TIME-WINDOW OPTIMIZATION FOR A CONSTELLATION OF EARTH OBSERVATION SATELLITES

by

CHRISTIAAN VERMAAK OBERHOLZER

submitted in part fulfilment of the requirements for the degree of

MASTER OF COMMERCE

in the subject

QUANTITATIVE MANAGEMENT

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF JS WOLVAARDT

FEBRUARY 2009

ABSTRACT

Satellite Scheduling Problems (SSP) are NP-hard and constraint programming and metaheuristics solution methods yield mixed results. This study investigates a new version of the SSP, the Satellite Constellation Time-Window Optimization Problem (SCoTWOP), involving commercial satellite constellations that provide frequent earth coverage.

The SCoTWOP is related to the dual of the Vehicle Routing Problem with Multiple Time-windows, suggesting binary solution vectors representing an activation of time-windows. This representation fitted well with the MatLab® Genetic Algorithm and Direct Search Toolbox subsequently used to experiment with genetic algorithms, tabu search, and simulated annealing as SCoTWOP solution methods. The genetic algorithm was most successful and in some instances activated all 250 imaging time-windows, a number that is typical for a constellation of six satellites.

Key terms:

Satellite scheduling; Genetic algorithms; Earth observation; Metaheuristics; Constellation of satellites; Time-window optimization; Tabu search; Simulated annealing; Vehicle Routing Problem; Multiple time-windows.

ACKNOWLEDGEMENTS

I would like to thank the following people without whom this dissertation would not have been possible:

My wife Christa for continued support and encouragement over the years.

My supervisor, Prof J. S. Wolvaardt, for his guidance, encouragement, and patience.

Prof Michael G. Kay of the North Carolina State University, for permission to use his MatLog Toolbox in my research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION AND MOTIVATION

1.1 General

A number of different problems involving the optimization of the use of either satellite resources only, or a combination of ground station and space satellite resources, have collectively become known as Satellite Scheduling Problems (SSP). The SSP has lately received more attention due to the increasing commercial exploitation of earth observation (EO) satellite imagery. Previously the purview of security and scientific communities, the commercial demand for imagery is fuelled by the spread of newly declassified technology that makes high-resolution images available for general use. The demand for this imagery has grown to the point where it easily surpasses supply, creating a need to optimise the use of the scarce space and ground resources used to produce earth observation imagery.

A survey of literature has shown that the term “Satellite Scheduling Problem” denotes a class of related problems rather than a single problem. The class of problems are also not necessarily “scheduling” problems in the sense that the word is normally used in operations research, although they are all at least related to this kind of problem. Different kinds of SSP arise depending on the object of optimization. Most frequently the objective is to optimize imaging operations of the satellites, i.e. the space resources or so called “space segment”. A related problem has the objective of optimizing the activities of receiving and control stations, i.e. the “ground segment” that receive data from, and send

commands to the satellites. Both these kinds of SSP have to deal with the hard constraints imposed by orbit geometry and the rotation of the earth and are therefore very similar, to the extent that they are sometimes combined in a single problem.

The focus of this dissertation is on the first kind of SSP, i.e. optimizing satellite imaging operations, and more specifically, on a new variant of this problem that arises when the imaging operations of a number of identical satellites that are orbiting as a constellation in the same orbit plane have to be optimized. In the past, problems that involve optimizing the imaging activities of a number of satellites have been investigated but these satellites were most often of different types and in different orbits. The commercial demand for, and myriad uses of, satellite imagery have lead entrepreneurs to the idea of constructing purpose-built commercial constellations of identical satellites that will cover areas of interest at suitably frequent intervals. In an effort to reflect more clearly the true nature of problem, we term the SSP applicable to constellations, the Satellite Constellation Time-Window Optimization Problem and will accordingly be using the acronym SCoTWOP. We retain however the less precise but more commonly used SSP as acronym when referring to the problem of optimizing satellite imaging operations in general.

The SSP has been proven to be NP-complete (Barbulescu *et al*. (2004)), and since its inception in the military and scientific spheres, optimization attempts have primarily focussed on heuristic techniques of various kinds. On another, more practical tack, the problem has been viewed as over-constrained i.e. having no or very few feasible solutions. The focus has therefore been on constraint satisfaction techniques to try and find at least one feasible solution or if not, to maximise the number of constraints satisfied. Both the optimization and constraint satisfaction strategies have met with varying degrees of success in practical systems.

Genetic algorithms have been employed with mixed success for finding good solutions to the SSP. As with many genetic algorithm applications, the challenge is to find a suitable representation of a solution to the problem in the form of

some genotype. A rather obvious and intuitive way of representation is to allocate a number to each image that has to be taken, and to represent a feasible sequence of imaging events as a genotype comprising an ordered subset of these image numbers. However, the benefit of a constellation of satellites in the same orbit plane is their collective higher coverage rate and frequent revisit period. This means that there is often more than one opportunity to image the same spot on the earth within a given time frame, whether by different satellites or the same one. This means that an image can no longer be represented by a single number making the representation of solutions as a relatively simple genotype no longer feasible.

1.2 Concepts in earth observation satellite operations

A brief overview of EO satellite operations is provided below as background to the SSP. More information can be found in Wertz *et al.* (1999) and an informal description of the global SSP problem is provided by Verfaillie *et al.* (2002).

A typical EO satellite orbits the earth in Low Earth Orbit (LEO) at a speed of about seven km/s at an altitude of 400 to 1200 km. The duration of each orbit is about 90 to 100 minutes, resulting in approximately 14 to 15 orbits in a 24-hour period. Most of these satellites are in a slightly elliptical orbit, with an orbit plane inclined at about 8 degrees from the pole axis. This near-polar orbit is designed to be sun-synchronous so that the satellite maintains a constant orientation with respect to the sun. While the satellite is orbiting the earth, the earth itself is of course rotating, so that the area of the earth beneath the satellite is constantly changing. Figure 1-1 and Figure 1-2 illustrate the orbit geometry of a typical sun-synchronous orbit in three and two dimensions respectively.

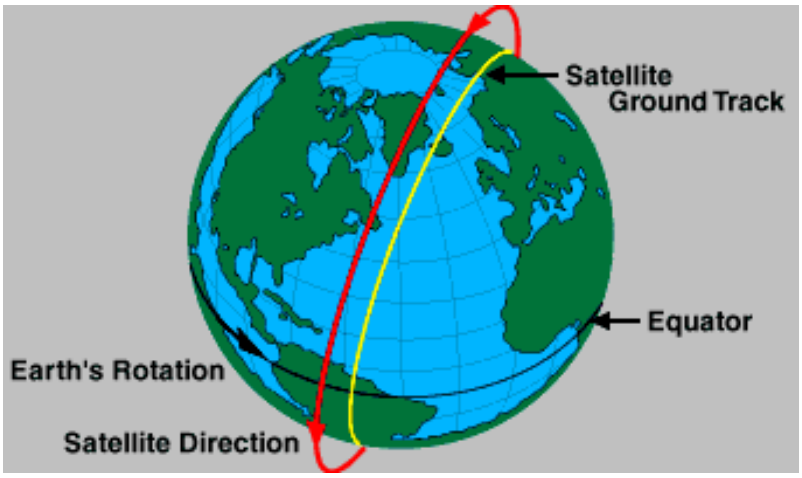

Figure 1-1: A Three Dimensional View of a Satellite in Sun-synchronous Orbit around the earth showing the Near Polar Orientation of its Orbit Plane (Copyright © CRISP, 2001)

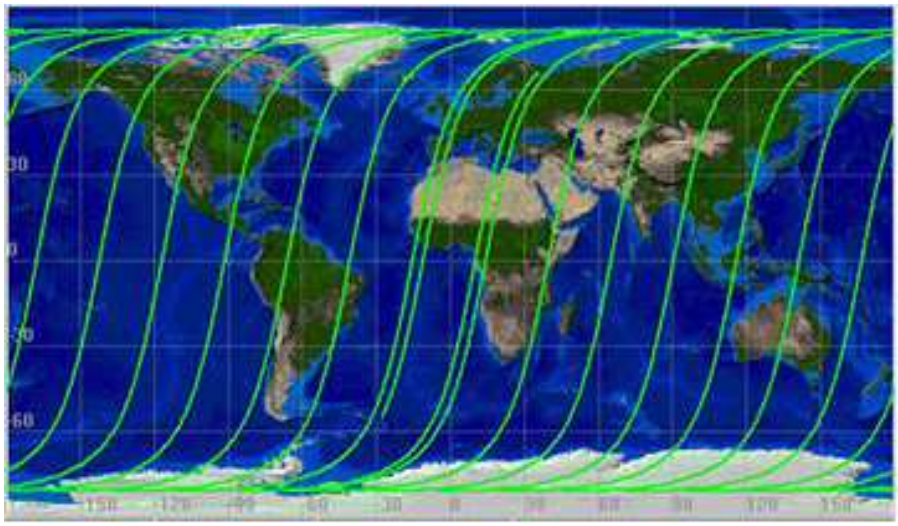

Figure 1-2: A Two Dimensional View of the earth showing the Ground Trace of a Satellite in Sun-synchronous Orbit (Copyright © Keldysh Institute of Applied Mathematics, Russian Academy of Sciences, 2004)

Many optical satellites typically use a movable mirror to scan the area of the earth up to some distance away from its nadir to provide adequate field of view. The scanning motion can be parallel to the flight direction (push-broom

scanning) or orthogonal to the line of flight (whisk-broom scanning). Newer satellites are much more agile and can point its instrument by changing its attitude through a rolling or pitching motion, a manoeuvre known as slewing. Modern synthetic aperture radar (SAR) satellites employ electronic focussing to steer the pulses from their phased array antennas in the range (cross-track) direction.

Depending on the field-of-view of the satellite's sensor, this means that every point on the earth's surface can be accessed within a finite period. This revisit period can vary from around 20 to 30 days on the equator to only hours near the poles. This means that, in order to be able to acquire daily images of the same geographical area, a constellation of satellites is required.

The satellite receives detailed instructions, on which activities to perform during each orbit, in the form of a schedule transmitted to it via radio signal from its controlling ground station. This can only happen when the satellite is in view of the ground station and is therefore not feasible on every orbit. Rather, instructions for a number of upcoming orbits are up-linked at the same time. Similarly, the satellite cannot always downlink imagery data obtained by its sensor in real time. The imagery data obtained during a number of orbits is recorded on-board and down-linked when the satellite comes into view of a ground station. Depending on the altitude of the satellite and the location of the ground station, time-in-view available for up and down-link averages about ten minutes.

From the above, it should be evident that actual imaging may only be possible for 30 minutes or less during each orbit. This may be restricted further by limitations on on-board data storage capacity, limits on electrical power or the capability of the satellite to get rid of dissipated heat.

Potential customers usually specify the image/s they require in terms of the geographical co-ordinates of its boundaries, a delivery time, resolution and, if required, type of processing. Before accepting a customer's order, these requirements are checked for executability. If executable, the image requested

are translated into basic imaging operations that suit the satellite's sensor and operational characteristics before being converted into commands to the spacecraft.

An area that needs to be imaged is only in view of the satellite for limited time periods and then only during some orbits. The frequency of imaging opportunities depends on the nature of the orbit and the latitude of the target area. Due to the near-polar geometry of a sun-synchronous orbit, areas near the Arctic and Antarctic can be imaged during every orbit while an area on the equator may only be in view occasionally. Since satellite orbits repeats periodically, it follows that over a period of weeks or months, there will be several opportunities to acquire an image of a specific target.

Usually a customer for an image has a delivery date in mind. The customer may want the image within 24 hours, a number of days or even weeks. Obviously, the longer the lead-time a customer provides the satellite operator, the more opportunities there will be to acquire the images. As time goes by, and the due date grows nearer, the number of imaging opportunities for a particular target obviously diminishes and it becomes more urgent to include that image in an upcoming imaging schedule.

On the other hand, some customers may require images of the same area at regular time intervals, e.g. daily, weekly, and monthly; to be able to monitor change – the monitoring of ice thickness in the Arctic during winter to guide shipping comes to mind. The image requests for customers like these obviously have to be included in every relevant imaging schedule. In short, any set of image acquisition orders for an upcoming number of orbits, may contain a mix of orders, ranging in urgency from high to low, and this mix makes the optimization process more tractable.

If more than one EO satellite are launched in such a way that they are phased in the same orbit plane, the resulting constellation of satellites can obviously provide better customer service by affording more frequent imaging opportunities of the same area on earth. There are numerous constellations in

geosynchronous orbit but, at time of writing, constellations of commercial EO satellites are still being planned or in early stages of commissioning. For example, the RapidEye constellation, one of the first purely commercial earth observation constellations, was launched on August 29, 2008, and has a primary purpose of providing daily coverage of Western Europe.

If we call the period between successive up / down-link opportunities an imaging cycle (consisting of say five orbits), we illustrate the SCoTWOP in the entity-relationship diagram shown in Figure 1-3.

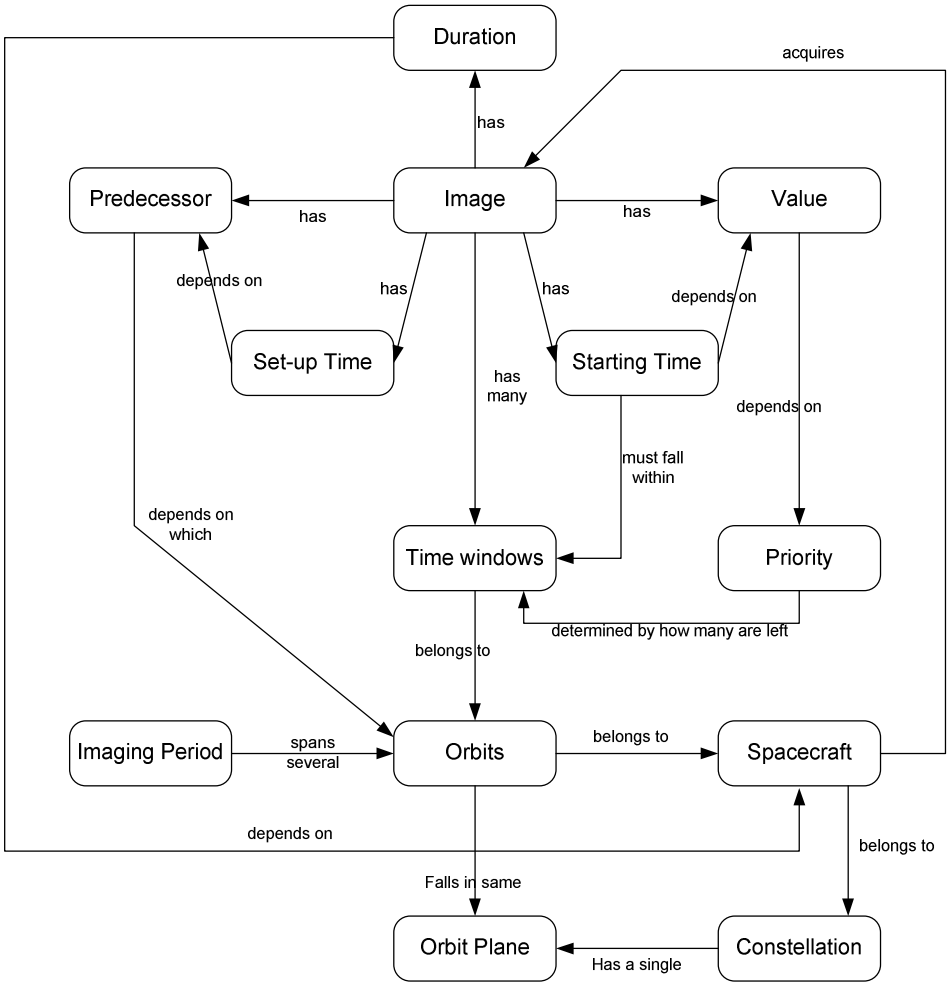

Figure 1-3: Entity-Relationship Diagram illustrating the various entities associated with a satellite image and the relationship between them

The diagram in Figure 1-3 indicates that every image has the following entities associated with it:

- A single **duration** (the time it takes to acquire the image);
- A **predecessor** (an image taken immediately before it – in the case of the first image of the planning cycle, the ground station is the predecessor);
- A **set-up time** (required to re-orientate the instrument – perhaps through slewing of the spacecraft)
- A **value**;
- One or more **time-windows** during which it would be feasible to acquire the image (i.e. when it is in view of the instrument) during the planning period;
- An earliest and latest **starting time** for each feasible time-window;
- A delivery or **due date**.

Note that:

- Set-up times depend on the specific **predecessor** and by implication also on the specific time-window;
- The value of an image is dependent on **priority**, which is determined by how many feasible time-windows are left before the image is due for delivery;
- Every time-window belongs to only one **orbit**
- Every orbit belongs to only one **spacecraft** in the **constellation**
- An **imaging period** span several upcoming orbits of one or more spacecraft
- Valid predecessors of an image belong to the same set of orbits (and hence the same spacecraft)

1.3 Research objective

The research reported on in this dissertation was aimed at finding answers to the following research question:

Investigate the feasibility and efficacy of a selection of metaheuristic algorithms to find global solutions for the commercial imaging scheduling problem for a constellation of multiple identical satellites intended to provide frequent coverage of the same geographical area.

By way of explanation the following should be emphasized:

- The focus is on the scheduling of commercial satellites that form part of the capital of a private business venture with the aim of making a profit. This is to be contrasted with scheduling of scientific or military satellites that usually belong to government institutions although, in some cases, their imagery may also be sold;

- The emphasis is on scheduling the activities of a constellation of identical or similar spacecraft that are phased along the same orbit, sharing the same orbit plane and having a common purpose: that of providing more frequent coverage of the same geographical area than a single satellite;

- The idea is to find optimal or near optimal global solutions for the constellation as a whole since optimal solutions for individual satellites do not necessarily translate to optimal solutions for the constellation;

- The problem is restricted to imaging activities as these are the source of revenue. Other on-orbit activities such as orbit maintenance will not be considered, and;

- The interest is on the performance of the different metaheuristics in finding solutions for the SSP.

1.4 Study outline

The rest of this dissertation addresses the following:

- Chapter 2 provides and overview of the state of the art in solutions for the SCoTWOP in the form of a review of current literature on the subject.
- Chapter 3 is devoted to a detailed analysis of the specific problem under view and culminates in a mathematical presentation of the problem.
- The implementation of the genetic algorithm is described in Chapter 4.
- Chapter 5 presents the results obtained, draws conclusions from the results, and discusses potential avenues for further study.

2 LITERATURE REVIEW

A review of available literature indicates that the SSP has been receiving attention from researchers for the past ten years or so. This chapter examines the findings of a review of publications with reference to the following:

- Types of SSP;
- Characterization or class of problem;
- Solution methods employed, and;
- Justification of the research objective.

2.1 Types of SSP

2.1.1 Different perspectives on the SSP

As mentioned in Chapter 1, the term *Satellite Scheduling Problem* does not refer to a single problem but in fact to a family of related problems that are all linked through the inviolable constraint of satellite orbit geometry. A satellite system for observation of the earth typically consists of one or more satellites in low earth orbit and one or more ground stations that receive image data from these satellites and/or transmit command and control data to the satellites. There are therefore two major categories of SSP:

- The first type of SSP pertains to the scheduling of the imaging activities of the satellite/s, i.e. it attempts to answer the question: Which images should be taken and when should they be taken? The activities that are scheduled happen in space.

- The second type of SSP focus on the activities of the ground station and attempts to answer the question: With which satellites should I communicate to up-link/down-link information and when should this communication happen? The activities that are scheduled are interactions between the ground and space.

Attempts have also been made to combine these two problems into a single problem that seeks to obtain an integrated schedule encompassing both ground and space activities. A third type of problem, though fairly rare, involves the scheduling of multiple instruments on the same satellite. This only applies to large satellites like SPOT-5 that have a number of different sensors on board that cannot necessarily be used at the same time. The SSP then needs to answer the question: Which instrument should I use to obtain a specific image and when should I use it? Alternatively, the question may be: Given that I have to use two sensors at the same time to acquire different images, which one should have priority?

Normally the generation of schedules for satellite activities take place on earth in a ground station. The schedule for upcoming orbit/s is prepared, tested for feasibility and safety, translated into spacecraft commands and then up-linked to the spacecraft when it passes overhead. There is, however, some research being done on moving at least some of the scheduling activities to the spacecraft itself. The problem remains an SSP of the first kind but the computing power to solve it moves from the earth to the spacecraft, giving the spacecraft greater autonomy. The idea is to make the problem dynamic in the sense that the spacecraft, given its knowledge of its own state and the images already taken, can perhaps make better informed decisions as to when to take images in response to new requests. This approach to the problem is also referred to as “automation of the scheduling

process". The survey below makes it clear that all these kinds of SSP have been topics of research in the past.

Barbulescu *et al.* (2002), (2004) and Howe *et al.* (2000) deal with the second type of SSP when they view the problem from the perspective of the Satellite Control Network of the United States Air Force. The network coordinates communications between 16 antennas, located at nine ground stations around the earth, and more then 100 satellites. User requests typically specify a time-window and duration for communication with a specific satellite. Similarly, Burrowbridge (1999) addresses the problem of scheduling multiple low altitude satellite communication contacts across one or more space tracking networks. The objective is to maximize the number of satellite contact opportunities.

Dungan *et al.* (2002), Frank *et al.* (2001) and Globus *et al.* (2002), (2003), (2004) address the problem of coordinating the imaging activities of NASA's growing fleet of EO satellites. This is an example of the first kind of SSP. Past practices of scheduling imaging activities on different satellites, or on different instruments on the same satellite independently of one another, required the manual coordination of observations by communicating teams of mission planners. Given the increasing number of satellites and the increasing demand for observation time, this approach is no longer viable, suggesting the need for a centralised scheduling function relying on automated techniques.

Pemberton (1999) and also Pemberton and Galiber (2000) view the broader problem of scheduling all operations on-board a satellite, and hence include not only instrument operations (sensor activities) in the scheduling problem, but also platform operations (e.g., maintaining the health, orbit and status of the spacecraft), and communications operations (between satellites or between the satellite and a ground station). This is therefore an instance of combining the first and second kind of SSP. In addition to EO satellites they also address the scheduling of operations of telecommunications and broadcast satellites. Dimitrov and Galiber (1998) also define satellite mission planning as the problem of mapping tasks (observation, communication, downlink, control

manoeuvres, etc.) to resources (sensor satellites, relay satellites, ground stations, etc.). Again, this combined the first and second kind of SSP.

Harrison and Price (1999) also have the first kind of SSP in view when they restrict themselves to the scheduling of a single Synthetic Aperture Radar (SAR) equipped satellite and seek to find out how many images can be acquired in a single (three minute) overpass over a typical area of interest and whether the scheduling process can be automated given a lead time of between ten and fifteen minutes. Ruan *et al.* (2005) and Cordeau and Laporte (2005) also address the single satellite problem without confining themselves to a specific type of satellite. Lemaître *et al.* (2000) also confine themselves to imaging operations but point to the fact that the greater flexibility of newer agile (body pointing) satellites, with their three degrees of freedom, makes them more difficult to schedule than the older mirror scanning satellites with their single degree of freedom.

Kitts (1994) addresses satellite scheduling from a completely different perspective: that of moving the scheduling function at least partially from the ground to the spacecraft itself i.e. an approach of automation of the scheduling process as alluded to earlier. The context is that of a number of satellites operating within a cross-linked constellation. This real-time control strategy merges the payload scheduling function, traditionally considered a high-level planning problem, with low level actuator control. This is a special instance of the first kind of SSP in which the scheduling function has been moved from the ground to spacecraft itself.

Khatib *et al.* (2003) also propose providing the satellite with a limited on-board scheduling capability. The idea is to allow the satellite to dynamically update up-linked schedules generated on the ground to allow for consideration of actual conditions or unforeseen changes in priority. Similarly, Pemberton and Greenwald (2002) advocate dynamic updating of up-linked static schedules to accommodate contingencies and recounts details of a number of potential approaches to effect this.

Vasquez and Hao (2000), (2001) investigate the problem of scheduling the imagining of a single satellite with multiple instruments such as SPOT-5, which is the latest in the series of SPOT satellites operated by France.

Muccio *et al.* (1999) take the position of a commercial satellite company that would like to find an optimal schedule for a constellation of satellites. They restrict themselves to imaging operations (the first kind of SSP) and solved the problem for a constellation of three identical satellites.

2.1.2 Commercial satellite operations

It is evident from the literature surveyed that the commercial aspects of satellite operations have received little attention. Most of the work in the field has focussed on military or scientific satellites. Examples of work in the military field are that of Barbulescu *et al.* (2002), (2004) and Howe *et al.* (2000) on the scheduling of the Satellite Control Network of the United States Air Force. NASA's focus has been primarily on scientific applications as the work of Dungan *et al.* (2002), Frank *et al.* (2001) and Globus *et al.* (2002), (2003), and (2004) attests.

