic radiation with wavelengths of millimeters to meters – from extraterrestrial sources. In contrast the eye observes photons of 300-700nm wavelength (Kitchin (1998)). Due to these more practical dimensions in radio astronomy, theoretically any conductor of suitable size can be used as an observation instrument. In practice radio astronomy instruments take various forms.

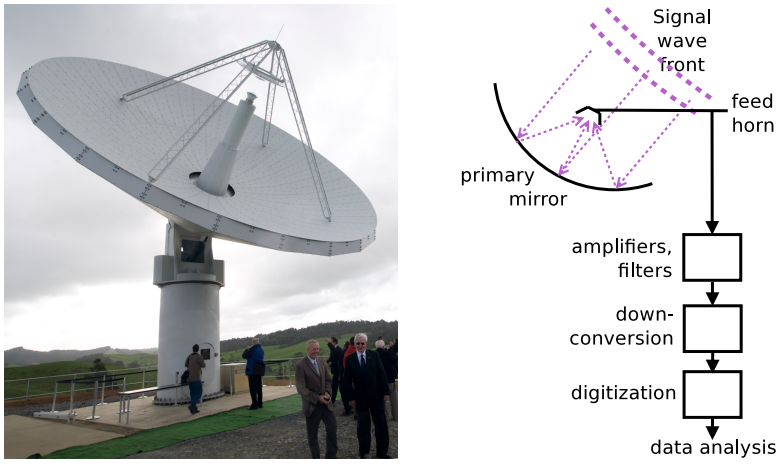


Figure 1: A typical radio telescope design with primary mirror and feed horn in the focus. The feed horn has to be supported by struts which interfere with the signal. The signal is filtered and amplified in place, and transferred to a data center, where it is down-converted, digitized and finally, analysed. Left panel: Warkworth 12m antenna. Picture courtesy of Institute for Radio Astronomy and Space Research (IRASR, AUT University). Right panel: A block diagram of a single dish antenna.

However, the most well-known radio astronomy observation instrument is the radio telescope dish. It is based on a mirror that reflects radiation onto a feed horn, which collects the radiation to be amplified and translated into science data (see Figure 1).

It is true in general that the signal to noise ratio is dependent on the number of sampling points obtained, since some form of averaging is involved: $Noise \propto \frac{1}{\sqrt{n_S}}$. The number of samples is proportional to the duration, the sampling rate. If several receivers are used, this number goes in too: $n_S = n_R \times T \times f_S$. Given fixed equipment, only the observation time can be increased, and only if it is quadrupled, the noise is halved. It would seem obvious that more collecting area / more telescopes contribute to a better signal to noise ratio roughly linearly (see Kraus (1966) for details).

1.1.1 Interferometry and Radio telescope arrays

Interferometry is a powerful tool to increase the (angular) resolution and to reduce the noise further. By linking two or more antennas together, and assuming the same signal reaches them – although with some delay – this common signal can be extracted through correlation, as all individual noise contributions are uncorrelated (Kitchin (1998)). The angular resolution increases with the distance between the telescopes. The image detail recoverable increases with diversity of baselines (distance vectors between antennas).

Hence, given a set of antennas, there is a trade-off to be made:

- In a *compact configuration*, the angular resolution is low. However, the combinations of antennas cover a small range of baselines well, a image would show large structure well. Since the photons travel through the same air paths, the noise can be removed very effectively through correlation.
- In a *distant configuration*, the angular resolution is high, so an image of the sky would show many details of a central region. But the combinations of antennas cover the range of baselines only sparsely, so the general structure would be less defined. Signals have higher noise. Very large baseline interferometry (VLBI) operates in such a distant configuration, by linking together telescopes throughout a continent or around the globe. Various VLBI networks are active.

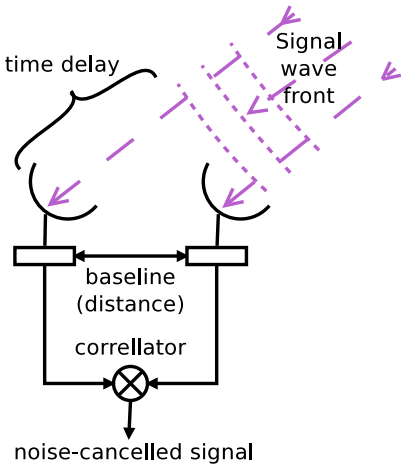


Figure 2: Simplified concept of an interferometer.

Arrays usually aim for a compromise between the two cases. For the case of the ATCA (Australian Telescope Compact Array), the 6 antennas can be moved to a compact star configuration of a few dozen meters maximum distance, or to an expanded configuration covering several kilometers. For the ATA (Allen Telescope Array), 42 antennas are distributed in an area of a few hundred meters in a quasi-random fashion, to allow good coverage of baselines.

1.1.2 Square Kilometre Array (SKA)

The Square Kilometre Array (SKA) will consist of several components, covering a range of radio wavelengths:

1. Dense aperture array,
2. Sparse aperture array and

3. Dishes.

Only the radio telescope dishes are considered in this work. Their configuration is made of two components:

- There will be a *core region*, similar in concept to today's radio telescope arrays, but several kilometer diameter in size and filled with thousands of dishes.
- Furthermore, 20 to 25 *stations* – clusters of telescopes – with about 30 to 50 telescopes each, will be distributed across the continent, extending the maximum distance to thousands of kilometers.

With those two components, both aspects mentioned above are well-covered, making the SKA an extremely powerful science tool.

The Square Kilometre Array, as a next-generation observation instrument, is magnitudes larger in size than current arrays, several of which, called precursors and/or pathfinders, are testing various concepts, materials, manufacturing, techniques and technologies for the SKA project. For instance, the ATA developed a cheap telescope manufacturing technique to get a high number of small telescopes (42 so far) at a low price.

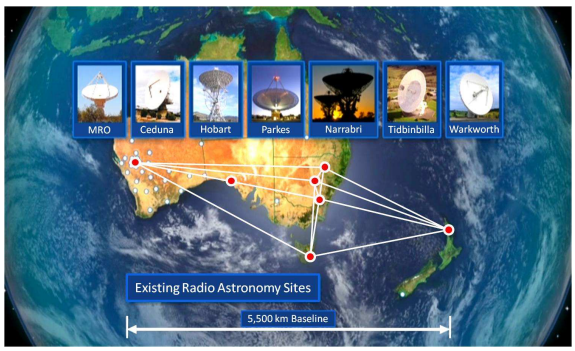


Figure 3: If placed in Australia, the core region would be where today's ASKAP site is, near Perth in Western Australia. Red dots indicate already operating telescope sites. White dots indicate the spirals of telescope centers extending up to New Zealand and providing an enormous baseline of 5500km. Courtesy of CSIRO/anzSKA/AUT.

Besides the hardware, much work is needed on data processing and data analysis. Radio telescopes are notorious for high data volumes (several gigabytes per second), and linking the data streams together requires new data centers and better algorithms. This is the reason computer scientists and the ICT industry are excited about this project, but such a projects incurs, with the required infrastructure and staff, consequences and opportunities for a country as a whole.

1.2 Planning, Scheduling, Control and Monitoring

Operating complex equipment is well studied from the manufacturing industry. Following Wall (1996), the process can be split up into four segments: Planning, which defines the tasks to do including their restrictions, Scheduling, where it is decided when and how to do these tasks, Control, where the tasks are executed, and Monitoring, which observes the execution and provides a feedback loop to alter the control process, or to demand a different schedule.

1.2.1 Planning

In most general terms, the tasks that could be done are separated from the tasks that are planned to be done. The outcome is the plan.

In manufacturing, the tasks or jobs are the items to produce. In planning, the variety of items and quantities are decided.

For observations, the tasks are proposals. Scientists interested in using the equipment submit an proposal, which can consist of one or more observations. These are specified with the scientific requirements, and the total hours needed to reach the science goals.

Typically, a committee decides to either grant execution of a set of observations, or assigns a priority or grade to each proposal and leaves it to the scheduler to select high-grade observations before low-grade ones.

1.2.2 Scheduling

A scheduler takes the plan and assigns execution times as well as resources (e.g. equipment) to the tasks. The result is called the schedule.

Two aspects are of importance in creating a schedules: Validity and Goodness.

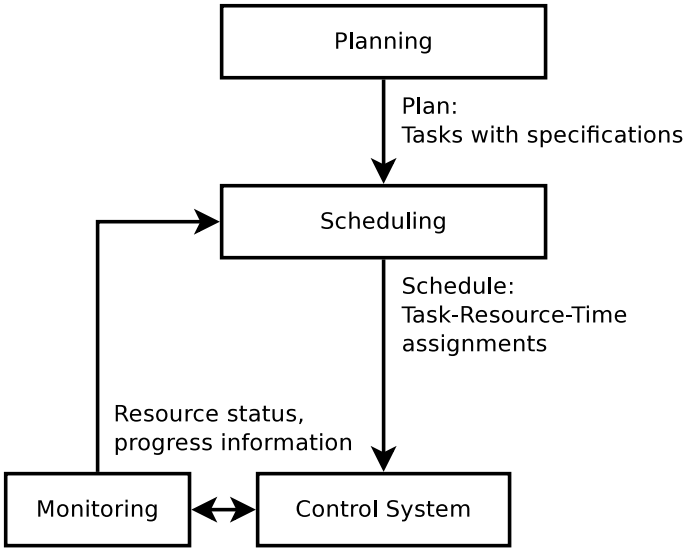


Figure 4: The feedback loop between planning, scheduling, control/execution and monitoring. Planning can be more integrated than depicted here.

Validity: The schedule has to be valid, i.e. each task can actually be executed as the schedule states. Time constraints, order constraints and resource constraints have to be adhered to.

In project scheduling for instance, there is high interdependence between the tasks (see Figure 5 on page 17 for an illustrative Gantt-diagram). The scheduler is strongly constrained in the possible orderings of tasks.

In job shop scheduling, the manufacturing items have to be assigned to machines. Here, the order of machines visited need to be taken into account, as well as the resources needed to do a certain job at a certain machine at a certain time.

However, for observation scheduling, the assumption can be made that there are no interdependencies, i.e. that the measurements are independent.

Goodness: The goodness of a schedule is the critical Figure of merit that a scheduler has to take into account. It is usually expressed as a cost function (to be minimized) or a preference function (to be maximized).

This measure is highly dependent on the application area, and has to be specified by stakeholders with domain knowledge (usually the operators). In general, the time needed and resources required are to be minimized. A full discussion on the cost function for the case of radio astronomy scheduling can be found in section 7 on page 65.

For a scheduler, the time interval considered can be unlimited or limited. In project scheduling for instance, the time axis is open but the total time is to be minimized. For observation scheduling, even if the time is split into limited terms, it is possible to carry on unfinished or unscheduled observations to the next term.

If planning is intertwined with scheduling, the schedule is potentially oversubscribed, meaning not all tasks can be completed. It is up to the scheduler to prefer completing high priority tasks before starting low priority tasks.

1.2.3 Control/Execution

The control system is a highly complex, often distributed system that has to make sure all the pieces of equipment work in accord. It takes the task to execute at the current time and sets up the machines and instruments to their correct state.

The relevant properties of a reliable control systems for the moment is that if the task can be executed, the control system executes it.

1.2.4 Monitoring

The monitoring system obtains qualitative and quantitative indicators from the pieces of equipment about the state of operation. This information is logged for later analysis, but also shared with the control system to allow it to react to misalignments and failures. The notifications can not only reach the control system, but also the scheduler, if resources are permanently unavailable and a change in schedule is required.

1.3 Thesis aim and structure

This work is concerned with the issue of planning and scheduling observations on radio telescope arrays. Planning is integrated, as, with oversubscription on observatories, the scheduler also has to decide which tasks to exclude.

With the Square Kilometre Array, the field of scheduling today faces the issue of scalability. It is clear that a semi-manual scheduling system for a dozen instruments that is updated once in a month will not work for a complex system with up to 1000 instruments that are not staying constant (construction process, equipment failure or other circumstances). This demands scalable and flexible solutions. Even though humans are in charge of decisions and preferences, the scheduling can not be accommodated by humans any more. This work is also the first to consider dynamic array splitting – part of the specification of the SKA – allowing parallel observations to be scheduled.

The aim of this work is to develop a statement for the real-world problem of scheduling observations, and to evaluate and compare various approaches to solving this problem. Finally, real-world application of the work on today's observatories is explored for various cases.

The rest of this work is organised as follows:

Chapter 2 defines scheduling, its terms and several real world problems, followed by an investigation of the fundamental problem and its properties.

In Chapter 3, specifics to the problem of scheduling in astronomy and radio astronomy are presented.

In Chapter 4, the specific use case of concern, Telescope Array Scheduling is discussed and related to existing scheduling problems and definitions. Furthermore, a survey on current usage in telescope arrays is presented. The available data is analysed and current solutions discussed.

In Chapter 5, the approach to radio astronomy scheduling and observation scheduling in general is laid out.

In Chapter 6, the implementations and test framework to compare and evaluate solutions as well as the new methods and approaches employed are presented.

In Chapter 7, the cost function (figure of merit for schedules) is discussed.

In Chapter 8, the results are presented and discussed in Chapter 9, including a thorough discussion on how the findings can be used in future telescope arrays.

In Chapter 10, the application of the approach in the real-world is explored.

2 Scheduling

Scheduling is an enormous field of research. Due to its importance in manufacturing and transportation, scheduling has received a lot of attention since the early days of computer science. To name but one example, Giffler & Thompson (1960) lay out all the foundations of what today would be called a non-preemptive scheduling problem with tasks, resources and machines. The issue of predecessor relationships, minimizing the total make-span and the complexity of the problem are already mentioned and analyzed. An attempt to give a comprehensive overview of scheduling history and the various areas can be found elsewhere, such as in Pinedo (2008).

Wall (1996) provides a modern account and very general overview of resource-constrained scheduling, defining a schedule as an assignment of resources (where), tasks (what) to periods of time (when). He notes scheduling to be a dynamic problem – not only the tasks to do may change, but the objectives may also change during execution of a schedule. No schedule is static until project is completed, and this is especially true when humans interaction is part of the process. A schedule is subject to constraints. They are a combination of temporal, precedence and availability considerations.

Scheduling has to deal with incomplete information, and often strongly connected with planning. Smith (2005) argues that only narrow aspects of scheduling are solved so far. One of the open fields he names specifically is the integration of planning and scheduling, either to be solved by special separated solvers, or as a single integrated search space. His earlier survey, Smith et al. (2000), goes into detail on existing planning and scheduling research.

To further illuminate the importance of scheduling research, illustrative examples from various domains are described below, in part taken from Wall (1996), Pinedo (2008) and Ernst et al. (2004). It should also be noted that bad scheduling – in industry, manufacturing, airports, and many other applications areas not mentions – can cost huge amounts of money, and consequently good scheduling can avoid these costs.

CPU Scheduling In CPU scheduling, the problem is to (quickly) select a task for the next execution timeslot. The time is open-ended, and it is assumed that all tasks are executed eventually. These algorithms can be called heuristics, making only local decision (what to put in the next time slice) based on priority, and previous run time. It goes without saying that CPU scheduling algorithms have to be very fast in deciding. This can result in the starvation of a task, and real-time and embedded systems in particular want to avoid this by strategies such as completing the task with the earliest-deadline first.

Network Scheduling In network scheduling, the problem is similar to CPU scheduling. For each connection, a stack of packets are waiting to be sent. The scheduler has to make a trade-off between them when ordering them. It is important to maximize throughput and to avoid starvation, as timeouts incur re-transmissions. These algorithms can take more time considering options. Additionally to heuristics, linear optimization can be employed.

Job-shop Scheduling In Job-shop scheduling, the machines are a fixed given, and so are the requirements on which machines a job needs to pass. There are single machine models, parallel machine models, where a job can be done on any of the machines, and job shop models, where a job has to go through some order of machines to visit. Usually, the make-span, the time for a job to finish, is to be minimized. This type of scheduling virtually always implies that a job occupies the resource fully, and that the job visits several machines in some, partially defined, order. In contrast, we require a job to be assigned to a set of machines at the same time.

The application area of this extremely prominent problem is manufacturing.

Project scheduling Another prominent problem in scheduling literature is project scheduling, where, for a project, tasks have to be ordered and assigned to people. The tasks each have predecessors, and resources (people) associated with their execution. The task is not only to find a valid ordering, but to minimize the total time to completing the project. In Figure5, a Gantt diagram is provided to illustrate predecessor relationships.

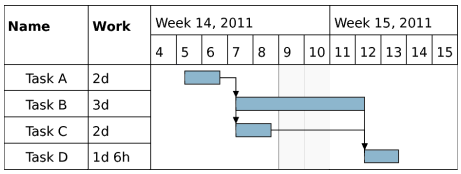


Figure 5: A Gantt diagram displays the predecessor relations and ordering required in project scheduling.

Airport/Transportation Scheduling In transportation scheduling, resource producers, resource consumers and a means of transportation have to be associated to meet demands. For instance, trucks need a schedule for delivery of goods from factories to stores. A variation is airport scheduling, where the time of arrival for airplanes has to be coordinated with staff availability on ground, and schedules are subject to varying conditions.

Classroom and Staff scheduling Classroom scheduling, or timetabling, is the task of assigning students and teachers to rooms at certain timeslots. In this context, the people and rooms can be thought of as the resources (see Schaerf (1999) for a survey). In schools and universities, the desired schedule is a repeating timetable

(e.g. weekly or biweekly). In the related exam scheduling problem however, a schedule for exams is sought, so that no student/teacher/room has overlapping exams.

In staff scheduling, a recurring timetable for staff is sought, so that shifts are filled while taking e.g. holidays into account. Another people scheduling problem is classroom assignment, where the task is to assign rooms and subgroups of people (teachers and students) to time intervals, subject to certain conditions. The teachers, students and rooms can be seen as three classes of resources / equipment. They are required for the execution of a class. They can only be used for one execution at a time. Another aspect that is very similar is that classes should be continuous.

2.1 Fundamental problem

Most of the theoretical literature focuses on the problem of creating a good and valid schedule, given constraints and quality measures. However, as Wall (1996) emphasizes, the problem is actually three-fold:

1. **Creating valid solutions**
   Solutions have to be valid, i.e. adhere to the "hard constraints".

2. **Creating valid and good solutions**
   Solutions should be close to the optimal solution, i.e. most of the "soft constraints" should be satisfied.

3. **Updating a schedule on changes**
   In real life, a schedule has to be modified when constraints or resources change. Ideally, a minimal change will make the schedule valid under the new constraints, and good.

Often times production/industry scheduling only address the first two steps, and in the area of theoretical computer science only the first step is addressed as the combinatorial problem of finding a valid solution, when studying NP-hard problems such as binary satisfiability, bin-packing, N-queens and others, scheduling means the association of tasks to times. This "constraint satisfaction problem" (CSP) is mostly concerned with uninterruptable jobs and the availability of resources. From this viewpoint, two real-world aspects complicate the problem: (1) Constraints and preferences may be subject to change, even during the execution of the schedule, requiring rescheduling; (2) As Schiex et al. (1995) emphasizes, the knowledge about the situation may be incomplete, and the scheduler may have to be able to work with these "uncertainties".

2.1.1 Complexity of finding a valid solution

The abstracted problem of finding valid solutions to a constrained domain is called a constraint satisfaction problem (CSP).

To allude to the complexity of the problem, lets consider $n = 100$ tasks of fixed duration. They can be ordered in $n! = 10^{158}$ different ways if they do not have interdependencies. A factorial grows larger than any polynomial, which already hints that we have a NP-hard problem at hand.

Since NP-hard problems can not be simplified, we can proof that finding a valid solution is NP-hard by re-mapping to another NP-hard problem:

The task is to assign N tasks to a time-interval. Each task has a duration and a benefit. The total time-interval length is given (the constraint), and for simplicity we assume only one task may be active at a certain time.

An analogous problem is Bin-Packing or the Knapsack problem: Given a backpack, the task is to fill it with items, each of certain weight and benefit, while not exceeding a weight limit. The best combination of items is sought.

It can be easily seen that the filling a backpack and filling a schedule are equivalent problems. Because the Knapsack problem is NP-hard, so is scheduling, as NP-hard problems can not be reduced in complexity.

2.1.2 Finding good solutions

After the effort of creating a valid solution, this might be far from the optimal solution. The complexity of finding valid solutions is at least bound by the complexity of listing all valid solutions. See Ullman (1975) and Aarup et al. (1994) for discussions on scheduling complexity. In some rare cases, see Frank et al. (2005) for instance, the scheduling complexity is in P. However, the very strict criteria for these cases are not applicable in practice.

2.2 Algorithm classes

Wall (1996) distinguishes exact methods from heuristic methods:

- Exact methods:
  - Critical path method (CPM, for dependencies within a single project)
  - Linear (Integer) Programming
  - Bounded enumeration
- Heuristic methods:
  - Scheduling heuristics: minimal slack, nearest latest finish time, shortest feasible duration, least resource proportions
  - Sequencing heuristics: order, based on decision trees
  - Hierarchical approaches: goal programming for multi-objective scheduling

- "AI" approaches: knowledge base and inference, or multi-agent based: each agent improves to his benefit
- Simulated annealing: based on neighborhood operator
- Evolutionary algorithms: binary or order-based representations

Scheduling methods are noted by Wall (1996) to often break when the structure of the problem changes; others scale poorly. One of the many scheduling books, Aarup et al. (1994), notes that finding a schedule can be seen as a constraint-directed search process, and adopts the following classification of algorithm:

1. constructive algorithms incrementally add to a schedule until completion
2. repair methods alter an existing schedule to solve conflicts or optimize.

Barbulescu et al. (2006) contributes a general insight on what well-performing over-subscribed scheduling algorithms seem to have in common: They are good because they make multiple moves at once. This lets them explore a plateau with little or no gradient. Randomization is also noted to be efficient in traveling the search space.

We identify the following major algorithm classes from the vast field of scheduling:

1. **Constrained Satisfaction Problem solvers**:
   (a) *The problem* is viewed as a combinatorial issue where the right order of a NP problem has to be found. Scheduling is a form of bin-packing.
   (b) *Algorithms* used are Backtracking, Local search, CPM, Constraint propagation. Very hard since they are often close to brute force.
   (c) *Constraints* limit the domain space of each variable (e.g. which bins a item can go to).
   (d) *Preferences* are expressed by choosing branches in a preferred order (determining the order of listing solutions).
2. **Heuristics, "CPU schedulers"**:
   (a) *The task* is to quickly assign a process to a CPU (or a data package to a network link)
   (b) *Algorithms* used are FIFO, Round-robin, Fair queuing. Simple and quick.
   (c) *Constraints* are expressed by creating queues: Only jobs that could run at this time slot are in the queue.
   (d) *Preferences* are expressed by creating several queues, and the order of picking from the queues (e.g. priority queues).

3. **Genetic/Evolutionary algorithms:** Wall (1996)

   (a) *The problem* is to find the fittest chromosome, which is an encoding of a schedule. Every piece/gene can have a value out of a set (e.g. $[t_{start}...t_{end}]$ or A, C, G, T).

   (b) *The algorithm* is mutating and crossing an existing population of solutions randomly. Based on a fitness function, the best are selected to the next generation. This is repeated. Many variations of this fundamental algorithm exist.

   (c) *Constraints* are expressed by the gene structure.

   (d) *Preferences* are expressed in the fitness function.

4. **Linear (Integer) Programming solvers**:

   (a) *The problem* is viewed as a assignment issue.

   (b) *Algorithms* used are branch-and-bound or Simplex methods. Solvable in linear time if the problem can be expressed with linear variables.

   (c) *Constraints* are expressed as linear relations ($\leq$,$\geq$,$=$)

   (d) *Preferences* are expressed as a (linear) cost function.

Table 1 summarizes different approaches and algorithms. Column 1 classifies them as problem class, approach of solution or requirement for a unsolved problem. Further columns indicate whether it is possible to update the solution (e.g. on changes), whether a closed time frame is considered, whether planning is integrated with scheduling, if the formulation is capable of parallel jobs and whether short-term or long-term scheduling is considered.

In 2.2.1 - 2.2.4, the use and formulation of these methods are discussed. Their applicability to observation scheduling is discussed in 6.1.

2.2.1 CSP solving

For the CSP approach, the main issue is finding valid solutions. Usually, CSP solvers try to find one solution, or list all possible solutions, and can be thus identified with search algorithms. Since it is a combinatorial problem rather than an optimization, there is usually no preference of solutions involved. Even without good initial guess heuristics and traversal methods, tackling this NP-hard problem is complicated in general, since the space complexity does not go away.

2.2.2 Heuristics, CPU-like schedulers

These algorithms make local decisions and usually do not consider multiple solutions, or in-depth, global analysis of possibilities, and so, may miss out on better solutions.

Table 1: A comparison table of some of the approaches and algorithms. The last two rows identify requirements for Array Scheduling for ATA and SKA.

Type	*Name*	*constructive or updatable / repair*	*closed / open-time*	*planning included*	*considers parallel jobs*	*short / long term*
Problem	CSP	repair	closed	no	yes	L/S
	Classroom scheduling	repair	closed	no	yes	L
	SOFIA	constructive	closed	yes	no	S (Night)
Solution	CPU	constructive	open	no	no[1]	L/S
	LP/IP	constructive	closed	no	yes	L/S
	JIC	repair	closed	no	no	S (Night)
	CERES	repair	closed	yes	no	S (Night)
	GA	repair	open / closed[2]	yes	yes	L/S
Requirements	ATA	repair[3]	open / closed	yes	no	L
	SKA	repair	open / closed	yes	yes	L

[1] The multi-core case is only multiple schedulers drawing from the same pool, without any resource resolution.
[2] The chromosome will be finite, but the cost function can be formulated to allow an open system.
[3] Ideally, the repaired schedule, e.g. after cancellation of a project, should look very similar to the previous schedule.

Algorithm 1 Genetic algorithm

```
Generation = 1
Generate initial population POP of size pop-size
until termination criterion reached
{
        Generation = Generation + 1
        Produce children CHI by applying operators to POP
       (through selection, mutation, etc.)
        Evaluate fitness for each I in CHI
        POP = POP + CHI
        Reduce population to size pop-size using selection method
}
return POP
```

For instance, consider two processes, A and B. The resource needed for process A is becoming unavailable soon. Typically, a CPU scheduler will not look ahead to find out whether, if B is run now, process A will starve. Zhao et al. (1987), for instance, discusses various strategies for multi-processor systems.

For instance, an allocation algorithm can just look at each time slot and pick a job based on certain preferences, e.g. priority, time-to-last date, etc. This would give a good solution very quickly, in time $O(T)$.

2.2.3 Genetic programming

Genetic algorithm (GA) solutions are simple to write, as any form of cost function can be expressed. An in-depth understanding of the problem at hand is not necessary for the GA to provide good solutions. However, optimality is not guaranteed. It is not completely understood why they work so well. The GA is more successful if the space of valid solutions is easy to explore.