Outside the United States, many satellites often have a dual purpose: a primary one of providing scientific or strategic imagery to government institutions, and a second, commercial objective to defray some of the costs of the government investment. The work of Vasquez and Hao (2000), (2001) on scheduling the various instruments aboard SPOT 5 applies to this kind of situation. The SPOT series of satellites were built and paid for by the French Space Agency CNES, but are operated for commercial purposes by the company SPOT Image. Similarly, the Radarsat satellites are paid for by the Canadian Space Agency but operated for profit by the private company Radarsat International. A similar arrangement is in place for the EROS satellites of Israel. While the purveyance of archived imagery is a prime source of income for these organisations, it is fair to assume that new strategic image acquisitions for government users will take precedence over commercial requests for new imagery. This aspect often makes

the scheduling problem less of a purely commercial one and more a matter of juggling externally imposed priorities.

The literature survey indicate that only Muccio *et al.* (1999) placed themselves in the position of a purely commercial, if fictitious, satellite company that would like to find an optimal schedule for a constellation of three satellites. This is perhaps not surprising seeing that purely commercial satellites have only recently come to the fore. The first high resolution satellite launched for purely commercial purposes was IKONOS (USA) in 1999, with QuikBird (USA) following two years later.

2.1.3 Constellations of satellites

For the purpose of this dissertation a distinction will be made between a *fleet* of satellites and a *constellation* of satellites. The term *fleet* will refer to a grouping of satellites, the orbits of which are not necessarily geometrically coordinated, and which do not share the same purpose although their activities may be coordinated for some reason or other. In general, the different satellites in the fleet may have different capabilities so that it may not be possible to allocate the same imaging task to different satellites. In contrast, the term *constellation* will be used for a number of satellites that share a common purpose and the orbits of which are geometrically coordinated to provide a specific rate of earth coverage. The satellites in the constellation are generally identical in capabilities so that all of them can perform the same imaging tasks.

Given this definition, the literature survey reveals that few, if any, instances of earth observation constellations exist. In contrast, constellations of geostationary satellites, that provide communication and global positioning services, are more common. An example of a fleet of satellites is the group of satellites operated by the United States Air Force which is being controlled by their Satellite Control Network. This fleet is alluded to in the work of Barbulescu *et al.* (2002), (2004) and Howe *et al.* (2000), but it is important to note that they focus on scheduling the activities of a network of ground stations for this fleet rather than the activities of the fleet itself (the second kind of SSP). The work is however still

relevant in that the scheduling problem also has to deal with the complexity induced by satellite orbit geometry.

The subject of the work of Dungan *et al.* (2002), Frank *et al.* (2001) and Globus *et al.* (2002), (2003), and (2004) is on the scheduling of NASA's diverse and large satellite fleet rather than on purpose built constellations.

At time of writing only a few groupings of EO satellites may conceivably qualify for the term commercial constellation: France's individual SPOT-2, SPOT-4, and SPOT-5 satellites are operated as a constellation of satellites, as are EROS A and B of Israel. However, these satellites do not share the same capabilities, nor were they launched closely together or simultaneously. In a certain sense they may almost be called accidental constellations since they came into being because the older satellites, like SPOT-2, simply lasted longer than expected. Since the operating life of a satellite is typically seven years, a true constellation can only be ensured if the satellites are launched within a reasonable time frame or even simultaneously on the same launch vehicle. If, for example, the constellation is to be comprised of seven satellites, each with a design life of seven years and they are launched at a rate of one per year, the full constellation may in practice be difficult to achieve, since the first satellite will have only one year of its life left by the time the last one is launched, leaving little margin to compensate for launch failures and project delays

At time of writing, the only real example of this type of constellation, built for purely commercial purposes, is the five satellite RapidEye constellation of Germany which has been launched in August 2008. The only other existing constellation that comes close to this is the five satellite Disaster Monitoring Constellation (DMC) collectively owned by the Algerian, Nigerian, Turkish, British and Chinese governments and which has been constructed to provide emergency earth imaging for disaster relief. Spare available imaging capacity of the DMC is sold under contract, but commercial schedules will obviously be pre-empted by the dictates of disaster relief.

2.2 Characterisation of the SSP in literature

Most researchers refrain from categorizing the SSP more specifically than an example of an *oversubscribed scheduling problem*, meaning there are more requests for a resource than can be satisfied, so that some requests remain unfulfilled. Given that it is impossible to schedule all the activities, the focus is on scheduling as many of the urgent activities as possible in a given time period by assigning appropriate weights to each activity. Activities relating to less urgent unfulfilled requests are then scheduled for a later time period. Notably this approach is taken both by those working from the ground station perspective, like Barbulescu *et al.* (2004), as well as those that view the problem from the satellite perspective like Globus *et al.* (2004)

There is also general agreement that this type of problem is intractable. Muccio *et al.* (1999) point out that one formulation for the SSP is the machine shop scheduling formulation which involves integer assignment variables, real start time variables, and potentially thousands of other variables and constraints. As a large machine shop scheduling problem the SSP can be classified as NP-complete.

Pemberton and Galiber (2000) emphasize the most obvious difference between traditional scheduling problems and the SSP: The fact that the resources (i.e., the satellites) are orbiting the earth places an additional set of constraints on when a task can be executed. In agreeing that the SSP satellite-scheduling problems are usually *over-constrained*, they approach it as a *constraint-optimization* problem rather than a *constraint-satisfaction problem.* The term over-constrained simply means that there are few (if any) feasible solutions when the objective is to schedule all requested activities. The constraint-satisfaction approach is therefore to try and find at least one feasible solution without regard for optimality. Another approach is to seek solutions that maximize the number of constraints that are satisfied. This is in essence the constraint-optimization approach which relies on the concept of *soft* vs. *hard* constraints. Hard constraints cannot be violated whereas there may be some leeway to violate soft constraints. In the case of the SSP, constraints that are imposed by orbit and

earth geometry are hard, whereas the customer's expected delivery time for an image may be somewhat flexible.

Vasquez and Hao (2000) provides a Knapsack formulation for the SSP following Bensana *et al.* (1996) who employed a Multi-Dimensional Knapsack formulation.

2.3 Solution methods employed for the SSP

Predictably, the methods employed for the SSP range from exact integer programming methods for smaller problems to stochastic search methods and constraint programming for larger ones.

Harrison and Price (1999) developed a partial enumeration method that exploited time constraints to prune the search tree to make a smaller problem more tractable. Similarly, Pemberton (1999) employed priority segmentation and then used branch-and-bound to solve sub problems. Muccio *et al.* (1999) also considered the problem in two parts, solving first the allocation and then scheduling problem through linear and integer programming.

Among those that investigated metaheuristics, Wolf and Sorenson (2000) found that their genetic algorithm offers considerable improvement when compared to a look-ahead algorithm, which in turn, performed better than a simple priority dispatch method. Barbulescu *et al.* (2004) found genetic algorithms and squeaky wheel optimization both successful at scheduling the Air Force Satellite Control Network.

Globus *et al.* (2004) compared thirteen scheduling algorithms including variants of stochastic hill climbing, simulated annealing, the genetic algorithm, squeaky wheel optimization, and iterated sampling. They found that simulated annealing outperformed the other methods.

Among those that employed constraint programming approaches are Pemberton and Galiber (2000) and Dungan *et al.* (2002). Lemaître *et al.* (2000) used two quite different methods: a constraint programming framework, and a local search

method. They found that each one has its own advantages: the former being very flexible, while the latter gives better performance.

The fact that scheduling activities to date have primarily focussed on single satellites and fleets of diverse satellites rather than true constellations makes its easy to understand why the focus has been on finding feasible solutions rather than optimal or near optimal ones. Hence the preference for constraint programming methods by some researchers. In fact, in a fleet of satellites there may not be much scope for optimisation at all. Since the satellites often do not have the same purpose or capabilities, they can not necessarily be employed for the same imaging tasks and a specific task can therefore not be switched between satellites to find a better scheduling solution.

In this case satellites can be treated as more or less independent when it comes to task allocation and the primary objective of the scheduling problem seems to be to find a solution where access to common scarce resources such as a ground station has to be scheduled. Again we refer to the work of Barbulescu *et al.* (2002), (2004) and Howe *et al.* (2000) on the Satellite Control Network of the United States Air Force and the work of Dungan *et al.* (2002), Frank *et al.* (2001) and Globus *et al.* (2002), (2003), (2004) on NASA's fleet of EO satellites. The same is often true for a single satellite with multiple instruments as pointed out by Vasquez and Hao (2000), (2001) for SPOT-5. In this case there is not so much a choice as to which instrument to use for which task, but rather when each instrument should be allowed to have access to spacecraft resources such as power.

The advent of commercial constellations that are operated to maximise profit, opens up new optimisation opportunities, since the focus is on acquiring as many images as possible in a given time period. The acquisition of some images may be more easily deferred to later planning periods. Since the satellites in the constellation are identical, and since they can, at least at higher latitudes, cover the same ground within a short space of time, there may also be more than one opportunity to collect imagery of the same area in the same planning cycle. As a result the problem is no longer over constrained and one or more feasible

solutions may exist, at least until demand overwhelms the extended supply offered by the constellation. There may therefore be opportunities for optimization in choosing which satellite to task with acquiring a specific image and when the image should be taken. Since these choices create a number of feasible solutions and the scheduling problem for commercial constellations is therefore not over constrained, the use of metaheuristic techniques becomes a more attractive proposition.

Even so, metaheuristics have been applied to the SSP by several researchers, sometimes with conflicting results. Notable among these is the fact that Barbulescu *et al.* (2004) found genetic algorithms and squeaky wheel optimization both successful at scheduling the Air Force Satellite Control Network whereas Globus *et al.* (2004) found that simulated annealing outperformed genetic algorithms and other methods when it comes to scheduling the NASA satellite fleet. The fact that these two groups view the problem from different perspectives, i.e. ground segment vs. space segment scheduling, may account for the different experience but it needs pointing out again that both problems share the constraints imposed by orbit geometry that make the SSP distinctive. This difference remains therefore intriguing.

Unfortunately most articles do not say much about the specific ways in which solutions were encoded as genotypes. Wolfe and Sorensen (2000) report a genetic approach that is novel in that it uses two additional binary variables, one to allow the dispatcher to occasionally skip an imaging task in the queue and another to allow the dispatcher to occasionally allocate the worst position to the imaging task. The resulting schedules seem to improve on the results of other methods they investigated.

2.4 Justification of the research objective

Given the literature overview above, it is evident that the problem of finding optimal schedules for a constellation of identical EO satellites has not received much attention in the past. The reason for this is that such constellations are mostly still in the planning stage. It is also evident that scheduling efforts in the

past have not been widely successful and that very little of the research in this field has found its way into practical scheduling systems. One reason for this is that in practice operational imperatives simply preclude any real optimization effort – there is simply not enough time.

To understand this, one may consider the case of a single satellite in near-polar orbit, with a ground station at high latitude so that the satellite is in contact with the ground station during almost every orbit. Typically there would then be 90 minutes or so between contacts with the satellite. During this period a schedule for the upcoming orbit is generated, tested, and codified into a set of commands for the satellite. Given the high premium placed on the safety of the satellite, it is not unreasonable for the satellite operator to demand that the schedule be finalized at least one hour before it is uplinked. This leaves time to ensure that demands resulting from the schedule would not place the satellite in any danger, for example: requiring impossible or risky manoeuvres. The duration of the contact with the satellite is typically ten minutes or so, leaving approximately 20 minutes out of every 90 minute orbit to find an optimal schedule.

The question may well be asked why these schedules are not prepared well in advance, say days before they are to be executed. The answer lies in the fact that new requests to confirm executability of potential image acquisitions arrive continuously at a rate of about 3000 per day and that about 500 of these result in confirmed orders every day. Furthermore, images that have already been taken may turn out to be defective in some way and may have to be rescheduled. The result is that scheduling operations take place on a full-time basis and that there is little or no time to optimize schedules. Satellite operators therefore merely seek to maintain a schedule in which each image confirmed request has been allocated a feasible timeslot. Any scheduling algorithm that is intended to find optimal or near optimal solutions will therefore have to be very fast.

Metaheuristics such as genetic algorithms that deal with a population of solutions are much more effective when they can deal with large populations of feasible solutions. If there are only a few feasible solutions, a lot of time is wasted on evaluating infeasible solutions in order to try and maintain a viable

population with enough genetic variability to complete a specified number of generations without stagnating. It follows that the higher the number of feasible solutions, the faster the genetic algorithm, all other factors being equal. The concept of a constellation of identical satellites in the same orbit plane not only enables the satellite operator to acquire and sell more images, it also affords more opportunities to obtain the same image. Depending on the latitude of the area of interest, it may be possible to acquire an image of the same spot on the earth a number of times by more than one satellite, albeit from different angles. The increased number of imaging opportunities for the same image should translate into a higher number of feasible solutions and therefore the potential for a faster algorithm.

However, while increasing the number of feasible solutions, the potential to acquire an image of the same area on multiple occasions complicates the representation of solutions in terms of genotypes that can be manipulated by the algorithm. In ensuring that an image is taken only once, the representation of the solution must now encode the information pertaining to a sequence of images each having multiple potential imaging time-slots which, in turn, belong to different satellites. The challenge is therefore to represent solutions in a form that is simple to manipulate while preserving the more comprehensive information it encodes.

In the light of the above it seems that the objective of determining which metaheuristic (among a selection of those employed for the SSP by other researchers) will be quick and effective at finding solutions for the scheduling of a constellation of satellites (i.e. the SCoTWOP), is a worthwhile endeavour. Of course a number of metaheuristics will have to be tested in order to provide a testing environment, and this literature overview suggests that the tabu search, simulated annealing, and genetic algorithm metaheuristics are prime candidates.

2.5 Conclusion

Different perspectives have yielded a number of different but related problems that all carry the SSP moniker and there is no consensus on a standardised or

canonical form for the problem. Characterizations of the problem in terms of more general classes of optimization problems also vary but there is a general consensus that the problem is NP-complete. In general it appears that constraint programming has been the favourite solution method for larger problems. The efficacy of results obtained though metaheuristics appears to depend on the type of problem.

Research into the problem of scheduling the imaging operations of constellations of commercial satellites using metaheuristics such as a genetic algorithm appears to be a worthwhile endeavour because:

- The first truly commercial constellations have only recently been launched;
- The nature of these constellations should make them more amenable to optimisation efforts and therefore perhaps more suitable for the application of metaheuristics, and
- The results obtained thus far with a variety of metaheuristics appear not to be conclusive, yielding mixed results as to their relative efficacy.
- Encoding of solutions in the form of a genotype for use in metaheuristics like a genetic algorithm becomes very challenging when multiple imaging opportunities exist for the same spot on the earth.

3 PROBLEM ANALYSIS

3.1 Problem context

The kind of problem of interest here involves a constellation of at least two but probably not more than ten, identical satellites that share the same orbit plane. The purpose of such a constellation would be to cover the same geographical area more frequently than is possible with a single satellite. This higher rate of coverage is sought after by customers who would like to monitor change over a certain geographical area. An example of this kind of use is the need to monitor the extent of damage caused by natural disasters. Whereas a particular area could be covered by a single satellite over a period of days, the constellation makes it possible to cover the same area within a day or even a few hours. The orbit of the satellites in the constellation would typically be sun-synchronous and the satellites would be phased equidistantly around the orbit so that they collectively provide coverage of the whole earth within a period of days. The nature of a sun-synchronous orbit results in a situation where areas of the earth closer to the poles will be covered more frequently than those closer to the equator. In addition, the number of satellites in the constellation will also determine the number of imaging opportunities of the same area over a given time period.

Customers order images from the satellite operator and agree on an acceptable lead-time, the length of which will factor into the price of the image. During the lead-time, which may be anything up to 120 days, there may be numerous

opportunities or time-windows when the area to be imaged may be imaged by the satellites in the constellation. Every satellite in the constellation completes 14 to 15 orbits every 24 hours. During some orbits the satellite comes within line-of-sight of the ground station and then commands may be up-linked to the satellite while raw imagery data and telemetry are down-linked to the ground station.

Among the commands up-linked to the satellite is a series of imaging instructions telling the satellite where to point the imager and when to turn it on and off during the upcoming number of orbits when it is not in contact with the ground station. It is the determination of series of imaging instructions that forms the heart of the SCoTWOP.

We assume the following:

- The number of orders waiting to be imaged exceeds the daily imaging capacity of the constellation and the satellite operator needs to prioritise the orders in terms of when they should be filled, and
- The satellite operator has at his disposal both orbit propagator and satellite simulator software that allows him to determine time-windows and instrument orientations for each image for each satellite.
- The operator has already accounted for other on-board activities such as warm-up periods, heat dissipation, orbit maintenance, etc. so that all that remains is to schedule the imaging activities.

3.2 The objectives of optimisation

The objective of the commercial satellite operator is to maximize income on the investment in the constellation of satellites. This can be achieved if the imaging activities for each planning period, i.e. the period between successive ground station-to-satellite contacts, are such that the constellation as a whole acquires those images that yield the highest value given the constraints.

The term *value* is not necessarily used in the strictly monetary sense i.e. the price of the image. Rather it is an appropriate weighing factor that incorporates aspects such as price, urgency, the number of imaging opportunities left and, in the case of optical satellites, the probability of cloud cover over the area to be imaged. Value should thus be seen as an indicator of the importance or priority that the satellite operator may attach to an image and this value may change over time depending on the specific planning period under consideration.

The obvious *primary* objective for optimisation is therefore to maximise the value of image data acquired over the planning period under consideration. Note that it does not make sense to consider a longer-term objective, since new orders will probably arrive while the current planned series of imaging instructions is being executed and may make any long term schedules obsolete. Some researchers would however add a *secondary* objective to that of maximising the value of image data, such as minimising the slewing of the spacecraft required to acquire the image data.

While it is possible to cope with the problem of multiple objective functions by, for example, assigning relative weights to the objectives, we note that time taken up by slewing leaves less time for imaging and thus can be handled by appropriate constraints.

The objective function is therefore:

Maximise the value of image data acquired by the constellation over the upcoming planning period.

3.3 The constraints imposed on optimisation

We assume that the images considered for the upcoming planning period have been selected from a larger set on the basis that they meet the orbit and visibility constraints. Furthermore, some constraints, such as those dealing with priority or cloud cover probability can be effectively incorporated by suitable adjustments in the value of the image. For example, if weather forecasts indicate a high probability of cloud cover for an image, the value of that image should be

adjusted such that the image is forced out of the solution. Similarly, if the upcoming orbit is the last or only opportunity to acquire a specific image, the value of the image should be adjusted such that it is forced into the solution. It stands to reason that some common sense is required here, for it doesn't matter how high the priority of an image is, if it cannot be imaged because of excessive cloud cover then increasing the priority cannot produce the image

We will now consider the constraints that need to be taken into account explicitly.

- **On-board memory**. The total amount of memory required by the images scheduled to be taken by a satellite during the planning period should not exceed the capacity of its on-board Solid State Recorder (SSR).

- **Visibility.** An image can only be imaged if it's within the sensor's field of view so that imaging should start and finish within a specific time-window. Note that there can be more than one time-window per image per satellite.

- **Sequence.** Since satellites only move in one direction around the earth it has to take images in the sequence that it encounters them.

- **Non-interference.** Two images cannot be taken at the same time by the same satellite.

- **Singularity.** Every image should only be taken once during the planning period.

3.4 Mathematical model for the SCoTWOP for commercial constellations

The realisation that optimisation problems from diverse disciplines can be expressed with similar mathematical models and that most problems can in some way be related to classes of standard problems had an organising influence on operations research. In keeping with this spirit, superficial examination of the SCoTWOP reveals that it too shares characteristics with well-known problems:

- The objective function and on-board memory constraint, taken together forms the classic Knapsack Problem.

- The remaining constraints relating to visibility, non-interference and singularity, reminds strongly of Machine Scheduling Problems.

Researchers such as Bensana *et al.* (1996) and Vasquez and Hao (2000) have recognised the knapsack characteristics of the problem, and most researchers seem to agree that it is a kind of scheduling problem without really questioning the truth of this characterization. However, while some of the constraints seem to justify this view, the problem does not really have the hallmarks of a classic scheduling problem. In the classic scheduling problem the task is typically to find a sequence of time slots during which a set of activities are to be performed and the objective is to complete all the activities before their respective deadlines. In the SCoTWOP, the sequence of the time-windows during which images can be taken is determined by orbit configuration and the sequence is therefore fixed. All that can really be decided is which of these time-windows to use for imaging, with the objective of taking as many images as possible or to maximize the value of the images. We therefore note that the SCoTWOP has more in common with what may be regarded as a version of the dual of the classic scheduling problem. This is the reason for referring to "time-window optimisation" rather than calling it a "scheduling" problem.

It would be more useful if the SCoTWOP could be related to a single type of standard problem rather than being viewed as a hybrid knapsack-scheduling problem. Careful examination reveals that this is indeed possible: the SCoTWOP is strongly related to the well known Vehicle Routing Problem (VRP), specifically to a variant of the VRP known as the Vehicle Routing Problem with Multiple Time-windows (VRPMTW). To see this, we need to digress and examine the VRPMTW more closely before we return to the mathematical formulation of the SCoTWOP.

3.4.1 The vehicle routing problem with multiple time-windows

The Vehicle Routing Problem (VRP) encompasses a whole class of problems in which a set of minimum-cost routes must be determined for a fleet of vehicles to deliver goods to a set of geographically dispersed customers with known demands. Vehicle routes originate and terminate at one or more depots. The VRP was recognised as early as 1959 by George Dantzig as a central problem in the fields of transportation, distribution and logistics and has become a well-known integer-programming problem, which falls into the category of NP Hard problems. The problem is considered to lie at the intersection of the well-studied Traveling Salesman Problem (TSP) and the Bin Packing Problem (BPP). Common variants of the VRP are:

- The Capacitated VRP (CVRP) in which every vehicle has a limited capacity, and;
- The VRP with time-windows (VRPTW) in which every customer has to be supplied within a certain time-window.

The variant of the VRPTW with more than one time-window per customer is known as the VRP with Multiple Time-windows or VRPMTW.

Normally the objective of the VRPMTW is to minimize the set-up cost of the size of the vehicle fleet (number of vehicles) plus the sum of travel time and waiting time needed to supply all customers in their required hours. The following formulation of the VRPMTW is adapted from that of De Jong *et al.* (1996). The notation is derived from that of Desrocher *et al.* (1988):

- G is a graph with a set of vertices V and a set of arcs A so that $G = (V, A)$;
- N is the set of n customers, with $N = \{1, \ldots, n\}$ and i is the index;
- V is the set of vertices so that $V = \{0\} \cup N$ where 0 corresponds to the depot;
- A is the set of arcs $A = (\{0\} \times N) \cup I \cup (N \times \{0\})$; where:

- I *is* the set of arcs connecting the customers, $I \subseteq N \times N$;
- $\{0\} \times N$ is the set of arcs from the depot to the customers, and
- $N \times \{0\}$ is the set of arcs from the customers to the depot.

- Z is any proper subset of the set of vertices V so that its complement $\overline{Z} = V - Z$;
- $F_i = \{1, \ldots, w_i\}$ is the set of time-windows associated with customer i ;
- f is the index for the time-windows of each customer i , so that $f \in F_i$;
- TW_{if} is time-window f of customer i , with $TW_{if} = \left[e_{if}, l_{if} \right]$ such that $e_{if} < l_{if}$ and $l_{if} < e_{if+1}$;
- q_i is the positive demand of customer $i \in N$;
- Q is the capacity of each of the (identical) vehicles;
- c_{ij} is the cost associated with arc $(i, j) \in A$; where c_{0i} includes the fixed vehicle cost. This enters the fixed costs of the vehicles into the objective function;
- t_{ij} is the travel time along arc $(i, j) \in A$ (the cost of which is included in c_{ij} above);
- d_i is the duration of time to serve the customer $i \in N$;
- C *is* the cost for one unit of waiting time.

The variables of the problem are the following:

- S_i is the time instant that service starts for customer $i \in N$;
- W_i is the waiting time at customer $i \in N$ if the vehicle arrives early;

- y_i is the load of the vehicle when it arrives at customer $i \in N$;
- x_{ij} is a binary variable equal to 1 if arc $(i, j) \in A$ is used by a vehicle, 0 otherwise;
- u_{if} is a binary variable equal to 1 if customer $i \in N$ is served in time-window $f \in F_i$.