Formulation for a Genetic Algorithm Genetic algorithms are a general framework for optimization, inspired by the evolutionary process in biology: A population of individuals is improved through a number of generations. Each individual is encoded as its chromosome, which is made up of individual genes. Each gene can take several values, identified as alleles of the gene.

The process runs in two steps that are repeated until some termination criterion is reached (e.g. time, goodness of solution): (1) The population is modified through genetic operators (2) The new population is truncated through a selection mechanism, and the fitness function. Algorithm 1 illustrates the working of a GA. Several decisions are left open for implementation: The size of the population, the termination criterion, the genetic operators used and how often and in which order they are applied, the fitness function and the selection method.

The "default" genetic operators, the most commonly used ones, are crossover and

mutation. Crossover takes two chromosomes, cuts them in half at some points, and exchanges the tails. The mutation operator selects genes randomly and sets them to a new, random value from the allele of the gene.

Model In the scheduling literature, two formulations are commonly found: Time-indexed and Order-based. The time-index gene encoding lets genes represent timeslots and the alleles are the jobs that can be run at this time (see Figure 6).

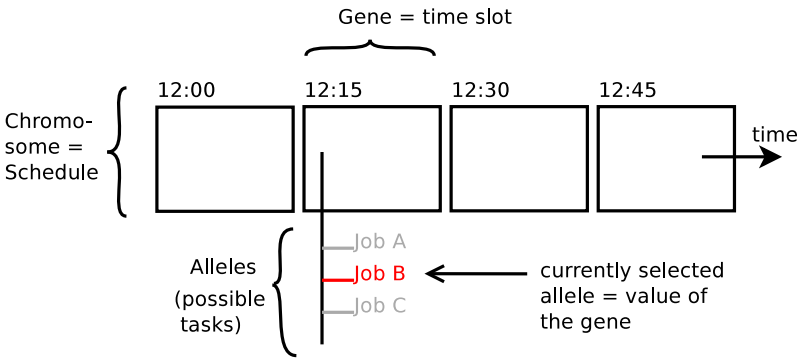


Figure 6: GA encoding using genes as timeslots.

A problem referred to as **fragmentation** here, is consistently found in GA scheduling studies. In this context we mean the undesired outcome that, for instance, the sequence of tasks is `(A, B, A, B, A, B)`, i.e. alternating or frequently changing. For calibration and setup, as well as good quality observations, long continuous observations are necessary and such solutions are undesired. The source of such solutions is that the GA, given independent genes doesn't have a understanding that neighboring genes should be preferred to be similar.

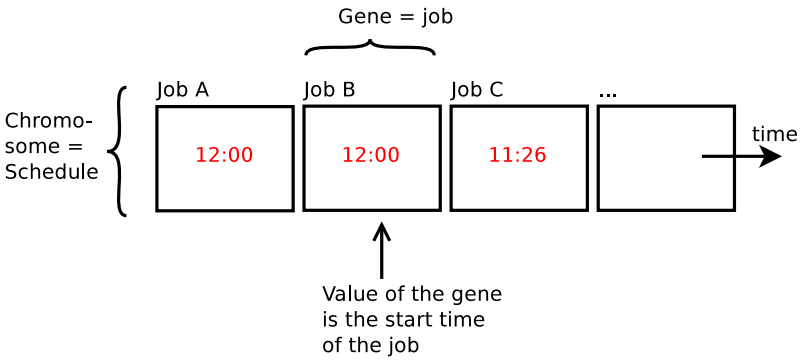


Figure 7: GA encoding using genes as jobs start time.

Trying to avoid this issue, the second popular formulation, order-based encoding, lets each gene identify the start time of the associated task (see Figure 7). This formalism seems natural for uninterruptible tasks, not allowing fragmentation at all. For interruptible tasks, the number of genes can be made variable.

It is clear that with this formulation, the mutation and crossover operators have to be programmed in specific ways to keep the solution valid. This can be rather

complex and it is more difficult for the GA to explore the space of solutions. Wall (1996) notes that this formulation is less successful than the time-indexed encoding.

Formulation Evolve generations genetically according to a cost function.

Result The result, after n iterations, is a population of good schedules. The best of these can be taken as the resulting schedule.

Comments The benefit of a genetic algorithm is that an arbitrary, non-linear fitness/cost function can be given, and the problem does not have to be well studied to give good solutions. Starting from an existing schedule (population), it can be easily modified by changing constraints to adjust for changes.

The genetic approach can be extended to a interactive phase where the user can order suggested schedules. For instance, after the automated phase, the scheduler suggests 6 schedules which are distributed to the affected personnel, which can order them and hand them back to the scheduler, to generate a new population of schedules until all participants agree on one schedule.

2.2.4 Linear / Integer programming (LP/IP)

In linear programming, the cost function and constraints are limited by linear expressions. If the cost function is not linear, a linear approximation may give initial solutions to start another algorithm from. Linear programming problems can be solved in linear time.

The general linear programming problem is to find the variables $\mathbf{a}$ that minimize the linear function $f(\mathbf{a})$ subject to relations $g(\mathbf{a}) \geq 0$.

Integer programming requires the variables $\mathbf{a}$ to be integers and is only in special cases solvable in linear time.

A second drawbacks of LP/IP is that it can not understand the issue of fragmentation (incontinuity between tasks).

Integer Programming problem formulation This formulation is often found in job shop scheduling and is provided here for comparison.

Model

- index k refers to the time slots of the scheduling term
- index i refers to the "machines", i.e. a resource of a certain resource class. a_j donates how many are needed for the execution of job j, and N is the total amount of resources available. If various resource classes are used (e.g. antennas, correlators, etc), more variables of this type have to be added.

- index j refers to the tasks ("job")

Let $x_{ijk} = \begin{cases} 1 & \text{if job j is scheduled on resource i for time slot k} \\ 0 & \text{otherwise} \end{cases}$

Formulation Minimize the cost function, e.g. (for a full discussion on the cost function see 7)

$$max : \sum_j priority_j \times \frac{1}{a_j} \sum_i \sum_k x_{ijk}$$

subject to:

1. No incompatible jobs are at the same time k, i.e. not requesting more resources than we have:
$$\sum_{i.j} a_j \times x_{ijk} \leq N \, \forall j \, \forall k$$

2. A job is on when it can do work, i.e. the source object is up:
$$x_{ijk} = 0 \, \forall k \notin LSTrange_j$$

3. A job gets its a_j antennas, at the time it is running, i.e. a_j resources must be allocated at the same time.
$$\sum_i x_{ijk} = a_j \, \forall k \, \forall j$$

4. A job gets its h_j hours: (this constraint can be moved to the cost function, see below)
$$\sum_k x_{ijk} = h_j \, \forall i \, \forall j$$

Result

x_{ijk} is the full schedule (<timeslots, job, resource>), specifying for each time k, whether the job j is using resource i (0 if not).

Linear Programming problem For completeness, this linear formulation allows linear subdivision of timeslots.

Model

- index k refers to the time slots of a day.

- index i refers to the "machines", i.e. a resource of a certain resource class. a_j donates how many are needed for the execution of job j, and N is the total amount of resources available. If various resource classes are used (e.g. antennas, correlators, etc), more variables of this type have to be added.
- index j refers to the tasks ("job")

We let x_{ijk} be linear, expressing for the scheduling term how many days are to be spent on job j on antenna i within time slot k.

Formulation Minimize the cost function
subject to:

1. No incompatible jobs are at the same time k, i.e. not requesting more resources than we have:
$$\sum_j x_{ijk} \leq days \, \forall i \, \forall k$$
2. A job is on when it can do work, i.e. the source object is up:
$$x_{ijk} = 0 \, \forall k \notin LSTrange_j$$
3. A job gets its a_j antennas, at the time it is running, i.e. a_j resources must be allocated at the same time.
$$\sum_i x_{ijk} = a_j \, \forall k \, \forall j$$

Result

x_{ijk} is a weighted allocation of the term, i.e. for time slots k, how many days should be spent with job j, resource k.

After the result is available, how much of job k is going to be done will be fixed, but the ordering is yet to be defined. This would be put in another scheduler based on preferences. The benefit of this formulation is that its complexity is in P. The drawback is that it will not pack jobs that efficiently because constraint (1) does not understand incompatibilities/oversubscription.

3 Scheduling in Astronomy

3.1 Literature review

Johnston et al. (1996) provides a good general introduction to scheduling for observatories. He notes that observatories today run either manual, partially automated or fully automated systems. Hence, "scheduler" can refer to a computer system or a person. If a computer system is meant specifically, we will refer to it as the "scheduling system". Next to the manual scheduling, the automated-interactive mode, where a human runs and interacts with a scheduling system is very common.

Furthermore, a scheduling system can be classified in a high-level way as (illustration in Figure 8)

1. Long-term scheduling – allocates blocks of time for a full term (several months)
2. Medium-term scheduling – helps operation staff respect the moon phase, target visibility, etc. (several days)
3. Short-term scheduling – determines the order of execution for the current night (within one night)

Johnston et al. (1996) continues to identify desired scheduler capabilities:

- adaptability: open for changing requirements and needs
- interactivity: control is given to the user
- connectivity: coupling to other systems

Future scheduling systems are desired to distribute their calculations in a computer network. For short-term schedules it is relevant to keep optimist/pessimist backup plans, and account for what-if scenarios (weather, seeing, early finish); this aspect can however, also affect medium-term scheduling.

Johnston et al. (1996) concludes by expressing the need for a multi-observatory interchange format for scheduling data, realistic benchmark problem sets and an overview of the tools available.

Leibundgut (1997) argues that current scheduling at the VLT favors certain execution modes, and leaves other astronomers out (for example targets of opportunity and surveys).

de Castro & Yáñez (2003) discuss the general problem of scheduling in the astronomy domain. They note that a Committee for Assignment of Time (CAT) determines must-do projects, assigns a priority and typically has an overcommitment of equipment at hand. Herein too, the difference between long-term vs. short-term scheduling is recognized. It is noted that the schedule is dependent on the scientific policy, which is not trivial, and has to be discussed in the community.

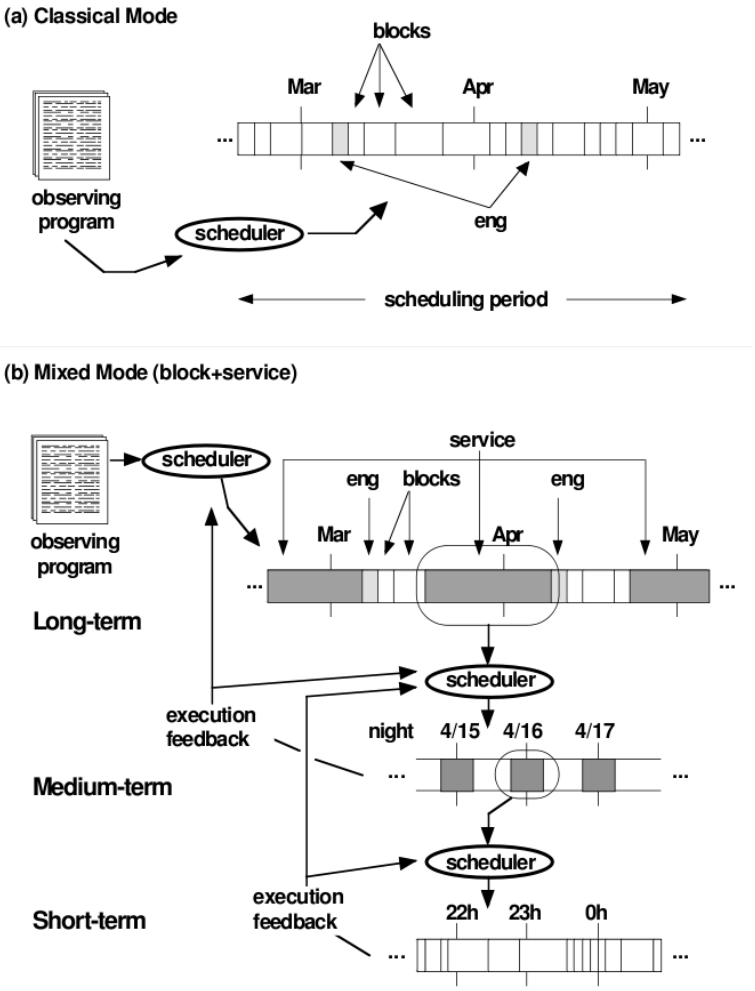


Figure 8: In a "classical mode", only blocks of times are given to astronomers at the start of the term. In a "mixed-mode" system, the long, medium and short-term schedulers interact, and adapt to changes. Taken from Johnston et al. (1996).

Clark et al. (2009) note that the short-term predictability is very important for observers of an observatory in visitor mode. Based on weather forecasts, a system at the Green Bank Telescope (GBT) gives transparency to its observers.

3.1.1 Fully-automated optical telescopes

ATIS is a popular command language to describe the execution/scheduling on automated optical telescopes. The commands are grouped into command groups, forming a observation. Scheduling on ATIS-based systems is about ordering these command groups. The command groups already have the necessary information (duration, target coordinates, etc.) in it, so the builtin scheduler, called APA ("Associate Principal Astronomer"), can do its work.

Most articles discuss improvements to the APA, or replacing it by other algorithms. Algorithms discussed here are usually short-term, for the current night only. When conditions change, the scheduler will rewrite the whole schedule without hesitation. This is due to the fact that no human interaction is planned for execution.

As Bresina et al. (1993) describes, the existing APA works on a sense-select-execute loop. First the "enabled groups" are determined, i.e. tasks whose preconditions are satisfied for the current sensed environmental conditions. Then the next group is selected and executed. The selection choice is based on task priority, number of observations remaining, which task is closest to its LST end, and finally, to break ties, the task ID.

As an alternative system, CERES is described in more detail by the same authors in Drummond et al. (1994). It is based on a temporal look-ahead: Alternative schedules are explored, and evaluated based on an objective function, which in turn is a weighted linear function of priority, fairness, airmass and duration. For generating schedules, three algorithms were evaluated: a greedy hill-climbing search, a random schedule generator, and the existing ATIScope algorithm. The greedy search outperformed both, and interestingly, the random schedule generator performed better than the ATIS algorithm. This is because the ATIS algorithm does not consider all aspects of the objective function, it operates primarily on priority.

The authors continue to describe the CERES system implementation. Since not always alternative schedules are available for the case something breaks, and the generating and searching process is time-consuming, a fallback to the original ATIS algorithm is used in such cases.

Robustness is another key aspect of schedules identified. A schedule is robust, if it does not have to change much in case a task can not be executed due to environmental perturbations. There seems to be a tradeoff between robustness and schedule quality (as measured by the objective function). Since short-term schedulers are discussed, it is important to have a backup plan to switch over to, rather than stop operation for a few minutes and elaborate a new schedule. Systems capable of such alternative plans are called "proactive" rather than "reactive". The authors argue for a mixed system, and continue their discussion in Drummond et al. (1995), where the "Just-in-Case Scheduler" (JIC) is discussed.

JIC starts out with an existing initial schedule. Since the duration of calibrations can vary, and possibly "break" the execution of a following task, it estimates the expected duration uncertainties to predict the reason for breaks. Then, it generates schedules in case the schedule breaks here, and merges these alternative schedules (called "multiply contingent schedule") with the current schedule. This is repeated for all tasks, and results in a more robust schedule. In case of a break, JIC can switch to a alternative schedule. In summary, JIC robustifies an existing schedule, by iteratively assuming one task will be delayed/fail, and creating a new schedule for that case.

Edgington et al. (1996) also implements a greedy algorithm for ATIS, which is based on "states". Each state is an independent, partial scheduled, and identified by an LST time window, and a counter. The algorithm goes on to calculate the tasks

available for execution ("enabled groups"). For each group, it applies the group, creates a successor state, and applies an evaluation function based on the preference requirements. The best state found becomes the current state, and the procedure is repeated. The evaluation function is a linear combination of airmass, priority, and the run count.
Another aspect the authors emphasize is that it is preferred to have a smooth west-east ordering of targets, to minimize slew time. The performance of their algorithm is noted to be load-dependent, but outperforms the standard ATIS scheduler.

Morris et al. (1997) developed another extension, genH (generates search heuristics) for ATIS, which is a meta-scheduler that iteratively mutates an existing schedule, finds the best mutation, schedules it with APA and tests its score. Finally, it keeps the best overall schedule. Morris et al. (1997) note other similar approaches, namely PALO by Greiner (1996), which generates problem elements that are expected to be improvements, and COMPOSER by Gratch & Chien (1996), a statistical hill-climbing algorithm.

Spragg & Smith (1993) report of a "tactical planning tool" called Nightwatch for the Royal Greenwich Observatory. It is also a short-term scheduler for optical astronomy, and can account for service or visitor mode observations. The technique is based on a CSP resolver with delayed evaluation and forward checking. Unfortunately, the project was yet to be completed at the time of the publication.

Gaffney & Cornell (1997) The Hobby-Eberly telescope constitutes a special case, because its altitude is fixed. Only the receiver is variable (similar to Arecibo). As a result, the hour angle is very limited, and a short-term scheduler is needed to determine when to observe objects out of a pool of observations, so that they are just within view. Unfortunately, the paper only describes the system, and no algorithms or decision procedures.

Boër et al. (2000) Another special telescope is the autonomous TAROT (Telescope a Action Rapide pour les Objets Transitoires) observatory. It has fast slewing ($<$3 s), and is meant for Gamma-Ray bursts followups, always listening to the notification system for such events. In the other time (90%) it is used for variability measurements. A request constitutes a set of observations that can be issued by an astronomer and is alive for up to 1 year. A group is defined as contiguous images to be scheduled together. Furthermore, an observation can be time-constrained by the issuer. Although the article does not explain if ordering of observations happens or how it works, it can be conjectured that an efficient order is not necessary due to the short slew times, so observations can be done in priority order. In the case a task can not be observed in the current night, its priority is growing.

3.1.2 Flight scheduling

The Stratospheric Observatory for Infrared Astronomy (SOFIA) is an airborne telescope. The short-term scheduler, described by Frank & Kürklü (2005), not only has to take care of the order of short, uninterruptable observations, but also the flight plan as well. Its objective is to maximize the number of observations of the night. This Single Flight Planning Problem (SFPP) is solved with "squeaky wheel optimization" (SWO).
The time window is specified by the two solutions of

$$\theta_{r,s}(o) = cos^{-1}\left(\frac{sin(20\,\mathrm{deg}) - sin(declination)\cdot sin(\gamma)}{cos(declination)\cdot cos(\gamma)}\right) + LST + ra$$

where γ refers to the position of the aircraft.

3.1.3 Space VLBI

HALCA is a radio-astronomy satellite that was the key element of the VLBI Space Observatory Programme (VSOP). It does Space VLBI which requires communication contact with ground tracking stations (GTS) and simultaneous observation of the same object from a radio telescope on the ground. As Meier (1998) explain, this requires special constraints:

- constant contact with at least one GTS is needed for stable LO signal uplink and data downlink,
- the observed source must be close to projected orbit normal for good UV coverage and
- 70 degrees or more away from the sun due to power panel shading by the antenna
- the observed source must be visible from both the satellite and the ground radio telescope.

These constraints result in severely limited time intervals for observing radio sources.

The tool SVLBSCHED is a long-term and short-term scheduler, where the long-term part (18-months) is a limited-resource algorithm: It allocates by scientific priority, top-down until no resources left. Equal priority ties are broken by observing days preference, i.e. which observation promises better (u-v) coverage.

3.1.4 Arrays

Grim et al. (2002) discuss the relevance for LOFAR to consider simultaneous observations. This is due to its structure and how it observes the sky, namely the beamforming is done digitally: By adding time-delays into the signals received from

the aperture array nodes, a certain position is producing an interference pattern and the receiving power from this portion in the sky is maximized. This can be done for eight beams simultaneously. The parallel scheduling problem formulation presented uses two abstractions: observation types (abstraction of execution modes) and virtual instruments (abstraction of resources necessary). In this case, the problem is very similar to the job-shop: a machine is a virtual instrument. For future research, the authors name oversubscription.

Gharote et al. (2009) look at the scheduling problem for the Giant Metrewave Radio Telescope (GMRT). For this, astronomical constraints (visibility, sun distance), equipment constraints, logistical constraints (preference for local observers, weekly maintenance/system tests) and people constraints (preferred dates by astronomer) are taken into account. A linear solver provides a first cut solution, and is repaired by heuristics, with the help of a human via a GUI.

3.1.5 Satellite scheduling

Spike is the name of the software used for scheduling on the Hubble Space Telescope (HST), and is one of the earliest works of scheduling in astronomy. Several articles, presented first in full in Miller et al. (1988), have this software as its subject.

Minton et al. (1992) present a repair heuristic for large-scale CSP/scheduling problems that uses value-ordering to prefer minimizing the number of constraint violations after each step. It is analysed to perform much better than backtracking, and as good as the previously implemented, more complicated neural network. This caused the HST team to replace the Neural network with this heuristic in the Spike software.

Johnston (2002) describes the inner working of the heuristic algorithms implemented in Spike. There is

- procedural search: best-first, or most-constrained-first
- rule-based heuristic search: select promising partial schedules
- commitment rules: decide how to extend schedules by scheduling decisions
- artificial neural networks: a set of discrete scheduling choices is represented by ANN

Furthermore, the authors note the dynamics of the problem: With increased experience, and over time, the problem and its constraints change.

Chavan et al. (1998) note that the Very Large Telescope (VLT, ESO) uses a medium and short-term scheduler (MTS, STS) where the STS component is based on Spike. Unfortunately, the inner workings and algorithms are not explained in this work.

For ROSAT, a satellite borne X-ray observatory, Nowakovski et al. (1999) found that relaxing the problem to a "generalized assignment problem" leads to good solutions. A relevant policy is that the observation times should, over time, be proportional to the ownership of the observatory. This scheduler plans out the order of micro-observations (target with duration) based on priority.

There are special satellite-based constraints, and slew times must be allowed as off-time between observations. It is desired to maximize the time used for observations, minimize partially scheduled observations and to maximize high-priority objects. A minimum observation time of 10 minutes is used, which results in 5800 slots for a 6 months term, where 1800 objects are placed, with 1000-125000 seconds integration time, and time slots between 10 and 40 minutes.

In a (non-parallel) ILP representation, this results in the special case where there are more objects than slots. The algorithm repeats solving the problem as a LP, removing successfully assigned (those without split) slots, and repeating the solving process for the rest. Afterwards, the schedule is repaired to make it valid. The benefit of this approach is that it uses an polynomial scale algorithm (primal simplex).

As alternative algorithms, a greedy first-fit heuristic, with back-tracking is presented, however the article comes to no final conclusion which of them, or which combination, is the best algorithm.

Kleiner (1995) presents an interactive planning interface for the Submillimeter Wave Astronomy Satellite (SWAS). The tool is mostly a helpful GUI to guide a human scheduler. No automated scheduling is described here, but an update is given in 1999, where Kleiner (1999) reports that the system efficiently calculates calculates orbit, planet positions, visibilities, gyration, satellite LOS velocity, guide stars, slew paths etc. through coordinate transformations only, without the need for timeslots/stepping. Also, the contradiction between robustness/efficiency is noted, so the system is designed to be dynamic and conservative: Instead of working on a schedule, a human scheduler only works on a so called "order form", which is the activity specification. From this, a schedule is just one instance, and can be easily constructed. Order forms are tolerant of change, and schedules are not. When the schedule breaks, a new one, that is equally efficient can be easily constructed, because the “order form” was fine tuned, not the (broken) schedule. Unfortunately, again, a scheduling algorithm is not described, and the website supposed to provide information is not online any more.

Harrison et al. (1999) discusses the requirements of the reverse situation, i.e. observation satellites imaging earth. This problem is very similar to the automated telescopes, yet here, the possible observation windows are small, in the minutes range. The problem is approached as knapsack problem, where branch-and-bound, greedy and tabu search methods are known. As objective function, the sum of

priority multiplied by utilized time for each project is maximized, i.e. $\sum_i p_i * t_i$. Since this paper used only a small problem set (10-30 tasks), enumeration was sufficient, but scales very badly.

Giuliano & Johnston (2008) explores schedulers for the next-generation James Webb Space Telescope (JWST). They identifies multiple objectives: to minimize the unscheduled time, to minimize angular momentum buildup and to minimize the number of observations missing their last opportunity to schedule.

There are a few spacecraft-specific constraints to be taken care of, namely momentum buildup and sun shielding. It is argued that single objective solutions loose information and fix the tradeoff between multiple objectives, such as number of observations dropped and amount of momentum correction needed. True multi-objective algorithms leave the trade-off open for the algorithm to discover. This approach is implemented using Pareto optimality. Pareto optimality is reached when no modification can be made without worsening one objective. In a n-dimensional parameter space, a Pareto frontier can be identified. For the Pareto frontier, non-dominated sorting into ranks is performed: rank n dominates individuals $> n$, i.e. rank 1 is the Pareto frontier (dominated by no other individual). Furthermore, a crowding distance is taken into account, meaning that members of crowded regions are eliminated more easily.

The evolutionary algorithm (EA) used by Giuliano & Johnston (2008) is based on Generalized Differential Evolution (GDE3), roughly described as follows: A parent and 3 members of population x_1, x_2, x_3 are selected. A new "trial vector" $y = x_1 + F * (x_3 - x_2)$ is calculated, with F being the differential weight parameter. In the last step, y is crossed with the parent, and a selection between parent and trial vector.

These vectors represent decision vectors, and decide for each operation whether or not to schedule it in a timeslot. The full system is based on Spike: an iterative repair search starts with an initial guess, whereas EA invokes Spike with initial information, but has no knowledge of constraints.

Spike goes ahead and schedules the first 22 days (one term) worth of observations, removes conflicts, and fills gaps. This is done 400 times (population size: 20, 20 generations). Next, the secondary search generates 400 momentum buildup schedules for each generated schedule in the first layer. These are initialized randomly to get a wide Pareto frontier sampling. The random "search" is compared with GDE3.

The conclusions drawn are that a separation of the objectives is superior to linear combinations (which only works on convex frontiers), and that it gives insight to problems/tradeoffs. For future research, distributed calculations are considered, as well as additional heuristics for sample creation.

3.1.6 Details of GA studies

In this section, genetic algorithm approaches taken by studies mentioned above are discussed in more technical detail.

de Castro & Yáñez (2003) compare a neighborhood search algorithm and a GA in a study of short-term scheduling of uninterruptable observations.
For uniformly distributed targets in the sky, the GA performs better. However, realistic targets preferably lie in the galactic plane. Therefore, the Infrared Space Observatory (ISO) target list is used as a test set, oversubscribing the period by factors of 1.5 and 3 (calculated without slew time). For duration, an exponential-like exposure between 0 and 4000 seconds is used.
The chromosome uses an order-based encoding, i.e. the vector of arranged targets. The selection criterion used is "roulette-wheel selection", i.e. the selection probability is proportional to the fitness value. The crossover (crossover probability of $p_c = 0.6$) uses a random selection of couples (n/2); the parents are crossed and replaced by children with probability p_c. In one tested crossover operator, the parent genes are split in two at a random point, and latter section exchanged, in the other the middle section of the parent gene is selected, and exchanged (prone to repeated observations).
The mutation (mutation probability of $p_m = 0.1$) simply permutes two genes (from the targets order). The GA is configured with a population size of 50 and 10 generations.
The authors find that the Lin-Kernighan method performs better when the input is sorted by RA, but the algorithm is slow. The GA performs well and should be chosen when the main constraints are not well-known. It is also noted that the used cross-over operators are not very successful. The main conclusions are:

- A high-quality observation is hard to achieve with restricting policies, e.g. keeping the ratio of observers proportional to the corresponding institutes financial involvement
- GAs are by far superior in large-scale problems
- To make the problem easier for the scheduler, a low oversubscription is needed, as well as a large number of observations with short exposure times

Unless the same GA representation is used, the found GA performance and cross-over problems are not generalizable however.