Normally the objective of the VRPMTW is to minimize the number of vehicles and the sum of travel time and waiting time needed to supply all customers in their required hours. Since fixed vehicle cost is accounted for in the arc cost c_{0i} of arcs leaving the depot, and travel time are accounted for in all the arc costs, c_{ij}, the VRPMTW problem can be formulated as:

Minimize

$$\sum_{(i,j)\in A} c_{ij} x_{ij} + C \sum_{i \in N} W_i \tag{1}$$

subject to:

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in N \tag{2}$$

$$\sum_{j \in V} x_{ji} = 1 \qquad \forall i \in N \tag{3}$$

$$\sum_{f \in F_i} u_{if} = 1 \qquad \forall i \in N \tag{4}$$

$$\sum_{f \in F_i} u_{if} e_{if} \le S_i \le \sum_{f \in F_i} u_{if} l_{if} \qquad \forall i \in N \tag{5}$$

$$\sum_{i \in V} x_{ij} \left(S_i + d_i + t_{ij} + W_i \right) = S_j \qquad \forall j \in V \tag{6}$$

$$\sum_{i \in V} x_{ij} \left(y_i - q_i \right) = y_j \qquad \forall j \in V \tag{7}$$

$$\sum_{i \in Z} \sum_{j \in \overline{Z}} x_{ij} \geq 1 \qquad \forall Z \subset V \qquad (8)$$

$$q_i \leq y_i \leq Q \qquad \forall i \in N \qquad (9)$$

$$x_{ij} \in \{0,1\} \qquad \forall (i, j) \in A \qquad (10)$$

$$u_{if} \in \{0,1\} \qquad \forall i \in N, f \in F_i \qquad (11)$$

$$S_i \geq 0 \qquad \forall i \in N \qquad (12)$$

$$W_i \geq 0 \qquad \forall i \in N \qquad (13)$$

Constraints (2), (3), and (10) ensure that every customer is visited exactly once. As a consequence no two vehicles ever use the same arc, ensuring that as many of the arcs (0, *i*) are used as there are vehicles, and that the fixed costs of all vehicles are counted. Constraints (4), (5), and (11) ensure that the service of each customer starts within one of its time-windows, and constraints (6) and (13) ensures that a vehicle has enough time to travel from customer *i* to customer *j*. Constraint (8) is a sub-tour elimination constraint that is retained here even though constraints (5) and (6) usually serve to eliminate sub-tours in practical applications. Constraints (7) and (9) ensure that the loads of the vehicles are feasible when they arrive at a customer. Note that constraints (5) and (9) can easily be written as two inequalities each and that the result would be a mixed integer program in standard form that can readily be solved exactly for small instances.

Many formulations of the VRPMTW do not attempt to minimize waiting time explicitly, or apply less rigid bounds on the time-windows. In this case the objective function reduces to:

Minimize

$$\sum_{(i, j) \in A} c_{ij} x_{ij} \qquad (1a)$$

In keeping with this formulation, constraint (6) changes to:

$$\sum_{i \in V} x_{ij} \left(S_i + d_i + t_{ij} \right) \leq S_j \qquad \forall j \in V \qquad (6a)$$

3.4.2 The analogy between the VRPMTW and the SCoTWOP

At first glance, the analogy between the VRPMTW and the SCoTWOP is at once evident and, at least superficially, fairly obvious. Apart from the objective function, which will be discussed later, the correspondence set out in Table 3-1 below seems logical.

Table 3-1: The Correspondence between the VRPMTW and the SCoTWOP

VRPMTW Entity	SCoTWOP Entity
Customers	Images
Routes (Sequence of customers)	Sequence of images / imaging opportunities
Vehicles	Satellites
Demands	Image memory requirement
Depot	Ground Station /s
Distance (travel time) between customers	Set-up (slewing) and travel time between images
Vehicle capacity	Satellite SSR capacity

VRPMTW Entity	SCoTWOP Entity
Customer service time-windows	Image time-windows

This analogy between the VRPTW and SCoTWOP can be misleading because there are subtle but important differences between the two problems:

- In the VRPMTW, the time-window constraint is imposed by the customer on the vehicles, but in the SCoTWOP the opposite is true: the times-windows are imposed by the satellite (vehicle) on the images (customers). The consequence of this is that the satellites in the SCoTWOP, unlike vehicles in the VRPMTW, are not necessarily interchangeable since their time-windows for the same image are not the same. In VRPMTW parlance it would mean that two vehicles (no constraints) cannot possibly visit the same customer at the same time.

- In the graph of the VRPMTW the distance between any two customers is normally the same regardless of direction of travel. In contrast, the constraints imposed by satellite orbits in the SCoTWOP results in a situation where the distance between two image locations A and B is not the same when travelling from A to B than when traveling from B to A. Once the satellite has passed location A on its way to B, it cannot return to location A without orbiting the earth at least once.

- Due to the nature of satellite orbits, the distance between image locations, and therefore travelling and slew time, is not constant but varies according to the satellites' progress through its orbit cycle. In graph-theoretical terms it means that there can be more than one arc connecting a pair of images. This concept is illustrated in Figure 3-1 below.

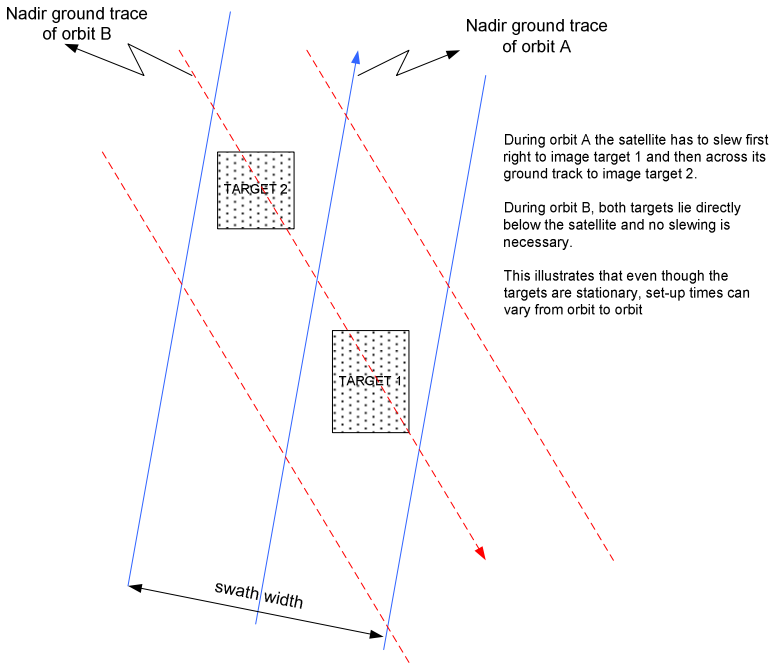

Figure 3-1: Although ground distances between image targets remain constant, travel and slewing distances may vary with the satellite's progress through its orbit cycle.

- The differences in objectives of the VRPMTW and the SCoTWOP extend beyond the fact that the one seeks to minimize cost while the other seeks to maximise value. More important is the fact the VRPMTW has a fixed number of customers that all have to be serviced by the minimum number of vehicles, whereas the SCoTWOP seeks to take as many images as possible given a fixed number of satellites. More about this later.

In addition to the important differences noted above, there is also a number of other, perhaps less significant differences that relate to slightly different interpretations of parameters and constraints that appear in both the VRPMTW and the SCoTWOP. As such they present no additional difficulties and do not in themselves define the unique characteristics of the SCoTWOP. These differences include the following:

- The time-windows in the VRPMTW apply to the start time of the service only and do not bound the completion time of the service. In the SCoTWOP imaging has to start and end within the bounds of the time-windows otherwise the image won't be in sight anymore. The time-window constraint in the SCoTWOP is a hard constraint whereas time-window constraints in the VRPMTW are often considered to be soft.

- Since satellites always travel forward in fixed orbits, their travel time is not really a factor because the satellite travels past an image target whether it intends to acquire the image or not. This also obviates the need to account for waiting time since the satellite determines the time-windows.

- Since the number of satellites is fixed there is no need to account for a fixed cost component.

- In the SCoTWOP, the satellites pick-up loads at service locations, whereas in the VRPMTW vehicles deliver loads at service locations.

3.4.3 Mathematical formulation of the SCoTWOP

Given the similarities and differences between the VRPMTW and the SCoTWOP pointed out above, the formulation previously given for the VRPMTW can now be adapted to the specific needs of the SCoTWOP. The key to this adaptation is that the set of vertices *A* of graph *G* should be associated with the time-windows of images rather than the images itself. In other words, the correspondence between the VRPMTW and the SCoTWOP should be between customers and time-windows rather than between customers and images. The arcs in the graph would then correspond to the non-imaging time periods between imaging time-windows. In this way there will still be a maximum of one arc connecting the members of any pair of vertices (time-windows) in subset *I* of the graph *G*.

The characteristics of each image, such as value, duration, memory requirement etc. can be associated with each of its time-windows. Similarly, parameters like the setup (slew) time should be associated with the arcs between time-windows

rather than the arcs between images. Furthermore, we should account for the fact that the number of satellites (vehicles) is fixed and that time-windows belong to satellites rather images. For this purpose we introduce the index *k* to denote the *k*-th member of the set of identical satellites *P*.

We also need to account differently for the setup time associated with each arc (time interval) between time-windows. During this time interval the satellite may have to slew or set up its instrument to acquire the next image. Since this is to be discouraged, we associate a penalty with the time spent slewing but not with any remaining travelling time if slewing is complete before the next time-window starts. To account for the highly asymmetric nature of the graph we also associate a large time penalty with the arcs connecting a specific time-window and those that preceded it in time. This is justified by the fact that the spacecraft has to circle the earth at least once before being able to acquire an image of the area that it has already passed. In other words, we allow the satellite to go "back" in time but only at the penalty of associating with this move a very large "setup" time. In this way the graph *G* is asymmetric but at least bi-directional with two arcs, a "long setup time" one and a "short setup time" one, connecting vertices associated with the same spacecraft. We account for this penalty by taking the maximum of the slew or setup time and the travel time. In the case of travelling "forward" the cost of the travelling time left after slewing (if needed) is zero.

Using similar notation as presented in 3.4.1 the SCoTWOP can now be defined as follows:

- M is the set of m images with index f, so that $f \in M$;
- P is the set of p satellites with index k, so that $k \in P$;
- N is the set of n time-windows with index i, so that $i \in N$;
- N_k is the set of imaging time-windows associated with satellite k, with $N_1 = \{1,\ldots,n_1\}$, $N_2 = \{n_1+1,\ldots,n_2\}$, $\ldots$, $N_p = \{n_{p-1}+1,\ldots,n_p\}$. It is

assumed that no two time-windows are identical. This is in any case highly unlikely but can be achieved by slight changes to the starting time of an offender. As a result the time-windows indexed by $N_1,\ldots,N_P$ form a categorical system with union the set of time-windows N.

- O_f is the set of imaging time-windows associated with image f, with $O_f = \{1,\ldots,n_f\}$;

- $G_k = (V_k, A_k)$ is a graph with a set of vertices V_k and a set of arcs A_k associated with satellite k;

- V_k is the set of vertices representing time-windows associated with satellite k so that $V_k = \{0\} \cup N_k$ where 0 corresponds to the time-window associated with ground station contact;

- A_k is the set of arcs representing time intervals between time-windows associated with satellite k so that $A_k = (\{0\} \times N_k) \cup I_k \cup (N_k \times \{0\})$; where:

    - $I_k = N_k \times N_k$ is the set of arcs corresponding to the time intervals between time-windows that belong to the same satellite k;
    - $\{0\} \times N_k$ is the set of arcs corresponding to the time intervals between the ground station contact time-window and the imaging time-windows of satellite k, and;
    - $N_k \times \{0\}$ is the set of arcs corresponding to the time intervals between the imaging time-windows of satellite k and the ground station contact time-window.

- $G = (V, A)$ is then a graph consisting of the union of all sub-graphs G_k so that $G = \bigcup_{k \in P} G_k$ with a set $V = \bigcup_{k \in P} V_k$ of vertices and a set of arcs $A = \bigcup_{k \in P} A_k$;

- Z is any proper subset of the set of vertices V so that its complement $\overline{Z} = V - Z$;

- $[e_i, l_i]$ denotes the time interval of time-window i, such that $e_i + d_i < l_i$ and $l_i < e_{i+1}$;

- d_i is the duration of the imaging activity associated with time-window $i \in N$;

- q_i is the positive memory requirement for time-window i if the image associated with it is taken;

- Q is the solid state recorder (SSR) capacity of each satellite k assuming that they are identical;

- t_{ij} is the slewing or manoeuvring time required by a satellite if it has to take image j immediately after image i. This time period has a finite value in cases where the same satellite can take both images during the planning period. In cases where image j precedes image i, t_{ij} assumes an infinite value;

- R is a large positive number used for the penalty time associated with every arc connecting a time-window with one already passed or belonging to a different satellite;

- c_{ij} is the time penalty associated with the time interval between two time-windows i and j where $c_{ij} = \max(t_{ij}, R)$;

- v_i is the value of the image associated with time-window $i \in N$, considered constant for this planning period;

The variables of the problem are the following:

- S_i is the time instant that imaging starts during time-window $i \in N$;

- x_i is a binary variable equal to 1 if an image is taken during time-window $i \in N$, 0 otherwise;

- $y_{ij}, (i,j) \in A_k$ is a binary variable equal to 1 if satellite $k \in P$ uses time-window j to take an image directly following on taking an image during time-window i without using an intervening time-window, i.e. $y_{ij} = 1$ if $(i,j) \in A_k$ is used (by satellite $k \in P$), otherwise $y_{ij} = 0$.

The SCoTWOP can be formulated as follows:

Maximize:

$$\sum_{i \in N} v_i x_i \tag{14}$$

Subject to:

$$\sum_{i \in O_f} x_i \leq 1 \qquad \forall f \in M \tag{15}$$

$$x_i e_i \leq S_i \leq x_i (l_i - d_i) \qquad \forall i \in N \tag{16}$$

$$\sum_{i \in N_k} q_i x_i \leq Q \qquad \forall k \in P \tag{17}$$

$$\sum_{i \in N} (S_i + d_i + c_{ij}) y_{ij} \leq S_j \qquad \forall j \in N \tag{18}$$

$$\sum_{i \in Z} \sum_{j \in \bar{Z}} y_{ij} \geq 1 \qquad \forall Z \subset V \tag{19}$$

$$x_i \in \{0,1\} \qquad \forall i \in N \tag{20}$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{21}$$

$$S_i \geq 0 \qquad \forall i \in N \tag{22}$$

Constraint (15) ensures that every image is taken at most once for all the time-windows associated with it. Constraint (16) ensures that imaging starts and ends within one of the time-windows associated with that image. Constraint (17)

ensures that the capacity of each satellite's recording device is not exceeded. Constraint (18) ensures that the satellite has enough time to slew between any pair of time-windows used by it. It also ensures that satellites only move in the forward direction in time and that attempts to move backward in time or excessive slewing is impossible because of the large value that $c_{ij} = \max\left(t_{ij}, R\right)$ assumes since R is very large. The value of c_{ij}, together with constraints (16) and (19) usually serve to eliminate sub-tours, but sub-tours are in principle still possible in cases where there is a large overlap between time-windows with short imaging durations. Constraint (19) is the usual sub-tour elimination constraint that is retained here for that purpose.

3.5 Conclusion

The VRPMTW appears to provide a useful paradigm for analysing the nature of the SCoTWOP. Instead of viewing the SCoTWOP as a hybrid problem that incorporates characteristics of a number of standard problems, the VRPMTW makes it possible to relate the SCoTWOP to a single kind of standard problem. The SCoTWOP may in fact be seen as a variant of the dual of the VRPMTW. To see this consider the following:

- The VRPMTW seeks to minimise the total cost of delivery, given it has to deliver to all of a fixed number of customers within time-windows dictated by the customers.

- The dual of the VRPMTW would seek to maximise the number of customers serviced (value), given it has a fixed number of routes (i.e. number of vehicles) to perform the service and a fixed set of time-windows dictated by the routes.

If we apply the correspondence set out in Table 3-1, we would replace customers with images and vehicles with satellites in the statement about the dual of the VRPMTW above. The result would be a statement that would conform to the SCoTWOP. As explained in the next chapter, the similarities between the VRPMTW and the SCoTWOP assisted greatly in arriving at a very useful way of

encoding potential solutions to the SCoTWOP in a manner that was amenable to the three metaheuristics investigated.

4 SOLUTION METHODOLOGY

This chapter describes the nature of heuristics and metaheuristics in general before providing details of the three metaheuristics used in this study, namely:

- The tabu search metaheuristic;
- The simulated annealing metaheuristic, and;
- The genetic algorithm metaheuristic.

Aspects that are common to the implementation of all three techniques are discussed with particular reference to the way solutions are represented and evaluated for fitness; the common input dataset and the way feasibility of solutions are verified. The software components peculiar to each method are discussed before the experimental procedure is outlined.

4.1 Heuristics and metaheuristics for combinatorial problems

Two of the methods employed to solve the SCoTWOP involve the use of a metaheuristic to guide a simpler search algorithm. We used simulated annealing and tabu search. The third heuristic used, a genetic algorithm, belong to a class of heuristics that is biologically inspired and includes the ALife (artificial life) algorithms of particle swarm optimisation and ant colony optimisation.

Reeves *et al* (1993) define a heuristic as:

"…a technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is."

The local neighbourhood search heuristic starts with a sub-optimal solution and searches a defined neighbourhood of the solution for a better one. However, this strategy alone cannot be relied upon to find a global optimal solution since it may converge to a local optimum. This problem may be overcome by employing an additional strategy, or metaheuristic, to guide the local neighbourhood search heuristic in its search. Such strategies, employed by some metaheuristics to deal with the problem of heuristics converging to a local optimum, include the following:

- Enlarging the search neighbourhood;
- Restarting the search from different initial solutions, and;
- Allowing uphill moves. (We solve the SCoTWOP reformulated as a minimization problem.)

The metaheuristics tabu search and simulated annealing that have been evaluated for solving the SCoTWOP, both allow uphill moves. The third, a genetic algorithm, operates on a population of solutions and uses evolutionary strategies to breed consecutive generations of better (fitter) solutions. These metaheuristics are discussed in more detail below.

4.1.1 The tabu search heuristic

The tabu search metaheuristic seeks to avoid convergence to local optima by declaring a limited number of recently generated solutions off-limits (tabu) for a number of upcoming iterations. It does this by keeping track of recent solutions and/or their attributes in a dynamic memory called the "solution history" or "tabu list." Potential solutions in the current iteration that matches those in the tabu list are ignored, so redefining the search heighbourhood and forcing the

algorithm to diversify even if it means temporarily allowing a worse solution. The tabu list is updated after every iteration so that no solution remains on the list for more than a specified number of iterations (the "tabu tenure"). The process ends after a specified maximum number of iterations have been reached. If the focus is on solution attributes rather than complete solutions "aspiration criteria" may be used to override the tabu tenure rule so that improved solutions that share attributes with those on the tabu list may be allowed.

The implementation of tabu search for the SCoTWOP focuses on complete solutions and aspiration criteria are not used. The outline of the algorithm is as follows:

```
Generate an initial feasible solution
Set the number of iterations and the tabu tenure
Initialize the iteration counter and the logging vectors
Initialize the tabu list
Initially, let the optimal solution be equal to the
initial estimate (current solution)
WHILE number of iterations is less than the maximum:
      Update tabu list
      Find an improved feasible solution
      WHILE an improved feasible solution is not yet
      found:
            Generate a candidate solution in the
            neighbourhood through modification of the
            current solution
            Repair the solution if the same image is
            taken more than once
            Test solution for feasibility
            Determine the fitness of the new solution
      END WHILE
      Replace the current solution with the new solution
      Declare the new solution off limits for the tenure
      period
      Update the iteration counter
END WHILE
```

4.1.2 The simulated annealing heuristic

The simulated annealing metaheuristic seeks to avoid convergence to local optima by occasionally allowing moves to worse solutions. It mimics the behaviour of a metal that is slowly being cooled to promote crystal growth and relieve stresses. The probability of replacing the current solution with a worse neighbourhood solution depends on the difference δ between the solution values and on a global parameter T (called the temperature):

- If the new solution is worse than the current one, accept it with probability $p(\delta) = e^{\left(-\frac{\delta}{T}\right)}$

- If the new solution is a better solution that the current one, always accept it (meaning $p(\delta) = 1$)

When T is large the current solution changes almost randomly but favours better solutions as T goes to zero. The outline of the algorithm as implemented for the SCoTWOP is as follows:

```
Generate an initial feasible solution
Set the number of iterations and initial temperature
Initialize the iteration counter and the logging vectors
Initially, let the optimal solution be equal to the
initial estimate (current solution)
WHILE number of iterations is less than the maximum:
      Update tabu list
      Find an improved feasible solution
      WHILE an improved feasible solution is not yet
      found:
            Generate a candidate solution in the
            neighbourhood through modification of the
            current solution
            Repair the solution if the same image is
            taken more than once
            Test solution for feasibility
            Determine the fitness of the new solution
      END WHILE
      If the new solution is superior, use it to replace
      the current solution
      If the new solution is inferior use it to replace
      the current solution with a probability equal to
      e^((old-new)/T)
      Update the iteration counter
END WHILE
```

4.1.3 The genetic algorithm metaheuristic

Genetic algorithms mimic the concept of survival of the fittest in sequential populations of solutions that are produced by replacing selected couples by their offspring (sometimes parents and other members of a generation are retained in the next generation) and introducing mutations.

Solutions are encoded as genotypes and compared using a fitness function. During each generation a selection of solutions are paired and allowed to reproduce through a crossover operation. A small portion of the parent

population is also allowed to change through mutation to prevent getting stuck in local optima. The new generation generated through crossover and mutation replaces some of the solutions in the previous generation that exhibit inferior fitness. The process terminates after a prescribed number of generations or some other stopping criteria. The outline of the algorithm as implemented for the SCoTWOP is as follows:

```
Set the number of iterations or other stopping criteria
Specify the fitness function
Select criteria for parent selection
Select crossover methods
Select criteria for mutation
Generate an initial population of feasible solutions
WHILE the stopping criteria have not been met:
      WHILE the new generation of solutions is still
      incomplete
            Select parent solutions
            Apply crossover method
            Apply mutation criteria and method
            Test solutions for feasibility
            Determine the fitness of the new solutions
            If the new solutions are superior, use it to
            replace the same number of inferior solutions
      END WHILE
      Update the iteration counter
END WHILE
```

4.2 Common aspects of applying the three metaheuristics

There are several common aspects to the application of the three metaheuristics to the SCoTWOP. It was convenient to represent solutions to the SCoTWOP in exactly the same way for all three algorithms, the feasibility of solutions were evaluated in the same way, and the relative merit or fitness of individual solutions were evaluated in the same way. These aspects are discussed in more detail below.

4.2.1 Solution representation

Given the similarities between the SCoTWOP and the VRPMTW it seems natural to encode the SCoTWOP in a similar way as is common for the VRPMTW. In that case a solution is typically encoded as a bit string in which a 1 would correspond to inclusion of the image (customer) in an image sequence

(route) and a 0 indicating the contrary. The length of the bit string would correspond to the number of images.

However, a problem arises in the case of the SCoTWOP where an image may have more than one imaging opportunity (time-window) in a specific planning period and where time-windows are associated with the satellite (vehicle) rather than the customer (image). In this case there is no longer a one-to-one correspondence between images and the bits in the bit string. This makes it extremely difficult to represent a solution as a genotype that can be easily handled by the genetic algorithm and can also be easily translated into a set of image sequences. The notion that the SCoTWOP is related to the dual of the VRPMTW as pointed out in Section 3.5, offers a way out of this problem. The solution of the dual of the VRPMTW could be represented by a bit string, the length of which corresponds to the number of time-windows. An occupied time-window can then be denoted by a 1 and a 0 would similarly denote an unused time-window. In this way a one-to-one correspondence can be retained making genotype transcription much easier.

A solution to the SCoTWOP, termed: an "activation of time-windows" is therefore represented as a vector $\bar{x}$ of N binary variables, $N-2$ of which correspond to image time-windows and the other two corresponding to the time-windows for ground station access. If a time-window is included in the planning period the corresponding variable would take the value 1 and 0 otherwise. The solution vector $\bar{x}$ is mapped to a vector containing the integers 2 to $N-1$ in ascending order which corresponds in turn to the indexes of a row consisting of time-window start times in chronological order. In other words, a solution vector of say, $\bar{x} = (0,1,1,0,0,1,\ldots,1,0)$, is translated through element-by-element multiplication with the vector $(2,3,4,\ldots,249)$, to a vector of active time-window indices: $(0,3,4,0,0,7,\ldots,246,0,248,0)$, meaning that time-windows 3, 4, 7,…, 246, 248 have been included in the solution.

4.2.2 Testing for feasibility

Feasible solutions to the SCoTWOP must of course meet all the constraints: that of restricted on-board memory, visibility, sequence, non-interference, and singularity. Each potential new solution generated by the algorithms therefore needed to be evaluated for its feasibility before its relative merit (fitness) could be evaluated. The similarities with the VRPMTW made it possible to use existing MatLab® code that had originally been developed to evaluate the feasibility of solutions for VRP and VRPTW problems. The utility, `rteTC.m`, that forms part of the MatLog Toolbox developed by Kay (2004), was adapted for the SCoTWOP application to test image sequences for feasibility. For VRP type problems, rteTC.m calculates the total cost of a route ensures that time-window constraints are not violated and calculates the starting times for each loading operation within the time-window. The utility returns the following exit flag indicating either that the route is feasible or if not, why it is infeasible:

```
XFlg    = exit flag
        = 1, if route is feasible
        = -1, if infeasible due to capacity
        = -2, if infeasible due to time-windows
        = -3, if infeasible due to custom feasibility function
```

"Custom feasibility function" refers to a provision in the software for the user to define an additional feasibility function that implements a constraint that is unique to the problem of interest, in addition to the capacity and time-window constraints. This option was not used for the SCoTWOP although it was contemplated to use it for the singularity constraint. In the end it was thought to be more efficient to either repair solutions that violated the singularity constraint, before submitting it to `rteTC.m`, or if that was not possible, to declare such a solution as infeasible and not submitting it to `rteTC.m` at all. More clarification on this approach is provided in Section 4.2.3.

Although `rteTC.m` is suitable for VRPMTW problems, it had to be slightly adapted and used slightly differently for solving the SCoTWOP. In the case of the SCoTWOP, a "route" is, of course, a sequence of time-windows for a particular satellite. The VRPMTW capacity constraint is the same as the on-

board memory constraint of the SCoTWOP. The notion of time-window feasibility in the case of the VRPMTW takes care of both the non-interference and visibility constraints for the SCoTWOP. The VRPMTW seeks to minimise total cost (including vehicle cost, travel cost, and waiting cost) of a network of routes subject to a constraint on the maximal allowable total cost per route. This constraint is required to prevent the inclusion of impractically long routes in the network and is represented by the `maxTC` parameter in `rteTC.m`. In the SCoTWOP, the number of routes (satellites) is a given and the focus is on maximising the number of customers (images) per route. This problem was resolved by setting `maxTC` to a value that will always allow feasible routes to pass the cost constraint test but which will cause infeasible routes to still be rejected. This provides a way to enforce the sequencing constraint in the SCoTWOP: By setting maxTC at a value that allows large finite values but not infinitely large values, any sequence of images that would require the satellite to move "backwards", would be excluded since the infinite cost of such an arc would cause `maxTC` to be exceeded.

This leaves the singularity constraint to be dealt with. In cases where a violation of this constraint was due to a crossover operation, this was overcome by simply "repairing" any solution that violated the singularity constraint before submitting it for feasibility evaluation by `rteTC.m`. This was achieved by eliminating all but one of any duplicate time-windows for a particular image from the proposed sequence. The specific time-window to be eliminated from a set of duplicates was chosen at random. In cases where some diversity would have been lost if a solution was repaired, such as when it was the result of a mutation operation, the solution was declared infeasible and penalized in accordance with the scheme outlined in Section 4.2.3.

The result of the above is that `rteTC.m` could be used to enforce all but the singularity constraint for the SCoTWOP. It was therefore not necessary to create a custom feasibility function for `rteTC.m` to account for any of the constraints. A listing of the modified `rteTC.m` is included in Appendix C.

In several of the experiments conducted with the genetic algorithm, some of the constraints were artificially relaxed to evaluate their impact on algorithm performance. This was achieved by appropriate adjustments to the input data rather than changing the code. This aspect is discussed in more detail in Section 4.2.4 below.

4.2.3 Determining the fitness of solutions

For all three algorithms the fitness of a feasible solution is determined by calculating the inner product: $-(\overline{v} \cdot \overline{x})$, where $\overline{v}$ is the vector consisting of the values of the images. The fitness function thus represents the negative value of the objective function of the SCoTWOP. This is necessary because the software has been designed to solve minimization problems as default. In the case of the SCoTWOP, we therefore minimize the negative of the objective function. For the genetic algorithm, the fitness function was encoded in the utility `Satfitness.m`.

To take full advantage of the genetic algorithm's ability to exploit inferior solutions to maintain genetic diversity, infeasible solutions were retained in the population of solutions but penalized by artificially changing the value of the fitness functions of infeasible solutions so that their fitness became worse. Since the value of the fitness function is calculated as the negative of the objective function value, infeasible functions were penalized by either declaring the fitness function value to be infinitely positive or increasing the (negative) fitness value by adding a fixed percentage of the (positive) value of the objective function. `Satfitness.m` evaluates solutions first for violation of the singularity constraint before submitting for further feasibility testing to `rteTC.m`. Solutions that violate the singularity constraint are immediately penalized and not submitted for further evaluation. All solutions returned by `rteTC.m` with an exit flag indicating violation of the other constraints are similarly penalized.

4.2.4 Input data

The SCoTWOP input data for the evaluation of the three algorithms comprises four groups of six data sets, one each for the case of one through six satellites, for a total of 24. The data in the datasets provides for the same, fixed number of 250 imaging time-windows that has to be serviced regardless of the number of satellites in the constellation. This correspond to the situation where the number of 250 time-windows can be serviced by one satellite over a period of days, by two satellites in about half that time, etcetera, so that imaging rate increases with the number of satellites until all 250 time-windows can be serviced by six satellites in a matter of hours. Keeping the number of time-windows constant is crucial for investigating the comparative merit of the different algorithms and the relative impact of the different constraints. One could, of course, increase the number of time-windows to be serviced as the number of satellites increases but this would obscure the effect of the singularity constraint that one expects to be more exacting as the number of satellites grows. Keeping the number of time-windows to be serviced at 250 therefore not only is realistic in terms of what is expected of constellations, but also enables comparison.

The four groups of data sets, denoted A through D, each contains data for 250 time-windows, the nature of which is such that the constraints on the SCoTWOP described in a specific group, become more restrictive than the constraints described in the previous group. The input data contains some fields that are associated with the constraints. Whether a particular constraint is active or not, depends on the value assigned to such a field. For example, by adjusting the available on-board memory to a value much larger than the sum of that required by all images under consideration, the constraint on on-board memory is effectively removed without having to change the algorithm computer code. By reducing the value assigned to the on-board memory field, the constraint can be introduced. In this way, all constraints are always considered, but the specific value assigned to a limiting parameter determines whether the constraint is implemented or not. This principle is implemented as follows for the four constraint groups A through D:

- Group A comprises six datasets, in which only the visibility and sequencing constraints are active (see Section 3.3). The data is such that all other constraints are relaxed. In other words, on-board memory is abundant, time-windows do not overlap (no interference constraint), and there is only one imaging opportunity per image (no singularity constraint).

- Group B data sets have the same constraints as Group A but also add a limit on on-board memory while the non-interference and singularity constraints are still relaxed.

- Group C data sets add the non-interference constraint to that of Group B, i.e. time-windows do overlap in many cases while the singularity constraint is still relaxed.

- Group D data sets represents the SCoTWOP in its complete form, adding to the Group C constraints the singularity constraint, which means that there may be several opportunities for the same image, only one of which must be selected.

The four groups of datasets were used as follows:

- Dataset D6, representing the most difficult problem in that it allows for 6 satellites and the most stringent set of constraints, was used to compare the relative performance of the three algorithms;

- All four groups of datasets were then used together with the genetic algorithm to investigate the influence of the types of constraints and the number of satellites on algorithm performance, and;

- Finally, Dataset D6 was once again employed to investigate the influence of changing certain genetic algorithm parameters.

Each dataset contains data that is typical of satellites in sun-synchronous orbit at an altitude of approximately 600 km with a capability to slew rapidly at about one degree per second. The satellites are assumed identical in capabilities but phased around the same orbit plane. The datasets were prepared in the form of

Microsoft Excel® spreadsheets, a format that is readable by MatLab®. Each dataset provides values for the following image attributes:

- `n` is the number of time-windows that have to be considered for scheduling during a planning period comprising a number of orbits by all satellites. It includes the time-window of initial ground station access. In all cases the value of `n` is 250;

- `lookangle` is the bore-sight angle of the imager, relative to nadir, associated with the image to which the time-window belongs. The value of `lookangle` in the datasets varies between -30 and +30 degrees;

- `ld` is the image duration associated with the image to which the time-window belongs. The value of `ld` in the datasets varies between 5 and 15 seconds;

- `TW` is a pair of values, denoting the start and end times of the time-window for each imaging opportunity. The duration of the time-widows in the datasets varies between 30 and 60 seconds;

- `q` is the memory requirement for the image to which the time-window belong. The value of `q` is 10 times that of the image duration `ld` so that it varies between 50 and 150;

- `Q` is the total on-board memory capacity of the solid state recorder on the satellite. The value of `Q` is set as the product of a multiplier and the average sum of the image memory requirement of the images that can theoretically be acquired by a specific satellite. After some experimentation, the value of this multiplier was set at 1.2 for the datasets belonging to Group A. This amounts to a 20 % overcapacity in available memory which was found to be sufficient to ensure that the constraint is relaxed. (Recall that Group A models the SCoTWOP with only the visibility and sequencing constraint – all other constraints are relaxed.) In all other cases the multiplier was set at 0.7 after some experimentation meaning that on average the memory is sufficient for only 70% of the images. This value was found to be sufficient to ensure that the constraint is active in all cases;

- `Value` is the value of the image associated with the time-window during the current planning period. `Value` varies between 100 and 200 times the value of the image duration `ld,` so that it varies between 500 and 3000;

- `Satno` is an index associated with the specific satellite associated with the time-window. The maximum value of `Satno` corresponds to the number of satellites in the constellation which varied from 1 to 6, and;

- `viewno` is the number of the earliest time-window where this particular image can be captured and can be any number between 2 and 249 since the index 1 denotes the ground station.

One of the 24 input data sets used for evaluating the algorithm is included in Appendix B

4.2.5 Reading and interpreting the input data

The utility `DataReader.m` was created in MatLab® to read the input data into the MatLab® workspace memory. It then proceeds to calculate the setup time between all possible image pairs as the difference between their slew angles. It also allocates a travelling time for each pair of time-windows in the network; time-windows that occur prior to any given time-window are allocated an infinite travel time, while those that are chronologically later are allocated a travel time of zero. In this way the highly asymmetric nature of the network is accommodated. The cost of each arc between time-windows is then calculated as the maximum of the setup time and travelling time for the arc. A listing of `DataReader.m` is included in Appendix C.

4.3 Implementation of the algorithms in MatLab®

4.3.1 Implementation of the tabu search algorithm

The tabu search Algorithm for the SSP is codified as the MatLab M-File `TabuSat.m` (see Appendix C) that is based on code originally created by Aurdal (2003)

The utility `DataReader.m` is used to read the input data into the workspace memory and to calculate the setup time between all possible image pairs.

`TabuSat.m` first generates a random solution vector $\bar{x}$ of N ones and zeros, $N-2$ of which correspond to image time-windows and the other two corresponding to the time-windows for ground station access. If a time-window is included in the planning period the corresponding variable would take the value 1 and 0 otherwise. It then proceeds to repair the vector by making sure that no images are taken twice. The repaired vector is then split into portions that correspond to the different satellites and formatted as image sequences. The set of image sequences is submitted to the `rteTC.m` utility and checked for feasibility. If any of the sequences are infeasible, the process outlined above starts anew.

Once the set of image sequences is feasible the tabu search is executed as outlined in Section 4.1.1 above. The feasible solution is subjected to a neighbourhood search in which one of the vector elements is randomly selected and changed to a 1, if it was 0, or to a 0 if it was 1. One such neighbourhood solution is generated at a time and this neighbourhood solution is then repaired if necessary and checked for feasibility by invoking `rteTC.m.` If the neighbourhood solution is feasible, its fitness function is then evaluated as the negative value of the objective function of the SCoTWOP since the algorithm was encoded to solve minimization problems. Improved solutions found in this way are put on the tabu list for tenure of 5 iterations. The process is repeated for a total of 100 iterations.

4.3.2 Implementation of the simulated annealing algorithm

The simulated annealing Algorithm for the SSP is codified as the MatLab M-File `SimulanSat.m` (see Appendix C) that is based on code created by Aurdal (2003)

The utility `DataReader.m` is used to read the input data into the workspace memory and to calculate the setup time between all possible image pairs.

The program is structured similar to `TabuSat.m` for the tabu search algorithm and operates in the same way apart from applying the simulated annealing algorithm as described above in Section 4.1.2. An initial solution vector $\bar{x}$ is generated randomly and repaired before being split into image sequences for the different satellites. The set of image sequences is submitted to the `rteTC.m` utility and checked for feasibility. If any of the sequences are infeasible, the process outlined above is repeated until the first feasible solution is found.

The feasible solution is then subjected to a neighbourhood search in which one of the vector elements is randomly selected and changed from to a 1 if it was 0 or to a 0 if it was 1. This neighbourhood solution is then repaired if necessary and checked for feasibility by invoking `rteTC.m.` If the solution is feasible, its fitness is then evaluated as the negative value of the objective function of the SCoTWOP. The initial temperature for the algorithm was set at a value of 100 and is decreased 0.1% per iteration until the maximum number of iterations (typically 1000) is reached.

4.3.3 Implementation of the genetic algorithm

4.3.3.1 General

As is the case for the tabu search and simulated annealing algorithms, the genetic algorithm also operates on solution vectors $\bar{x}$ consisting of $N-2$ ones and zeros. In genetic algorithm terminology, the vector $\bar{x}$ can be called a *genome* and its one or zero entry of the vector can be called a *gene*. Rather than evaluating a single solution at a time, the genetic algorithm operates on a

population of individual solution vectors simultaneously. For the SCoTWOP, this population of solutions comprised 20 individual solutions. The initial population of 20 solutions was generated through the utility `SequenceBuilder.m`, described in Section 4.3.3.4 below. The genetic algorithm uses a fitness function to determine the relative merit of each solution in the population. In the case of the SCoTWOP, the fitness function used was the negative of the objective function. The solutions are subsequently ranked in accordance with their fitness function values. Some of the solutions, with an emphasis on higher ranking solutions, are then paired and allowed to reproduce through a crossover operation. Some other solutions are altered through a mutation operation. The solutions resulting from the crossover and mutation operations are then allowed to replace lower ranked solutions in the population so that the number of solutions in the population remains constant. This is repeated until one of various stopping criteria is reached. Stopping criteria used for the SCoTWOP were the total number of generations of the solution population (set at 100), the number of generations without fitness function improvement (set at 50) and time period in which there was no fitness function improvement (set at 20 minutes).

4.3.3.2 Solving the SCoTWOP with the MatLab® Genetic Algorithm and Direct Search Toolbox

Implementation of the genetic algorithm for solving the SSP was achieved through the use of two off-the-shelf "toolboxes" created for the MatLab® programming language. The first toolbox is the Genetic Algorithm and Direct Search Toolbox for by MathWorks the creators of MatLab®. This toolbox works in conjunction with the second, the more general Optimization Toolbox for MatLab®. The Genetic Algorithm and Search Toolbox allows for the easy specification of combinatorial problems and the implementation of various options to customize the algorithm to the problem peculiarities. Those of importance to this application are as follows:

- **The Population Type Option** provides for representation of the genome (solution vector $\bar{x}$) as a *Double Vector* (default) in which each gene (vector

element) is a real number of double precision (i.e. 64 bits), *Bit String* in which each gene is either the whole numbers 1 or 0, or a *Custom* representation of the users own choice.

- **The Fitness Scaling Option** specifies the function that performs the scaling or relative merit of individual solutions in the population. The default fitness scaling function, *Rank*, assigns merit based on the rank of an individual in its position in the sorted scores. *Proportional* scaling makes the merit of an individual proportional to its raw fitness score. *Top* scaling gives a specified number of the top individuals equal merit. *Shift Linear* scaling adjusts the raw scores so that the merit of the fittest individual is equal to a constant multiplied by the average score. *Custom* provides for the user to create a customized scaling function.

- **Selection Options** specify how the genetic algorithm chooses parents for the next generation. The default selection function, *Stochastic Uniform*, lays out a line in which each parent corresponds to a section of the line of length proportional to its scaled value. The algorithm moves along the line in steps of equal size. At each step, the algorithm chooses a parent from the section it lands on. *Remainder* selection chooses parents deterministically from the integer part of each individual's scaled value and then uses roulette selection on the remaining fractional part. *Uniform* selection chooses parents using the expectations and number of parents. *Roulette* selection chooses parents by simulating a roulette wheel, in which the area of the section of the wheel corresponding to an individual is proportional to the individual's fitness. *Tournament* selection chooses each parent by choosing a fixed number of solutions at random and then choosing the best individual out of that set to be a parent.

- **Reproduction Options** specify how the genetic algorithm creates children for the next generation. *Elite Count* specifies the number of individuals that are guaranteed to survive to the next generation. The default value is 2. *Crossover Fraction* specifies the fraction of the next generation, other than elite children, that are produced by crossover. The default value is 0.8.

- **Mutation Options** specify how the genetic algorithm makes small random changes in the individuals in the population to create mutation children. The default mutation function, *Gaussian*, adds a random number taken from a Gaussian distribution with mean 0 to each entry of the parent vector. The standard deviation of this distribution is determined by the parameters *Scale* and *Shrink*, which respectively specifies the initial standard deviation and the rate at which it changes over generations. If the **Population Type** option is set at *Bit String* as was the case for the SCoTWOP, *Scale* automatically assumes the value 1 and *Shrink* the value 0. If the result of adding the random number falls outside the permitted range of the gene, the value is rounded to the nearest range limit, i.e. a negative value becomes 0, and so would 0.3, while values of 0.65 or 2.5 become 1. *Uniform* mutation is a two-step process. First, the algorithm selects a fraction of the vector entries of an individual for mutation. In the second step, the algorithm replaces each selected entry by a random number selected uniformly from the range for that entry. Since this mutation results in real numbers, it cannot be used for the *Bit String* **Population Type** as is the case for the SCoTWOP. *Custom* enables the user to write your own mutation function.

- **Crossover Options** specify how the genetic algorithm combines two individuals, or parents, to form a crossover child for the next generation. The default crossover function *Scattered*, creates a random binary vector and selects the genes where the vector has a 1 from the first parent, and the genes where the vector has a 0 from the second parent, and combines the genes to form the child. In other words: a vector consisting of random 1 and 0 entries is created first and its entries are then used as an index to select genes from two parent solutions A and B. If a specific entry of the index vector is 1, a gene for the new child is selected from the corresponding entry of parent A. If a specific entry of the index vector is 0, the gene for the new child is selected from the corresponding entry of parent B. *Single Point* chooses a random integer n between 1 and the number of variables or genes `N` and then selects vector entries numbered less than or equal to n from the first parent. Vector entries numbered greater than n are selected from the second parent. It

then concatenates these entries to form a child vector. *Two Point* selects two random integers *m* and *n* (*m*<*n*) between 1 and the number of variables *N*. Vector entries numbered less than or equal to m are selected from the first parent, those numbered from *m+1* to *n*, inclusive, from the second parent and entries numbered greater than *n* from the first parent. The algorithm then concatenates these selections to form a single gene. *Intermediate* creates children by taking a weighted average of the parents. *Heuristic* returns a child that lies on the line containing the two parents, a small distance away from the parent with the better fitness value in the direction away from the parent with the worse fitness value. *Custom* enables one to write a special crossover function. This option was used for the SCoTWOP since it was necessary to repair child solutions that violated the singularity constraint as discussed below and elsewhere.

- **Stopping Criteria Options** determine what causes the algorithm to terminate. The *Generations* stopping criterion specifies the maximum number of iterations the genetic algorithm will perform. The default is 100. *Time Limits* specifies the maximum time in seconds the genetic algorithm runs before stopping. *Fitness Limit* provides for the algorithm to stop if the best fitness value is less than or equal to the value of a known fitness limit. This is a useful option if the lower bound of a function to be minimized is known (for maximization problems, this would correspond to the negative value of a known upper bound since the toolbox only works for minimization problems as a default). *Stall Generations* lets the algorithm stop if there is no improvement in the best fitness value for a specified number of generations. Stall Time causes the algorithm to stop if there is no improvement in the best fitness value for a specified interval of time in seconds.

4.3.3.3 Implementation options

Table 4-1 below contains the values of the various MatLab® GA Toolbox implementation options used, as defined in Section 4.3.3.2 above. In most cases these correspond to the default values for the particular option. These values were used for comparing the genetic algorithm with the other two algorithms as

well as for exploring the effect of different constraints and number of satellites on the performance of the genetic algorithm.

Table 4-1: Implementation Options Selected for the MatLab® Genetic Algorithm and Direct Search Toolbox

Implementation Option	Value
Population Size	20
The Population Type	Bit String
Initial Population Creation Function	`SequenceBuilder.m`,
Fitness Function	`Satfitness.m` (calculates the negative of the objective function: $-(\overline{v} \cdot \overline{x})$)
Fitness Scaling	Rank
Selection	Stochastic Uniform
Reproduction	Elite Count = 2. Crossover Fraction = 0.8.
Mutation	Gaussian
Crossover	Custom: `crossoverspecial2.m`
Stopping Criteria	Generations = 100 Time Limits = Infinity Fitness Limit= - Infinity Stall Generations = 50 Stall Time limit = 20

4.3.3.4 Creating the initial population of solutions

The Genetic Algorithm Toolbox provides the option of specifying a utility to create an initial population of feasible solutions. The utility `SequenceBuilder.m` was created for this purpose. This program first generates a random vector consisting of *n-1* ones and zeros where *n* is the number of time-windows as defined above. The first time-window, corresponding to the initial ground station contact, is not included in the vector generated. A one in a particular position in the vector means that that time-window is used. It then proceeds to repair the vector by making sure that no images are taken more than once. The repaired vector is then split into portions that correspond to the different satellites and formatted as image sequences. The set of image sequences is submitted to the `rteTC.m` utility (see Section 4.2.2 below) and checked for executability, i.e. to see whether the satellite can in fact capture this sequence of images. If any of the sequences are not executable, the process outlined above starts anew. The process continues until an initial population of 20 feasible solutions has been generated. A listing of `SequenceBuilder.m` is included in Appendix C.

4.3.3.5 The custom crossover function

In order to maintain a population of feasible solutions it was necessary to create a custom crossover function, called `crossoverspecial2.m` , to repair any children resulting from a crossover operation that violates the singularity constraints. Based on the default crossover function *Scattered*, provided in the toolbox, this function tests for violations of the singularity constraint after crossover and enacts repairs to ensure that a proposed solution does not contain duplicate images before submitting it to the `rteTC.m` utility for further determination of its feasibility. The code for `crossoverspecial2.m` is included in Appendix C.

4.3.3.6 Running the genetic algorithm

The utility `GeneticSat.m` was created to automate the initialization and running of the genetic algorithm. The utility was created using an automated feature of the toolbox and resulting listing of `GeneticSat.m` is included in Appendix C. The fitness function for the SCoTWOP was encoded in the utility `Satfitness.m`, which is called by `GeneticSat.m`. This utility also first tests for violations of the singularity constraint to ascertain that a proposed solution does not contain duplicate images before submitting it to the `rteTC.m` utility for further determination of its feasibility. If a positive result is returned from `rteTC.m`, `Satfitness.m` proceeds to calculate the value of the objective function associated with the proposed solution. A listing of `Satfitness.m` is also included in Appendix C.

4.4 Computing resources

The computing resources used for evaluating the efficacy of the genetic algorithm for the SCoTWOP comprised the following:

- **Hardware:** Desktop Personal Computer with AMD Sempron® processor running at a clock speed of 1.8 GHz and having 384 MB of RAM.
- **Operating System**: Microsoft Windows XP Home Edition® Version 2002 Service Pack 2.
- **Software:**
    - MathWorks MatLab® Student Version 7.0.1.15 (Rev 14) Service Pack 1, 13-Sep-2004.
    - MathWorks MatLab® Genetic Algorithm and Direct Search Toolbox Version 1.0.2 (R14SP1) 05-Sep-2004.
    - MathWorks MatLab® Optimization Toolbox Version 3.0.1 (R14SP1) 05-Sep-2004.

4.5 Experimental procedure

Three sets of experiments were conducted using the software tools and input datasets described above:

- The first set of experiments was aimed the comparing the performance of the three metaheuristics in solving the SCoTWOP. The experiments consisted of using each of the three heuristics in turn to solve the SCoTWOP for the Group D dataset with six satellites, Dataset D6. A minimum of 10 runs were made per heuristic (30 in total);

- The second set of experiments was aimed at determining the impact of constraint type on the time taken to find a solution to the SCoTWOP when using a Genetic Algorithm. The SCoTWOP was solved for each of the 24 sets of input data. A minimum of 10 runs were made per data set (240 in total), and;

- The third set of experiments focused on determining the efficacy of using different reproductive options (other than the default) for the Genetic Algorithm. Several runs were made for different values of the *Elite Count* and *Crossover Fraction* parameters, while two alternative cross-over techniques were also tried. In each case the Group D dataset with six satellites Dataset D6 was used.

The last two sets of experiments were conducted following on the success of the genetic algorithm in the first set of experiments. With the exception of the last step below, each computer run in the experiments used the following simple procedure:

- Run `Readdata.m` to first clear all variables and then read the input data into the MatLab® Workspace.

- Run `GeneticSat.m (`or `TabuSat.m` or `SimulanSat.m)` to run the applicable algorithm, using the `tic-toc` function to measure CPU

time. In the case of the genetic algorithm the option to plot a graph of the algorithm's progression was enabled.

- When the algorithm terminates, verify that the final solution is still feasible (all XFlg = 1) before recording the total number of images scheduled and CPU time expended.

- Save the Workspace and graph (in the case of the genetic algorithm only).

The results obtained in this way are presented and discussed in the next chapter.

5 RESULTS AND DISCUSSION

5.1 Comparing the performance of the three metaheuristics

5.1.1 Results of the performance comparison experiments

This set of experiments was aimed at comparing the performance of the three metaheuristics in solving the SCoTWOP. The experiments consisted of using each of the three heuristics in turn to solve the SCoTWOP for the Group D dataset with six satellites. Each heuristic was evaluated over ten computer runs, and for each run the processing time, number of time-windows scheduled, and objective function value were recorded. The results thus obtained are recorded in Table 5-1 and graphically displayed in Figure 5-1 and Figure 5-2.