Sponsler (1989) tried to map the Spike CSP problem to a chromosome, by identifying each gene with an activity having an associated starting time slot. The following genetic operations are in use:

- vertical crossover: partial matched crossover, exchanges one gene, i.e. the activity times of one activity, between two chromosomes

- horizontal crossover: on a single chromosome, swap two genes
- mutation: reset a randomly selected gene to a new legal segment

The used fitness function is a combination of preference and constraint satisfaction, where latter is weighed exponentially. To the surprise of the author, the GA performs very poorly, especially in comparison to neural network, both in time used and goodness of solutions.

For the genetic algorithm tested in Grim et al. (2002), a gene is also associated with a micro-observation. The fitness function consists of two parts, the constraint-fitness and the preference-fitness. The weight of the constraint-fitness is adapted stepwise during the GA run. The constraints are resources, observation frequencies, execution order, that the distance to earlier observation must not be too small, and relations to other observation. The preference fitness is defined as $\sum_{obs}(obslen/(windowlen)+1)$.
The crossover operation is not explained, but said to be complicated (due to order-based chromosome structure).
The GA is configured to use a population size of 30 and 20000 generations. The selection method used is Tournament selection between the two worst of the current and two chromosomes of the next generation. The GA is run in steady-state, meaning only few individuals are replaced each generation. The authors cite an article saying it supposed to be better than a generational model.

For discussing the poor results of these two studies, we recall the issues of this chromosome representation mentioned in section 2.2.3. The structure of the chromosome allows invalid solutions too easily, making operators very difficult to write. Furthermore, the search space is too constrained making it hard to explore, and GAs are known to perform badly in tightly-constraint problems. The authors in Grim et al. (2002) themselves conclude that the algorithm is not really suited because it always ends up with invalid solutions that have to be repaired. A time-indexed chromosome encoding would be more native and results in a smaller search space. The fragmentation problem of this alternative encoding, is known to Grim et al. (2002).

Furthermore Wall (1996) states that in scheduling, a struggle (generational model) always performs better than steady-state. He performed a thorough study in this field of resource-constrained scheduling, while the book cited by Grim et al. (2002) for the opposite argument is a general scheduling book.

3.1.7 Summary

Scheduling in astronomy is not an issue in many areas, as it can be solved sufficiently by a manual scheduler. However, in some areas where the problem of finding schedules is very difficult (e.g. satellites, Space VLBI) or the scheduling is part of an

automated system (automated optical telescopes), research has evolved. However, in recent years another niche has developed, due to the development of large-scale projects and ever-more complicated instruments. For modern telescope arrays such as LOFAR, ATA and ALMA, there is a demand for simplifying the work of a scheduler, as parallel observations, split arrays and more dynamic scheduling becomes commonplace. These demands are highlighted by Mora & Solar (2010), Jones (2004) and Fomalont & Reid (2004). The last discusses science based on splitting the Square Kilometre Array into 10 sub-arrays.

3.2 Domain description of Observation Scheduling

In this work we consider long-term scheduling for observations. In the case of astronomy, this would be over several days up to several months. As such, the individual steps within an observation (e.g. calibration, observing object A, then B, etc) is not as relevant as making the necessary conditions available; The individual objects to observe are abstracted away to the field of view required. The observations may be interrupted, although this might incur a penalty that can be expressed through the re-calibration required.

A description of the domain follows:

- (I) Astronomers send in proposals.
- (II) Proposals contain observation tasks.
- (III) Observation tasks specify
  - (IIIa) the desired field of view: In the usual case of a small astronomical object, this constrains the observation times by the visibility of the object (earths rotation).
  - (IIIb) the required resources / equipment. Among others, this contains the required number of antennas.
  - (IIIc) the desired hours of observation. For good data quality, long observations are desired by the astronomer.

  Proposals are to be executed by the radio telescope array. (IV) Due to oversubscription, possibly not all proposals can be executed; (V) proposals are given a priority by a committee, and higher-priority proposals are preferred. The committee can also amend the proposal (shorter observation, less equipment).
- (VI) At the beginning of the term (quarter or semester), the accepted proposals are given to the scheduler (human or system).
- (VII) The desired outcome of the scheduling process is a schedule

  - (VIIa) which is an association between time intervals, proposals and resources
  - (VIIb) that fulfills the requirements of the proposals
  - (VIIc) that is feasible to execute by the operating staff
    * (VIIc1) it must contains regular, scheduled maintenance periods defined by the operating staff (e.g.
    * (VIIc2) it must contain a certain share of free times, for Targets of Opportunity (TOO), short-term observations, maintenance, tests

- (VIII) In reality, the schedule will go back to the committee, is checked with the operating staff, amendments to the schedule and/or the proposals are made.

- (IX) During the term changes can occur

  - (IXa) Targets of opportunity (TOO) can occur: A star might suddenly flare, a supernova might explode, etc. Suddenly, operation has to be preempted and this target is to be observed.
  - (IXb) A proposal might be canceled
  - (IXc) A new, important proposal might be added

- (X) However, changes must not completely rewrite the schedule, so that astronomers and operation staff can plan ahead.

3.3 Symmetries

Inherent properties of celestial mechanics make the time symmetric around the sidereal time. In other words, the visible sky is the same at the same time the next "day", if day isn't used as 24 hours (Julian day) but about 23 hours 56 minutes (sidereal day). The symmetry is due to earths rotation around its axis, and the shift compared to solar days is due to earths revolution around the sun.

Since the visible sky, namely which side is facing a star is location-dependent (longitude), the local sidereal time (LST) is used, which incorporates the location.

This symmetry suggests the use of time measured in sidereal time of day and sidereal day. An observation can usually be done on any day if the local sidereal time range is the same.

3.4 System description

An **automated long-term scheduling** system is sought that allows **parallel observations**, is **reactive to resource availability** and operator preferences, can re-schedule once these change and, last but not least, is **scalable**.

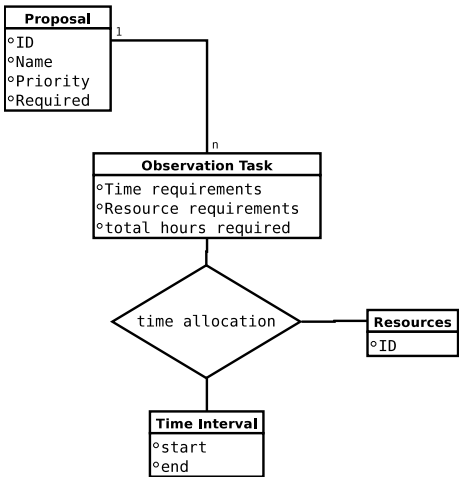


Figure 9: Domain model of the scheduling problem.

3.4.1 Definitions

Proposal A proposal may consist of several observations to be done. A proposal has a priority, and each observation specifies the total hours of observation time required. A proposal can also contain a "required" flag, indicating it must be included in the schedule for this term.

Observation task contains a definition of possible time intervals for execution, an expression of the resource requirements (for instance the number of antennas and the backends required), and the total hours the task has to run. The task can be split. The task belongs to a proposal. Each resource requirement take the form "need any n out of this set" or "need exactly this/these one(s)" and they all have to be fulfilled simultaneously.

Term the current scheduling period, e.g. a quarter or a semester.

Schedule associations of time intervals with execution modes. The time intervals are non-overlapping and cover the whole term.

Execution mode a set of observation tasks with the resources associated to them. A execution mode can only be realized when the resources are not oversubscribed.

Resource equipment in a certain state needed for an observation. E.g. antennas, antenna configurations, front-ends, receivers, filters, back-ends, correlators.

3.4.2 Functional requirements

Assumptions

1. Assume maintenance periods (VIIc1) are expressed by a proposal that has to be executed (required flag set).

2. Assume surveys that desire to map x degrees (all-sky surveys: x=360) for h hours are expressed by n observation tasks which observe patches of x/n degrees for h/n hours (x/n small, e.g. 10 degrees).

Definition

A scheduling system is desired that produces one or multiple schedules $\mathcal{S}'$, given

- the terms time frame $\mathcal{I}$,
- a date in the future t, after which execution modes may be planned for,
- a set of proposals to be considered for scheduling $\mathcal{P}$,
- a frequency of free times $f = \frac{T_{free}}{T_{term}}$,
- and optionally, one or more existing schedules $\mathcal{S}$, each with a preference value attached to it.

Then a scheduling system as defined above, when used at term start and whenever changes occur, can be capable of suggesting a schedule for the term. This satisfies I-X functionally.

3.4.3 Non-functional requirements

A system must not only produce a schedule to be acceptable, it must produce a good schedule. Defining what a good schedule is, and being able to express so, is key to a scheduling system.

- If given an initial schedule $\mathcal{S}$ to alter due to changed conditions. The produced schedules **must** not make modifications before the current time t, but **must** take past allocations into account.
- The scheduled free times **should** be close to f. $f' \approx \frac{T_{free}}{T_{term}}$.
- The required time requirements of observation tasks **must** be respected.
- The required resources for observation tasks **must** be available. If a resource has to be made available, e.g. an antenna reconfiguration is necessary, the time necessary for this has to be included. This implies that reconfiguration time should be minimized (e.g. tasks for a certain configuration should be run in temporal proximity).
- The observation tasks **should** be as continuous as possible. Observations usually are made up of a setup period that includes initial calibration (e.g. 10-30 minutes), and the actual target observation, which is preempted by recalibration (e.g. for 3 minutes every 20 minutes). Since recalibration is

required every time a execution mode is started, the execution mode should continue as long as possible to minimize the setup periods.

- The observation task's FOV **must not** have the sun within a angular distance of s degrees.
- The proposal priority **should** be respected, i.e. higher priority proposals should be executed. Furthermore, the observation tasks should get the time they require.

3.4.4 Special case: Large-Scale Radio Telescope Arrays

Certain special requirements apply for large-scale radio telescope arrays:

- Radio telescope arrays have the benefit over optical arrays that the atmospheric effects are smaller, and that observations are possible 24×7.
- Most of the work is long surveys with little interaction.
- An array with more than half a dozen telescopes is so complex that manual scheduling becomes undesirable.

3.4.5 Additional requirements

A good scheduling framework can allow

- the flexibility of reorganising parts of the array, or splitting into sub-arrays to do several observations at the same time with fewer telescopes
- parallel execution of tasks
- Re-scheduling when changes occur (e.g. operator staff, telescopes become unavailable, maintenance required or completed, a new important proposal comes in).

3.5 Problem mapping

As can be seen from Figure 9, the task of a scheduler remains the mapping of tasks with resources and time intervals, given constraints and some method of assessing the quality of schedules. This section deals with mapping the specific problem of Radio telescope array scheduling to well-known problems in the literature.

The most prominent scheduling formulation is job-shop scheduling. The machines are a fixed given, and so are the requirements on which machines are needed to pass. A mapping would equate machines to telescopes, tasks to observations. A mapping of a reorganisable array, where an observation can be completed on any

sufficiently large set of machines is not applicable. The biggest issue is that the notation of a task requiring two machines at the same time is not expressible.

In more general terms, a formulation of many scheduling problems such as job-shop scheduling given by the Graham notation (Rogers & Graham (1982)). The observation scheduling problem discussed here is *outside this notation*, because the notation assumes that one job uses one machine at a time. In our case, multiple resources are required for execution at a time.

The second prominent scheduling formulation is project scheduling. The case where resource constraints are to be considered is called resource-constrained project scheduling problem (RCPSP). This is applicable and the situation is RCPS with no dependencies between tasks. This makes it a more "trivial" case as the possibilities are less constrained, but increases the number of possible orderings. Following Herroelen et al. (1997), the problem can be classified as m,1,va|pmtn|av, a m-resource-type preemptive RCPSP.

Astonishingly little research has been done in this area since Herroelen et al. (1998). Only recently, PRCPSP is being started to be explored, although with serious limitations. For instance, with Ballestín et al. (2008), m_PRCPSP has been developed, allowing m interrupts, an option that will clearly introduce bias longer tasks vs shorter tasks. See Hartmann (2002) for a review of subclasses of the RCPSP.

The problem is also equivalent to classroom scheduling. Here the resources are of several types (classrooms, teachers, pupils) that, adhering to quite complex rules, have to be assigned to timeslots. The problem is usually applied to short, repetitive intervals (one or two-week rhythms).

Solution methods used in the literature are explained in the methodology section.

4 Current scheduling in Radio Telescope Arrays

This section presents a survey of the current situation in today's working telescope arrays.

4.1 Survey of Scheduling Techniques

Virtually all telescope arrays working today have been contacted and inquired about their scheduling, control and monitoring systems. I am grateful for the effort and patience while allowing me an insight in their systems.

The surveyed observatories included:

- ATCA – *Phil Edwards*, Mark Wieringa – Australia Telescope Compact Array in Narrabri, Australia. Part of the research a travel to the array and an observation.
- LOFAR – *Ruud Overeem, Arjo de Vries* – Low Frequency Array in the Netherlands
- ATA – *Colby Gutierrez-Kraybill* – Allen Telescope Array in Hat Creek, California, USA. Part of the research was a travel to the array and close collaboration with *Colby Gutierrez-Kraybill*.
- HST – Robert Hawkins – Hubble Space Telescope, Space.
- CARMA – Steve Scott – Combined Array for Research in Millimeter-wave Astronomy, California
- Wettzell Observatory – Alexander Neidhardt, Martin Ettl – Germany
- LHC – Stefan Schlenker – Large Hadron Collider, Switzerland
- IVS – International VLBI Service
- LBA – Long Baseline Array, Australian VLBI

Most of these observatories use manual scheduling, partially supported by visualization software.

4.1.1 ATCA

The ATCA is a array of 6 dishes with 22 meters diameter. ATCA is run in observer mode, meaning the astronomers come to the site. For the scheduler, this means that new observers should not be placed between Friday and Sunday, so that they can get a introduction from the staff maintenance between 8am-4pm on weekdays.

The schedules terms are 6-month long. Proposals are rated between 0 and 5 (highest), the median being 3.6.

Most important is that the observation is within the correct LST range. A portion of idle time is held free for Targets Of Opportunity (TOO).

The ATCA employs a Java-based tool called "TermScheduler" developed by Dave McConnell and maintained by Mark Wieringa.

The main feature of the tool is that it allows assigning proposals to slots, while showing the requirements of the current proposal to be placed next. The scheduler (human) iteratively generates timetables and checks for better solutions and errors (similar to the workflow of writing latex).

Contrary to ATCA, the SKA will not be run in observer mode. The ATCA allows no splitting of the array (using some antenna for one observation, and some for another). No expansion to more dishes is planned.

Unfortunately, a data set of the ATCA proposals could not be obtained.

4.1.2 HST

The Hubble Space Telescope uses Spike for automated scheduling, as mentioned in the literature review. Its scheduler is a search and repair heuristic that mostly tries to find valid solutions that satisfy all constraints.

4.1.3 LOFAR, LHC

LOFAR and the LHC both use PVSS as its control system. However, unlike the LHC, LOFAR also uses a scheduling system, named SAS (Specification, Administration and Scheduling).

Its main functions are (1) specifying an observation (target, frequencies, durations), (2) scheduling: Assigning resources to observation request (stations, disks, cluster nodes, timeslots) and (3) administration, registration of metadata: data products, statuses produced in the process and progress is stored in the database (postgres).

SAS interacts to PVSS through MACScheduler, which fetches the next observation from SAS and starts and configures the PVSS system.

Although some scheduling algorithms have been implemented, it is unclear how much of the scheduling is in fact done automatically.

4.1.4 ATA

ATA plans to introduce automated scheduling for their 42 dishes of 6 meter diameter each.

Maintenance is currently held on Wednesdays, blocking the array from observations for testing, repair and configuration.

The observations done at the ATA include mostly surveys.

Every 6 months, a number of projects come in. A scheduling term is 3 months long. All proposals that pass the committee are done, and they are not assigned a

priority. Each observation specifies the total hours required, the number of antennas, IF and correlators requested.

Projects can change over time; Observations are given a grade after they have been done from A to F, where low grade observations (E-F) are repeated. Also possible are targets of opportunity and cancellation of projects.

4.2 ATA data set

The ATA deals mostly with surveys, and is operated in service mode. The data set contained 215 approved proposals with information on LST start and end time, number of antennas required, and total hours requested. These are all approved proposals over the past years. The empirical distributions are shown in Figure 10 (durations) and Figure 11 (total hours).

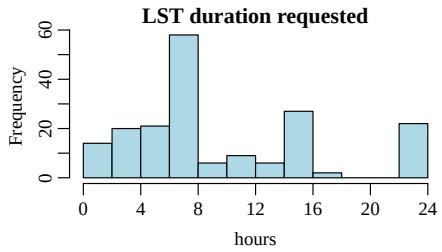


Figure 10: The distribution of durations. 7-8 hour durations are observations of a region, while higher durations refer to surveys.

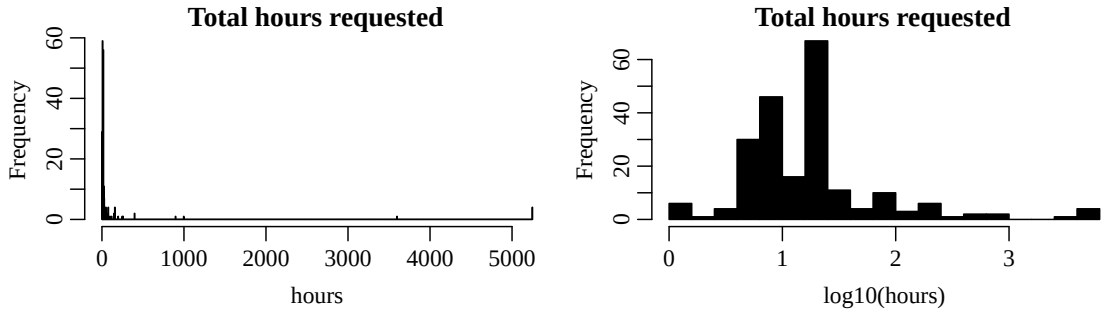


Figure 11: The total hours requested per project (uniform scale on left panel, log-scale in right panel).

4.2.1 Quartal 3, 2010

To give a specific, detailed insight, a quartal from 2010 contained the observations laid out in table 3 and was scheduled as shown in Figure 12.

Title	Total Time Requested	Frequency [MHz]	Backend Requested	Posted
Intensive monitoring of Cygnus X-3	28	3040	2 Correlators	9-Jun-2010
Demonstrating a fast-dump 8-input correlator for transient studies with the ATA (resubmit)	30	800	1 Correlator	9-Jun-2010
Commensal correlator observations with SETI projects	0	1430	2 Correlators	9-Jun-2010
Galactic Center Survey correlator engineering time	9	3040	2 Correlators	9-Jun-2010
Weekly "hex" calibration observations	13	700	2 Correlators	9-Jun-2010
Test of new method to equalize primary beams in mosaiced images	16	4442	2 Correlators	10-Jun-2010
SETI Engineering tests of SonATA	100	1420	Beamformer	11-Jun-2010
REU Project: Fast variability of Methanol Masers	14	6660	Beamformer	11-Jun-2010
Engineering for Validation of Multiple Beam Dump-To-Disk Observing	15	6660	Beamformer	11-Jun-2010
Beamformer Calibration Studies 3040 - 3440	16	3040	Beamformer	11-Jun-2010
SETI Targeted Search at Two Times the Waterhole	150	2840	Beamformer	11-Jun-2010
SETI Observations of Candidate “Dyson Spheres” Selected from IRAS	30	1420	Beamformer	11-Jun-2010
SETI Survey of a Region Near Cygnus X-3	80	1420	SETI/Prelude	11-Jun-2010
PiGSS	5250	3100	2 Correlators	12-Jun-2010

Table 3: ATA proposals relevant for the 3rd quartal of 2010

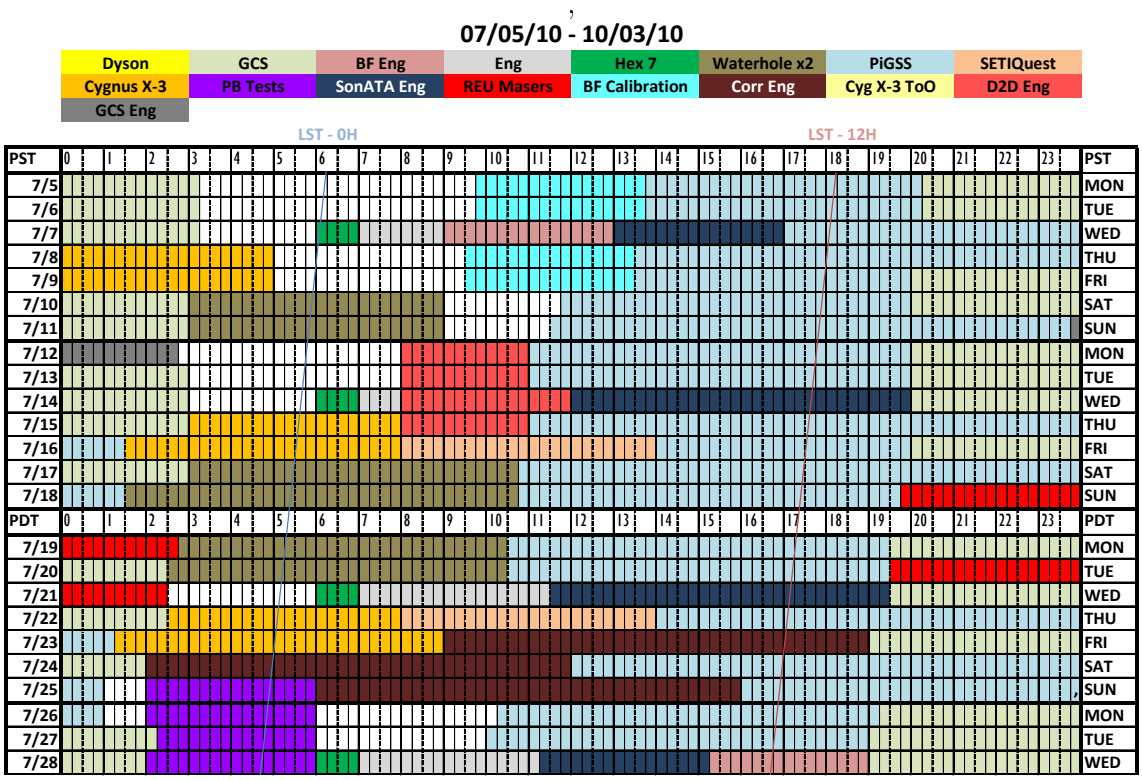


Figure 12: A section of how the quartal was scheduled. Rows represent days, columns represent a time slot of this day (in solar time). In comparison, the LST shifts each day by 4 minutes, which is why the observations begin to wander to the left. White areas indicate free time for targets of opportunity.

4.3 Test set generator

The model was chosen to be a sum of 4 populations. The ratios were selected so that the result matches the ATA data set.

Population name	Properties	Total share
"Full-Sky"	A proposal consists of 8 jobs to cover all LST ranges with equally length. Total hours are uniformly chosen between 100 and 1000 hours.	23%
"Galactic"	A proposal aims for a the galactic LST range. Since it doesn't matter which preferred direction is used, 14 ± 2 was chosen. Total hours are uniformly chosen between 4 and 50 hours.	40%
"Random"	A proposal aims at a source anywhere in the sky. The LST range window size is uniformly chosen between 6 and 10 hours. The total hours is drawn $h = (1./(u * 200 + 1)) * 6000$ where u is a uniform variable between 0 and 1. A total hours less than 4 hours is rejected.	32%
"Daytime"	These tasks may be real observations, maintainance or test observations. They start between 9 and 16 local time. Total hours are chosen as $h = a * 10 + b + 1$ where a is uniformly chosen as either 0 and 1 (long task), and b is uniformly chosen between 0 and 10.	4.6% (16% of a day)

The priority is in all cases chosen uniformly between 1 and 5. As many proposals are generated as needed, basically filling either the number of days available completely (by hours), or a multiple of them.

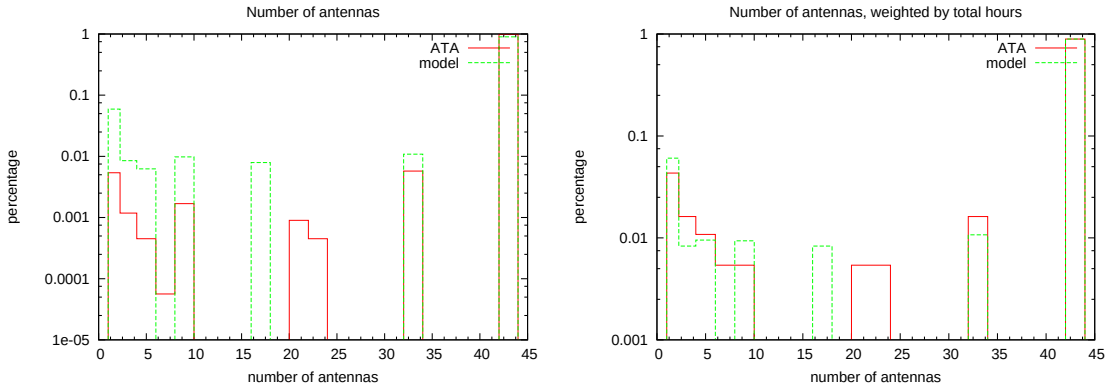


Figure 13: ATA data set and generated data set from model. Top panel: distribution of number of antennas in proposals. Bottom panel: distribution of number of antennas in proposals, weighted by number of hours. The ATA antenna distribution can be roughly described to have 4% for 1 antenna, and about 1% at each log2 decade, and the rest at the total number of antennas. This is how the model was chosen. It can be assumed that the number of antennas is the same for test observations as for real ones, and thus this model was used across all four populations. The distributions agree well.

Antenna model

Total hours model Other authors used a log-uniform distribution here, and the ATA dataset looks very similar to this (see Figure 11, left panel).