Table 5-1: Comparison of the performance of the three metaheuristics for the same set of input data over 10 computer runs each

Run #	Tabu Search			Simulated Annealing			Genetic Algorithm		
	CPU Time (Sec)	No of Time-Windows	Objective Function Value	CPU Time (Sec)	No of Time-Windows	Objective Function Value	CPU Time (Sec)	No of Time-Windows	Objective Function Value
1	1973.7	121	221524	170.7	127	200421	4663.3	175	251208
2	2035.3	130	224663	185.3	136	201720	2825.3	176	246290
3	2272.0	129	227120	129.8	137	204751	5970.3	183	251428
4	1821.2	147	217321	170.7	121	192955	5361.9	174	246015
5	1747.7	131	231496	239.8	122	214228	2366.8	174	250426
6	1850.4	134	220770	202.8	134	205765	2496.4	173	250781
7	1766.2	135	217299	340.0	127	206371	3050.7	173	246589
8	2377.5	133	221757	125.8	136	208325	1785.0	171	250964
9	2333.1	139	223588	119.0	140	198598	6705.3	173	252096
10	1652.2	129	220936	212.8	131	196984	1628.8	177	249986
Mean	1982.9	133	222647	189.7	131	203012	3685.4	175	249578

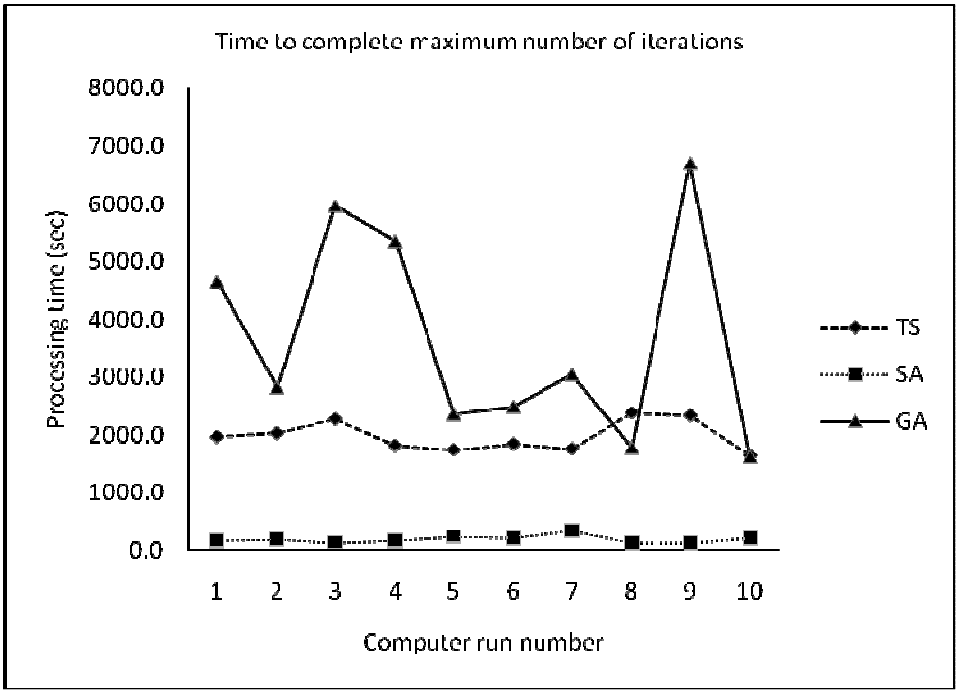

Figure 5-1: The computer processing time needed to complete the maximum number of iterations for each of the three metaheuristics recorded over 10 computer runs

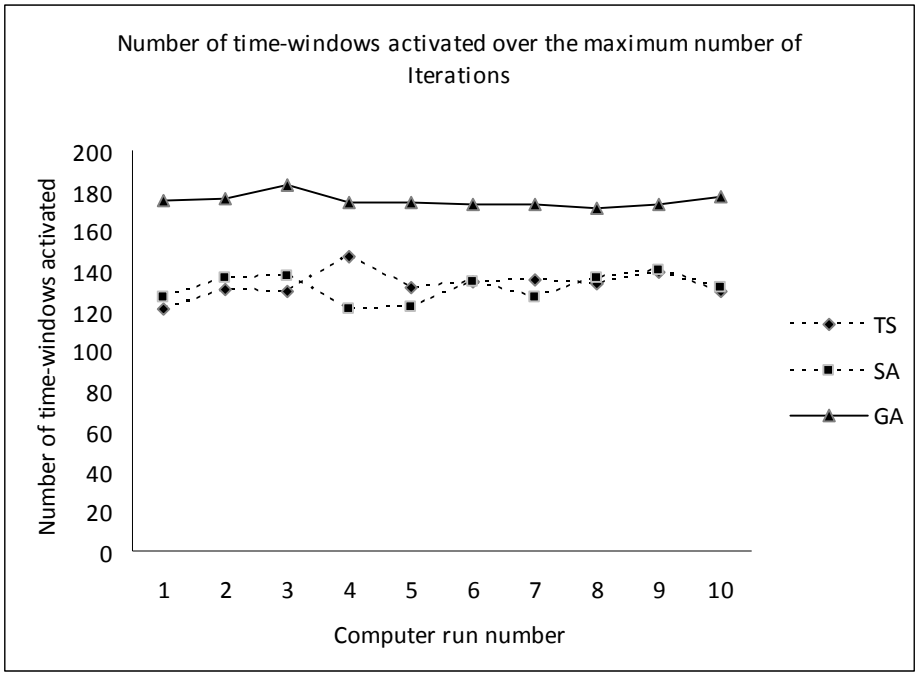

Figure 5-2: The number of time-windows activated over the maximum number of iterations for each of the three metaheuristics recorded over 10 computer runs

5.1.2 Conclusions on the performance comparison experiments

When evaluating the computing time needed to complete the maximum number of iterations, it should be kept in mind that the number of iterations was not the same for all three methods. The default number of iterations was used in all cases: 100 for tabu search, 1000 for simulated annealing and 100 generations for the genetic algorithm. Sensitivity to the number of iterations was investigated by performing runs using up to 10 times the default number of maximum iterations. While computing time increased commensurately, no appreciable difference was found in the number of time-windows activated in each case, indicating that the default number of maximum iterations was sufficient for each of the methods to converge to a solution.

The differences in the maximum number of iterations however pales into insignificance when it is appreciated that the simulated annealing algorithm activates more or less the same number of time-windows as the tabu search algorithm in as little as one tenth of the computing time, even though it completes ten times the number of iterations. If the time taken per iteration is taken as a measure of performance, the simulated annealing algorithm can therefore be said to be roughly 100 times faster than the tabu search algorithm. The genetic algorithm performs even worse in terms of computing time. Not only is there considerable variation in computing time, but in the worst case, the algorithm takes almost three times as long as even the tabu search algorithm.

Although computing time is an important measure of performance, it need not be the determining factor in selecting an algorithm. A poor performance in terms of computing time can potentially be improved through more efficient software design and implementation, faster processors and other computing resources, or techniques such as parallel computing. A more important measure of performance is the number of time-windows activated since this reflects the performance of the constellation of satellites (representing a huge investment), and in terms of this indicator, the simulated annealing and tabu search algorithm perform equally well, both activating on average 53 % of the time-windows. However, the genetic algorithm, activates 70 % of time-windows on average, and does so consistently, even though computing times vary considerably. Since computing time is not as important as number of time-windows activated, and the genetic algorithm activates on average about 17 % more time-windows than the other two algorithms, it was decided to limit further experiments on the other heuristics to explore the performance of the genetic algorithm in more detail. The results of these experiments are presented in the sections that follow.

5.2 Determining the impact of constraint type on performance of the genetic algorithm

5.2.1 Results overview

The raw results obtained through the 240 computer runs are summarized for each of the constraint groups A through D in Table 5-2 through Table 5-5 that follow below. The tables show the computing time, number of time-windows activated, and final and best fitness (objective function) value for each of the computer runs. In the paragraphs that follow the results obtained are discussed from two orthogonal perspectives.

Table 5-2: Results obtained for Constraint Group A

Run #	1 Satellite			2 Satellites			3 Satellites			4 Satellites			5 Satellites			6 Satellites		
	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value
1	110.3	219	322292	130.6	215	318894	129.4	212	317198	130.3	219	328135	128.9	215	338737	129.7	210	318962
2	111.3	215	335802	112.1	210	324715	131.9	203	306646	125.0	219	321174	151.0	216	341057	155.4	216	321773
3	115.2	205	330739	129.7	207	317861	114.3	202	312931	132.7	220	326400	143.8	217	335236	122.3	221	324363
4	112.5	221	326052	125.9	212	314606	116.3	210	315075	143.3	210	318192	126.9	204	326978	130.8	218	321679
5	106.2	216	337380	124.3	221	321844	123.5	216	317797	124.3	210	313903	130.4	216	338720	130.1	220	324249
6	117.7	204	333758	116.1	214	331974	152.3	213	318056	128.0	212	314122	126.4	216	339039	140.0	214	316781
7	109.9	216	319923	156.3	218	314059	131.1	219	326935	120.9	215	324754	117.7	212	336846	122.4	201	310972
8	109.5	209	327315	120.1	224	330593	116.0	210	315885	129.3	207	312344	119.7	205	324854	133.0	215	315416
9	111.8	210	322292	118.8	199	332782	118.9	212	316104	121.7	208	312209	95.3	215	338737	152.5	217	320491
10	129.6	209	324597	139.9	213	301629	118.4	213	316672	125.4	213	318481	117.7	217	330558	160.3	218	323449
Mean	113.4	212	328015	127.4	213	320896	125.2	211	316330	128.1	213	318971	125.8	213	335076	137.7	215	319814

Table 5-3: Results obtained for Constraint Group B

Run #	1 Satellite			2 Satellites			3 Satellites			4 Satellites			5 Satellites			6 Satellites		
	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value
1	159.5	182	296682	120.3	182	300516	112.6	173	293367	118.8	177	279484	124.1	179	274661	193.1	173	266608
2	115.8	177	299719	124.9	179	298772	113.1	174	279859	127.1	179	280882	241.9	178	262030	169.7	178	259767
3	113.0	183	288547	124.4	185	301523	109.5	171	284388	124.1	180	280422	240.9	176	267329	164.1	179	266980
4	97.9	176	288409	138.7	175	302406	109.2	176	285797	130.5	174	275749	191.0	177	277447	125.6	175	252309
5	157.8	178	293490	186.8	183	301660	109.7	175	283691	131.0	177	267726	130.6	181	269559	133.5	175	266807
6	106.5	174	285284	115.7	185	299476	120.0	176	288897	225.8	174	277923	247.2	181	283971	143.6	178	259909
7	100.8	184	291693	106.5	183	305262	139.8	178	291359	228.9	181	275868	186.6	181	278619	125.1	172	253771
8	110.8	180	289487	123.1	181	302418	116.9	178	294021	234.2	179	278350	186.8	180	272898	141.3	176	262172
9	98.8	177	286553	116.7	185	299513	133.5	175	293761	231.0	180	280823	182.3	176	275304	119.0	173	268966
10	108.3	181	293159	116.3	180	305400	117.2	178	282811	229.9	175	281340	137.1	180	272794	156.6	178	256098
Mean	116.9	179	291302	127.3	182	301695	118.1	175	287795	178.1	178	277857	186.9	179	273461	147.2	176	261339

Table 5-4: Results obtained for Constraint Group C

	1 Satellite			2 Satellites			3 Satellites			4 Satellites			5 Satellites			6 Satellites		
Run #	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value
1	112.6	176	282549	115.4	177	272784	162.4	185	297839	124.1	171	259673	116.4	175	267294	257.4	175	272788
2	120.3	172	281004	113.8	176	281085	146.7	181	286244	119.7	171	252930	118.4	173	274118	255.3	172	272520
3	181.9	172	281822	113.3	178	276701	126.9	178	287371	112.2	169	255300	115.9	173	263905	255.4	171	279598
4	116.3	172	277810	109.6	180	280386	115.8	182	284437	108.9	163	253914	119.6	176	274102	124.3	171	277772
5	117.9	168	277894	123.5	178	274932	219.1	185	292703	113.1	170	252380	122.1	173	273906	261.2	169	269401
6	114.7	170	281294	118.3	172	278957	210.6	181	292575	116.5	171	266192	114.6	172	264897	262.8	177	277664
7	117.1	170	285982	126.5	177	273012	151.5	184	287762	123.5	171	258602	119.1	172	278280	139.2	176	286414
8	113.8	175	280862	125.7	175	278723	223.0	182	291784	123.4	173	264360	122.4	174	276712	132.1	173	270132
9	111.8	172	283238	114.5	178	279926	208.5	180	276384	115.7	177	268097	119.5	162	257948	155.3	171	271374
10	122.5	174	278051	114.8	180	282250	218.3	185	292703	112.0	174	260684	236.0	172	266926	121.0	176	272383
Mean	122.9	172	281051	117.5	177	277876	178.3	182	288980	116.9	171	259213	130.4	172	269809	196.4	173	275005

Table 5-5: Results obtained for Constraint Group D

Run#	1 Satellite			2 Satellites			3 Satellites			4 Satellites			5 Satellites			6 Satellites		
	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value	CPU Time (Sec)	No of Time-windows	Objective Function Value
1	1160.9	181	281109	1673.0	182	279781	2571.5	177	276132	3018.5	180	263668	1900.9	177	243982	4663.3	175	251208
2	1005.2	181	281109	2211.6	181	274653	2056.3	173	267017	2662.0	178	258878	1954.4	177	243982	2825.3	176	246290
3	1166.9	181	281109	2107.5	175	265874	1428.4	173	267461	3951.5	178	259355	1501.1	184	243552	5970.3	183	251428
4	1792.3	176	286137	2253.0	176	271169	2054.8	175	265146	4296.9	177	258408	1691.7	177	241692	5361.9	174	246015
5	1582.6	180	290868	1880.0	178	280463	2178.0	177	265413	2212.3	184	259393	4220.0	183	244134	2366.8	174	250426
6	1310.0	179	296192	1954.2	179	272167	3866.0	174	265418	3081.8	173	261281	4220.0	179	243518	2496.4	173	250781
7	1539.0	174	283913	3634.1	175	274792	4052.8	174	262471	1704.2	178	259045	5680.7	174	243047	3050.7	173	246589
8	1606.3	178	280999	1930.1	177	273106	5614.6	177	264343	3960.2	175	260130	5581.4	179	244802	1785.0	171	250964
9	964.9	176	289652	2027.3	173	275860	2136.0	173	268217	4201.1	180	257756	1797.3	176	248209	6705.3	173	252096
10	1730.8	179	289639	2968.7	178	277522	3996.9	178	264123	3135.8	175	259540	3896.3	177	243372	1628.8	177	249986
Mean	1385.9	179	286073	2263.9	177	274539	2995.5	175	266574	3222.4	178	259745	3244.4	178	244029	3685.4	175	249578

5.2.2 The influence of the number of satellites

The first perspective on the results looks at the influence of the number of satellites in the constellation on algorithm performance for each of the four groups (A though D) of data sets. The influence of the number of satellites were analysed in terms of its effect on computing time and its impact on the number of time-windows activated.

5.2.2.1 The influence of the number of satellites on computing time

Figure 5-3 through Figure 5-6 below show the influence of the number of satellites on computing time for each of the constraint groups A through D.

Each figure shows the range of computing times obtained for a given number of satellites and also indicates, within the range, the mean value for the 10 experiments associated with the given number of satellites. A trend line is fitted though the mean values. As is to be expected, the trend line shows that computing times increase with the number of satellites. The rate of increase as indicated by the slope of the trend line is however not particularly steep and the ranges of computing times overlap. More experiments are required. At this stage one would cautiously say that it appears that computing time increases with an increase in the number of satellites but not to the extent that it presents a problem in practice.

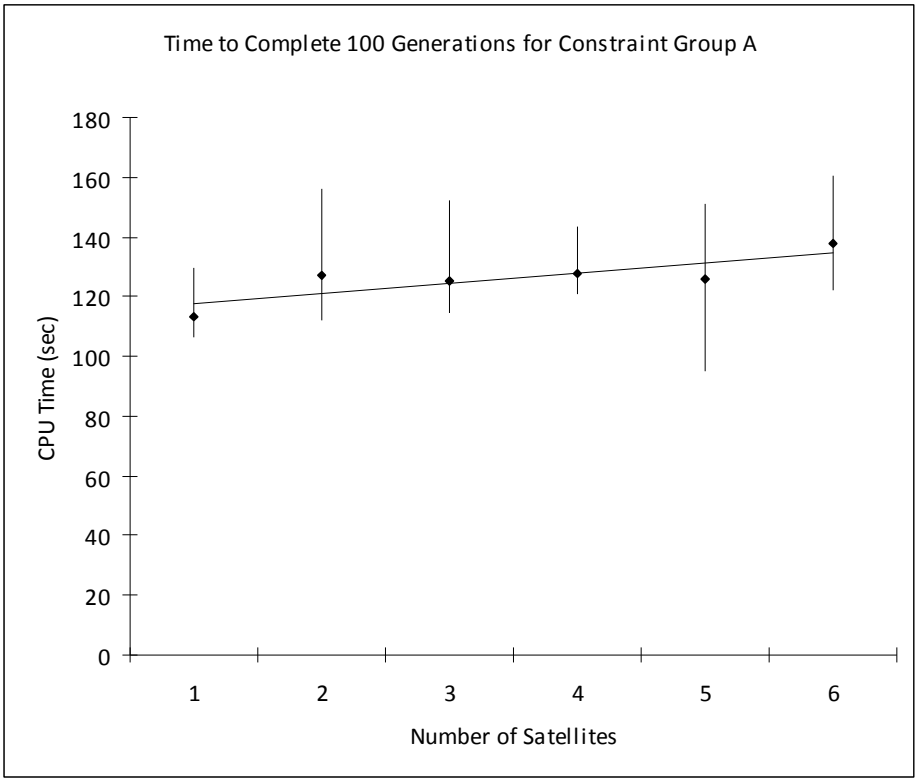

Figure 5-3: The computing time to find 100 generations of feasible solutions for Constraint Group A as a function of the number of satellites

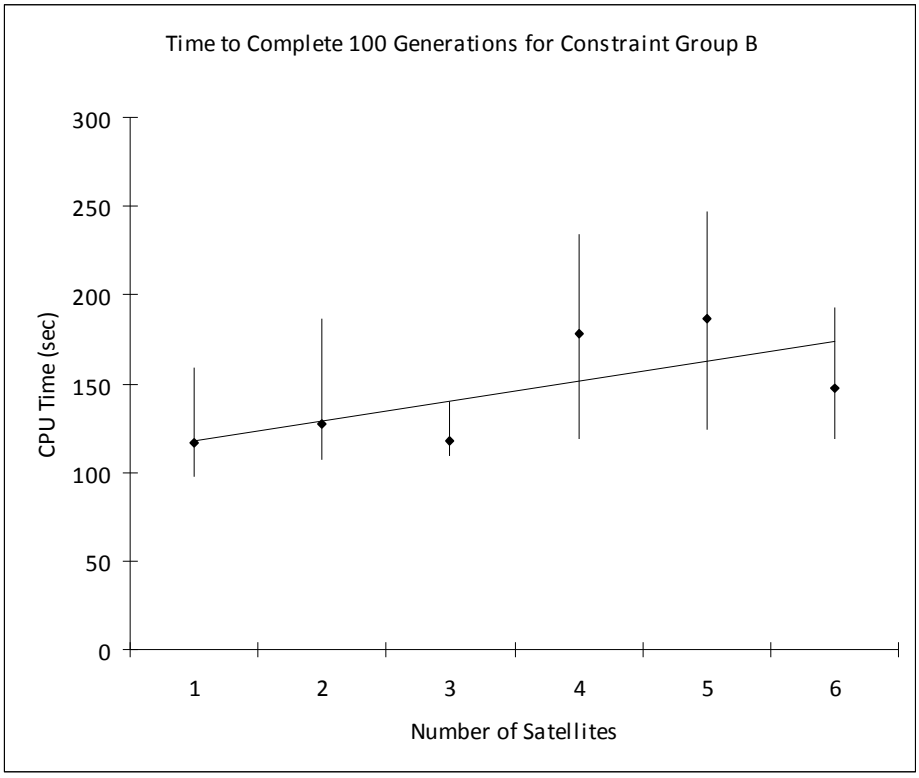

Figure 5-4: The computing time to find 100 generations of feasible solutions for Constraint Group B as a function of the number of satellites

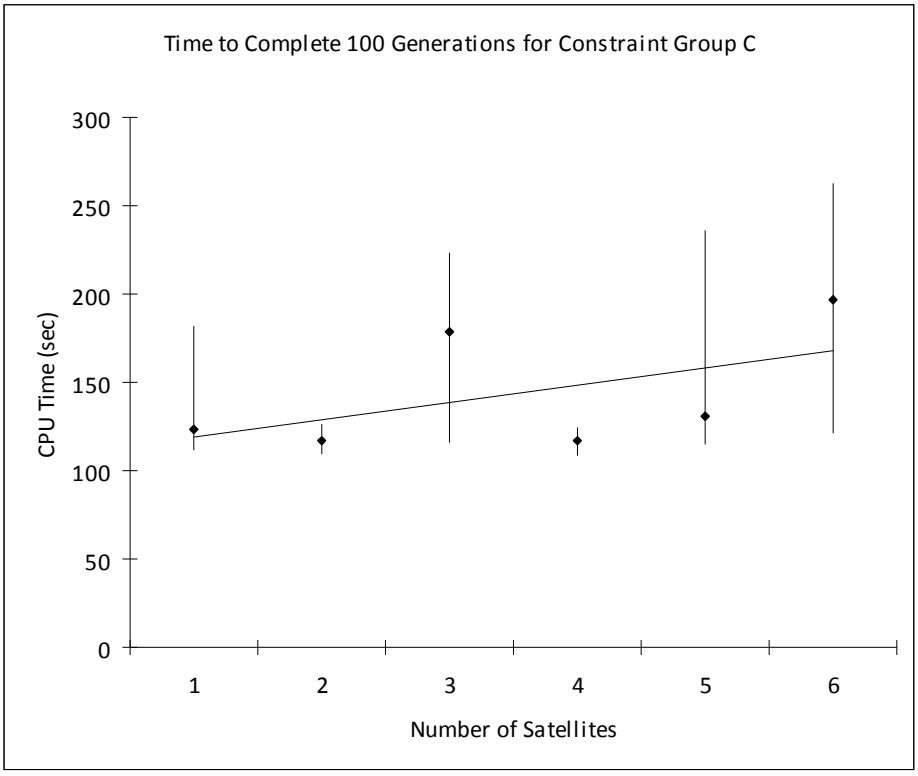

Figure 5-5: The computing time to find 100 generations of feasible solutions for Constraint Group C as a function of the number of satellites

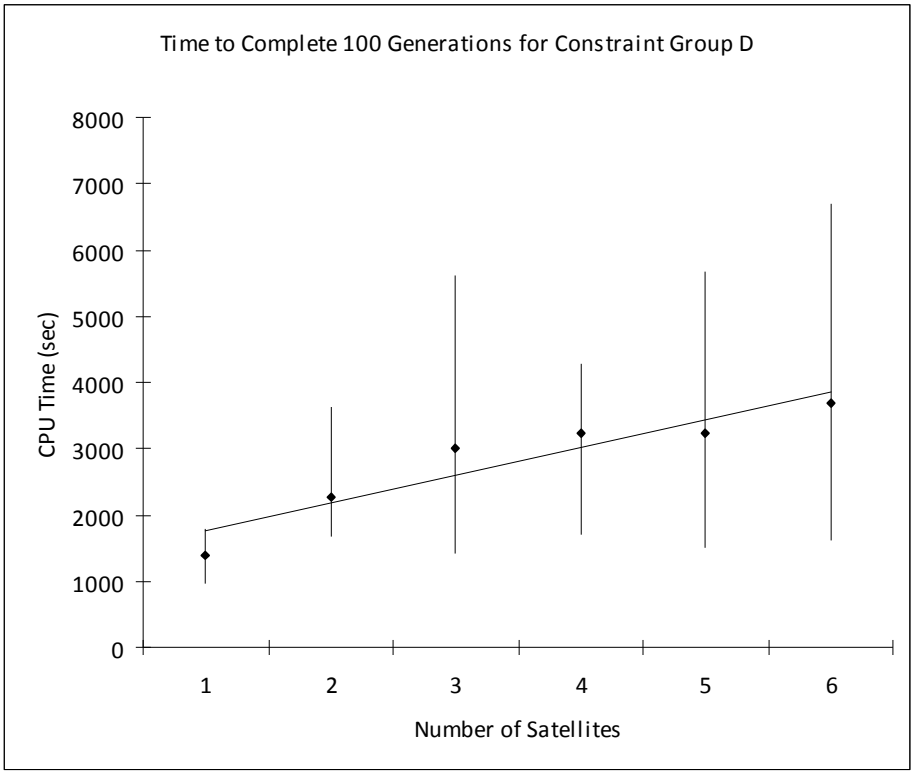

Figure 5-6: The computing time to find 100 generations of feasible solutions for Constraint Group D as a function of the number of satellites

5.2.2.2 The influence of the number of satellites on the number of time-windows activated

Figure 5-7 through Figure 5-10 below show the influence of the number of satellites on the number of time-windows successfully activated for each of the constraint groups A through D. We note that the algorithm performs reasonably well, in that it consistently activates about 86 % of time-windows successfully for constraint group A, where there are no limits on capacity and only non-interference constraints hold. This number drops to about 70 % when on-board memory is limited as is the case for constraint groups B, C, and D. This reduction corresponds very well with the reduction in memory capacity which was set at 70 %.