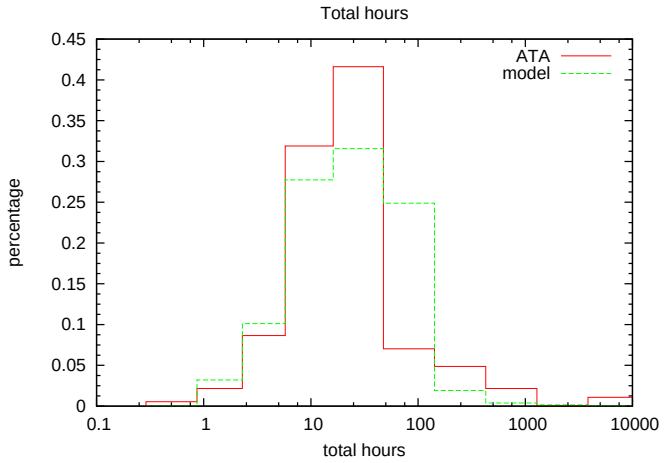


Figure 14: Distribution of total hours requested in proposals. The distributions agree well.

LST window size model The duration or LST window size is the absolute difference between LST start and end. It can be associated with the duration of how long a source is visible.

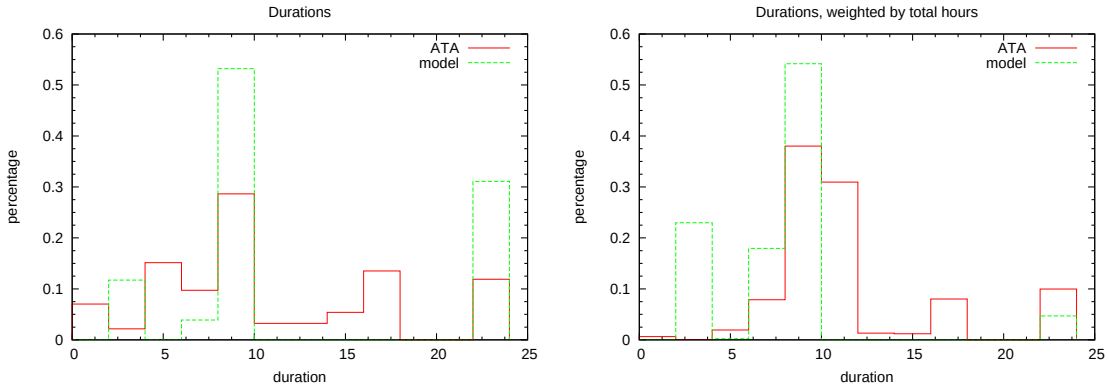


Figure 15: ATA data set and generated data set from model. Top panel: distribution of LST window duration in proposals. Bottom panel: distribution of LST window duration in proposals, weighted by number of hours. The distributions agree to an acceptable degree; The excess at 3 hour durations are the all-sky surveys (covering 24 hours, split up in 8 subtasks of 3 hours). Maintenance tasks are nominally written as 24 LST hour durations, as they have different time constraints. The main effort in reproducing ATA data is to have an expected duration window size of 8 hours.

Part II
Methodology

5 Approach

5.1 Time granularity

Although in principle the problem could be stated in continuous time, this problem approach uses timeslots ΔT based on the granularity of the problem: In radio astronomy it is not a big deal to lose 15 minutes of observations. The integration time for 15 minutes usually doesn't give any new results, in fact this is about the time needed for setup, calibration, etc. This defines the granularity of the observation problem and the solutions. here, for certainty, we set $\Delta T = 15\,minutes$, although easily modifiable as different domains may come up with different granularities through the same logic.

5.2 No interdependencies

We assume all observations are independent in their time requirements (no predecessor relationships). On a long-term scheduling system this is true on large-enough time-scales, when all micro-observations (calibration and relative checks) are combined in larger observation tasks. Preferences may be used to prefer an ordering, however all orderings are taken as valid.

5.3 Preemption

Unless an observation target is circumpolar, or the requested time is shorter than a day, the observation will be interrupted. Thus, only preemptive scheduling can be of use. This seems obvious to astronomers, but much of the literature on resource-constrained (project) scheduling deals with strong predecessor relation ships and non-preemptive tasks, which is not applicable here.

5.4 Parallelism

One novel aspect of this work is to allow parallel execution of observations. This means that a job needs several machines at a time (e.g. any subset of a given size), while another uses another, distinct, subset.

This formalism is incompatible with job-shop scheduling that requires an item (job) to go through machines in order, but doesn't allow it to use multiple (possibly varying) machines sets.

5.4.1 Job-Combinations

Before placing jobs in parallel, it has to be determined whether they are in principle compatible. With n jobs, there are 2^n combinations to be tested. This can be simplified by assuming that a superset of an incompatible job combination is again incompatible, but the order of the complexity remains unless rigid requirements are made. For instance, if every job needs an equipment that can only accommodate m jobs at a time (e.g. storage devices or transmission lines), or if every job needs an item for itself of a resource set of size m, the complexity is reduced to n^m combinations.

A virtual priority can be assigned to the "companions". Since the priorities are usually logarithmic, i.e. two "priority 4" proposals are worth as much as one "priority 5" proposal, for instance$exp(p_{a+b}) = exp(p_a) + exp(p_b)$ can be used.

5.4.2 Resource-sets

The formalism adopted here lets each observation specify a number of resource requirements which all have to be fulfilled: $R_A \wedge R_B \wedge ... \wedge R_Z$, where, for example, A stands for antennas, and B for backends.

Each resource requirement consists of a set of usable items S, and $0 < a \leq |S|$, the number of items required from it $R : (a, S)$. For instance $R_A : (2, \{X, Y, Z\})$ would specify that any 2 of X,Y,Z fulfills requirement A of a certain job.

This formalism allows the expression of "need any n out of this set" $R_B : (n, S)$ and "need exactly this/these one(s)" $R_C : (1, \{X\})$ requirements.

Computation of resource-compatible jobs A job J_1 and a job $J_2$are only compatible if every single of their resource requirements are compatible:

$$compatible(J_1, J_2) \Leftrightarrow \bigwedge_{R_i \in \mathcal{R}} compatible_{R_i}(J_1, J_2)$$

. The general case of $n+1$ jobs requires n jobs to be compatible with themselves and with in sum with another job and can be reduced to the following scenario.

Consider two jobs with each resource requirements on A (e.g. antennas): $J_1 : R_A : (a_1, S_1)$ and $J_2 : R_A : (a_2, S_2)$.

If $S_1 \neq S_2$, there are m_1 resources in S_1, namely $S_1 \backslash S_2$, that can be assigned to J_1, reducing a_1 to $a_1' = a_1 - m_1$, and m_2 resources in S_2, namely $S_2 \backslash S_1$, that can be assigned to J_1, reducing a_1 to $a_1' = a_1 - m_1$.

Hence we have $S = S_1 = S_2$. Then

$$compatible_R(J_1, J_2) \Leftrightarrow a_1 + a_2 \leq |S|$$

.

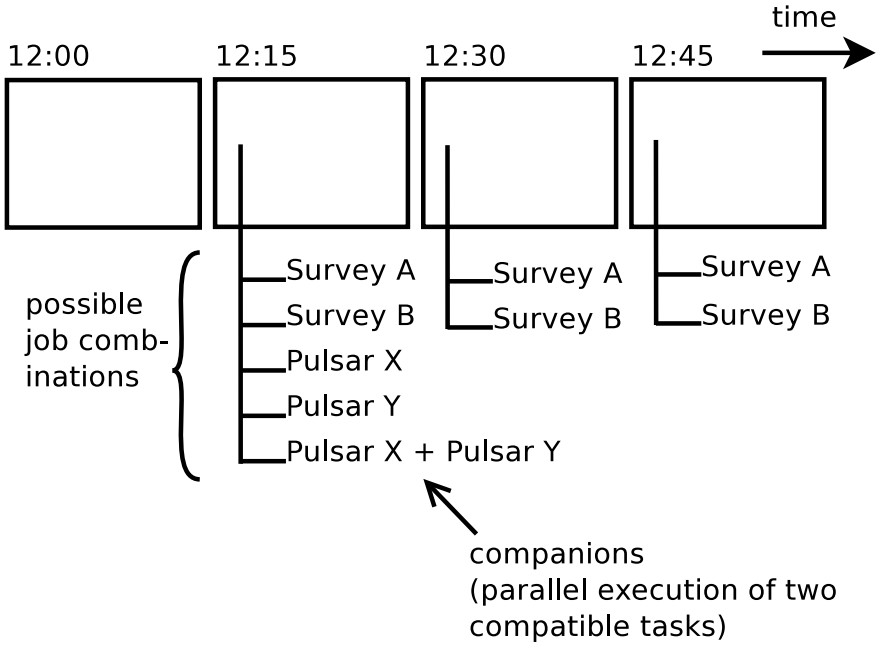


Figure 16: Illustration of the "schedule space" (space of valid schedules).

In the general case with a job vector **J** and without exactly same resource sets but a strict superset $S = \bigcup S_i$,

$$compatible_R(\mathbf{J}) \Leftrightarrow \left(\sum_{s \in S} \sum_i \begin{cases} 1 & s \in S_i \\ 0 & otherwise \end{cases} \right) \leq |S|$$

.

Proof (indirect): After assigning all resources R around the tasks, there is at least a resource item s_i left that is wanted by two tasks J_1 and J_2, i.e. $s_i \in S_1$,$s_i \in S_2$. Both tasks still need another resource: i.e. $a'_1 < a_1$ and $a'_2 < a_2$, but all other tasks have their resources $a'_i = a_i$. So $\sum_{k>2} a'_k + a'_1 + a'_2 = |S \backslash s_i|$, which is at least (both 1 short) $\sum_{k>2} a_k + a_1 - 1 + a_2 - 1 \leq |S| - 1$. This contradicts the (supposedly fulfilled) requirement $a_1 + a_2 \leq |S|$. ■

The time complexity for calculating the compatibility of a job set is $O(|S|) = O(|\bigcup S_i|)$, hence a simple (sub-)problem.

5.5 Space of valid schedules

Due to the assumption that observations are independent, at each timeslot, the observations whose time requirements are fulfilled can be found. The "schedule space" maps time onto sets of job-combinations $\mathcal{S} : t \rightarrow \{JC\}$. A job-combination is just a set of compatible jobs $JC : \{J\}$. An illustration is given in Figure 16.

The size of the schedule space can be given as a sum of all job combinations at each slot, which can be estimated by the multiplication of the number of slots

$(T/\Delta T)$ multiplied by the number of job combinations without considering time constraints. With T the total time considered (e.g. 91 days) and ΔT the time slot size, we can write the space complexity of the schedule space as

$$O(\sum_{t=1}^{T/\Delta T} |S_t|) \approx O(\frac{T}{\Delta T} \times |S|)$$

Checking all constraints over the whole scheduling term is proportional to the length of the term. Checking the constraints for all companions in a timeslot is proportional to the number of companions, as only one set of tasks will be selected. Checking the constraints for a companion set in a timeslot is proportional to the number of companions, as all that is needed is for each task to (1) check if the time constraints for the task are fulfilled, (2) to check if the necessary resources are available and (3) to count down the available resources at that timeslot.

The time complexity of this calculation, excluding the number of job combinations (2^n in general, n^k simplified), is similar to the size of the schedule space:

$$O(\frac{T}{\Delta T} \times |S|)$$

Due to LST symmetry (see 3.3) this time complexity can be usually be reduced to calculating only one day:

$$O(\frac{1}{\Delta T} \times |S|)$$

All scheduling algorithms presented will work on the schedule-space, so that a valid schedule is always guaranteed.

Complex time requirements

Sidereal time requirements are just a special case of time requirements. To take any time requirements $f(t) \rightarrow \textit{boolean}$ into account, the schedule space is to be browsed, and at each timeslot where a job can not run for some reason (resources not available at that time, bad date for observers, ...), all job combinations that contain this job are to be removed.

To force certain observations, or to free up a certain time interval, the space of valid schedules can be edited by either emptying the relevant time slots or setting them to the desired task.

Special cases

- **Surveys** and other observations may need certain LST interval distributions, for instance equal coverage of all visible portions of the sky. The proposal can be broken into LST- observations that cover the sky, e.g. in four LST intervals of six hours length, each requesting a quarter of the total proposal time. Such

a split will assure that the scheduler can not bias the requested time towards one LST interval.

- **Distant locations**; Extended arrays, VLBI: For an interferometer, the source has to be visible to all locations. The LST is different from location to location, but one can be chosen for scheduling. Then, the LST range of visibility is not the complete, say, 8 hours it is visible from that location, but has to be shortened to a range where it is visible to all locations. The further apart the locations in east-west, the shorter the visible range.

- **Maintenance**. Maintenance can be put in formally as a proposal. There might be some freedom in assigning time, or it might be a required maintenance period (day). In the latter case, the schedule space for that range is to be forced to include task. The scheduling formalism allows maintenance of some resources to go on while others are used for observations, if the resource requirements are specified for both.

6 Implementation of a multi-paradigm framework

The implementation was done using Java. The LP/IP solver incorporated is lp_solve 5.5, from Berkelaar et al. (2010). The largest GA libraries available for Java are ECJ, JGAP and the Watchmaker Framework. They all were tested, but the last one was chosen as it allows easy runtime configuration and freedom in choosing the form of the chromosome. The Watchmaker Framework can be found at Dyer (2010). Genetic operators and the genetic history were easy to implement there. For correctness, JUnit tests were incorporated. The implementation is available under `http://johannes.jakeapp.com/research/observation-scheduling/` .

6.1 Algorithms

6.1.1 CSP solvers

These algorithms have been excluded in this work since they do not scale well and finding valid solutions is simple in the case considered here. However, their initial, greedy guess approaches can be used as heuristics.

6.1.2 Heuristics

The simple heuristics used here can be written as listing scheduling algorithms. Stork & Uetz (2000) distinguishes listing scheduling algorithms on whether they iterate through time slots (serial listing scheduling) or jobs (parallel listing algorithms).

Serial listing schedulers (see Listing 2) go through the time slots and select a suitable choice based on some strategy.

Algorithm 2 Serial listing scheduler

```
def serial-listing-scheduler():
        schedule = {}
        while not done:
                t = getNextUnallocatedTimeslot()
                choices = getChoices(t)
                task = select(choices)
                schedule[t] = task
        return schedule
```

Strategies for selecting a task (function select in Listing 2) used here:

- *Prioritized*: Task with highest priority is selected.
- *ShortestFirst*: Task with shortest time left is selected.
- *Random*: A random task is selected.
- *First*: The first task of the available tasks is selected (FIFO strategy).
- *KeepingPrioritized*: If possible, the task from the last timeslot is continued, otherwise the highest priority task is selected.
- *FairPrioritized*: Randomly select a task with a probability proportional to its priority.
- *EarliestDeadline*: Select the task with the least number of possible time slots left
- *MinimumLaxity*: Select the task with the least laxity, i.e. number of possible time slots - number of time slots still required (compare with Zhao et al. (1987))

Strategies for moving through the schedule used here (function getNextUnallocatedTimeslot in Listing 2):

- *Serial*: simply iterate from the first time slot to the last
- *SerialLeastChoice*: The next time slot selected is the one with the fewest number of choices. The number of choices is calculated from the schedule space, with the already-finished tasks removed.
- *ContinuousUnlessOneChoice*: Acts like *Serial*, unless there are time slots with only one choice, in which case it acts like *SerialLeastChoice.*
- *ContinuousLeastChoice*: This strategy acts like *SerialLeastChoice*, but subsequent calls return the adjacent unallocated time slots to which the same task can be extended. This should allow task selection strategies like *KeepingPrioritized* to use more continuous observations.

- *ExtendingLeastChoice*: Similar to *ContinuousLeastChoice*, but fills neighboring unallocated time slots immediately with the same task as far as possible.

Parallel listing schedulers (see Listing 3) order the tasks based on some strategy, and try to accommodate them on the timetable one by one.

Algorithm 3 Parallel listing scheduler

```
def parallel-listing-scheduler():
        schedule = {}
        for j in getTasksOrdered():
                for time t in getPossibleSlots(j):
                        jc = findJobCombinationSuperset(schedule[t],
                        if jc available:
                                schedule[t] = jc
                        if isComplete(j):
                                break
        return schedule
```

Strategies for ordering tasks (getTasksOrdered in Listing 3):

- *by pressure*: For each task, a number called "placement pressure" is calculated, which refers to the ratio of the number of possible time slots to the requested total hours. Observations with high pressure are first in the ordering, as they are difficult to place in the schedule. The pressure calculation has to be corrected for very long observations that can only be accommodated partially in any case. This strategy is similar to the minimum laxity defined above.
- *by priority*: Ordered by priority, highest priority first.
- *shortest first*: Ordered by requested hours, shortest first.

The variant of the algorithm used here, *TrivialFirst*, fills time slots with only one choice first, but otherwise works as described in Listing 3.

Another heuristic algorithm is *"GreedyPressure"*, shown in Listing 4. At each time slot, as many as possible of the preferred tasks are scheduled. For the task ordering strategy, *by pressure* from above is used.

6.1.3 Genetic Algorithm (GA)

For a general introduction to GA, see section 2.2.3. Here, the specific choices of chromosome representation, genetic operators, selection method etc. are discussed.

Algorithm 4 "Greedy pressure" scheduler

```
def greedy-pressure():
        schedule = {}
        pressureOrderedTasks = {}
        for t in time:
                pressureOrderedTasks[t] =
                        filterUnavailable(t, getTasksOrdered())

        for t in time:
                tasks = filterFinished(pressureOrderedTasks[t])
                jc = createBestJobCombination(tasks)
                schedule[t] = jc

        return schedule

def createBestJobCombination(tasks):
        values = {}
        for jc in jobCombinations()
                values[jc] = 0
                for t in tasks:
                        if t in jc:
                                values[jc] = values[jc] + 1
                        else:
                                break
        return jc with highest value
```

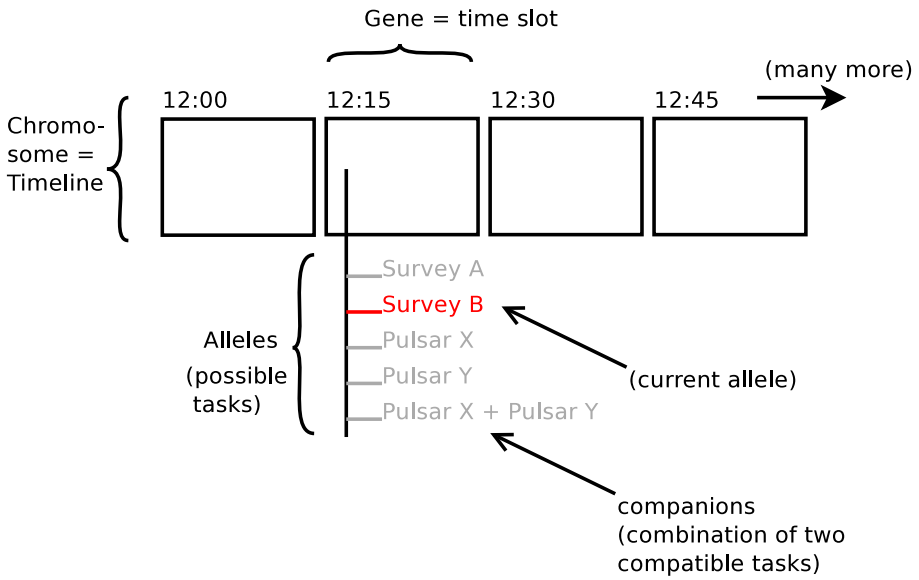


Figure 17: An illustration of the timeline as a chromosome for a Genetic Algorithm.

Chromosome A time-indexed encoding is chosen. Each time slot is represented by a gene, which can be set to a a job combination. The alleles – possible values of this gene – are defined by the corresponding slot in the schedule space (see 5.4.1). The justification for this selection is two-fold: First, it is native and straight-forward due to the independence of observations. Secondly, the alternative representation, order-based encoding has been consistently found to perform poorly (see section 3.1.6).

Crossover operator A crossover operator creates a child from two parents by using the values from parent A for some genes and the rest from parent B. A corresponding alternate child can be immediately produced.

A typical implementation is 1-point-crossover. This crossover operator makes random pairings of two chromosomes, and, with probability $p_{Crossover}$, cuts both chromosomes at a random gene, and exchanges their tails (see Figure 18).

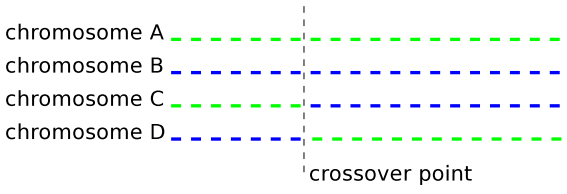


Figure 18: 1-point crossover

The generalization, n-point-crossover uses n points for cutting. Figure 19 shows 2-point-crossover.

Let $p_{DoubleCrossover}$ specify the probability of using 2-point-crossover and $ndays_{Crossover}$ the maximum distance between crossover points. Both 2-point and 1-point crossover can be used.

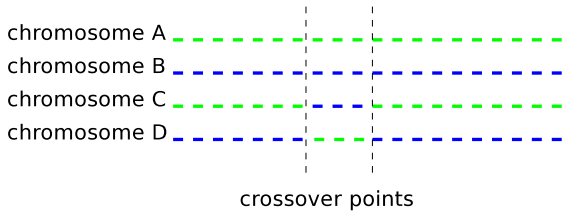


Figure 19: 2-point crossover

Mutation operators

- The default **Mutation** (*Mut*) operator would reset a gene with probability p_{Mut} to a random allele.

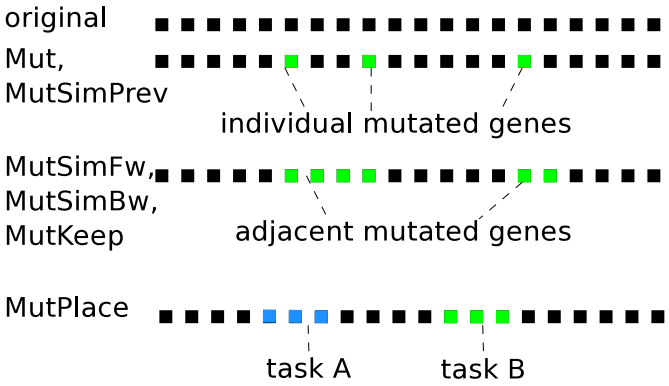


Figure 20: Illustration of the mutation operations: Mut and MutSimPrev modify individual genes while MutSimFw, MutSimBw, MutKeep, MutPlace change blocks of genes.

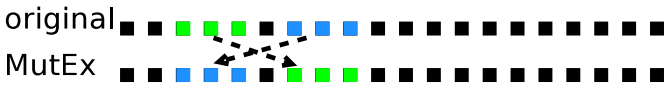


Figure 21: Illustration of the mutation operation MutEx.

To avoid fragmentation, several additional operators are implemented for evaluation. These are operators that make multiple moves. Compare for instance with Sponsler (1989) and Tanomaru (1995).

- **MutationSimilarPrev** (*MutSimPrev*): Selects a gene with probability $p_{MutSimPrev} \times \frac{60}{\Delta T}$ and sets it to a job-combination most similar to the previous timeslot's job-combination.

- **MutationSimilarForward** (*MutSim*Fw): Selects a gene with probability $p_{MutSimFw} \times \frac{1}{15}$ and sets all the following timeslots to a similar job-combination for as long as possible (max: 1 day). If m genes were changed, the next m genes are not selected.

- **MutationSimilarBackward** (*MutSim*Bw): Same as MutationSimilarForward, but back in time. $p_{MutSimBw} \times \frac{1}{15}$.

- **MutationKeeping** (*MutKeep*) is a combination of MutationSimilarForward and the default mutation: It selects a gene with probability $p_{MutKeep} \times \frac{1}{15}$ and sets it to a random allele. All following timeslots are set to the most similar job-combination for as long as possible (max: 1 day). If m genes were changed, the next m genes are not selected.

- **MutationExchange** (*Mut*Ex): Selects a gene with probability $p_{Mutex}/9$ and tries to exchange the gene with one of the previous days (3 days).

- **MutationJobPlacement** (*MutPlace*): For each job, selects the job with probability $p_{MuiPlace}$. For the selected job, selects a random gene where the job can be placed. Sets all surrounding genes to the same value (like MutationSimilarForward and MutationSimilarBackward).

Figure 20 and Figure 21 illustrate the working of the various mutation operators. It is thought a priori that Mut will introduce fragmentation, whereas the MutSim operations will make the schedule more continuous, reducing and avoiding fragmentation. It is expected that the additional mutations are more successful in changing the schedule compared to the default mutation operator, because they can make modifications more appreciated by the fitness function, namely long, continuous observations.

Each of these operator has an associated probability that it is selected, some modified by a factor to make the resulting probabilities comparable.

Fitness The fitness function has to assign a value to a schedule, which the GA tries to maximize. The measure is discussed in section 7.

Selection Selection via Roulette Wheel Selection, with 2 elite individuals. Roulette Wheel Selection seems to be a standard selection mechanism, and when used with a few (2) elite individuals outperformed Cutoff Selection in all test with a relaxed Sudoku toy problem. Elitism means that the best individuals always make it to the next round. This avoids degradation of the population and having to rediscover good solutions.

Initial population The initial population is a set of schedules which may contain randomly generated solution schedules and/or solution schedules produced by other algorithms (e.g. heuristics, LP).

GA parameters

- **Population size, Number of iterations**: To make results and computational effort comparable, the product of population size and number of iterations is kept constant: $n_{iter} \times n_{pop} = 1000$.

- **Genetic operator probabilities**: Starting from default values $p_{Crossover} = 0.1$, $p_{Mut} = 0.1$ (all others 0, disabled), the best probability combination is to be determined.

6.1.4 LP/IP

Integer Programming problem, with parallelism In this formulation, we incorporate the calculated schedule space (from Section 5.5), which for each time slot k provides JCS_k, the set of job combinations that can be run at this time.

Model Let $JC_{k,j} \in JCS_k$refer to the job combination j that can run at time k, and $priority(JC_{k,j})$ be its (combined) priority.

Let $x(JC_{k,j}, k) = \begin{cases} 1 & \text{if job combination } JC_{k,j}\text{is scheduled to run at time slot } k \\ 0 & \text{otherwise} \end{cases}$

.

Formulation Minimize the cost function (see 7), e.g.

$$min: -\sum_k \sum_j priority(JC_{k,j}) \times x(JC_{k,j}, k)$$

subject to:

1. Only one companion is run at a time

$$\sum_{JC_{k,j}} x(JC_{k,j}, k) \leq 1 \, \forall k$$

2. Each job is on when it can do work, i.e. the astronomical target object is up. The set of job companions where all jobs' time constraints are fulfilled, JCS_k, already respects this property (for the calculation of the schedule space, see section 5.5).

3. Each job gets its required resources: Already ensured when finding companions.

4. Each job j gets at most the h_j hours it requested

$$\sum_{j \in JC_{k,j}} \sum_k x(JC_{k,j}, k) \leq h_j \, \forall j$$

Result

$x(JC_{k,j}, k)$ describes the full schedule, specifying for each time k, if the jobs of $JC_{k,j}$ are active (0 if not). Specifically, $y(k) := \{JC_{k,j} \in JCS_k | x(JC_{k,j}, k) = 1\}$ provides the job combination to run for each time slot k.

6.2 Combinations and Comparison of Algorithms

This section presents the framework used to compare scheduling algorithms described above.

The main part is a genetic algorithm, that is initially filled with all other solutions (heuristics and LP/IP), and a number of random initializations. The idea is to allow the GA to select and evolve better solutions based on the guesses from the other algorithms.

The tool of evaluation is based on test data sets generated from the random model described in section 4.3, corresponding to an oversubscription of 100%, 200%, 300%, and 400% of a 91-day term. It may seem that a 100% filled schedule is easy to create, but the proposal bias on the galactic core region makes certain time intervals oversubscribed.

A **quality measure** of an algorithm is the fitness function of its (best) output schedule. Evaluation is done across all test sets and for varying parallelism, allowing 1, 2, 3 or a maximum of 4 tasks in parallel at a time. Averaging over the 16 test cases provides a stable measure of algorithm quality.

6.2.1 Analysis of initial population influence

Genetic history Each chromosome of the initial population can be labeled "100% Algorithm A" according to the algorithm it originated from. While the GA makes mutations and crossovers of schedules, these labels can be carried along and updated accordingly. For instance, a crossover at 70% of the chromosome would make the mixture 30% of parent As and 70% of parent Bs properties, whereas a random mutation of 5% of the genes would reduce the parents influence by this value. The final surviving population can thus indicate what algorithms contributed most to the solutions.

Similarity measure As a different measure of influence, each surviving schedule can be compared to the initial population by timeslot similarity.

6.2.2 Research questions

1. *Question*: What are the best GA parameters, which are useful operators?

   (a) *Method*: Through meta-optimization, the values that maximize the quality measure.
   Starting from start values, a optimization algorithm was set to improve the GA parameters, namely n_{pop}, $p_{Crossover}$, $p_{DoubleCrossover}$, $ndays_{Crossover}$, p_{Mut}, $p_{MutSimPrev}$, $p_{MutSimFw}$, $p_{MutSimBw}$, p_{MutEx}, $p_{MutPlace}$. Starting from $n_{pop} = 50\, p_{crossover} = 0.1$, $p_{mut} = 0.1$ (all others 0, disabled), the SciPy COBYLA implementation – see Jones et al. (01) and Powell

(1979) – was used to probe the continuous parameter space. The correctness was verified using local hill-climbing.

(b) *Interpretations*: If $p_{op} \approxeq 0$ for a operator *op*, this operator is not helpful in improving the schedules.

2. *Question*: Is the GA comparable to other techniques?

   (a) *Method*: By ranking of GA output and algorithms fitness value, it can be decided if the GA with a random initial population can outperform (some) other algorithms.

3. *Question*: Does the GA improve other algorithms' solutions?

   (a) *Method*: Running the GA using other algorithm's output as the initial population. Comparing initial and final population of the GA using the genetic history and similarity measure.

   (b) *Interpretations*: If the final population (and the best output schedule) are influenced by the GA, the GA improved upon other algorithms. If the output population is basically one of the initial population, the GA wasn't able to improve the solution.

4. *Question*: What are the best algorithms?

   (a) *Method*: Running the GA using other algorithm's output as the initial population. Comparing initial and final population of the GA using the genetic history and similarity measure.

   (b) *Interpretations*: The genetic history and the final population show how influential and successful a initial schedule was, and how influential/useful its algorithm. Algorithms whose output dies during evolution are less useful than algorithms whose output is very similar to the final population.

Additionally, it is to be tested how the runtime compares between algorithms, however, this is not so much a concern for long-term schedulers. If the results are good, this can be up to several hours.

7 Cost function

To develop a solution towards optimality, a cost or benefit (depending on the sign) has to be assigned to solutions. Scheduling algorithms will aim to optimize this measure. This subsection discusses various telescope array scheduling desires as aspects that can be integrated into the cost function.

Let T_j refer to the total time requested for job j, in number of timeslots.

Let $x_{jk} = \begin{cases} 1 & \text{if job j is scheduled at time k} \\ 0 & \text{otherwise} \end{cases}$.

1. Completed hours for a job: The deviation from demanded hours is minimized (not linear).

$$min : \sum_j |T_j - \sum_k x_{jk}|^2$$

2. Idle times: in general, not using the available resources is bad (linear):

$$min : \sum_k (1 - \sum_j x_{jk})$$

3. Task priority:

   - Proportionally (in hours, linear)

$$min : \sum_j priority_j \times (T_j - \sum_k x_{jk})$$

   which just translates to

$$max : \sum_j priority_j \times \sum_k x_{jk}$$

   If the observatory is oversubscribed, a scheduler might end up with many started tasks, but no finished ones. Thus only counting completed jobs would be a good measure. However, this very discontinuous function would be is hard to optimize. Hence, adding a bonus for completed jobs would be better (yet still non-linear), for instance:

$$max : \sum_j \sum_k x_{jk} \times priority_j \times \begin{cases} 1 & \text{if } j\text{completed} \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

4. Best observation conditions: A job should ideally observe when the object of interest is ideal to observe (linear):

$$max : \sum_j \sum_k x_{jk} \times r_{jk}$$

   where r_{jk} identifies the return for job j to be scheduled at time k.

   In the case of astronomy, this benefit can done accurately by determining the airmass and the elevation angle, and not allowing it to go below a certain

wavelength-dependent angle (e.g. 12° for $6\,GHz$).

The relevant equations of the transformation are:

$\sin(altitude) = \cos(zenith) = \sin(latitude) \cdot \sin(dec) + \cos(latitude) \cdot \cos(dec) \cdot \cos(LST - ra)$

$airmass\, X = \frac{1}{\sin(altitude + 244/(165 + 47 altitude^{1.1}))}$[1]

5. Continuous observation: A job would like a block of time to be as long as possible. At the start of the observation, extra calibration may be necessary that will not hold true for the next time slot if not adjacent (nonlinear).

$$max : \sum_j priority_j \times \left(\sum_{b \in B_j} (|b| - \Delta T) \right)$$

where $B_j \ni b$ are the lengths of time blocks allocated for job j, and ΔT refers to the duration of one time slot.

6. Resource switching: It could be expressed that reconfiguring the array (changing baselines) or changing receivers, etc. takes time and effort (=cost). This has to account for neighboring time slots and their equipment difference. To give an example, at the ATCA it takes a full day to change the antenna configuration.

It should be noted that apparently several constraints (elevation maximization), hour completion for a job, job completion, continuous observations, resource switching are not expressible in a linear way. Hence, a linear solution might be a good start, but a nonlinear solver is needed.

Another approach is to weaken some of the constraints, and add the deviation from the ideal to the cost of a solution: $max : -\sum \left(\frac{real - ideal}{tolerance}\right)^2$ (non-linear).

Chosen cost/fitness function The chosen cost function for the linear solver, as shown in section 6.1.4, is:

$$max : \sum_j priority_j \times \sum_k x_{jk}$$

For the GA, to decrease fragmentation, the fitness function is adapted to:

$$max : \frac{1}{normalization} \sum_j priority_j \times \left(\sum_{b \in B_j} (|b| - \Delta T) \right)$$

The implementation of calculating the fitness function in algorithm form can be found in Listing 5.

[1] approximation formula

Algorithm 5 Fitness function

```
def calculateFitness(schedule):
        timeSlice = 15 # minutes
        penalty   = 15 # minutes, for setup, etc.
        previousJc = None
        fitness = 0
        for Time t in schedule:
                JobCombination jc = schedule[t]

                usefulTime = 0
                for Job j in jc:
                        if not isFinished(j):
                                if j in previousJc:
                                        usefulTime += timeSlice
                                else:
                                        usefulTime += timeSlice - per

                fitness = fitness + jc.priority * usefulTime
                previousJc = jc

        return fitness / calculateNormalization()
```

This could be easily extended with a benefit term r_{jk}, but in radio astronomy if the required elevation (LST range requirement) is achieved, quality is practically independent from the elevation and state of the atmosphere.

The normalization is calculated by assuming the scenario of not having any time-constraints, and just placing as many high-priority tasks as possible. Listing 6 shows the calculation of this upper bound. A fitness value of 1 can not be reached, as time constraints have to be adhered to that are not considered in the normalization calculation.

The cost function is meant to be adaptable by staff during a term, to accommodate changing needs.

Algorithm 6 Normalization calculation

```
def calculateNormalization():
        normalization = 0
        timeDone = 0

        for Job j in getAllJobsSortedByPriorityDescending():
                timeChunk = min(totalTermDuration - timeDone,
                        j.requestedTotalDuration)
                normalization += j.priority * timeChunk
                timeDone = timeDone + remainingTime

        return normalization
```

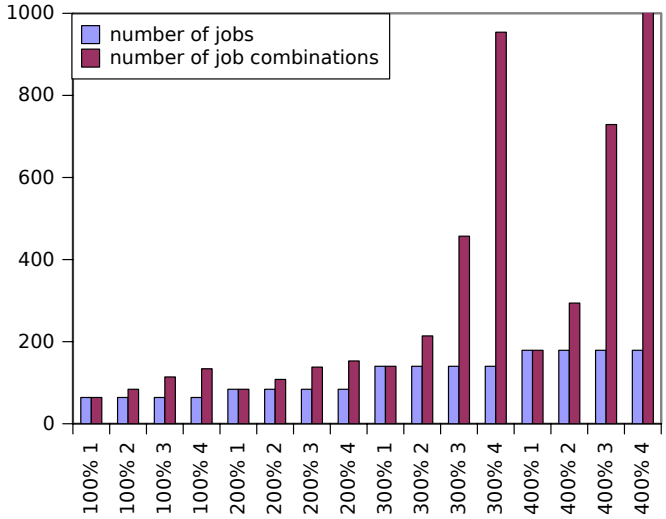


Figure 22: Plot of table 4 on the next page.

Part III

Results

In this section the GA is referred to as "filled" if the solutions from other algorithms were used, and as "empty" otherwise. The default configuration of the GA refers to using a population size of 50, a mutation probability $p_{Crossover} = 0.1$ and a crossover probability of $p_{Mut} = 0.1$, and not using any other operators (a reasonable a priori configuration).

Generated based on the ATA historical data (see section 4.3), the characteristics of the 16 specific test scenarios used are outlined in Table 4. These scenarios reflect various oversubscriptions (planning pressure) and possible parallelism. A value of 300% for oversubscription means that the sum of all requested hours from all proposals exceeds the term duration by a factor of 3. As expected, the number of tasks (column 2) grows roughly linearly with the oversubscription (column 1), and the number of job combinations (column 4) grows exponentially with parallelism (indicated in Figure 22).

The number of days is held constant at 91 days (a quarter of a year), hence the number of time slots is $\frac{T}{\Delta T} = \frac{91\,days}{15\,minutes} = 8736$. In column 5, the number of possible schedules is indicated, calculated from the choices in the schedule space. This provides a measure of the search space size and can also be estimated by $log(\#\text{ of job combinations}) \times \frac{T}{\Delta T}$. Note that the number of job combinations is lower for the generated proposals for 200% oversubscription than it is for those in the 100% case. This is due to more shorter jobs in the 100% case.

Another measure of scheduling difficulty is plotting the number of demanded resources over time. Figure 23 shows, averaged over the days, the number of antennas requested over time, with the available number of antennas (42) indicated as well.

Oversubscription	Number of jobs	Max. Parallel jobs	Number of job combinations	log(# of Possible Schedules)
100%	64	1	64	20119
		2	84	20795
		3	114	20888
		4	134	20891
200%	84	1	84	18186
		2	108	18574
		3	138	18589
		4	153	18589
300%	140	1	140	24160
		2	214	25308
		3	457	26795
		4	954	28034
400%	179	1	179	24539
		2	294	25457
		3	729	26701
		4	1779	27768

Table 4: Test scenarios

The simulated bias towards the galactic center is immediately visible.

8 Evaluation results

8.1 Algorithm performance

Due to their simplicity, all heuristic algorithms show good scalability in relation with the problem complexity (number of job combinations), keeping the total execution time below a minute. However, the linear solver does not cope well with the stated problem. Its execution duration is in the days range up to a week. Hence, its solution is not used for the GA. A typical GA run of 1000 evaluations ($n_{gen} \times n_{pop} = 1000$) was between 500 and 5000 seconds.

The plots in Figure 24 show the algorithms fitness value in comparison to their runtime. Only the non-dominated algorithms are given markers. For non-dominated algorithms, there is no algorithm that provides a better result in shorter time. To understand which algorithms suffer more fragmentation than others, Figure 25 shows a plot of a fitness function without penalty for a change of observations vs. a fitness function with a high penalty for a change in observation (60 minutes as opposed to 15 minutes in the unmodified fitness function). These measures are indicated by "observatory-biased fitness value" and "observer-biased fitness value" respectively, as the observatory wants to accommodate many high-priority projects, and the observer wants uninterrupted observations.

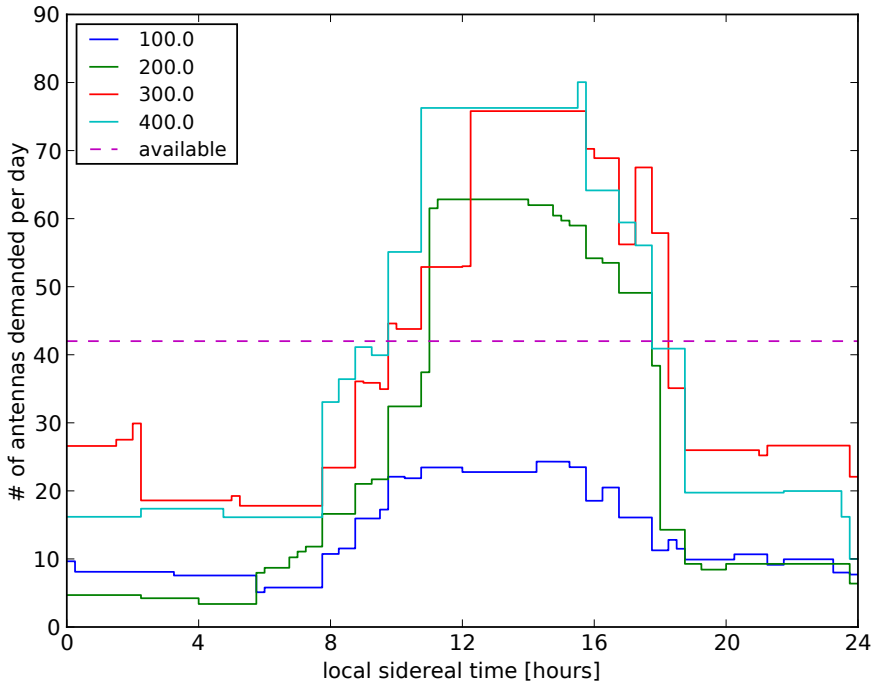


Figure 23: Resource demand of the test scenarios.

8.2 GA parameter meta-optimization

The "best" genetic operator probabilities found using meta-optimization are shown in table 5. The meta-optimization always disables the mutation operator (Mut), and prefers 2-point-crossover to 1-point-crossover. For the filled GA, only the MutationJobPlacement operator is left to introduce changes.

8.3 Algorithm qualitative comparison

Figure 26 shows how the various algorithm perform qualitatively – using the fitness function as the figure of merit. The GA is outperformed by a number of heuristics, even in the optimized configuration. The filled GA provides, of course, at least as good a result as the heuristics it started out with. The highest-scoring heuristics in this plot are "TrivialFirstParallelListing by Priority" and "SerialListing Prioritized". It should be noted that the fitness values are not directly comparable across the scenarios, as a scenario with more choice (higher oversubscription) and higher parallelism can usually achieve higher fitness than a scenario with less choice or stricter limits on parallelism. Additionally, as mentioned above, the specific, generated proposals of 200% oversubscription have less possible schedules than the other cases.

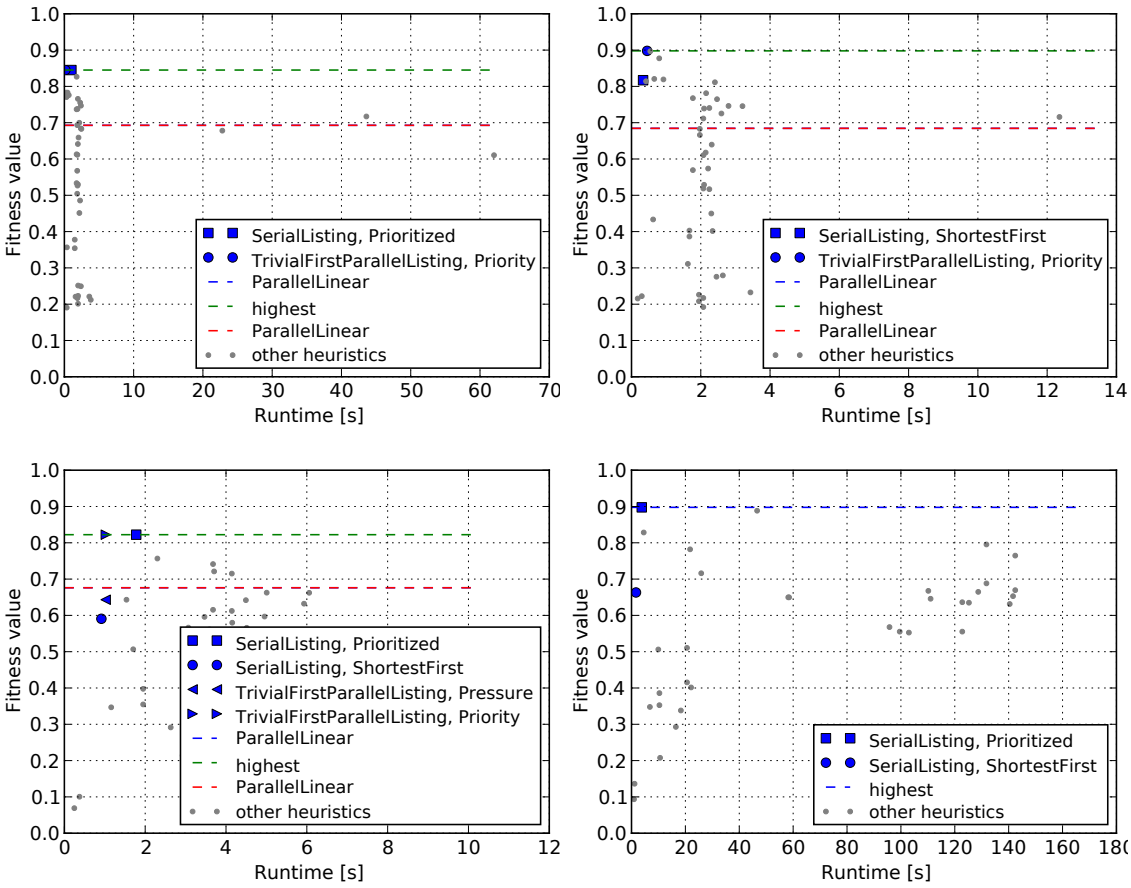


Figure 24: Algorithm quality vs. runtime. Top left panel: 100% oversubscription, 1 parallel task, Top right panel: 100% oversubscription, 4 parallel tasks, Bottom left panel: 400% oversubscription, 1 parallel task, Bottom right panel: 400% oversubscription, 2 parallel tasks.

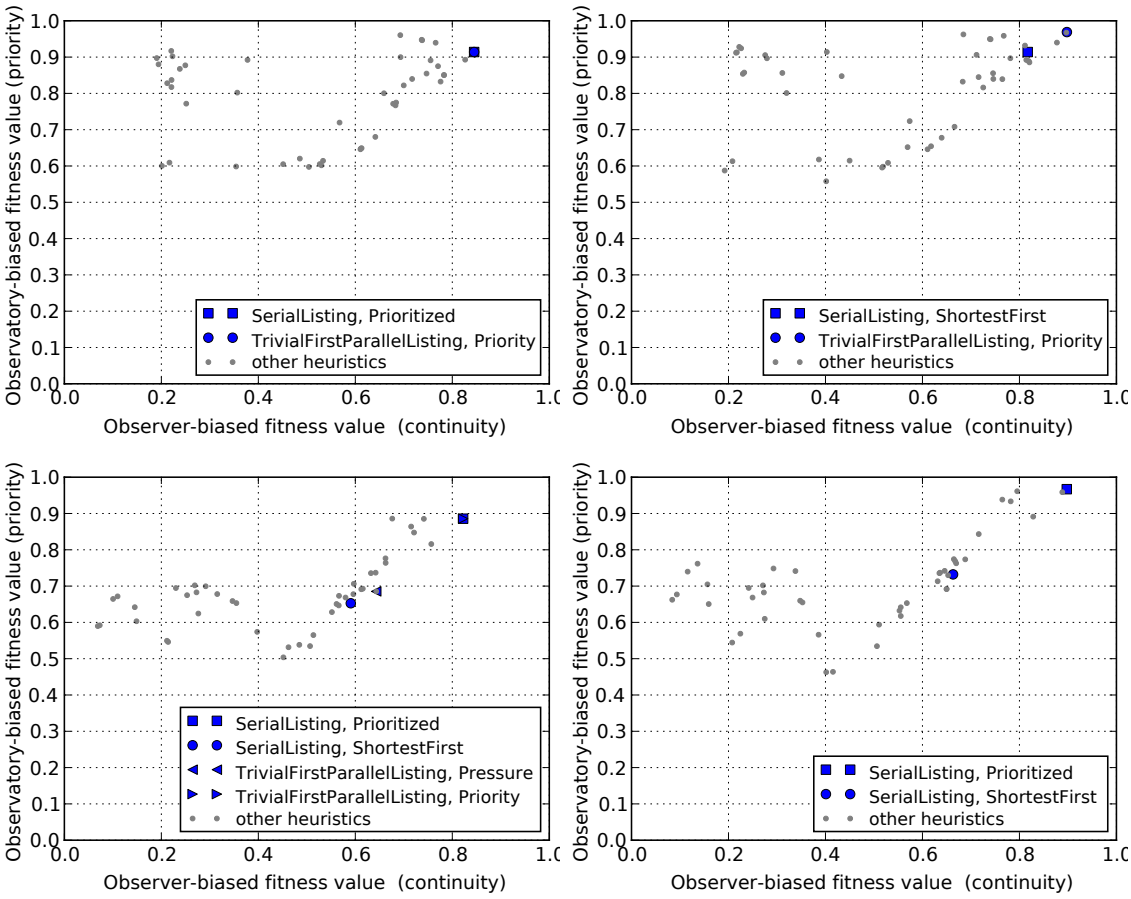


Figure 25: Algorithm quality with varied fitness function. Top left panel: 100% oversubscription, 1 parallel task, Top right panel: 100% oversubscription, 4 parallel tasks, Bottom left panel: 400% oversubscription, 1 parallel task, Bottom right panel: 400% oversubscription, 2 parallel tasks.

Parameter		Reference, Default configuration	Best Configuration, Empty GA	Best Configuration, Filled GA
Population	*size*	**50**	**41**	**61**
Crossover	*operator probability*	**0.1**	0	0
DoubleCrossover	*operator probability*	0	**0.1**	**0.099**
DoubleCrossover	*max days*	0	**1**	**15**
Mutation	*operator probability*	**0.1**	0	0
MutKeep	*operator probability*	0	**0.019**	0
MutSimFw	*operator probability*	0	**0.012**	0
MutSimBw	*operator probability*	0	**0.02**	0
MutSimPrev	*operator probability*	0	**0.048**	0
MutEx	*operator probability*	0	**0.019**	0
MutPlace	*operator probability*	0	**0.031**	**0.006**

Table 5: Summary of best GA configuration

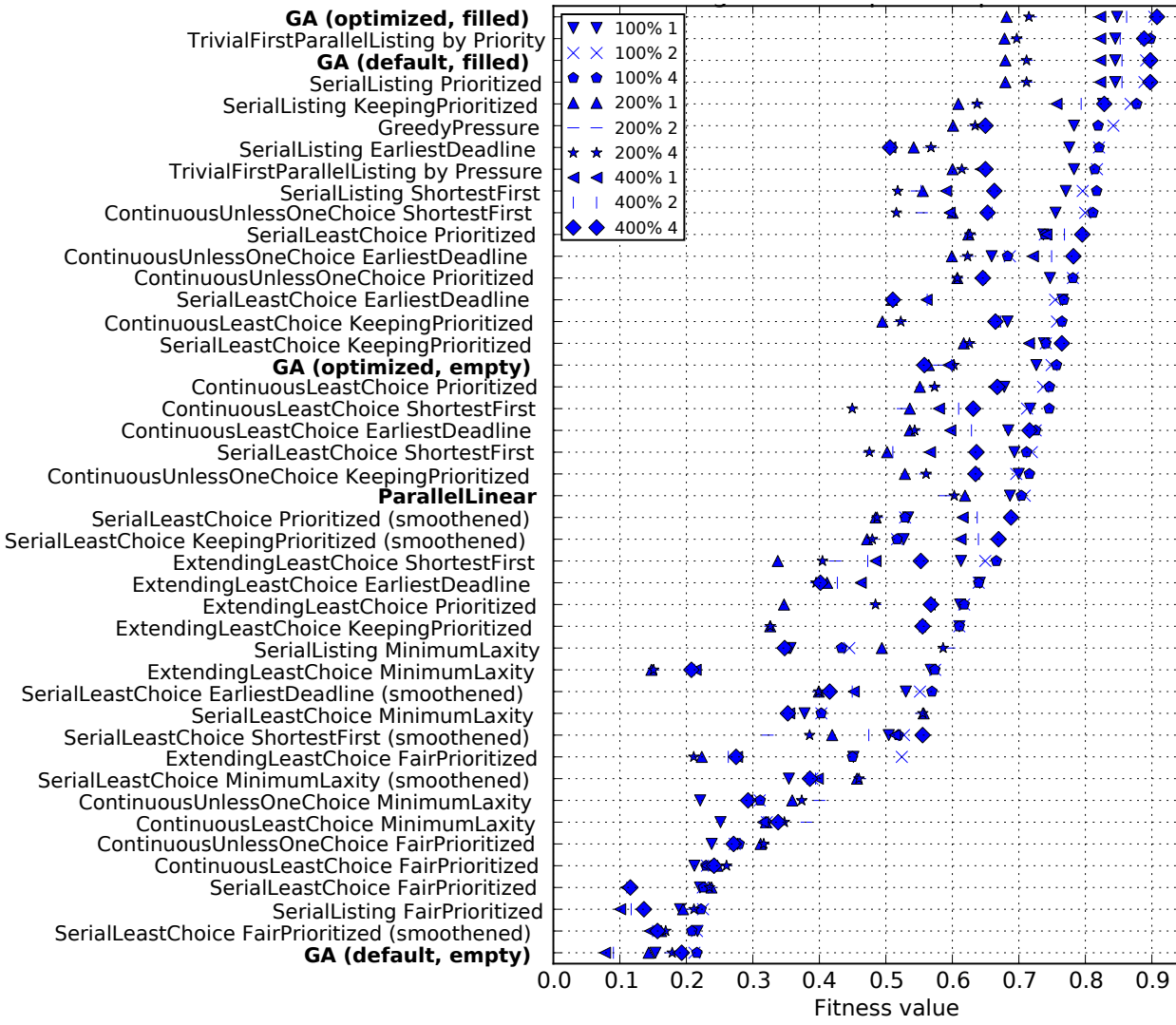


Figure 26: Qualitative comparison of algorithm output by fitness value.