Each figure shows the range of the number of time-windows activated obtained for a given number of satellites and also indicates, within the range, the mean value for the 10 experiments associated with the given number of satellites. A trend line is also fitted though the mean values. With the exception of the case for constraint group A, the trend lines appear to show that the number of time-windows activated decrease with the number of satellites. This seems counterintuitive, since one would expect that more satellites will, in general, be able to service more time-windows, as is indeed the case for constraint group A. To investigate the significance of this apparent trend for constraint groups B, C and D, the Pearson product moment correlation coefficient, r, was calculated using all 60 data points in the results obtained for the experiments conducted for each of these constraint groups. The Pearson coefficient is a dimensionless index that ranges from -1.0 to 1.0 inclusive and reflects the extent of a linear relationship between two data sets. Values close to -1.0 and 1.0 would indicate strong linear relationships while values closer to 0 would indicate weak or no relationship. The values of r obtained for constraint groups B, C and D respectively are 0.157, 0.143 and -0.201. The corresponding values of r^2 are 0.025, 0.021, and 0.041 respectively, indicating that 2-4 % of the variation between the 60 data points for each constraint group can be explained by the number of satellites. This is considered to be statistically insignificant, and the apparent trend need not be investigated further.

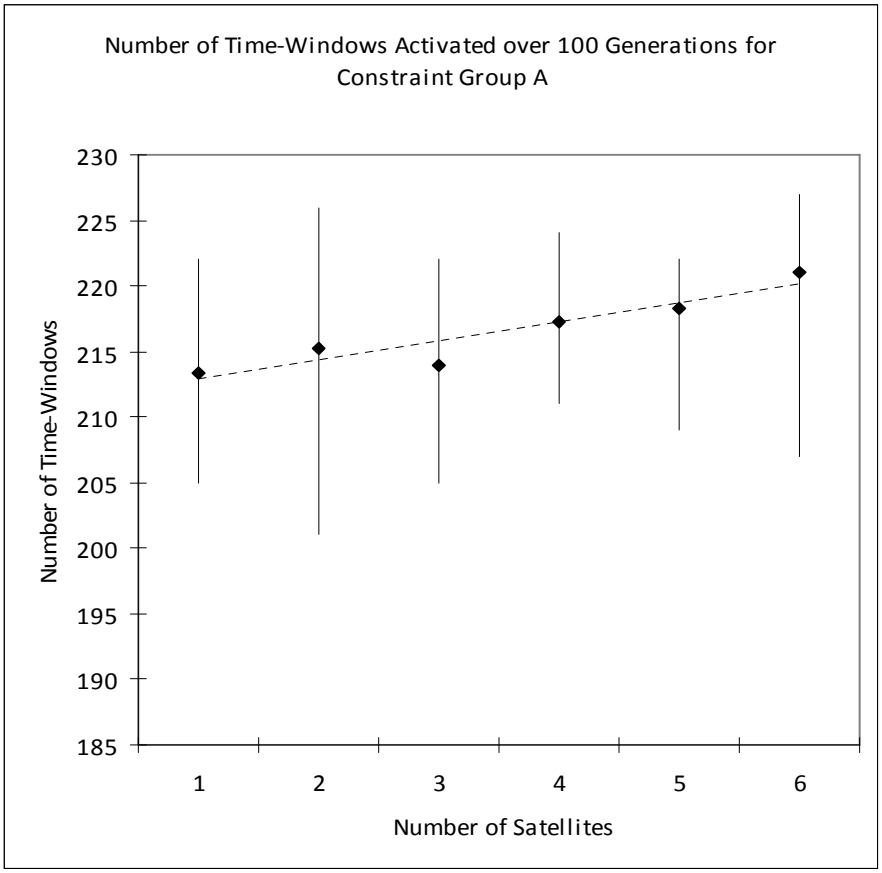

Figure 5-7: The number of time-windows activated during 100 generations of feasible solutions for Constraint Group A as a function of the number of satellites

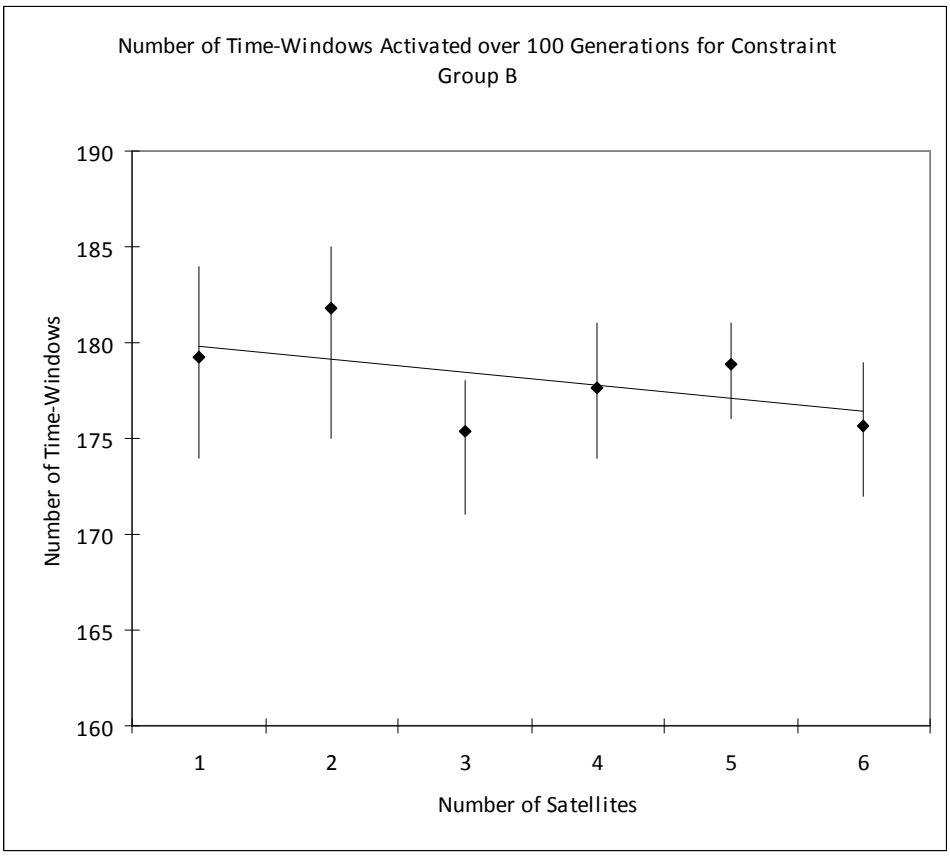

Figure 5-8: The number of time-windows activated during 100 generations of feasible solutions for Constraint Group B as a function of the number of satellites

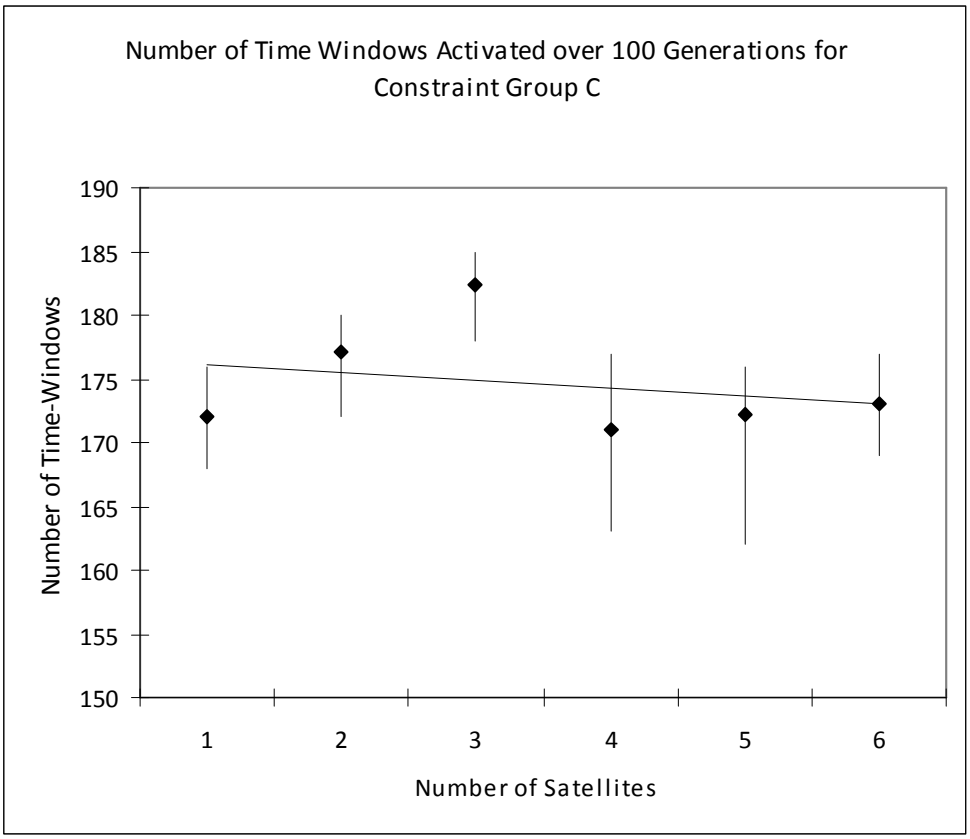

Figure 5-9: The number of time-windows activated during 100 generations of feasible solutions for Constraint Group C as a function of the number of satellites

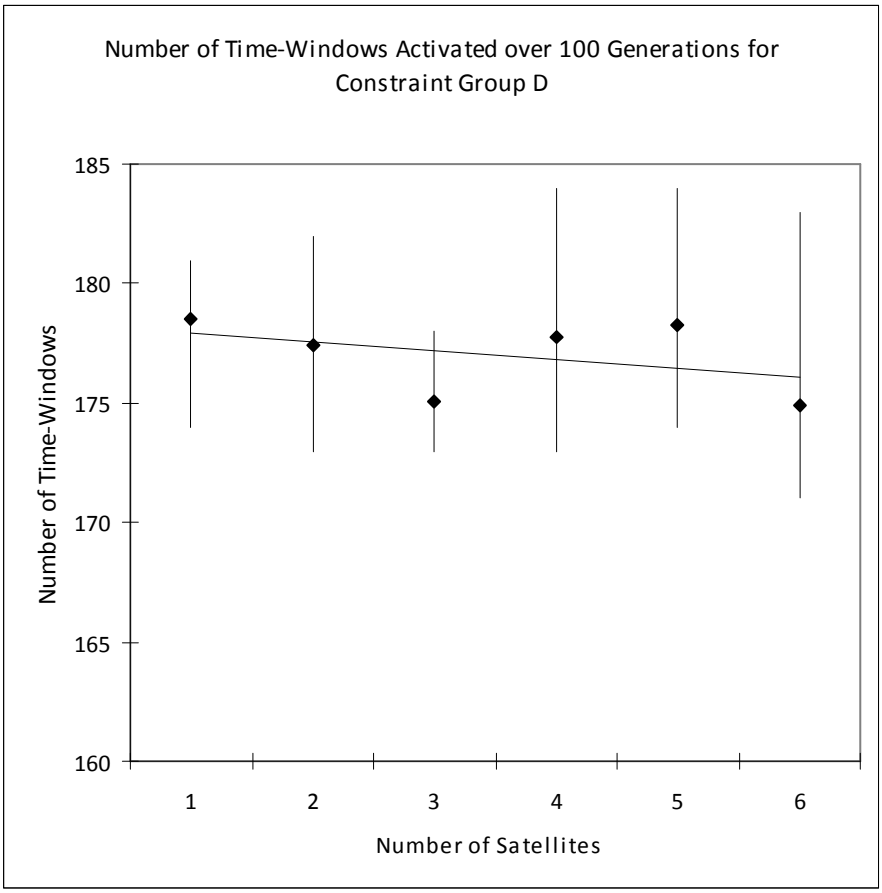

Figure 5-10: The number of time-windows activated during 100 generations of feasible solutions for Constraint Group D as a function of the number of satellites

5.2.3 The influence of progressively tightening constraints

The second perspective on the results looks at the influence of successively more restrictive constraints on algorithm performance for each of the six cases where the number of satellites in the constellation is kept constant.

5.2.3.1 The influence of tightening constraints on computing time

Figure 5-11 through 15 below show the influence of successively applying more restrictive constraints on computing time for each of the constellation configurations of one though six satellites.

Each figure shows the range of computing times obtained for a constraint set and also indicates, within the range, the mean value for the 10 experiments associated with the given constraint set. Contrary to expectations the graphs consistently shows that computing times are virtually the same for constraint groups A, B and C but that there is an order of magnitude jump in computing times when constraint set D is introduced. In the latter case the computing times increase between 12 and 20 fold and the change is so marked that no trend line could be fitted. This is significant in that it shows that the requirement to select a single time-window for a specific image from a number of possible options introduces considerable inefficiency in the algorithm. Also of great interest is the fact that this is the case regardless of the number of satellites.

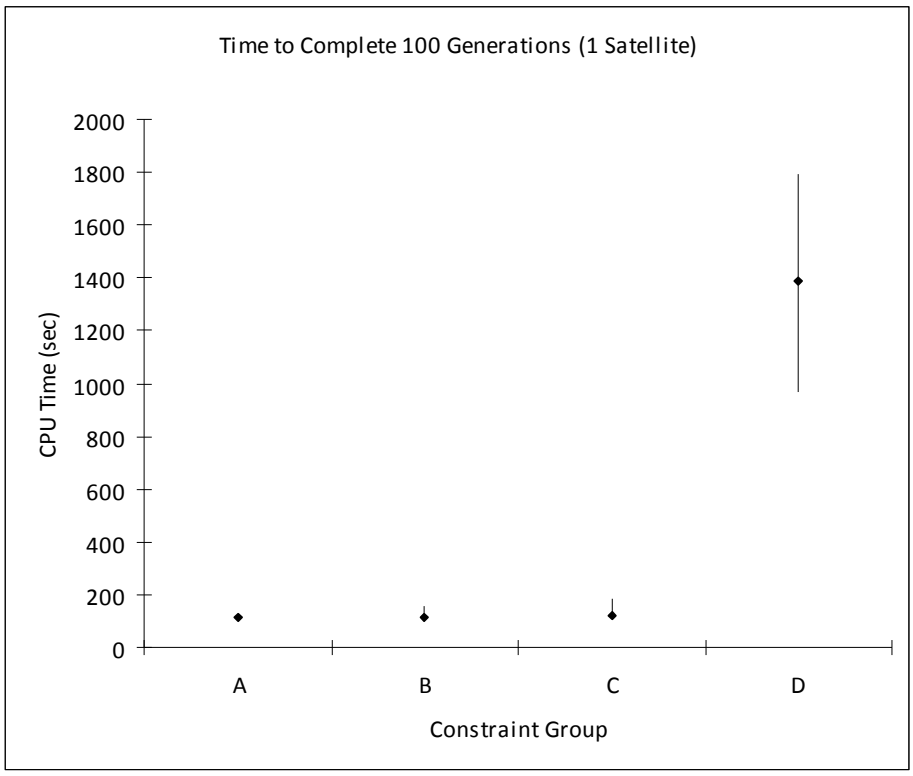

Figure 5-11: The computing time to find 100 generations of feasible solutions for a single satellite as a function of tightening constraints

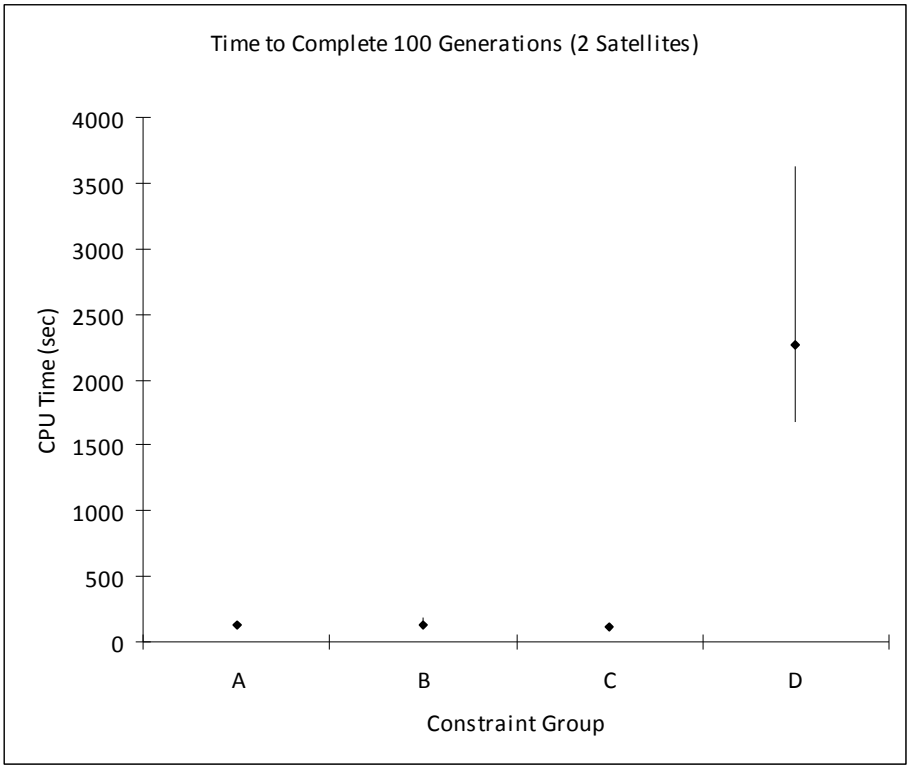

Figure 5-12: The computing time to find 100 generations of feasible solutions for two satellites as a function of tightening constraints

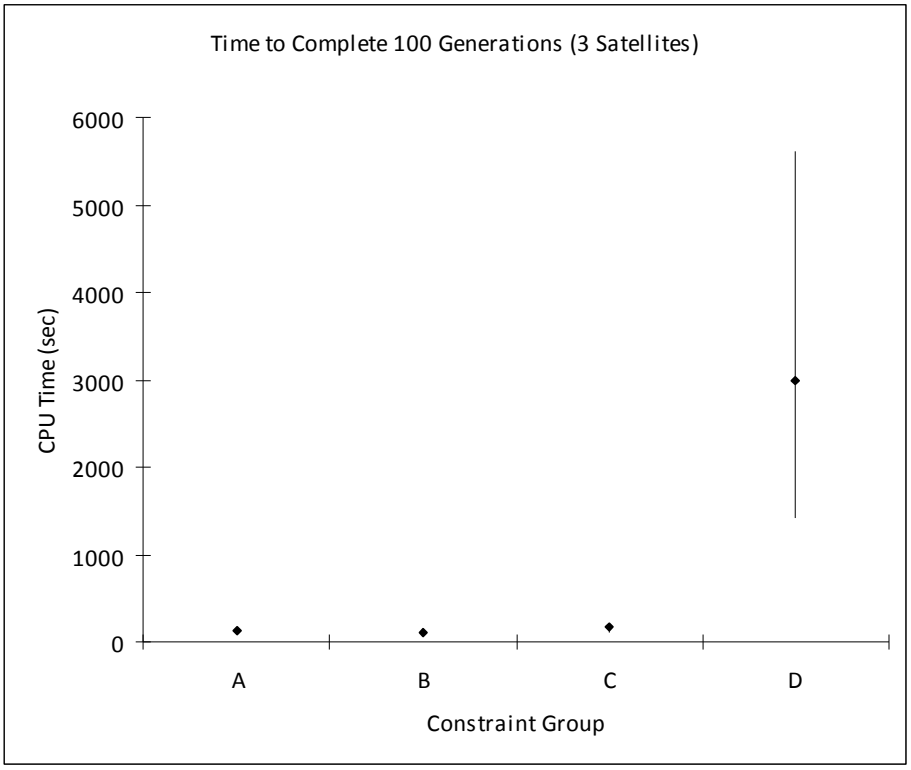

Figure 5-13: The computing time to find 100 generations of feasible solutions for three satellites as a function of tightening constraints

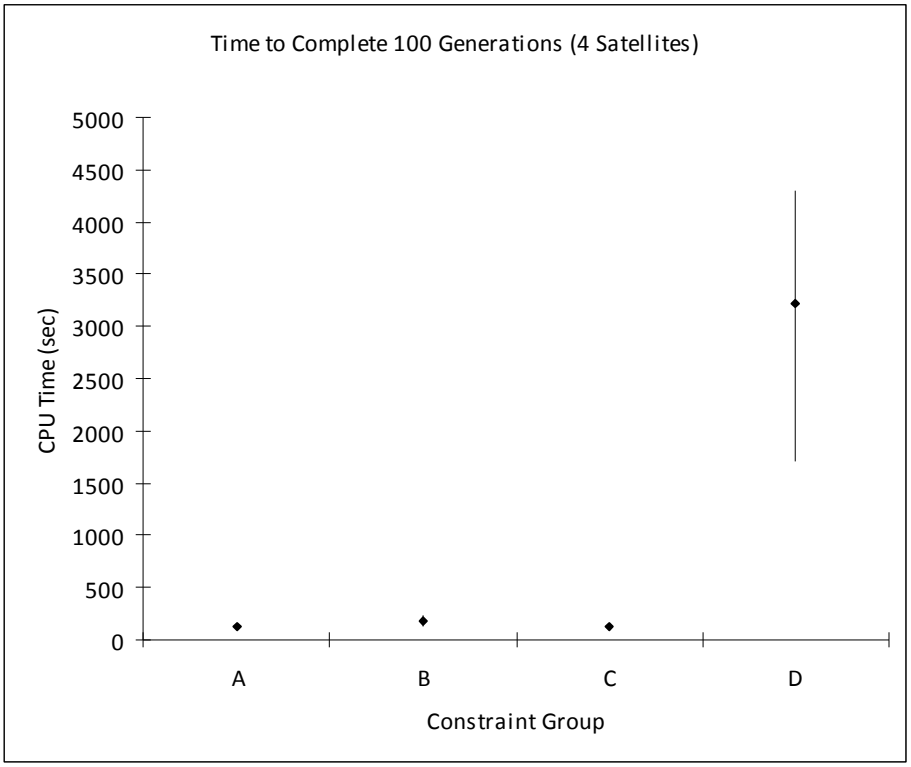

Figure 5-14: The computing time to find 100 generations of feasible solutions for four satellites as a function of tightening constraints

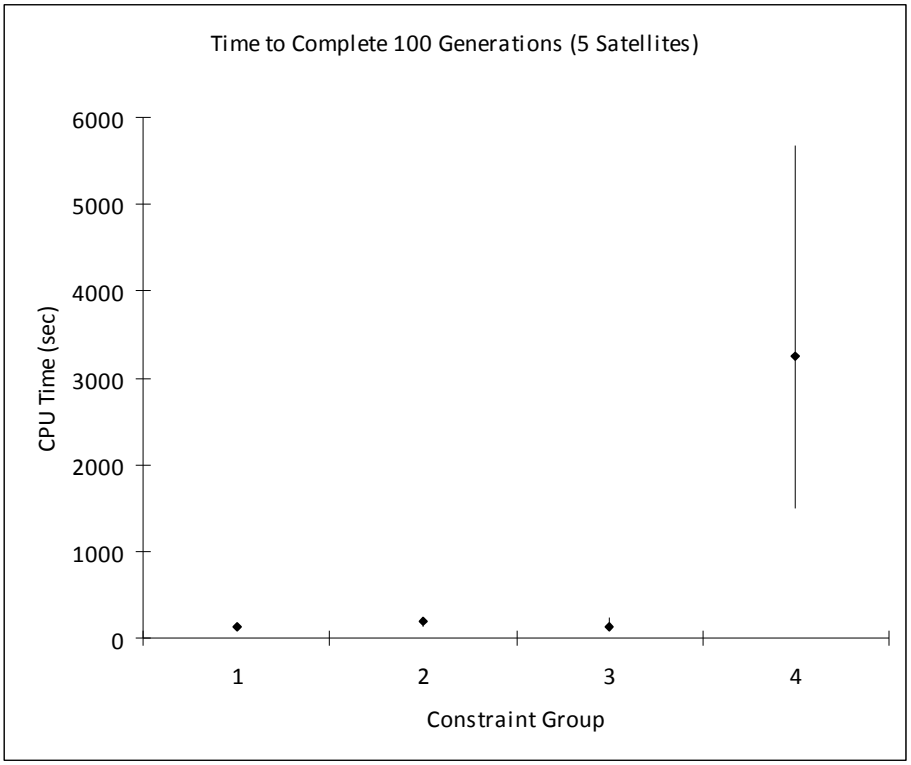

Figure 5-15: The computing time to find 100 generations of feasible solutions for five satellites as a function of tightening constraints

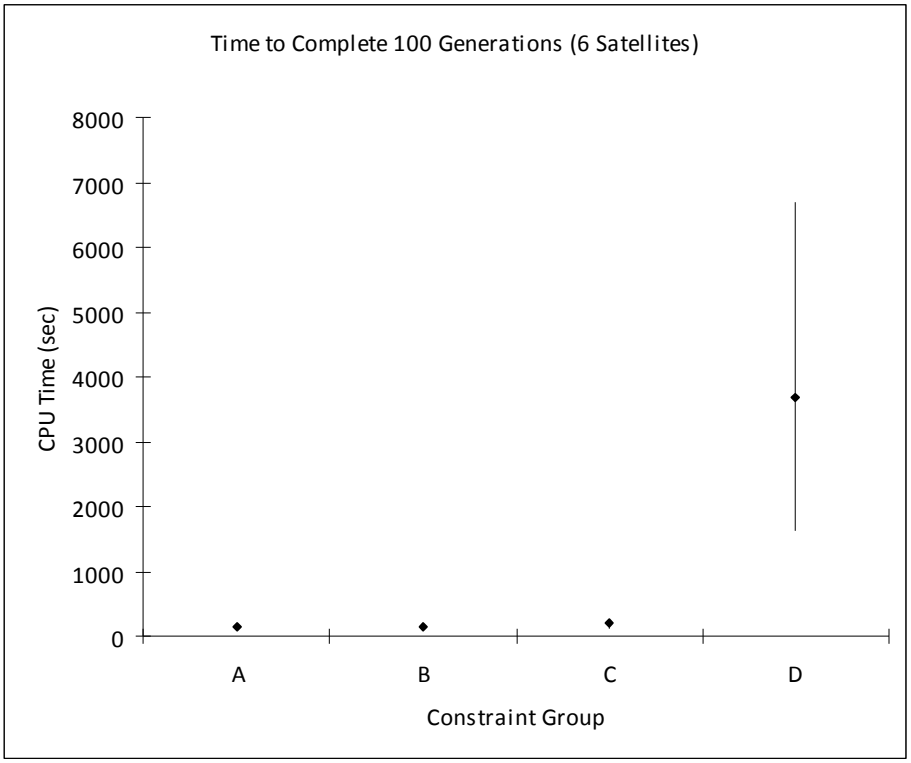

Figure 5-16: The computing time to find 100 generations of feasible solutions for six satellites as a function of tightening constraints

5.2.3.2 The influence of tightening constraints on the number of time-windows activated

Figure 5-17 through Figure 5-22 below show the influence of successively applying more restrictive constraints on the number of time-windows activated for each of the constellation configurations of one though six satellites.