To give an insight into the progress of the GA in the various configurations,

Figure 27 on the following page, Figure 28 on page 77 and Figure 29 on page 78 show the development of various configurations over time. The x-axis refers to a time measure, which is, to make the lines comparable for different population sizes, measured in the number of schedules generated (generation number multiplied by population size). The data points represent the best chromosome's fitness value, the error bar shows the standard deviation of the population. Runs of the GA configurations with the default and optimized configurations are shown, each in the empty variant (starting from bottom left in the graph) as well as the filled variant (starting from the best heuristic value, top left). It can be seen that the GA by itself does not do better than the best heuristic.

To answer the question whether the GA can build upon the heuristic solutions, how and it improves them, the following plots in table 7 on page 79 show an insight to the influences of those initial solutions, as well as the operators involved in creating the best schedule. The left column shows various configurations used. In the second column, the ancestors to the best solution are summarized. These schedules from other algorithms, have been crossed, mutated and subsequently selected throughout the run of the GA. The bar indicates how often the algorithm is a parent (over all test sets). The last column shows, for the best schedule, how often each operator was applied to an average gene. These modifications led to the surviving, best individual.

Obviously, in the "empty" GA case, there is no ancestry (hence the N/A entries). The "SerialListing Prioritized" and "TrivialFirstParallelListing by Priority" heuristics prove to be most influential. This is in agreement with the above finding, that they achieve the highest fitness value. In the filled default configuration case it is interesting to see that no modifications survive, i.e. the GA only selects the best heuristic but is not able to improve it.

8.4 Individual algorithm output

In this section, the actual schedules produced by various algorithms are illustrated. The schedule visualization displays days (vertically) and hours (horizontally). Each cell represents a timeslot of 15 minutes. Jobs that have already completed, but were still scheduled are shown in white font.

The *SerialListingScheduler* using the *FairPriority* strategy shown in Figure 30 on page 80 obviously produces a large amount of fragmentation – only few observation blocks last longer than one timeslot.

In contrast, the *SerialListingScheduler* using the *Priority* strategy scheduler shown in Figure 31 on page 81 produces more continuous observation. The result from the *TrivialFirstParallelListingScheduler* using the *Priority* strategy scheduler is very similar (shown in Figure 32 on page 82).

The GA with the default configuration, shown in Figure 33 on page 84, also

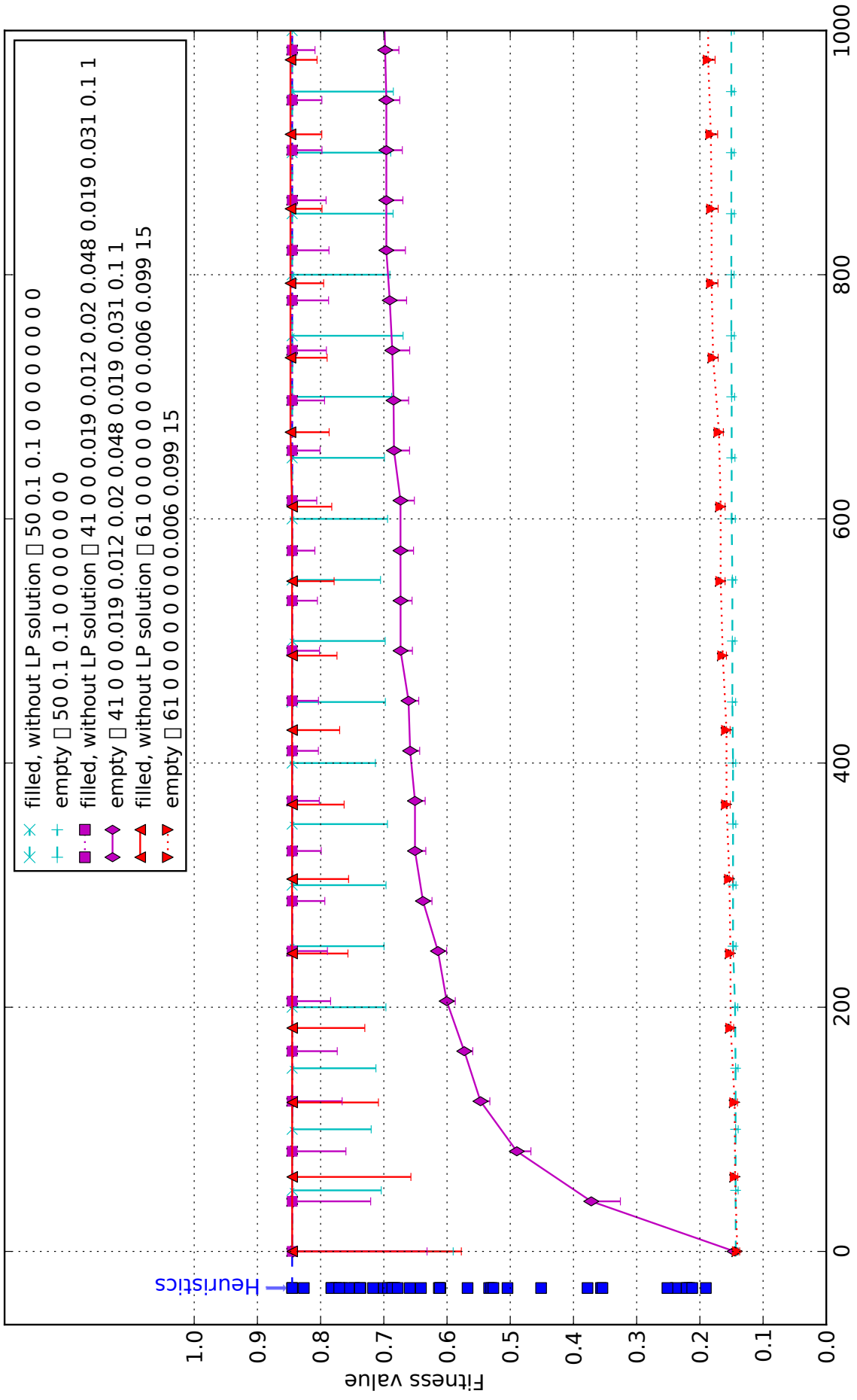


Figure 27: GA population development over time in various configurations. 100% oversubscription, no parallel tasks.

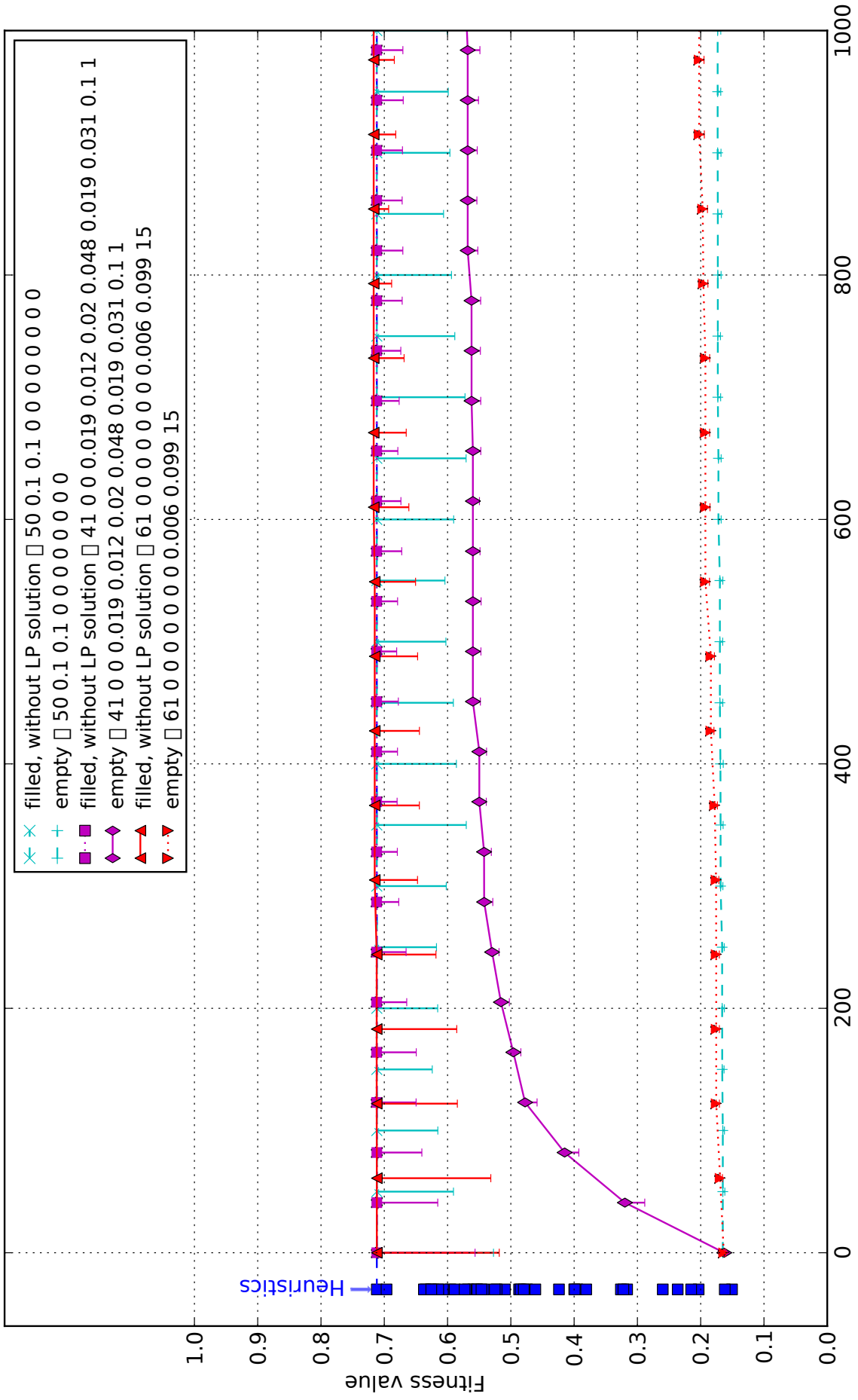


Figure 28: GA population development over time in various configurations. 200% oversubscription, max 2 parallel tasks.

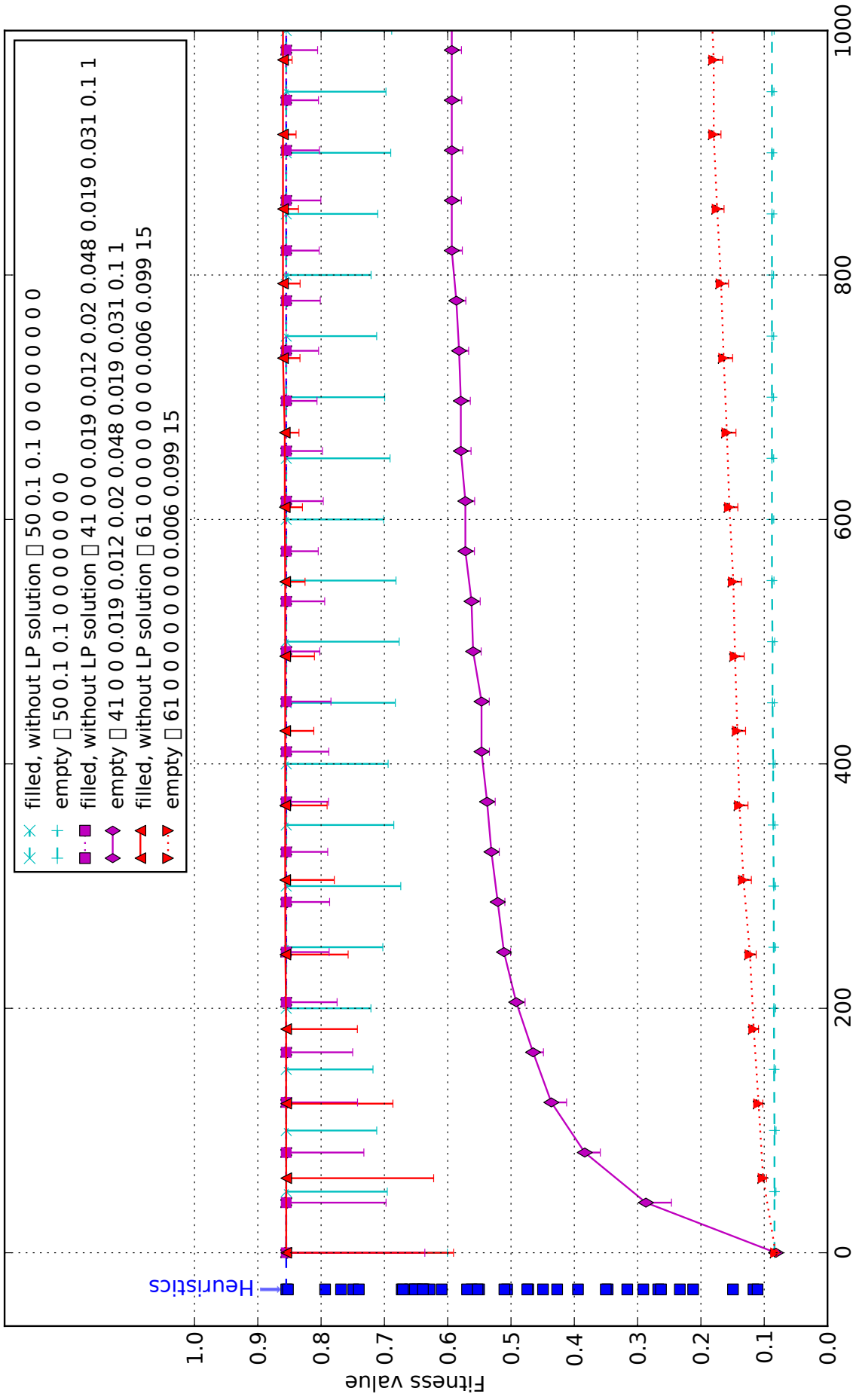


Figure 29: GA population development over time in various configurations. 400% oversubscription, max 4 parallel tasks.

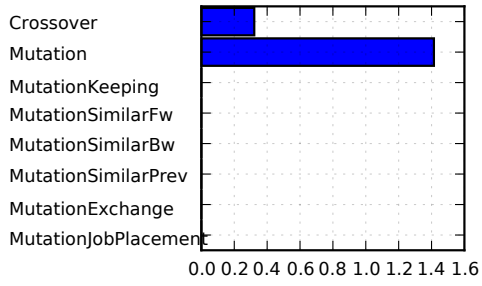
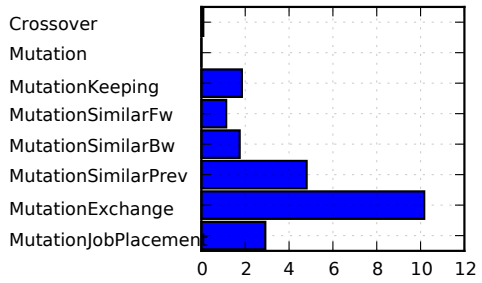
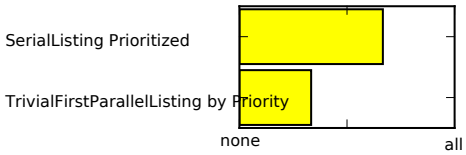
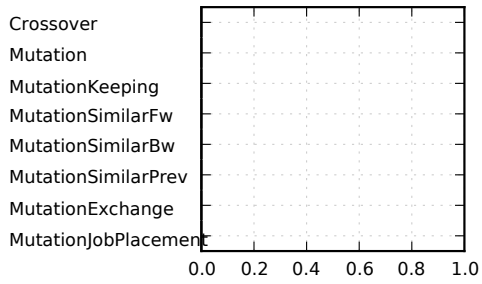
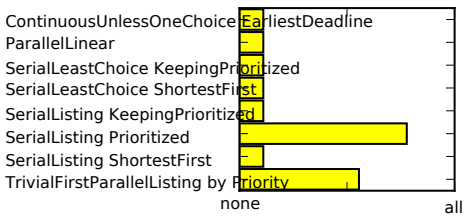
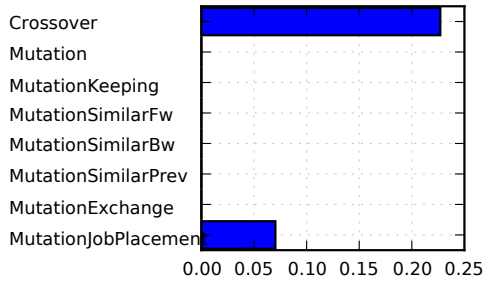
GA configuration	Genetic Origin of the Surviving Population	Operator Counters of the Surviving Population
empty, default configuration	N/A	
empty, optimized configuration)	N/A	
$n_{pop} = 41$, $p_{Crossover} = 0$, $p_{Mut} = 0$, $p_{MutKeep} = 0.019$, $p_{MutSimFw} = 0.012$, $p_{MutSimBw} = 0.02$, $p_{MutSimPrev} = 0.048$, $p_{MutEx} = 0.019$, $p_{MutPlace} = 0.031$, $p_{DoubleCrossover} = 0.1$, $ndays_{Crossover} = 1$		
filled, default configuration		
filled, optimized configuration		
$n_{pop} = 61$, $p_{Crossover} = 0$, $p_{Mut} = 0$, $p_{MutKeep} = 0$, $p_{MutSimFw} = 0$, $p_{MutSimBw} = 0$, $p_{MutSimPrev} = 0$, $p_{MutEx} = 0$, $p_{MutPlace} = 0.006$, $p_{DoubleCrossover} = 0.099$, $ndays_{Crossover} = 15$		

Table 7: An insight into the GA genetic history.

day \ LST	0			1			2		
0	Survey 3	SETI	Proposal1 08/2739	SETI			Waterhole	Survey 3	
1	Proposal1 08/2739	SETI		Proposal1 08/2739	Survey 3	Waterhole	Proposal1 08/2739	Survey 3	Proposal1 08/2739
2	Survey 3	Proposal1 08/2739	Waterhole	Survey 3	SETI	Survey 3	Proposal1 08/2739		
3	Waterhole	Proposal1 08/2739	Survey 3		Waterhole	Proposal1 08/2739	Survey 3	Proposal1 08/2739	Survey 3	Waterhole
4	Survey 3	Proposal1 08/2739			SETI	Proposal1 08/2739	SETI	Waterhole	Survey 3	Proposal1 08/2739
5	SETI	Survey 3	Waterhole	Survey 3	Proposal1 08/2739	SETI	Proposal1 08/2739	SETI	Survey 3	Waterhole
6	SETI	Proposal1 08/2739		Survey 3	Waterhole	SETI	Waterhole	Proposal1 08/2739		

Figure 30: A section of the schedule produced by *SerialListingScheduler* using the *FairPriority* strategy.

day \ LST	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16			17
0	Proposal108/2739					Survey 3		Survey 3	NGC7539	NGC2815 NGC9223 NGC2625 NGC3143	NGC2815 NGC9223 NGC8838 NGC3143	NGC1536 NGC9223 NGC3143 NGC7846					NGC1536 NGC8838 NGC3143 NGC7846			NGC6211
1	Proposal108/2739					Survey 3		Survey 3	NGC7539	NGC2815 NGC9223 NGC2625 NGC3143	NGC2815 NGC9223 NGC8838 NGC3143	NGC1536 NGC9223 NGC3143 NGC7846					NGC1536 NGC8838 NGC3143 NGC7846			NGC6211
2	Proposal108/2739					Survey 3		Survey 3	NGC7539	NGC2815 NGC9223 NGC2625 NGC3143	NGC2815 NGC9223 NGC8838 NGC3143	NGC1536 NGC9223 NGC3143 NGC7846					NGC1536 NGC8838 NGC3143 NGC7846			NGC6211
3	Proposal108/2739					Survey 3		Survey 3	NGC7539	NGC2815 NGC9223 NGC2625 NGC3143	Maintainance 157/6 NGC2815 NGC9223 NGC3143	NGC1536 Maintainance 157/6 NGC3143 NGC7846			NGC1536 Maintainance 129/8 NGC3143 NGC7846		NGC1536 NGC8838 NGC3143 NGC7846			NGC6211
4	Proposal108/2739					Survey 3		Survey 3	NGC7539			NGC1536 NGC2815 NGC7846 NGC3143					NGC1536 NGC8838 NGC3143 NGC7846			NGC6211
5	Proposal108/2739					Survey 3		Survey 3	NGC7539			NGC1536 NGC2815 NGC7846 NGC3143		NGC1536 NGC8838 NGC3143 NGC7846			NGC6211			
6	Proposal108/2739					Survey 3		Survey 3	NGC7539								NGC6211			
7	Proposal108/2739					Survey 3		Survey 3	NGC7539								NGC6211			
8	Proposal108/2739					Survey 3		Survey 3	NGC7539								NGC4134			
9	Proposal108/2739					Survey 3		Survey 3	NGC7539					NGC8918			NGC4134			
10	Proposal108/2739					Survey 3		Survey 3	NGC8918		Maintainance 131/7									NGC4134
11	Proposal108/2739					Survey 3		Survey 3	NGC8918								NGC4134			
12	Proposal108/2739					Survey 3		Survey 3	NGC8918					NGC1053			NGC4134			
13	Proposal108/2739					Survey 3		Survey 3	NGC1053								NGC4134			

Figure 31: A section of the schedule produced by *SerialListingScheduler* using the *Priority* strategy.

day \ LST	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	Proposal 108/2739					Survey 3	Survey 3		NGC7539								NGC6211	
1	Proposal 108/2739					Survey 3	Survey 3		NGC7539								NGC6211	
2	Proposal 108/2739					Survey 3	Survey 3		NGC7539								NGC6211	
3	Proposal 108/2739					Survey 3	Survey 3		NGC7539								Maintainance 131/7	NGC6211
14	Proposal 108/2739					Survey 3	Survey 3		NGC1053				NGC4134					
15	Proposal 108/2739					Survey 3	Survey 3		NGC4832		NGC4134							
16	Proposal 108/2739					Survey 3	Survey 3		NGC4832		NGC4134				NGC4832		Jupiter	NGC4045
17	Proposal 108/2739					Survey 3	Survey 3		NGC4832			Maintainance 151/5				Maintainance 157/6 NGC9223 NGC7846 NGC3143	NGC1536 Maintainance 157/6 NGC3143 NGC7846	NGC1536 Maintainance 157/6 NGC8838 NGC7846	NGC4045

Figure 32: A section of the schedule produced by *TrivialFirstParallelListingScheduler* using the *Priority* strategy.

produces large amounts of fragmentation. In comparison, the operations of the optimized GA, shown in Figure 31 on page 81, were more successful in keeping observations continuous. The result of the filled GA in optimized configuration is shown in Figure 35 on page 86.

day \ LST	0			1			2					
0	Waterhole		SETI	Survey 3	SETI		Survey 3	Waterhole	Proposal1 08/2739	Survey 3		Proposal1 08/2739
1	Survey 3	Proposal1 08/2739	Survey 3	Waterhole			Proposal1 08/2739	SETI	Waterhole	Proposal1 08/2739	SETI	Proposal1 05/62
2	SETI		Survey 3	SETI	Proposal1 08/2739		Survey 3	Waterhole	Proposal1 05/62	Waterhole	Proposal1 05/62	Proposal1 08/2739
3	Survey 3	Waterhole		SETI	Waterhole		Survey 3	Waterhole	SETI	Waterhole	SETI	Proposal1 05/62
4	Proposal1 08/2739	SETI	Survey 3	Waterhole			Proposal1 08/2739	Survey 3		Proposal1 05/62		
5	Survey 3	Waterhole	Proposal1 08/2739	SETI	Survey 3		Waterhole	SETI		Proposal1 08/2739		

Figure 33: A section of the schedule produced by the empty GA in default configuration.

day \ LST	0	1	2	3	4	5	6	7	8	9	10	11
0	Waterhole			SETI			Survey 3		Waterhole	NGC2625 NGC764	NGC9223 NGC3143 NGC5846	NGC2721
											NGC8838	
1	Proposal108/2739						Proposal105/62	SETI		NGC9223 NGC2625 NGC764 NGC2815	NGC9223 NGC3143 NGC5846 NGC8838	
											NGC9223 NGC2625 NGC2815	NGC7
2	Proposal108/2739						SETI		NGC9223 NGC2625 NGC764 NGC2815	NGC9223 NGC3143 NGC2625 NGC2815	NGC3143 NGC5846 NGC8838 NGC764	
												NGC3143
3	Proposal108/2739						Waterhole		NGC9223 NGC2625 NGC764 NGC2815	NGC9223 NGC2625 NGC2815	NGC9223 Maintainance 157/6 NGC2625 NGC2815	
4	SETI						Waterhole		Proposal105/62	Proposal103/175		
5	Proposal108/2739			Waterhole			Survey 3		NGC9223 NGC2625 NGC764 NGC2815	NGC9223 NGC3143 NGC2625 NGC2815	NGC3143 NGC5846 NGC8838 NGC764	
												NG
6	Proposal108/2739					Waterhole		SETI		NGC9223	NGC9223 NGC3143 NGC2625	NGC8838
												NGC5846
7	Proposal108/2739	SETI						Waterhole		SETI	NGC5846 NGC2625 NGC764	NGC9223 NGC3143 NGC5846
												NG
8	Proposal108/2739					SETI		Survey 3			NGC9223 NGC3143 NGC764 NGC2815	
												NGC8
9	Waterhole						SETI			NGC9223 NGC3143 NGC5846	NGC9223 NGC3143 NGC5846 NGC8838 NGC764	NGC3143 NGC5846 NGC8838
												NGC3448

Figure 34: A section of the schedule produced by the empty GA in optimized configuration.

day \ LST	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	Proposal108/2739					Survey 3	Survey 3	NGC7539	NGC9223 NGC3143 NGC2625 NGC2815	NGC9223 NGC3143 NGC8838 NGC2815	NGC7846 NGC9223 NGC3143 NGC1536					NGC7846 NGC3143 NGC8838 NGC1536	NGC6211
1	Proposal108/2739					Survey 3	Survey 3	NGC7539	NGC9223 NGC3143 NGC2625 NGC2815	NGC9223 NGC3143 NGC8838 NGC2815	NGC7846 NGC9223 NGC3143 NGC1536					NGC7846 NGC3143 NGC8838 NGC1536	NGC6211
2	Proposal108/2739					Survey 3	Survey 3	NGC7539	NGC9223 NGC3143 NGC2625 NGC2815	NGC9223 NGC3143 NGC8838 NGC2815	NGC7846 NGC9223 NGC3143 NGC1536					NGC7846 NGC3143 NGC8838 NGC1536	NGC6211
3	Proposal108/2739					Survey 3	Survey 3	NGC764 NGC2815	NGC3143 NGC5846 NGC764 NGC2815	Maintainance 124/10 NGC5846 NGC764 NGC2815						Maintainance 124/10 NGC3143 NGC5846	NGC7846 NGC3143 NGC5846 NGC1536	NGC7846 NGC8838
4	Proposal108/2739					Survey 3	Survey 3	NGC7539			NGC7846 NGC3143 NGC1536 NGC2815						NGC7846 NGC3143 NGC8838 NGC1536	NGC6211
5	Proposal108/2739					Survey 3	Survey 3	NGC7539			NGC7846 NGC3143 NGC1536	NGC7846 NGC3143 NGC8838 NGC1536					NGC6211	
6	Proposal108/2739					Survey 3	Survey 3	NGC7539									NGC6211	
7	Proposal108/2739					Survey 3	Survey 3	NGC7539									NGC6211	
8	Proposal108/2739					Survey 3	Survey 3	NGC7539									NGC4134	
9	Proposal108/2739					Survey 3	Survey 3	NGC7539						NGC8918			NGC4134	
10	Proposal108/2739					Survey 3	Survey 3	NGC9223	NGC9223 NGC3143 NGC5846	NGC9223 Maintainance 148/7 NGC3143 NGC5846						NGC7846 Maintainance 148/7 NGC5846	NGC7846 NGC5846 NGC1536	NGC4134
11	Proposal108/2739					Survey 3	Survey 3	NGC8918									NGC4134	
12	Proposal108/2739					Survey 3	Survey 3	NGC8918						NGC1053			NGC4134	
13	Proposal108/2739					Survey 3	Survey 3	NGC1053									NGC4134	
14	Proposal108/2739					Survey 3	Survey 3	NGC1053									NGC4134	

Figure 35: A section of the schedule produced by the filled GA in optimized configuration.