Each figure shows the range of time-windows activated for a constraint set and also indicates, within that range, the mean value for the 10 experiments associated with the given constraint set. A trend line is fitted though the mean values. As can be expected, the trend line shows that the number of time-windows activated decrease as constraints are tightened for a given number of satellites. The rate of decrease as indicated by the slope of the trend line is not particularly steep but given the results discussed above is nevertheless consistent.

As one would expect, given the same number of time-windows to be activated, but more constraints, the number of feasible solutions decreases resulting in a corresponding reduction in the number of time-windows activated over the same number of 100 generations of solutions. Again, the ranges of time-windows activated overlap significantly, regardless of the number the constraint so that the trend may not be statistically significant. Interestingly, the algorithm continue to perform well in terms of the number of time-windows activated, as seen in the previous section, efficiency suffers greatly when constraint Group D is introduced.

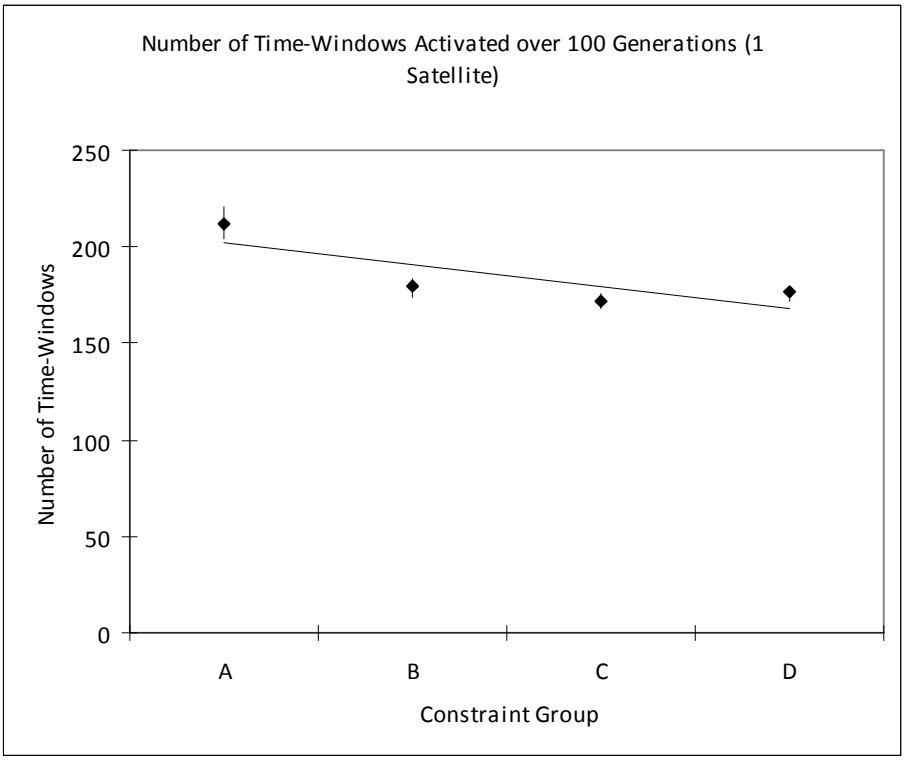

Figure 5-17: The number of time-windows activated during 100 generations of feasible solutions for a single satellite as a function of tightening constraints

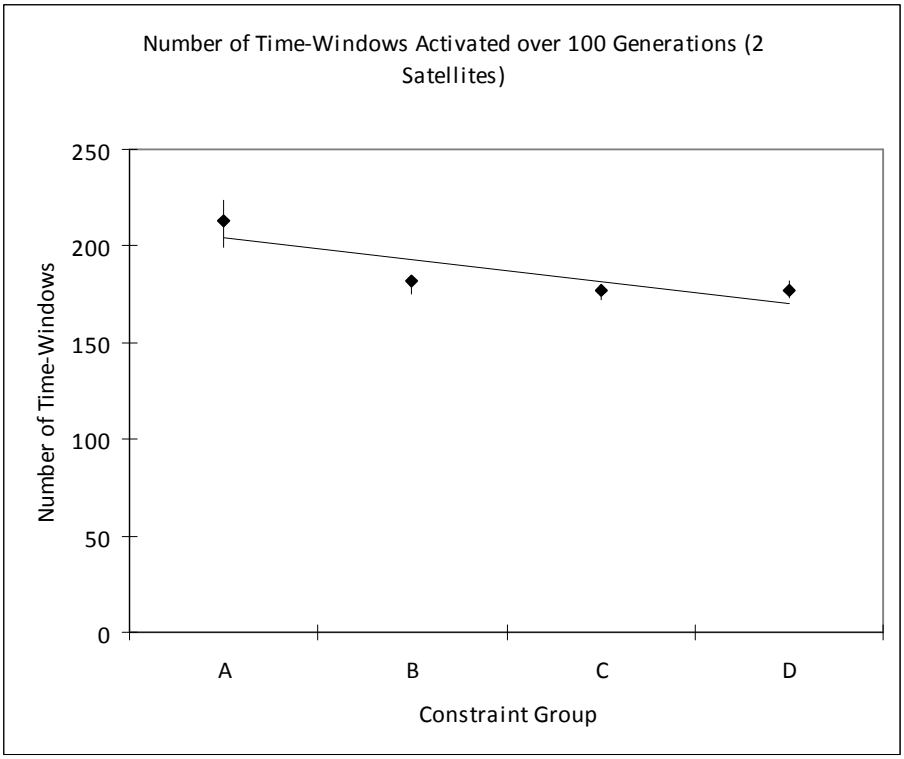

Figure 5-18: The number of time-windows activated during 100 generations of feasible solutions for two satellites as a function of tightening constraints

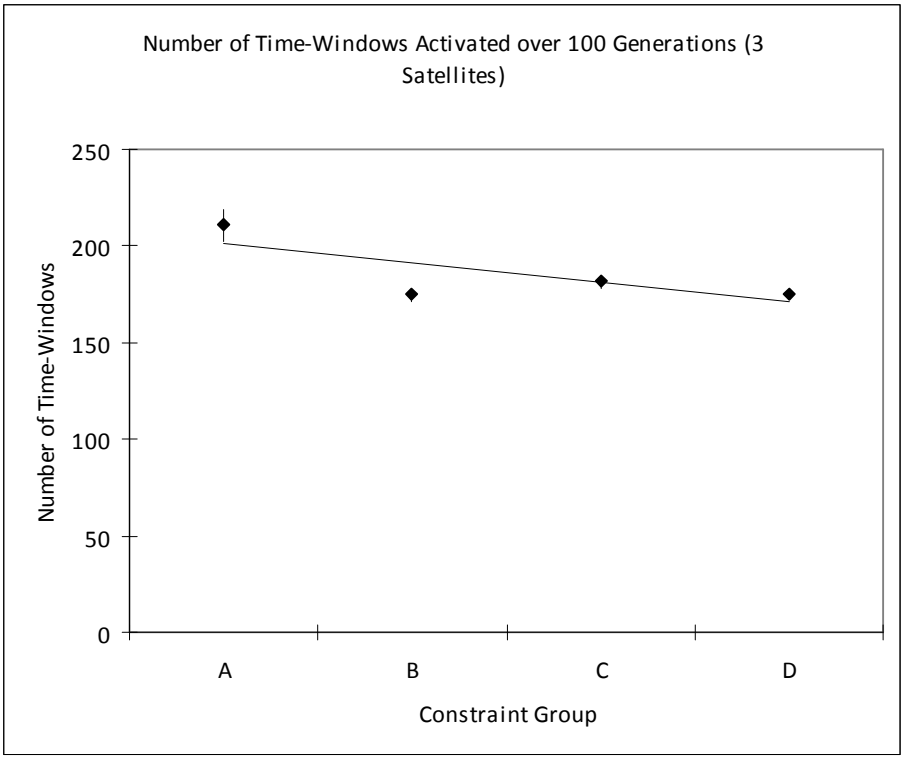

Figure 5-19: The number of time-windows activated during 100 generations of feasible solutions for three satellites as a function of tightening constraints

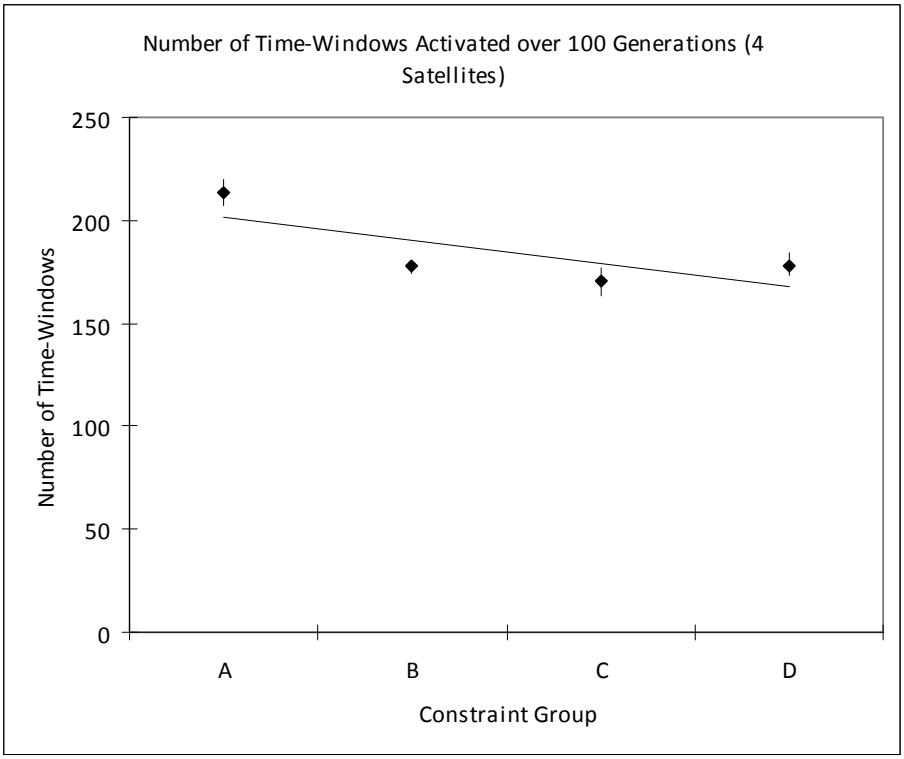

Figure 5-20: The number of time-windows activated during 100 generations of feasible solutions for four satellites as a function of tightening constraints

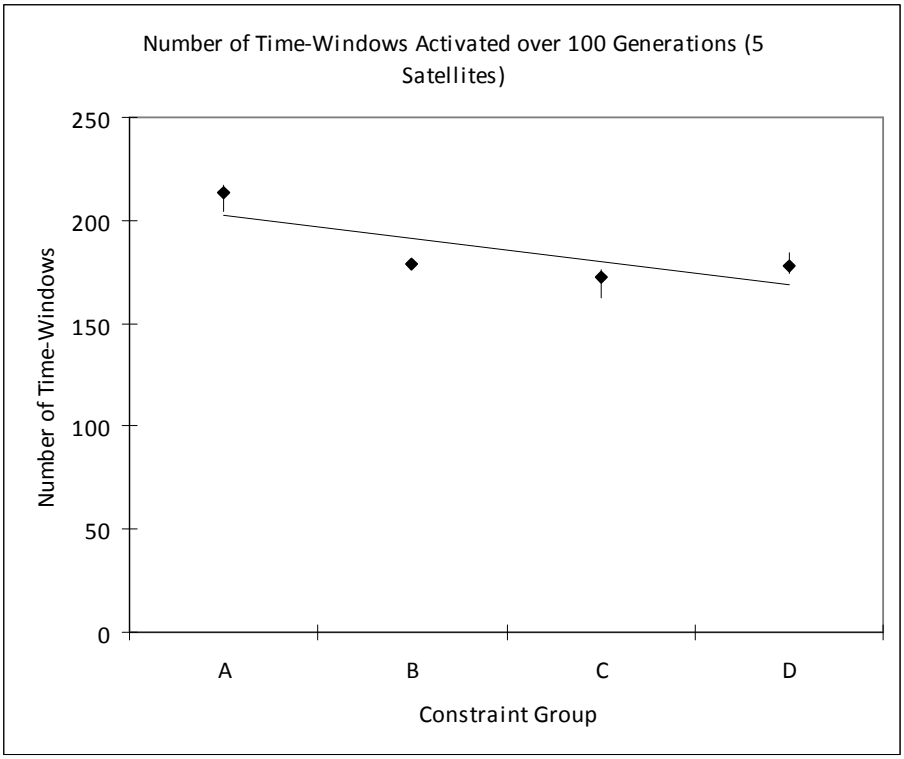

Figure 5-21: The number of time-windows activated during 100 generations of feasible solutions for five satellites as a function of tightening constraints

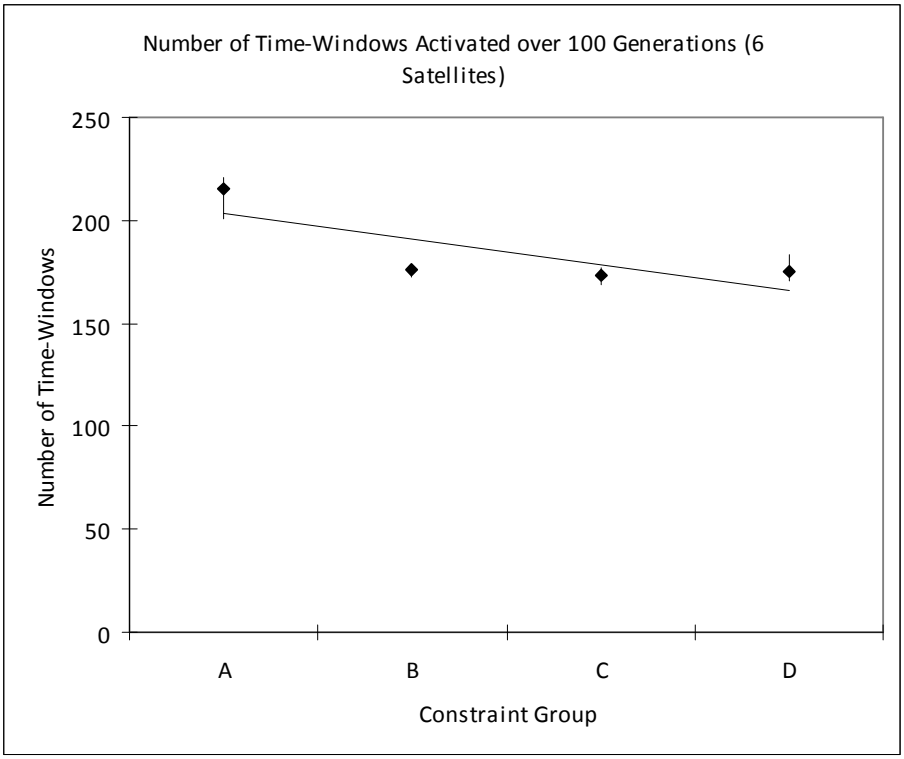

Figure 5-22: The number of time-windows activated during 100 generations of feasible solutions for six satellites as a function of tightening constraints

5.3 Determining the efficacy of using different reproductive options

Given the considerable deterioration in performance of the genetic algorithm when the full set of constraints is applied, an investigation was launched into possible ways of improving the performance by adjusting some of the algorithm parameters. Three avenues of investigation were identified, all of them relating to the way in which new solutions are derived from the parent population. These investigations comprised the following;

- The effect of changing the elite count parameter on algorithm performance;

- The effect of changing the crossover fraction parameter on algorithm performance, and;
- The effect of changing the crossover mechanism on algorithm performance.

The results of these investigations are outlined below. Given the long computing times required to run these experiments, the approach was to perform only a single computer runs if initial results indicate worse or similar performance than for the default case.

5.3.1 The influence of varying the elite count parameter

This set of experiments was aimed at investigating the influence of the elite count parameter on the performance of the genetic algorithm in solving the SCoTWOP. The elite count parameter specifies the number of individuals that are guaranteed to survive to the next generation and is a positive integer less than or equal to the population size. The default value is 2. The experiments consisted of varying the elite count from a value of 1 to 10 when using the genetic algorithm to solve the SCoTWOP for the Group D dataset with six satellites. A single computer run was performed for each value of the elite count parameter and for each run the processing time and number of time-windows activated were recorded. The results thus obtained are graphically displayed in Figure 5-23 and Figure 5-24.

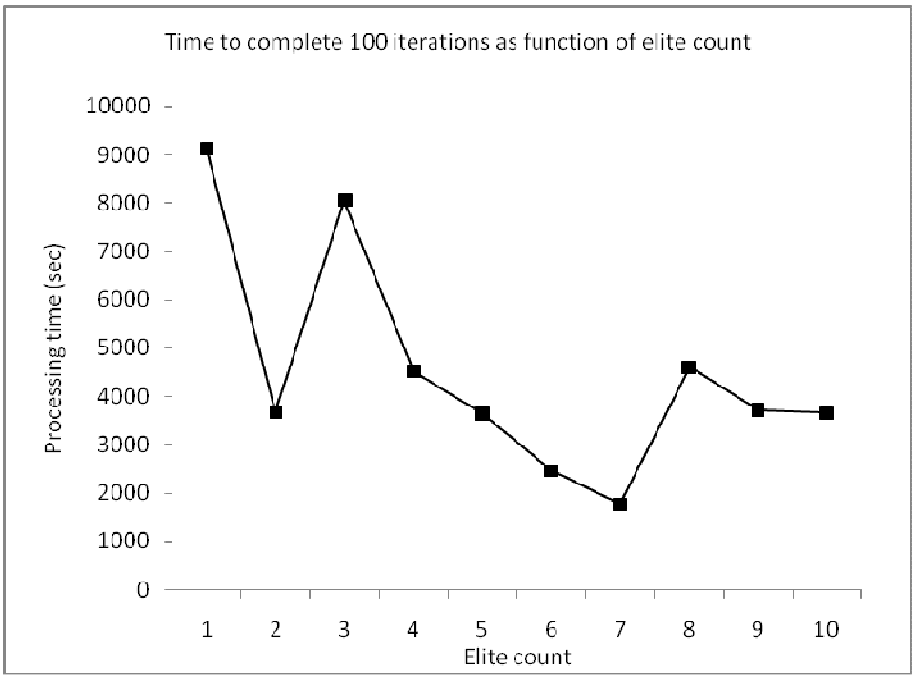

Figure 5-23: The computing time to find 100 generations of feasible solutions for six satellites as a function of the elite count parameter

Figure 5-23 shows the variation of the computing time required to find 100 generations of feasible solutions as function of elite count. When interpreting this data, it should be remembered that, for all values of the elite count parameter except its default value of 2, only one computer run was performed. The computing time at an elite count of 2, is however the average obtained from 10 computer runs. The computing times for the other values of the elite count parameter should therefore be interpreted relative to the more firmly established value of 3685 seconds obtained for an elite count of 2.

Superficially, it appears that a case can perhaps be made for a trend that will have computing times decrease with increasing elite count until a minimum is reached at an elite count of 7, after which computing times increase again. However, the better established default value, does not agree with such a trend. It would therefore be more cautious to say that the computing times of other elite

counts bracket that of the default count of 2, with considerable variation. In fact, this caution is reinforced when it is realized that the default value itself is the average of a series of values that range from as high as 6705 seconds to as low as 1628 seconds.

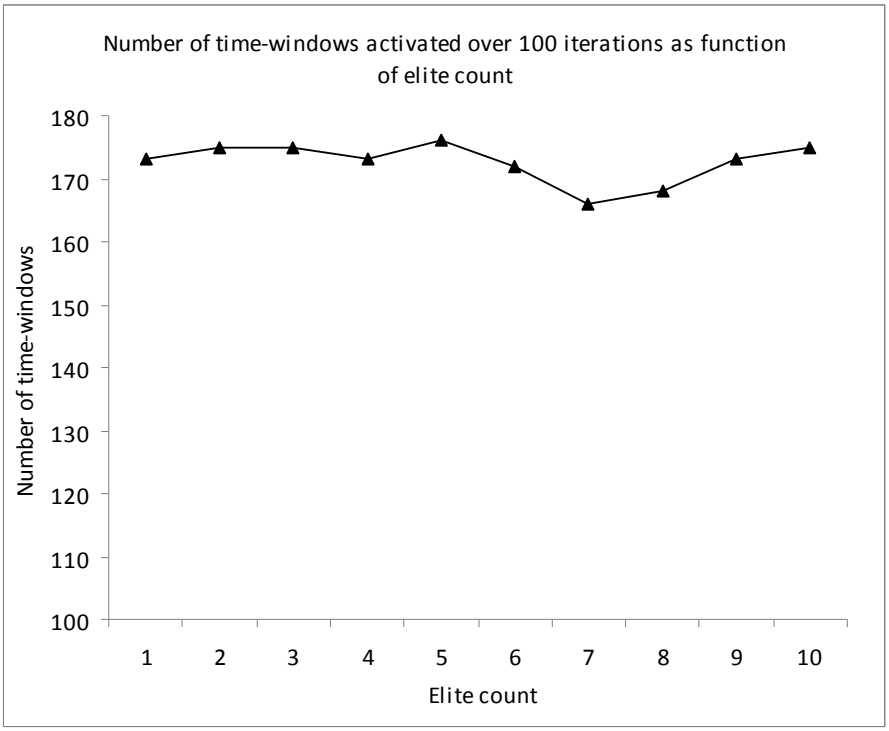

Figure 5-24: The number of time-windows activated during 100 generations of feasible solutions for six satellites as a function of the elite count parameter

Examination of Figure 5-24 confirms that the interpretation of Figure 5-23, given above, is probably correct. The figure shows the variation in the total number of time-windows activated with elite count. Comparison of this figure with Figure 5-23 reveals that lower computing times are in general associated with a lower number of time-windows activated.

More numerical experiments are needed to explore what may be a possible improvement in computational time.

5.3.2 The influence of varying the crossover fraction parameter

This set of experiments was aimed at investigating the influence of the crossover fraction parameter on the performance of the genetic algorithm in solving the SCoTWOP. The crossover fraction parameter specifies the fraction of the next generation, other than elite children, that are produced by crossover. The default value is 0.8. The experiments consisted of varying the crossover fraction from a value of 0.5 to 0.8 when using the genetic algorithm to solve the SCoTWOP for the Group D dataset with six satellites. A single computer run was performed for each value of the crossover fraction parameter and for each run the processing time and number of time-windows activated were recorded. The results thus obtained are graphically displayed in Figure 5-25 and Figure 5-26.