Part IV
Discussion

9 Performance of Scheduling algorithms

The problem of observation scheduling discussed here seems simple when compared to project scheduling for instance, as there are no predecessor relationships. However, in other aspects it is more difficult than the RCPSP as we allow arbitrary preemption, but would like to avoid such interrupts. A similar situation is CPU scheduling, where each task should get a turn, but swapping is costly. In long-term scheduling the number of combinations making up the solution space is large (up to $exp(28000) = 10^{12160}$ in the cases looked at). This number will quickly increase for longer scheduling terms and for a higher number of (shorter) proposals, as well as allowing more parallel observations (e.g. 8-12 for SKA).

Unfortunately, the LP solution, which is expected to give an optimal result (if only to a simplified cost function) does not deal well with this complexity and takes an unreasonable time to finish (days - weeks). Furthermore, it is vastly outperformed by heuristics and the GA.

"SerialListing, Prioritized" and "TrivialFirstParallelListing, Priority" are fast heuristics that provide excellent solutions, both accommodating high-priority proposals and avoiding interrupts. The former is especially easy as it just goes through all time slots and picks the job combination with the highest priority. Continuous observations are just an emerging effect. The parallel listing algorithm however is placing jobs in priority order on the schedule. These algorithms are straight-forward, easy to implement and very fast.

9.1 Genetic algorithm

Concerning the genetic algorithm, the issue of fragmentation was immediately reproducible. In fact, it is visible that the standard crossover and mutation operators are not successful in probing the problem space towards a better solution. This can be, as known from e.g. Barbulescu et al. (2006) and Wall (1996), attributed to the fact that only single, independent moves are made, and that it is very unlikely for the single moves to fall in such an order that a improvement is found.

With the new, added mutation operators – *MutationKeeping*, *MutationSimilarForward*, *MutationSimilarBackward*, *MutationSimilarPrev, MutationExchange, MutationJobPlacement* – that do multiple moves at once, e.g. overriding and extending a block, the GA was much more successful. In fact, meta-optimization disabled the ineffective single-gene operations (simple *Mutation*), and enabled several of the other ones.

The mutation operators *MutationKeeping*, *MutationSimilarForward*, *MutationSimilarBackward* are very similar. The only difference between them is that the first randomly picks an observation and keeps it forward, while the other two use the already-in-place observation. The difference between the latter is only the direction of extension in time.

The "filled" GA is only successful when most operators are disabled. Apparently, the GA needs to be careful to keep the solutions at this high fitness value, so that only few modifications at a time are possible. Mostly crossover is used, with the *MutPlace* operator placing random jobs. This optimized GA is capable of making minor improvements to the heuristics.

Comparing Figure 33 and Figure 35, it is also observable in the displayed schedules that fragmentation is effectively removed and avoided by these block-wise operators as intended. However, the “empty” GA is clearly outperformed by the heuristics.

Hence, it might seem that selecting the "best" heuristic is sufficient and the problem is solved. However, one should keep in mind that the problem might change, and the heuristics are not smart enough to understand complex preferences. Hence we recommend running "SerialListing, Prioritized" and "TrivialFirstParallelListing, Priority" first, and running the GA filled with those solutions for several minutes. This has the benefits, that

- The schedule can be updated when constraints change. The re-scheduling can take the previous scheduling solution into account. This allows open-ended update-able scheduling.
- The schedule can also be updated when preferences change.
- Even if the GA can not make any improvements, it at least selects the heuristic that works best in this problem setting.

We believe that the graphs in Table 7 show an interesting insight into the working of the GA. When an initial population is used, it can be tracked for a measure of influence. The number of successful applications of an operator can be used as a measure of the operators success. In an extreme case, “filled” GA with default configuration, the GA could not improve the initial solution at all, as the corresponding graph in Table 7 shows. We recommend these visualizations for other studies. In future works, these measures might for instance be useful when developing an adaptive GA that measures the effectiveness of operations and modifies their probabilities based on it. See e.g. Hartmann (2002) for a self-adapting GA. Furthermore, as the best configuration for the empty GA introduces many changes, this might improve its performance where it currently saturates.

It should be stressed that endless more operators and scenarios could be thought of, but the presented results give an insight to the problem space and the inner workings of the genetic algorithm under various circumstances. However, already,

the issue of fragmentation can be claimed to be solved to some degree, by replacing the simple gene-wise Mutation operator.

Furthermore, many complications to the approach can be thought of. To mention just one, the problem can be stated so that the task duration is dependent on the resources committed to it. In radio astronomy, this is directly a result of the radiometer equation, which relates the integration time with the number of observation samples obtained (which is related to the number of detectors).

The code used to track the genetic linage and initial populations influence can be found in 16. For completeness, the operators implementations are attached: *MutationExchange* in Listing 11, *MutationKeeping* in Listing *14*, *MutationSimilarForward* and *MutationSimilarBackward* in Listing *12*, *MutationJobPlacement* in Listing *15*.

10 Real-world integration

10.1 Planning and Scheduling

With an oversubscribed schedule, we inevitably find ourselves in the research field of integrated planning and scheduling (Smith et al. (2000)). The scheduling algorithm has to decide in some way, which observations to drop.

However, in observatories, the situation is often like with CPU schedulers: The point of view is that everything planned will be done eventually. Hence, since time is open, unfinished observations can be carried along to the next term, possibly with a higher priority.

10.2 Scheduling and Control Systems & Monitoring

After the long process of scheduling, having a schedule is of course not satisfactory as such.

First of all, it has to be executed. Hence some *loose* (e.g. manual steering based on a print-out) or *tight integration* (fully automated fetching next task) of the control system with the scheduling system is necessary. Since the first is trivial, we demonstrate the second kind.

The NRAO "*fieldsystem*" is a popular control system in radio astronomy observatories, especially for those related to the VLA and VLBI networks. A task is correctly executed at a specific time by writing a file in "*keyin*" format, which describes for each involved antenna the procedure necessary – the configuration of all devices, and the timings of the observations.

This file goes through *SCHED*, which, despite the name, does not do any scheduling. It is more a high-level control system preprocessor. *SCHED* produces files for each station, which *drudg* in turn makes into executable scripts at each antenna.

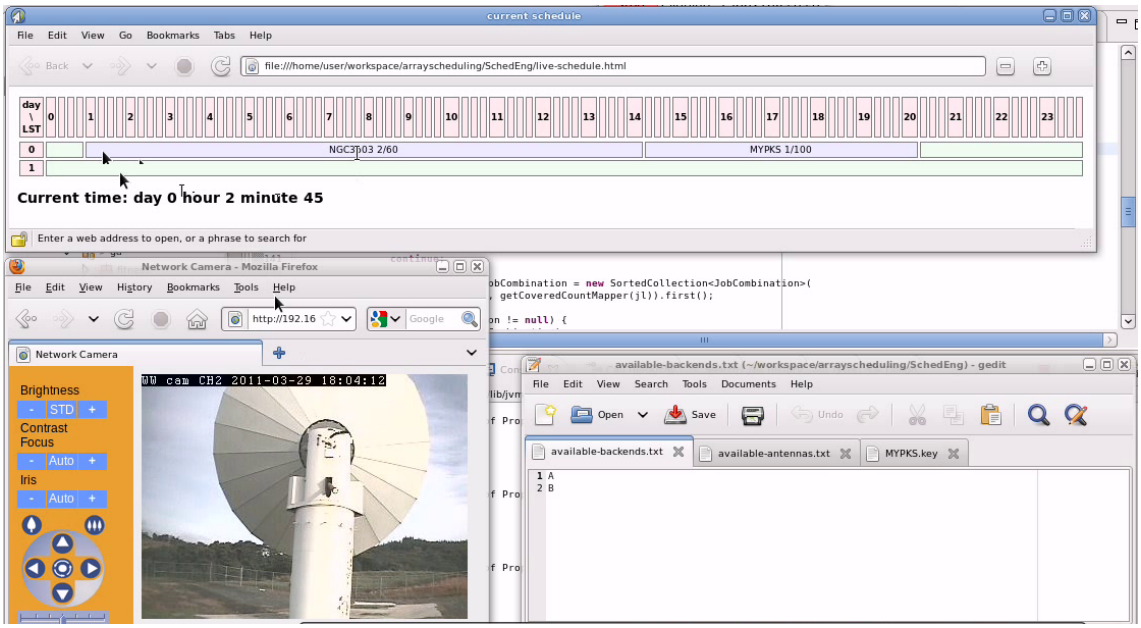


Figure 36: Screen shot of the demo environment after initial scheduling. The current schedule is shown on the top with the current time: 2h45m. Currently, the higher-priority observation is scheduled and executed. Bottom left: Web-cam view of the Warkworth 12m telescope pointing and observing NGC3603. Bottom right: the monitoring system currently shows both backends as available.

These are then finally run on the fieldsystem, which, being run at each station independently, coordinates all devices there.

Tight integration demo Listing 7 on the following page shows an observation template that is ready to be preprocessed by SCHED. The scheduler only has to fill in time and equipment allocated (e.g. using m4), and then this task can be run. We prepared two such templates for two different locations – PKS 0539-691 and NGC3603. Additionally, a proposal for each is prepared, the one with of NGC3603 having a higher priority.

A simple mock monitoring system that reads text files was implemented. Furthermore, a Control system frontend was made that replaces the station and current time in the right template, sends the file through SCHED, drudg and into the fieldsystem for immediate execution.

With these components, the feedback loop between planning, scheduling, control system and monitoring was demonstrated as shown in the screen shot figures 36 and 37 on page 94.

10.3 User interaction in scheduling

Naturally, a complete automated system without human checks is not the goal. A tradeoff is to be made between allowing manual interaction, and making schedule changes easy and not too time-consuming.

Algorithm 7 A minimalistic fragment of a observation task specification for SCHED. Only the text STARTYEAR, STARTMONTH, STARTDAY and STARTTIME were replaced, and the ACTIVESTATIONS has to be set to a list of stations involved in this observation.

```
! SCHED template to observe NGC3603
srccat /
EQUINOX = J2000
SOURCE='J1115-6116','NGC3603'
      RA=11:15:09.1 DEC=-61:16:17 RAERR=    0.1 DECERR=
0.1 CALCODE='V'
      REMARKS='NGC 3603 is a beautiful circumpolar HII region
'
      FLUX =    2.30,   0.50,   0.34,    8.60,   0.21,   0.09
FLUXREF = 'X/S  rfc_2010c '
 /
endcat /

! ============================================================
!       Cover information  (PI, experiment ...)
! ============================================================
version  = 1
expt     = 'We observe NGC3603!'
expcode  = EXPCODE
obstype  = 'VLBA'

piname   = 'Johannes Buchner'
email    = 'johannes.buchner.acad@gmx.com'
obsmode  = '128-4-2'

! ============================================================
!       Catalogs.  These are the standard ones and are the defalts.
! ============================================================
stacat /
     STAtion=WARKWRTH STCode=Ww DBCODE=Ww  DBNAME=Ww
     AXISTYPE=ALTAZ AXISOFF=0.01
     FRAME='random '
     LAT=-36:25:48
     LONG=174:39:36
     ELev=132.0
     TSETTLE=6  DAR=VLBA  NBBC=8 ! RECORDER=VLBA  NDRIVES=2
NHEADS=1
     DISK=MARK5A    MEDIADEF=DISK     TSCAL=CONT
/
endcat /
year   = STARTYEAR
month  = STARTMONTH
day      = STARTDAY
start    = STARTTIME

stations = ACTIVESTATIONS
setup  = '$SCHED/setups/v4cm-128-4-2.set'
source = 'NGC3603'  dur = 300 gap = 0  record /
```

Algorithm 8 Scheduling loop for demo. If resources change, the schedule is updated, taking past observations into account and continuing the operation.
The ScheduleExport provides a mean of visualization (HTML export). The monitoring system (on a high level) gives information about the resource availability. In reality, this would be event-based. The Control system is told the tasks (job combination) to execute, and is guaranteed that the resources to do so are available.

```
1  ReusableScheduler scheduler = new ReusableScheduler();
2  scheduler.setNdays(ndays);
3
4  ControlSystem cs = new NraoBasedControlSystem();
5  MonitoringSystem mon = new TextFileMonitoringSystem();
6  ScheduleExport ex = new HtmlScheduleExport();
7
8  LSTTime currentTime = null;
9  Schedule s = null;
10
11 while (true) {
12      scheduler.setAvailableResources("antennas",
13          mon.getAvailableAntennas());
14      scheduler.setAvailableResources("backends",
15          mon.getAvailableBackends());
16
17      scheduler.updateScheduleSpace(proposals, currentTime, s);
18
19      log.info("advancing schedules ...");
20      scheduler.advanceSchedules();
21      s = scheduler.getCurrentSchedule();
22      ex.export(s, currentTime);
23      log.info("executing ...");
24
25      for (Entry<LSTTime, JobCombination> e : s) {
26          LSTTime t = e.getKey();
27          // on re-entry, skip forward
28          if (currentTime != null && t.isBeforeOrEqual(currentTime)) {
29              continue;
30          }
31          currentTime = t;
32          ex.export(s, currentTime);
33          cs.execute(e.getValue());
34
35          measureEnvironment(t);
36
37          if (mon.haveResourcesChanged()) {
38              log.debug("resources have changed");
39              break;
40          }
41      }
42 }
```

Algorithm 9 A scheduler that allows reentry, and combines all algorithms available. updateScheduleSpace carves the schedule space, removing the choice from the past, but keeping them for calculation. The subsequent methods run the relevant schedulers and store the best result.

```
1  public class ReusableScheduler {
2
3      public void updateScheduleSpace(Collection<Proposal> proposals,
4              LSTTime currentTime, Schedule previousSchedule) {
5          log.debug("creating schedule space");
6          ITimelineGenerator tlg = new SimpleTimelineGenerator(
7                  getRequirementGuard());
8          space = tlg.schedule(proposals, ndays);
9          log.debug("created schedule space " + space.findLastEntry());
10
11         for (Entry<LSTTime, Set<JobCombination>> e : space) {
12             LSTTime t = e.getKey();
13             // remove choice from past
14             if (currentTime != null && t.isBeforeOrEqual(currentTime)) {
15                 space.clear(t);
16                 JobCombination jc = previousSchedule.get(t);
17                 if (jc != null)
18                     space.add(t, jc);
19             }
20         }
21     }
22
23     public void advanceSchedules() {
24         // creating heuristic initial population
25         Map<IScheduler, Schedule> schedules2 = HeuristicsScheduleCollector
26                 .getStartSchedules(space);
27         heuristicschedules = new HashMap<String, Schedule>();
28         for (Entry<IScheduler, Schedule> e : schedules2.entrySet()) {
29             heuristicschedules.put(e.getKey().toString(), e.getValue());
30             if (e.getKey().toString().contains(PREFERRED_HEURISTIC)) {
31                 bestSchedule = e.getValue();
32             }
33         }
34
35         schedules = heuristicschedules.values();
36         if (GA_ENABLED) {
37             advanceSchedulesWithGA();
38         }
39     }
40
41     private void advanceSchedulesWithGA() {
42         SimpleScheduleFitnessFunction f = new SimpleScheduleFitnessFunction();
43         f.setSwitchLostMinutes(15);
44
45         WFScheduler scheduler = new WFScheduler(f);
46         scheduler.setPopulation(schedules);
47         scheduler.setNumberOfGenerations(numberOfEvaluations / populationSize);
48         scheduler.setEliteSize(2);
49         scheduler.setCrossoverProbability(crossoverProb);
50         scheduler.setMutationProbability(mutationProb);
51         scheduler.setPopulationSize(populationSize);
52         scheduler.setMutationExchangeProbability(mutationExchangeProb);
53         scheduler.setMutationSimilarBackwardsProbability(mutationSimilarProb);
54
55         bestSchedule = scheduler.schedule(space);
56         schedules = scheduler.getPopulation();
57     }
58 }
```

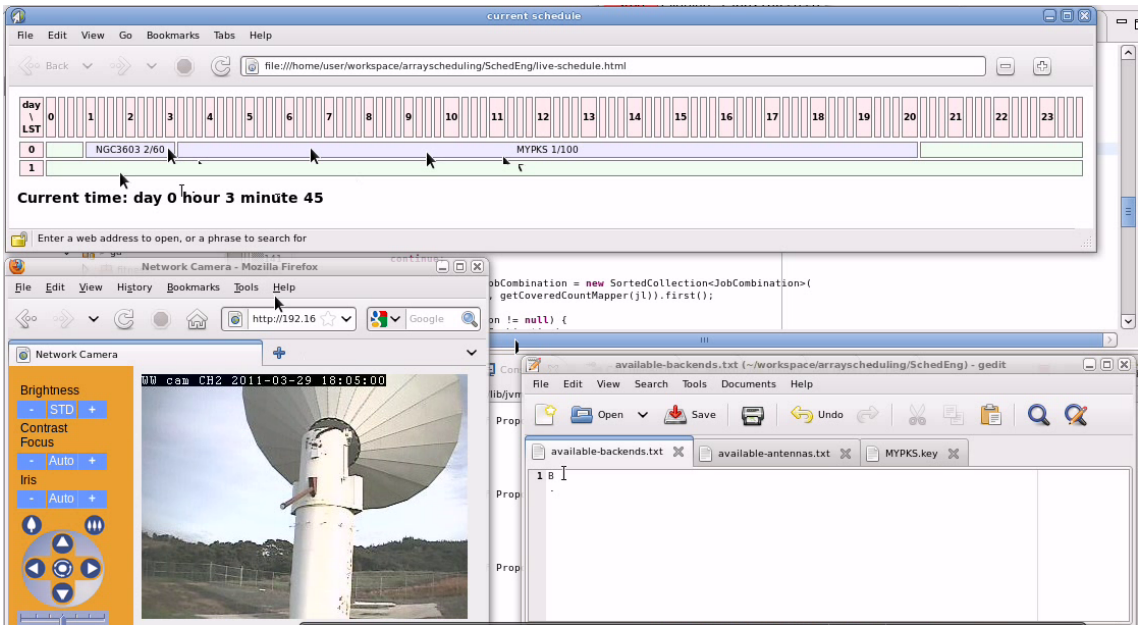


Figure 37: Screen shot of the demo environment after re-scheduling. Between the previous screen shot and this, we let one backend fail (as shown on the bottom right). Rescheduling immediately and automatically occurred (see top). Now, the lower-priority observation is scheduled because it can still be executed with only backend A. Bottom left: Telescope pointing and observing PKS 0539-691. Bottom right: the monitoring system currently shows only backend A as available.

Current (astronomy) scheduling systems show the following workflow with humans depicted in Figure 38 on the next page.

On the one hand, a human scheduler has to be in ultimate control of the schedule, on the other hand, as Kleiner (1999) discuss, it is very difficult to reschedule using a computer-based system once a human made direct changes. Following the recommendation of Kleiner (1999), it is best to not let the human scheduler edit the schedule directly, but provide a method of adding preferences and constraints to advise the scheduling system. In this approach it is required from the user to make explicit why she or he would like a certain change to occur. If this is in the form "this task is preferred here" it can be a preference, modifying the priority in that time period. It may turn out that these decisions, when articulate, require a change in the specifications, e.g. "the task can not be executed in this time period". Extracting these decisions as such allow re-scheduling and provide documentation.

A demonstration of such a UI is shown in Figure 39. This user interaction suggestion accesses the scheduling framework through JRuby on Rails. At any time, the user is presented with the schedules (center left), to be ordered in preference and displayed in the big area on the right. Through marking time ranges, preferences can be added, and viewed on the top left. Once done, the next iteration can be launched by the first button on the bottom left, which lets scheduling algorithm suggest additional schedules. The specification, ranking and selection is left to the user, the crossover/mutation/generation to the scheduler.

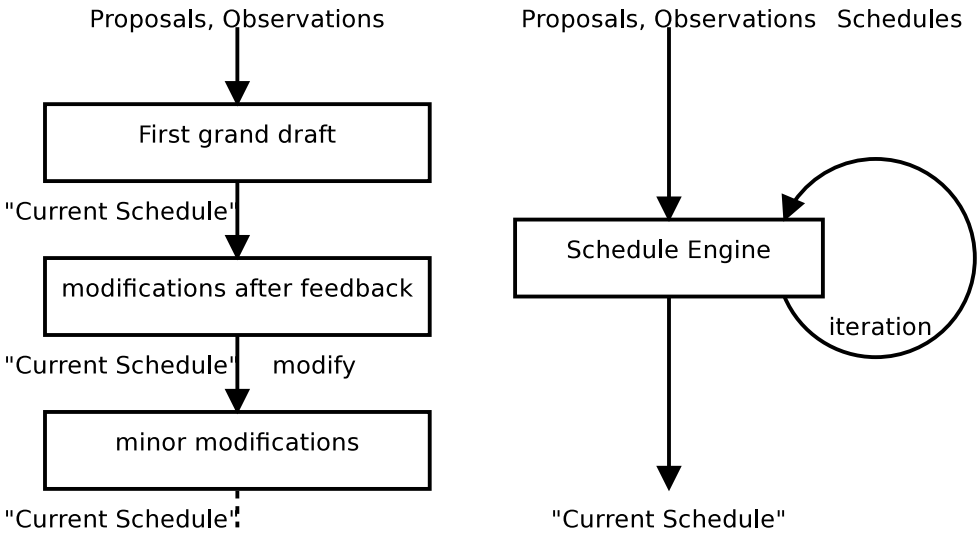


Figure 38: User interaction patterns. Left: Current usage. First draft, time-consuming scheduling sessions, followed by only minor modifications. A change in specifications is hard to accommodate.
Right: Proposed usage pattern. The current information and demands are always incorporated. The drawback could be that the schedule can be less predictable – if a predictable schedule is needed, the scheduler has to prefer schedule changes that affect the least number of people.

In this method, re-scheduling is easy, yet the user is left in control.

10.4 Applicability to ATA, ALMA and SKA

Once a user interface has matured, the framework is directly applicable to ATA. From the outline of the scheduling problem in Mora & Solar (2010), ALMA is in the very same scenario of wanting parallel observations at a time, in combination with a dynamic scheduler.

For SKA, the initial challenge is working on an incomplete system during growing construction. The scheduler allows such changing requirements specifically. As demanded by Johnston et al. (1996), the system is adaptive to changing demands and can be coupled to other system.

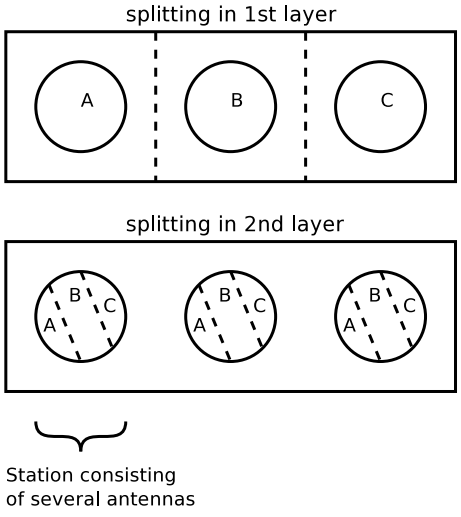


Figure 40: Sub-array splitting

With sub-arrays, there can be various ways of splitting. For instance, if we think of the stations as the first hierarchical layer, and of the antennas as the second (each station consisting of a number of antennas), a sub-array can be formed by assigning

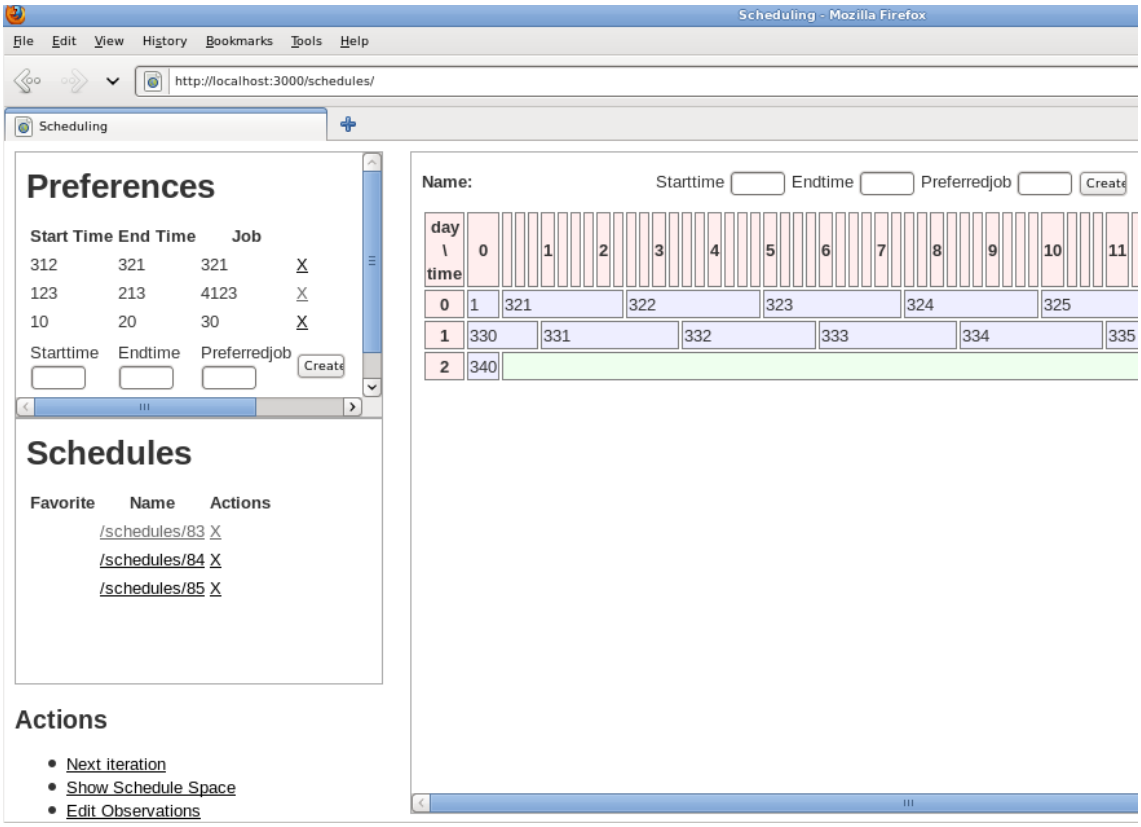


Figure 39: User interaction suggestion

full stations or by proportionally subdividing stations (see Figure 40). For using the full extent of the SKA, the second kind will be relevant, but there may be instances where the first kind is required. For instance, when observatories have their own observations to attend to, or due to maintenance of a station.

In either case, such splitting is possible with the approach taken here, as follows. To show 1st-layer splitting, let proposal A request any one of the "stations" resource type, and all of the "antennas" (e.g. 20 of the 20 per station). As many of such proposals can be scheduled in parallel as there are stations. To show 2st-layer splitting, let proposal B request all of the "stations" resource type, and some of the "antennas" (e.g. 4 of the 20 per station). Then 5 of such proposals can be scheduled in parallel through the approach taken in this work.

Part V

Conclusions

We provided a definition and formulation of the scheduling problem faced by modern radio astronomy observatories. In particular, we present

1. a complete literature review of the past approaches to scheduling in astronomy,
2. characteristics of the scheduling problem based on records and operating staff experience,
3. manual strategies used by human schedulers in currently operating observatories and
4. today's requirements for the complex machines radio telescope arrays like the ATA and SKA are. These include dynamic and scalable (re-)scheduling, management of many resources, parallel observations, dynamic parallel observations. With these in mind, we compared ideas from all major approaches based on existing scheduling algorithms.

The GA is well-known to perform poorly in order-based encodings. This is due to the fact that mutation operations are difficult to write so that the GA can efficiently sample the problem space. Hence the recommended time-indexed encoding has been used here. This formulation has the well-known issue of "fragmentation", a undesired high amount of interrupts. This problem was successfully solved here using a penalty in the fitness function, and creating mutation operators that work on multiple, neighboring genes at once.

Due to the complexity of the problem, exact algorithms such as branch-and-bound are impractical. The simplest and fastest of these approaches, Linear (Integer) Programming, was found to take unreasonable amounts of time (days) and to provide poor results (fragmentation).

We found that simple algorithms, in particular just going through the schedule and allocating the highest-priority observation possible at each time slot, lead to very good first-cut solution in the test dataset looked at. The GA can select the best heuristic and thus keep the approach robust to changes in the characteristics of the problem. It may help to make minor improvements, but also allow to update the schedule on changed requirements.

With this work we provided a first solution to scheduling observations where parallel observations are considered. We successfully demonstrated dynamic rescheduling on changed conditions, by integration of the scheduling framework with the control system used in the AUT Warkworth 12m telescope.

The scheduling framework presented here is relevant for currently operated telescopes like the ATA, currently constructed telescopes like ALMA and the future

large-scale project SKA. In particular, during the development phase of SKA, the dynamic, scalable scheduling framework can accommodate changing conditions.

Part VI

Appendix

References

Aarup, M., Zweben, M., & Fox, M. (1994). *Intelligent scheduling*. Morgan Kaufmann Publishers.

Ballestín, F., Valls, V., & Quintanilla, S. (2008). Pre-emption in resource-constrained project scheduling. *European Journal of Operational Research*, 189(3), 1136–1152.

Barbulescu, L., Howe, A., Whitley, L., & Roberts, M. (2006). Understanding algorithm performance on an oversubscribed scheduling application. *Journal of Artificial Intelligence Research*, 27(1), 577–615.

Berkelaar, M., Eikland, K., Notebaert, P., et al. (2010). lpsolve 5.5: Open source (mixed-integer) linear programming system.

Boër, M., Bringer, M., Berassain, J., Fontan, G., & Merce, C. (2000). The New MAJORDOME: Efficient Scheduling of Autonomous Telescopes. In N. Manset, C. Veillet, & D. Crabtree (Ed.), *Astronomical Data Analysis Software and Systems IX*, volume 216 of *Astronomical Society of the Pacific Conference Series* (pp. 115–118).

Bresina, J., Drummond, M., Swanson, K., & Edgington, W. (1993). Automated Management and Scheduling of Remote Automatic Telescopes.

Chavan, A. M., Giannone, G., Silva, D., Krueger, T., & Miller, G. (1998). Nightly Scheduling of ESO's Very Large Telescope. In R. Albrecht, R. N. Hook, & H. A. Bushouse (Ed.), *Astronomical Data Analysis Software and Systems VII*, volume 145 of *Astronomical Society of the Pacific Conference Series* (pp. 255–258).

Clark, M., Balser, D., Sessoms, E., Bignell, C., Condon, J., McCarty, M., Marganian, P., O'Neil, K., Shelton, A., & Maddalena, R. (2009). The GBT Dynamic Scheduling System: "When do I observe?" Guiding Users' Expectations. In *Astronomical Society of the Pacific Conference Series*, volume 411 (pp. 338–+).

de Castro, A. & Yáñez, J. (2003). Optimization of telescope scheduling. *Astronomy and Astrophysics*, 403(1), 357–367.

Algorithm 10 . MutationSimilarPrev operator. Randomly mutates a slot and tries to extend the previous Job if possible.

```
1  public class ScheduleSimilarPrevMutation extends AbstractScheduleMutation {
2      protected Schedule mutateSchedule(Schedule s1, Random rng) {
3          Schedule s2 = new Schedule();
4          int i = 0;
5          int n = 0;
6          JobCombination lastJc = null;
7          for (Entry<LSTTime, JobCombination> e : s1) {
8              LSTTime t = e.getKey();
9              JobCombination jc = s1.get(t);
10
11             Set<JobCombination> jcs = possibles.get(t);
12             if (e.getValue() != null && !jcs.isEmpty()) {
13                 s2.add(t, jc);
14                 if (lastJc != null && !lastJc.equals(jc)) {
15                     if (mutationProbability.nextValue().nextEvent(rng)) {
16                         jc = getMostSimilar(lastJc, jcs);
17                         if (jc != null) {
18                             s2.add(t, jc);
19                             i++;
20                         }
21                     }
22                     n++;
23                 }
24             }
25             lastJc = jc;
26         }
27         if (history != null) {
28             history.derive(s2, s1, i * 1. / n);
29             // rest is random
30         }
31         updateCounters(s2, s1, i);
32         return s2;
33     }
34 }
```

Algorithm 11 MutationExchange operator. Randomly selects slot and tries to exchange with the same slot from a previous day

```
1  public class ScheduleExchangeMutation implements EvolutionaryOperator<Schedule> {
2      private Schedule mutateSchedule(Schedule s1, Random rng) {
3          Schedule s2 = new Schedule();
4          int i = 0;
5          int n = 0;
6          Probability prob = mutationProbability.nextValue();
7          Probability u = new Probability(1. / 3);
8          for (Entry<LSTTime, JobCombination> e : s1) {
9              LSTTime t = e.getKey();
10             s2.add(t, e.getValue());
11             Set<JobCombination> jcs = possibles.get(t);
12             if ((i * 1. / n) < prob.doubleValue() && u.nextEvent(rng)) {
13                 if (!jcs.isEmpty()) {
14                     JobCombination jc = e.getValue();
15                     List<LSTTime> timeCandidates = new ArrayList<LSTTime>();
16                     if (t.day > 3)
17                         timeCandidates.add(new LSTTime(t.day - 3, t.minute));
18                     if (t.day > 2)
19                         timeCandidates.add(new LSTTime(t.day - 2, t.minute));
20                     if (t.day > 1)
21                         timeCandidates.add(new LSTTime(t.day - 1, t.minute));
22
23                     Collections.shuffle(timeCandidates);
24                     // find a exchange partner
25                     boolean foundPartner = false;
26                     for (LSTTime t2 : timeCandidates) {
27                         JobCombination jc2 = s1.get(t2);
28                         Set<JobCombination> jcs2 = possibles.get(t2);
29                         // partners should be exchangeable, but not the same,
30                         // and not both null
31                         if ((jc == null || jcs2.contains(jc))
32                                 && (jc2 == null || jcs.contains(jc2))
33                                 && !(jc2 == null && jc == null)
34                                 && (jc == null || !jc.equals(jc2))) {
35
36                             // switch
37                             if (jc != null)
38                                 s2.add(t2, jc);
39                             if (jc2 != null)
40                                 s2.add(t, jc2);
41                             foundPartner = true;
42                             break;
43                         }
44                     }
45                     if (foundPartner) {
46                         i++; /* we let this count */
47                     } else
48                         s2.add(t, e.getValue());
49                 }
50             }
51             n++;
52         }
53         if (history != null)
54             history.derive(s2, s1, i * 1. / n);
55         if (counter != null) {
56             counter.derive(s2, s1);
57             counter.add(s2, this.toString(), i);
58         }
59         return s2;
60     }
61 }
```

Algorithm 12 The MutationSimilarForward and MutationSimilarBackward operators are instances of this class. The operator randomly selects blocks and tries to extend them backwards in time as far as possible (maximum 1 day).

```
1  public class ScheduleSimilarMutation extends AbstractScheduleMutation {
2      private boolean forwardsKeep = false;
3      private boolean backwardsKeep = true;
4
5      protected Schedule mutateSchedule(Schedule s1, Random rng) {
6          Schedule s2 = new Schedule();
7          int i = 0;
8          int n = 0;
9
10         JobCombination lastJc = null;
11
12         int toSkip = 0;
13         for (Iterator<Entry<LSTTime, JobCombination>> it = s1.iterator(); it
14                 .hasNext();) {
15             Entry<LSTTime, JobCombination> e = it.next();
16             LSTTime t = e.getKey();
17             JobCombination jc = e.getValue();
18             Set<JobCombination> jcs = possibles.get(t);
19             if (jc != null && !jcs.isEmpty()) {
20                 s2.add(t, jc);
21                 if (toSkip > 0) {
22                     toSkip--;
23                 } else {
24                     /* only if we have a change, we should consider it */
25                     if ((lastJc == null || !lastJc.equals(jc))
26                             && (mutationProbability.nextValue().nextEvent(rng)))
27                         log.debug("mutating around " + t);
28                         toSkip = makeSimilarAround(t, jc, possibles, s2);
29                         i += toSkip;
30                     }
31                 }
32                 n++;
33             }
34             lastJc = jc;
35         }
36         log.debug("changed " + i + " of " + n);
37         if (history != null) {
38             history.derive(s2, s1, i * 1. / n);
39             // rest is random
40         }
41         updateCounters(s2, s1, i);
42
43         return s2;
44     }
45
46     protected int makeSimilarAround(LSTTime t, JobCombination thisjc,
47             ScheduleSpace template, Schedule s2);
48 }
```

Algorithm 13 makeSimilarAround function used by MutationSimilarForward, MutationSimilarForward, MutationKeeping and JobPlacementMutation operators.

```
1  public class ScheduleSimilarMutation extends AbstractScheduleMutation {
2      // ... continued ...
3      protected int makeSimilarAround(LSTTime t, JobCombination thisjc,
4              ScheduleSpace template, Schedule s2) {
5          boolean posContinue = forwardsKeep;
6          boolean negContinue = backwardsKeep;
7          int countChanged = 0;
8          LSTTime last = template.findLastEntry();
9          // iterate for 1 day maximum
10         LSTTimeIterator it = new LSTTimeIterator(new LSTTime(0, 1),
11                 new LSTTime(1, 0), Schedule.LST_SLOTS_MINUTES);
12         for (; it.hasNext();) {
13             LSTTime tDelta = it.next();
14
15             if (posContinue) {
16                 LSTTime tPlus = new LSTTime(t.day + tDelta.day, t.minute
17                         + tDelta.minute);
18                 if (tPlus.minute > Schedule.LST_SLOTS_PER_DAY
19                         * Schedule.LST_SLOTS_MINUTES) {
20                     long extraDays = tPlus.minute
21                             / (Schedule.LST_SLOTS_PER_DAY * Schedule.LST_SLOTS_MI
22                     tPlus.day += extraDays;
23                     tPlus.minute -= extraDays
24                             * (Schedule.LST_SLOTS_PER_DAY * Schedule.LST_SLOTS_MI
25                 }
26                 if (tPlus.isAfter(last)) {
27                     posContinue = false;
28                 } else {
29                     Set<JobCombination> jcs = template.get(tPlus);
30                     if (!jcs.isEmpty()) {
31                         JobCombination jc = getMostSimilar(thisjc, jcs);
32                         if (jc != null) {
33                             log.debug("jc " + jc);
34                             s2.add(tPlus, jc);
35                             countChanged++;
36                         } else {
37                             posContinue = false;
38                         }
39                     } else {
40                         posContinue = false;
41                     }
42                 }
43             }
44             if (negContinue) {
45                 // similar to above section, but in negative time direction ...
46             }
47         }
48         return countChanged;
49     }
50 }
```

Algorithm 14 The MutationKeeping operator. Randomly mutates a slot and tries to extend the selection as long as possible (max 1 day).

```
1  public class ScheduleKeepingMutation extends ScheduleSimilarMutation {
2      @Override
3      protected Schedule mutateSchedule(Schedule s1, Random rng) {
4          Schedule s2 = new Schedule();
5          int i = 0;
6          int n = 0;
7          int toSkip = 0;
8          Probability u = new Probability(2. / 5.);
9
10         for (Entry<LSTTime, JobCombination> e : s1) {
11             LSTTime t = e.getKey();
12             JobCombination jc = s1.get(t);
13
14             Set<JobCombination> jcs = possibles.get(t);
15             if (e.getValue() != null && !jcs.isEmpty()) {
16                 s2.add(t, jc);
17                 if (toSkip > 0) {
18                     toSkip--;
19                 } else {
20                     if (u.nextEvent(rng)
21                             && mutationProbability.nextValue().nextEvent(rng)) {
22                         // randomly choose a task
23                         jc = (JobCombination) jcs.toArray()[rng.nextInt(jcs
24                                 .size())];
25
26                         s2.add(t, jc);
27                         toSkip = makeSimilarAround(t, jc, possibles, s2);
28                         i += 1 + toSkip;
29                         log.debug("mutated and made " + toSkip + " similar");
30                     }
31                 }
32                 n++;
33             }
34         }
35         log.debug("changed " + i + " of " + n);
36         if (history != null) {
37             history.derive(s2, s1, i * 1. / n);
38             // rest is random
39         }
40         updateCounters(s2, s1, i);
41
42         return s2;
43     }
44 }
```

Algorithm 15 The MutationJobPlacement operator. Randomly selects a job, and finds a suitable slot to schedule it. Neighboring slots are changed too.

```
1  public class ScheduleJobPlacementMutation extends ScheduleSimilarMutation {
2      protected Map<JobCombination, List<LSTTime>> possibleSlots =
3          new HashMap<JobCombination, List<LSTTime>>();
4      protected List<JobCombination> jobs;
5
6      public ScheduleJobPlacementMutation(ScheduleSpace possibles,
7              Probability probability) {
8          fillPossibleSlots();
9          setForwardsKeep(true);
10         setBackwardsKeep(true);
11         jobs = new ArrayList<JobCombination>(possibleSlots.keySet());
12     }
13
14     @Override
15     protected Schedule mutateSchedule(Schedule s1, Random rng) {
16         Schedule s2 = new Schedule();
17         int i = 0;
18         int n = 0;
19         // copying schedule
20         for (Entry<LSTTime, JobCombination> e : s1) {
21             LSTTime t = e.getKey();
22             JobCombination jc = s1.get(t);
23             Set<JobCombination> jcs = possibles.get(t);
24             if (e.getValue() != null && !jcs.isEmpty())
25                 s2.add(t, jc);
26             n++;
27         }
28         // place jobs
29         for (int j = 0; j < jobs.size(); j++) {
30             if (this.mutationProbability.nextValue().nextEvent(rng)) {
31                 JobCombination jc = jobs.get(j);
32                 List<LSTTime> slots = possibleSlots.get(jc);
33                 LSTTime t = slots.get(rng.nextInt(slots.size()));
34                 s2.add(t, jc);
35                 i += 1 + makeSimilarAround(t, jc, possibles, s2);
36             }
37         }
38         updateHistory(s2, s1, i, n);
39         updateCounters(s2, s1, i);
40         return s2;
41     }
42 }
```

Algorithm 16 The genetic history used to track the linage and influence of the initial population to the final population. The crossover operator calls the derive method for every child, which passes on the properties of the parents onto the child.

```
1  /**
2   * @param <K> chromosome type
3   * @param <V> property type
4   */
5  public class GeneticHistory<K, V> {
6      private static Logger log = Logger.getLogger(GeneticHistory.class);
7
8      private Map<K, Map<V, Double>> properties = new HashMap<K, Map<V, Double>>();
9
10     public void initiated(K key, V property) {
11         if (log.isDebugEnabled())
12             log.debug("adding initial member with property '" + property + "'");
13         ensureKnown(key).put(property, 1.);
14     }
15
16     private Map<V, Double> ensureKnown(K key) {
17         if (!properties.containsKey(key)) {
18             Map<V, Double> m = new HashMap<V, Double>();
19             properties.put(key, m);
20             return m;
21         } else {
22             return properties.get(key);
23         }
24     }
25
26     public void derive(K newKey, K parent, Double parts) {
27         if (properties.containsKey(parent) && !properties.get(parent).isEmpty()) {
28             Map<V, Double> p = ensureKnown(newKey);
29             for (Entry<V, Double> e : properties.get(parent).entrySet()) {
30                 if (log.isDebugEnabled())
31                     log.debug("handing over " + e.getKey() + " " + e.getValue()
32                             + " --> " + e.getValue() * parts);
33                 if (p.containsKey(e.getKey())) {
34                     p.put(e.getKey(), p.get(e.getKey()) + e.getValue() * parts);
35                 } else {
36                     p.put(e.getKey(), e.getValue() * parts);
37                 }
38             }
39         }
40     }
41
42     public Map<V, Double> getProperties(K key) {
43         return properties.get(key);
44     }
45 }
```

Drummond, M., Bresina, J., & Swanson, K. (1995). Just-in-case scheduling. In *Proceedings of the National Conference on Artificial Intelligence* (pp. 1098–1104).: American Association for Artificial Intelligence.

Drummond, M., Swanson, K., & Bresina, J. (1994). Robust scheduling and execution for automatic telescopes. *Intelligent scheduling*, (pp. 341–369).

Dyer, D. W. (2010). The watchmaker framework for evolutionary computation (evolutionary/genetic algorithms for java), v0.7.1.

Edgington, W., Drummond, M., Bresina, J., Henry, G., & Drascher, F. (1996). Improved Scheduling of Robotic Telescopes. In T. Boroson, J. Davies, & I. Robson (Ed.), *New Observing Modes for the Next Century*, volume 87 of *Astronomical Society of the Pacific Conference Series* (pp. 151–157).

Ernst, A., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1), 3–27.

Fomalont, E. & Reid, M. (2004). Microarcsecond astrometry using the SKA. *New Astronomy Reviews*, 48(11-12), 1473–1482.

Frank, J., Crawford, J., Khatib, L., & Brafman, R. (2005). Tractable optimal competitive scheduling. In *Fifteenth International Conference on Automated Planning and Scheduling* (pp. 73–82).

Frank, J. & Kürklü, E. (2005). Mixed discrete and continuous algorithms for scheduling airborne astronomy observations. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, (pp. 183–200).

Gaffney, N. I. & Cornell, M. E. (1997). Planning and Scheduling Software for the Hobby Eberly Telescope. In G. Hunt & H. Payne (Ed.), *Astronomical Data Analysis Software and Systems VI*, volume 125 of *Astronomical Society of the Pacific Conference Series* (pp. 379–382).

Gharote, M., Deshpande, A., Lodha, S., Kantharia, N., Wadadekar, Y., Katore, S., & Pramesh, A. (2009). Automated Telescope Scheduling. In *Astronomical Society of the Pacific Conference Series*, volume 407 (pp. 438–441).

Giffler, B. & Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, 8(4), pp. 487–503.

Giuliano, M. & Johnston, M. (2008). Multi-Objective Evolutionary Algorithms for Scheduling the James Webb Space Telescope. In *International Conference on Automated Planning and Scheduling (ICAPS)* (pp. 107–115).

Gratch, J. & Chien, S. (1996). Adaptive Problem-Solving for Large-Scale Scheduling Problems: A Case Study. *Journal of Artificial Intelligence Research*, 4, 365–396.

Greiner, R. (1996). PALO: a probabilistic hill-climbing algorithm* 1. *Artificial Intelligence*, 84(1-2), 177–208.

Grim, R., Jansen, M., Baan, A., van Hemert, J., & de Wolf, H. (2002). Use of evolutionary algorithms for telescope scheduling. In *Proceedings of the Workshop on Integrated Modeling of Telescopes* (pp. 51–61).

Harrison, S., Price, M., & Philpott, M. (1999). Task scheduling for satellite based imagery. In *Proceedings of the Eighteenth Workshop of the UK Planning and Scheduling Special Interest Group*, volume 78 (pp. 64–78).

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval research logistics*, 49(5), 433–448.

Herroelen, W., De Reyck, B., & Demeulemeester, E. (1998). Resource-constrained project scheduling: a survey of recent developments. *Computers & Operations Research*, 25(4), 279–302.

Herroelen, W., Demeulemeester, E., & De Reyck, B. (1997). A classification scheme for project scheduling problems. *DTEW Research Report 09727*, (pp. 1–25).

Johnston, M. (2002). Spike: AI scheduling for NASA's Hubble Space Telescope. In *Artificial Intelligence Applications, 1990., Sixth Conference on* (pp. 184–190).: IEEE.

Johnston, M. et al. (1996). Scheduling Tools for Astronomical Observations. In *Astronomical Society of the Pacific Conference Series*, volume 87 (pp. 62–71).

Jones, D. (2004). SKA science requirements: Version 2. *Square kilometre Array US-SKA Memo*, 45.

Jones, E., Oliphant, T., Peterson, P., et al. (2001–). SciPy: Open source scientific tools for Python.

Kitchin, C. R. (1998). *Astrophysical Techniques*, volume 85. Springer Netherlands, 3 edition. 10.1023/A:1005017829159.

Kleiner, S. (1995). A Graphical Planning and Scheduling Toolkit for Astronomical Spacecraft. In *Astronomical Data Analysis Software and Systems IV*, volume 77 (pp. 136–139).

Kleiner, S. (1999). Small, Fast and Reusable: A Satellite Planning and Scheduling System. In *Astronomical Data Analysis Software and Systems VIII*, volume 172 (pp. 77–80).

Kraus, J. D. (1966). *Radio astronomy.* Cygnus-Quasar Books, 2nd edition.

Leibundgut, B. (1997). Operational concept of large telescopes. In A. L. Ardeberg (Ed.), *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 2871 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* (pp. 755–761).

Meier, D. L. (1998). Operating a Telescope Larger than the Earth: How to Schedule the HALCA Space VLBI Mission. In J. A. Zensus, G. B. Taylor, & J. M. Wrobel (Ed.), *IAU Colloq. 164: Radio Emission from Galactic and Extragalactic Compact Sources*, volume 144 of *Astronomical Society of the Pacific Conference Series* (pp. 421–422).

Miller, G., Lindenmayer, K., Johnston, M., Vick, S., & Sponsler, J. (1988). Knowledge based tools for hubble space telescope planning and scheduling: Constraints and strategies*. *Telematics and Informatics*, 5(3), 197–212.

Minton, S., Johnston, M., Philips, A., & Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3), 161–205.

Mora, M. & Solar, M. (2010). A survey on the dynamic scheduling problem in astronomical observations. *Artificial Intelligence in Theory and Practice III*, (pp. 111–120).

Morris, R., Bresina, J., & Rodgers, S. (1997). Automatic generation of heuristics for scheduling. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, volume 15 (pp. 1260–1266).

Nowakovski, J., Schwärzler, W., & Triesch, E. (1999). Using the generalized assignment problem in scheduling the ROSAT space telescope. *European Journal of Operational Research*, 112(3), 531–541.

Pinedo, M. (2008). *Scheduling: theory, algorithms, and systems.* Springer Verlag.

Powell, M. (1979). Variable metric methods for constrained optimization. *Computing Methods in Applied Sciences and Engineering, 1977, I*, (pp. 62–72).

Rogers, B. & Graham, M. (1982). Similarities between motion parallax and stereopsis in human depth perception. *Vision Research*, 22(2), 261–270.

Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review*, 13(2), 87–127.

Schiex, T., Fargier, H., & Verfaillie, G. (1995). Valued constraint satisfaction problems: Hard and easy problems. In *International Joint Conference on Artificial Intelligence*, volume 14 (pp. 631–639).

Smith, D., Frank, J., & Jónsson, A. (2000). Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, 15(1), 47–83.

Smith, S. (2005). Is Scheduling a Solved Problem? In G. Kendall, E. K. Burke, S. Petrovic, & M. Gendreau (Eds.), *Multidisciplinary Scheduling: Theory and Applications* (pp. 3–17). Springer US.

Sponsler, J. (1989). Genetic algorithms applied to the scheduling of the hubble space telescope*. *Telematics and Informatics*, 6(3-4), 181–190.

Spragg, J. & Smith, B. (1993). Nightwatch: a telescope observation scheduler using constraint satisfaction with delayed evaluation. In *Resource Scheduling for Large Scale Planning Systems, IEE Colloquium on* (pp. 1–4).

Stork, F. & Uetz, M. (2000). Resource-constrained project scheduling: From a Lagrangian relaxation to competitive solutions. In *Proc. of the 7th International Workshop on Project Management and Scheduling (Osnabrück, Germany)* (pp. 254–257).

Tanomaru, J. (1995). Staff scheduling by a genetic algorithm with heuristic operators. In *IEEE International Conference on Evolutionary Computation*, volume 1 (pp. 456–+).

Ullman, J. (1975). NP-complete scheduling problems*. *Journal of Computer and System Sciences*, 10(3), 384–393.

Wall, M. (1996). *A genetic algorithm for resource-constrained scheduling.* PhD thesis, MIT Mechanical Engineering Department.

Zhao, W., Ramamritham, K., & Stankovic, J. (1987). Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, 100(8), 949–960.