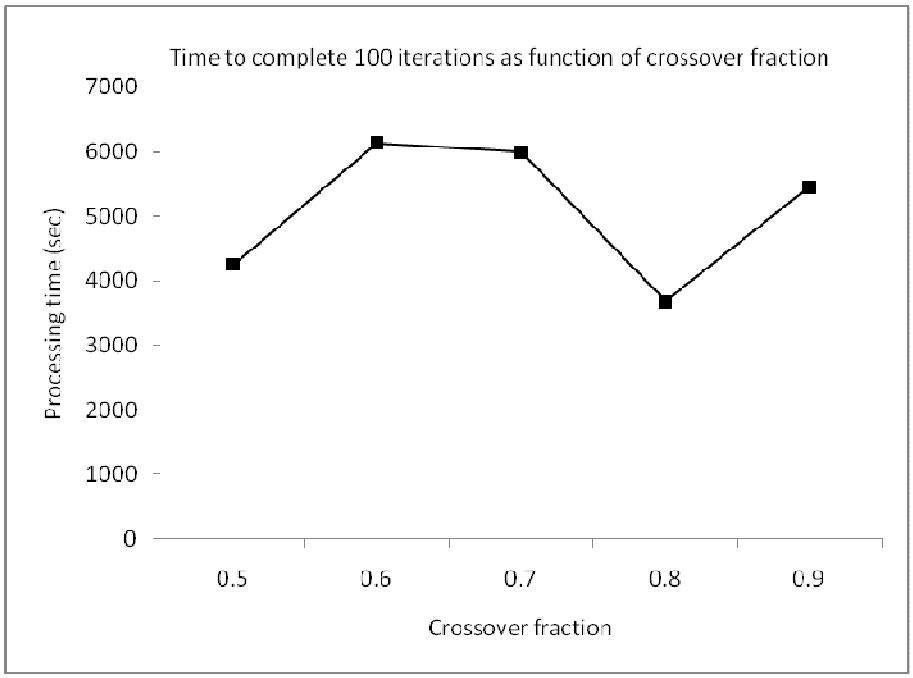

Figure 5-25: The computing time to find 100 generations of feasible solutions for six satellites as a function of the crossover fraction parameter

Figure 5-25 shows the variation of the computing time required to find 100 generations of feasible solutions as a function of the crossover fraction. As was the case above for the elite count experiments, it should be remembered that the computing times for other values of the crossover fraction parameter should be

interpreted relative to the more firmly established value of 3685 seconds obtained as the average computing time for the default crossover fraction of 0.8. Given this, it appears that very little can be said about the possible influence of crossover fraction on computing time. Although computing times for crossover fractions other than the default value are higher than that of the default value, the set of experimental data is too sparse to reach a definite conclusion.

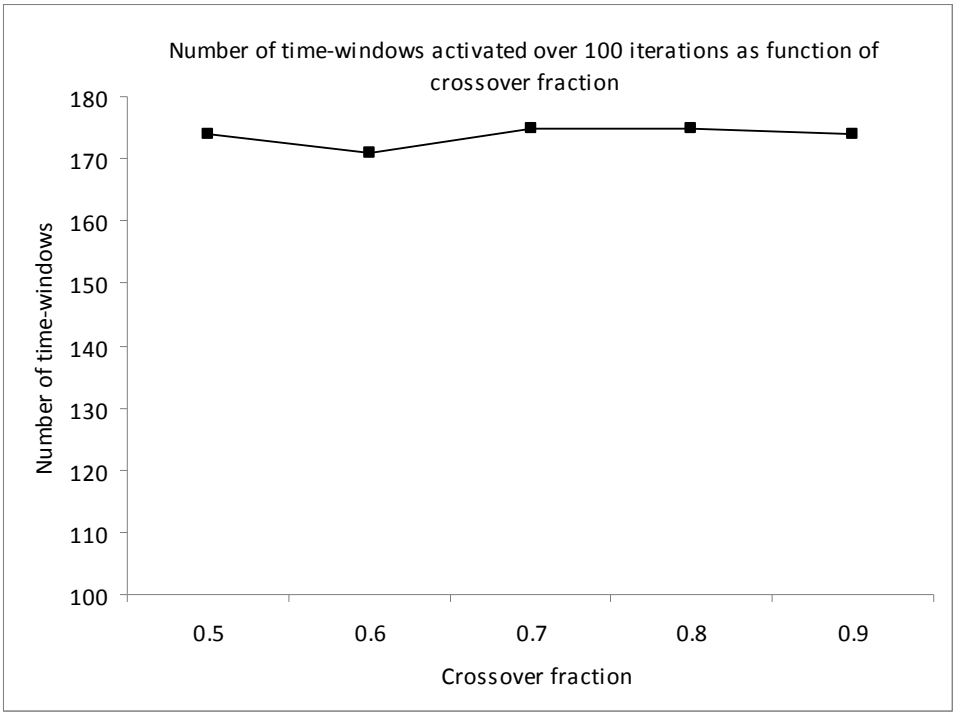

Figure 5-26: The number of time-windows activated during 100 generations of feasible solutions for six satellites as function of the crossover fraction parameter

Examination of Figure 5-26 once again confirms that the results should be interpretation with caution. The figure shows the variation in the total number of time-windows activated with crossover fraction. It seems that the crossover fraction has very little, if any, influence on the number of time-windows activated. Once again it can only be concluded that more experiments are necessary.

5.3.3 The influence of selecting different crossover options

The MatLab® Genetic Algorithm and Direct Search Toolbox provides a number of options for different crossover mechanisms through which the genetic algorithm combines the genome of two parents, to form a crossover child for the next generation. The following three mechanisms are suitable for use with bit string genomes, as is the case for the SCoTWOP:

- Scattered crossover, the default crossover option, creates a random binary vector and then selects a gene from the first parent if the corresponding entry of the random vector is a 1 and from the second parent if the corresponding entry of the random vector is a 0; Single point crossover chooses a random integer n between 1 and the total number of genes in the genome and then selects genes numbered less than or equal to n from the first parent and genes numbered greater than n from the second parent, and;

- Two point crossover selects two random integers m and n between 1 the total number of genes in the genome and then selects genes numbered less than or equal to m from the first parent and, genes numbered m+1 to n, inclusive, from the second parent.

Table 5-6 below contains the results obtained for the SCoTWOP when the single point and two point crossover mechanism are used. The results are compared with the average obtained for the scattered crossover mechanism, used for all other experiments.

Table 5-6: Results obtained for different crossover mechanisms

Crossover mechanism	Computing time needed for 100 generations (sec)	Number of time-windows activated over 100 generations
Scattered crossover	3685	175
Single point crossover	5388	175
Two point crossover	7431	181

Given the fact that only one computer run was conducted for each, the results show that single point and two point crossover do not have an obvious advantage over the scattered crossover mechanism, since computing times appear to be at least as long or longer than that obtained for the scattered crossover mechanism. In terms of number of time-windows activated, the three crossover mechanisms appear to perform equally well. The number of 181 time-windows activated when a two point crossover is used, is the best performance observed for all experiments conducted. It does however come at the price of an additional hour (a doubling) of computing time.

5.4 The influence of the penalization scheme for infeasible solutions

The results discussed above were obtained using the severest penalty for infeasible solutions, i.e. allocating a fitness value function of positive infinity to infeasible solutions. Potentially infeasible solutions arising from crossover operations were, except where otherwise noted, also repaired if they violated the singularity constraint. To investigate the influence of this on computing time and the quality of solutions, a number of computer runs were made using the default *Scattered* crossover option (i.e. no repair) and also using a more conventional penalty scheme in which a fixed percentage of the objective

function was added to the fitness function value. The Group D dataset with six satellites was used in all cases and the results are presented in Table 5-7 below.

Table 5-7: Results obtained for different penalty fractions

Penalty Fraction (percentage of the objective function value added to fitness value)	Computing time needed for 100 generations (sec)	Number of time-windows activated over 100 generations
10 %	4322	175
20 %	3119	117
30 %	4289	176
40 %	3687	175
50 %	3572	176
100 %	9156	175

Comparison of the results presented in Table 5-7 with those obtained in Table 5-5 for the main set of experiments show that the use of unrepaired solutions and less stringent penalties have no discernable effect on the results. Computing times did not improve and neither did the number of images scheduled. In fact, in some cases it was noted that the algorithm stagnated after approximately 50 generations and in two cases even terminated prematurely because of this. Its appears that penalizing a solution in proportion to its "degree" of infeasibility is counterproductive in that it leads to less diversity in the population of solutions.

5.5 A note about optimality

Given the lack of standardized test data against which to compare the performance of the algorithm, it is difficult to evaluate the experimental results in terms of optimality. In an effort to investigate this issue, the algorithm was run for 1000 generations for each of the cases of one through six satellites. Constraint Group D was used and the algorithm was allowed an infinite amount

of time and 1000 stall generations so that it would not terminate before 1000 generations have been reached. The results obtained are presented in Table 5-8 that provides a comparison between the results obtained for 1000 generations and those obtained for 100 generations.

From Table 5-8 it is evident that there is not a marked difference between the total number of time-windows activated over 1000 generations compared to the total over 100 generations. We also note the percentage of time-windows activated over 1000 and 100 generations is very similar. There is however a significant difference in the fitness value between the 1000 and 100 generation cases. While the number of time-windows activated remains more or less the same, the fitness value improves significantly. It appears that the algorithm spends approximately the first 50 generations improving the fitness value by adding time-windows to the sequence, after that it seems to seek out more valuable time-windows while keeping the number of time-windows the same. This observation seems to be borne out by Figure 5-27 and Figure 5-28 respectively for the one and six satellites.

The fact that the number of time-windows scheduled remains more or less the same regardless of the number of iterations may indicate that the algorithm has reached a near optimal solution, at least in terms of the number of time-windows to be scheduled. Although there is an improvement in fitness value, the rate improvement eventually becomes very slow taking about 100 generations between successive improvements in the later stages of the experiment. This may be an indication that optimality is being approached.

Finally it is noted that, if all the images are scheduled, a theoretical upper bound on the objective function value would be 358 453 for the case of six satellites. The achieved value of 260 559 achieved over 1000 generations represents 72.7 % of the upper bound. If we remember that the collective on-board memory capacity for all satellites is limited to approximately 70 % of the total that would be required to acquire all images, it seems that optimality is not far away.

Table 5-8: Comparison of results obtained for constraint Group D over 1000 generations with average results obtained over 100 generations

No of Satellites	1000 Generations				100 Generations			
	CPU Time (sec)	No of Time-Windows Activated	Proportion of Time-Windows Activated (%)	Objective Function Value	CPU Time (sec)	No of Time-Windows Activated	Proportion of Time-Windows Activated (%)	Objective Function Value
1	12789	186	75.3	313220	1386	179	72.5	286073
2	26999	184	74.8	297024	2264	177	72.0	274539
3	24393	175	71.4	273554	2996	175	71.4	266574
4	14074	176	72.1	267662	3222	177	72.5	259745
5	51637	177	72.8	250186	3244	178	73.3	244029
6	59870	178	73.6	260559	3685	175	72.3	249578

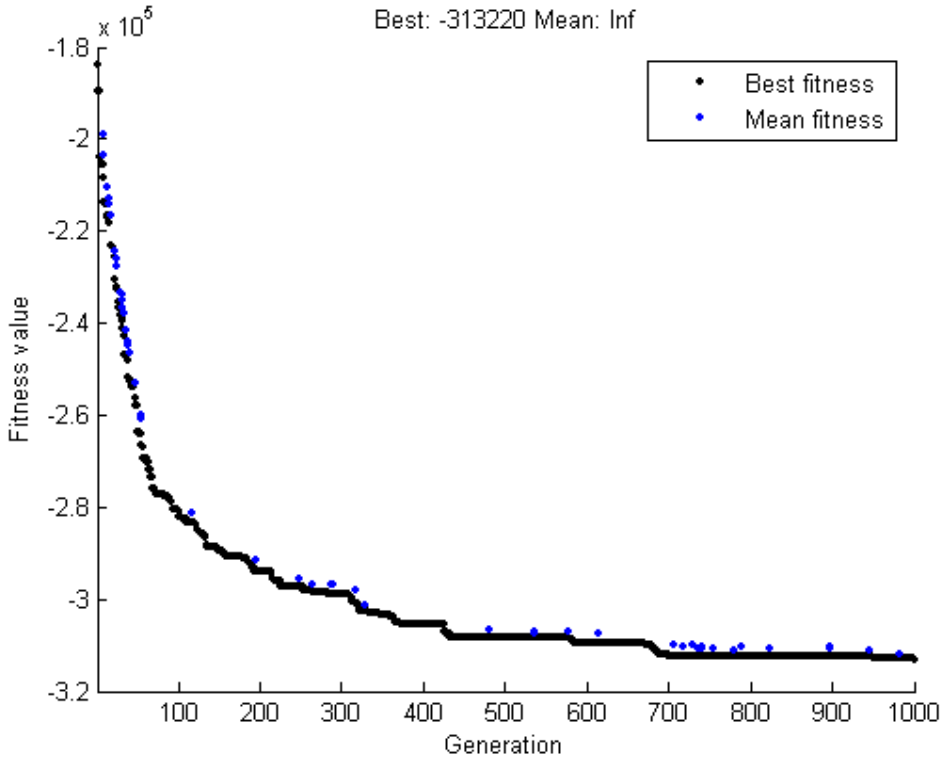

Figure 5-27: Genetic algorithm results for one satellite over 1000 generations

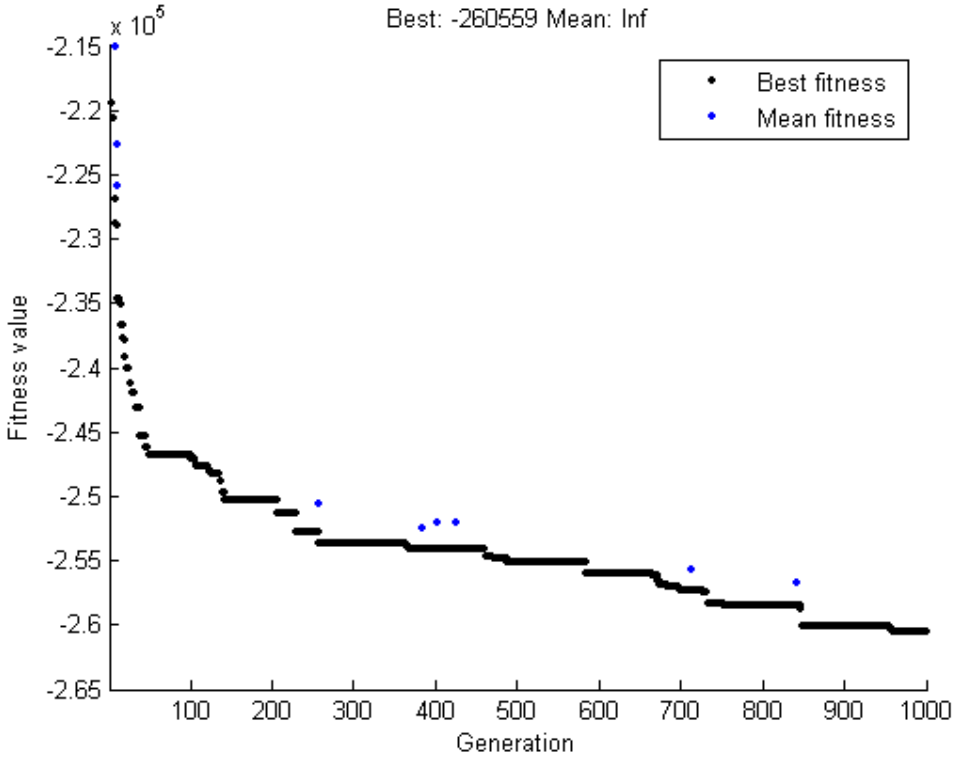

Figure 5-28: Genetic algorithm results for six satellites over 1000 generations

5.6 Conclusions and suggestions for further study

The results presented in the previous sections yield a number of interesting insights, some expected and others not:

- Although simulated annealing and tabu search performs better than the genetic algorithm in terms of computing time, the genetic algorithm performs much better in terms of the number of time-windows activated.

- The coding of the imaging times-windows instead of the actual images as a binary string used in all three algorithms appears to be effective on the whole, yielding consistent results. In terms of the number of time-windows activated

this representation and the genetic algorithm performs very well and consistently activates a very high number of time-windows.

- The solution representation and genetic algorithm performs very well in all cases except for when multiple imaging opportunities are present for the same image. In the former cases computing times range around two to three minutes or so, making for a practical system. In the latter case, a marked inefficiency is introduced, leading to computing times as high as 6000 seconds which is clearly not practical, given that a satellite completes its orbit in about 5400 seconds.

- The few experiments that were conducted using different reproductive options for the genetic algorithm did not yield any real improvements in computing time. Apparent improvements in computing time always came at the expense of a lower number of time-windows activated.

- It appears that the algorithm approaches optimality, especially if it is allowed to generate 1000 generations of solutions. However, the computing time in this case approaches 18 hours which make the algorithm impractical.

It can be concluded that viewing the SCoTWOP as related to the dual of the VRPMTW provides a valuable framework, primarily because it leads to a simple representation of a solution as a binary genotype that leads to an effective and, for the most part efficient, solution process. Complications arise when multiple imaging opportunities are present for the same image and an investigation into a solution for this problem may be worth pursuing, in particular since this is probably the single most important complication that discerns the SCoTWOP from all other SSPs.

In this regard consideration should perhaps be given to a scheme in which the problem is split into two parts which can be solved separately. The first problem would be dealing with selecting optimal time-windows for images with multiple time-windows and the second problem with selecting optimal time-windows for the majority of images that have only one time-window. In other words separate the images that are only subject to constraint groups A, B, and C from those that

are also subject to constraint group D. A solution arising from such a separation may not be optimal, but that is true for most heuristics in any case. The quality of such a solution may be improved if a suitable number of the neighbouring images (in the time-window sense) of the images that have multiple time-windows are included in the first problem.

6 REFERENCES

Aurdal, L., (2003), *Lars Aurdal's Home Site*, http://www.aurdalweb.com/, last accessed on November 15, 2008.

Barbulescu, L. Whitley, D.L., and Howe, A.E., (2004), *Leap Before You Look: An Effective Strategy in an Oversubscribed Scheduling Problem*, in American Association for Artificial Intelligence, Proceedings of the Nineteenth National Conference on Artificial Intelligence, July 25-29, 2004, San Jose, California Published by The AAAI Press, Menlo Park, California.

Barbulescu, L., Howe, A.E., Watson, J.P., and Whitley, D., (2002), *Satellite Range Scheduling: A Comparison of Genetic, Heuristic and Local Search*, in Proceedings of 7th International Conference on Parallel Problem Solving from Nature - PPSN VII, Granada, Spain, September 7-11, 2002, Springer Berlin / Heidelberg.

Barbulescu, L., Watson, J.P., Whitley, L.D., and Howe, A.E., (2004), *Scheduling space-ground communications for the Air Force Satellite Control Network*, Journal of Scheduling, Vol. 7, no. 1, pp. 7-34, Jan.-Feb 2004.

Bensana, E; Verfaillie, G.; Agnèse, J.C.;. Bataille, N.G.; and Blumstein D., (1996), *Exact and Inexact Methods for the Daily Management of an Earth Observation Satellite*, in SpaceOps96, 1996.

Burrowbridge, S.E., (1999), *Optimal Allocation of Satellite Network Resources,* Thesis submitted in partial fulfillment of the requirements of the degree of Master of Science in Mathematics, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, U.S.A., December 17, 1999.

Cordeau, J.F. and Laporte, G., (2005), *Maximizing the value of an Earth observation satellite orbit*, Journal of the Operational Research Society, Vol. 56, pp 962–968.

De Jong, C., Kant, G., and Van Vliet, A., (1997), *On Finding Minimal Route Duration in the Vehicle Routing Problem with Multiple Time-windows*, Working paper, Department of Computer Science, Utrecht University, The Netherlands.

Desrocher, M., Lenstra, J.K., Savelsbergh, M.W.P., and Soumis, F., (1998), *Vehicle Routing with Time-windows: Optimization and Approximation*, in: Vehicle Routing: Methods and Studies, Golden B.L., Assad A.A. (eds.) North-Holland, Amsterdam, pp 65-84.

Dimitrov, S.A., and Galiber, F., (1998), *Visualizing Satellite Mission Planning Problems*, Proceedings of the ILOG Users' Conference, Paris, France.

Dungan, J., Frank, J., Jónsson, A., Morris, R., and Smith, D.E., (2002), *Advances in Planning and Scheduling of Remote Sensing Instruments for Fleets of Earth Observing Satellites,* in Proceedings of the International Workshop on Future Intelligent Earth Observing Satellites. Denver, Colorado, USA. November 10-14, 2002.

Frank, J., Jónsson, A., Morris, R., and Smith D.E., (2001), *Planning and Scheduling for Fleets of Earth Observing Satellites,* in Proceedings of the 6th International Symposium on AI, Robotics and Automation for Space.

Globus, A., Crawford, J., Lohn, J., and Morris, R., (2002), *Scheduling Earth Observing Fleets using Evolutionary Algorithms: Problem Description and Approach*, in Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space.

Globus, A., Crawford, J., Lohn, J., and Pryor, A., (2003), *Scheduling Earth Observing Satellites with Evolutionary Algorithms*, in Conference on Space Mission Challenges for Information Technology (SMC-IT).

Globus, A., Crawford, J., Lohn, J., and Pryor, A., (2004), *A Comparison of Techniques for Scheduling Earth Observing Satellites*, Proceedings of the Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI-2004), San Jose, July 2004.

Harrison, S.A., and Price, M.E., (1999), *Task Scheduling for Satellite Based Imagery*, Defence Evaluation and Research Agency, Malvern, United Kingdom.

Howe, A.E., Whitley, D.L., Barbulescu, L., and Watson, J.P., (2000), *Mixed Initiative Scheduling for the Air Force Satellite Control Network*, in Second International NASA Workshop on Planning and Scheduling for Space, March 2000.

Kay, M.G., (2004), *Matlog: Logistics Engineering Matlab Toolbox*, http://www.ise.ncsu.edu/kay/matlog/index.htm, last accessed on November 15, 2008.

Khatib, L., Frank, J., Smith, D.E., Morris, R., and Dungan, J., (2003), *Interleaved Observation Execution and Rescheduling on Earth Observing Systems*, in Proceedings of the ICAPS-03 Workshop on Plan Execution, Trento, Italy, June 2003.

Kitts, C.A., (1994), *The Integrated Control Of Satellite Payloads*, in Proceedings of the SPIE Small-Satellite Technology and Applications III Conference, Rome, Italy, 26-30 Sept. 1994.

Lemaître, M., Verfaillie, G., Jouhaud, F., Lachiver, J., and Bataille, N., (2000), *How to Manage the New Generation of Agile Earth Observation Satellites?*, in Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space.

Muccio, J., Olson, B., and Starego, S., (1999), *A Generic Imagery Satellite Scheduling Process*, OR 680 Operations Research Seminar, Operations Research Department, George Mason University, Fairfax, Virginia, USA, May 12, 1999.

Pemberton, J.C., and Galiber, F., (2000), *A Constraint-Based Approach to Satellite Scheduling*, Workshop on Constraint Programming and Large Scale Discrete Optimization, Center for Discrete Mathematics and Theoretical Computer Science Rutgers University, Piscataway, New Jersey, United States.

Pemberton, J.C., (1999), *Solving Over Constrained Satellite Remote Sensing Scheduling Problems*, Electronic Notes in Discrete Mathematics, Volume 4, pp 54, CP99, Post-Conference Workshop on Large Scale Combinatorial Optimisation and Constraints, January 2000..

Pemberton, J., (2000), *Towards Scheduling Over Constrained Remote Sensing Satellites,* in Proceedings of the 2d International Workshop on Planning and Scheduling for Space.

Pemberton, J.C., and Greenwald, L.G., (2002), *On the Need for Dynamic Scheduling of Imaging Satellites*, in Proceedings of the International Workshop on Future Intelligent Earth Observing Satellites (FIEOS).

Ruan, Q., Tan, Y., He, R., and Chen, Y., (2005), *Simulation-Based Scheduling for Photo-Reconnaissance Satellite,* in Proceedings of the 2005 Winter Simulation Conference.

Reeves, C. R. (editor), (1993), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford.

Vasquez, M., and Hao, J.K., (2000), *A "Logic-Constrained" Knapsack Formulation and a Tabu Algorithm for the Daily Photograph Scheduling of an Earth Observation Satellite*, Journal of Computational Optimization and Applications.

Vasquez, M., and Hao, J.K., (2002), *Upper Bounds for the SPOT 5 Daily Photograph Scheduling Problem*, Submitted to Journal of Combinatorial Optimization.

Verfaillie, G., Lemaitre, M., Bataille, N., and Lachiver, J.M., Management of The Mission of Earth Observation Satellites: Informal Description of the Global Problem, May 22, 2002, http://www.ime.usp.br/~fr/roadef/informal.pdf, last accessed on May 28, 2009.

Wertz, J.R., Anderson, W.J., (editors), (1999), *Space Mission Analysis and Design*, Third Edition, Microcosm Press, El Segundo California.

Wolfe, W.J., and Sorensen, S.E., (2000), *Three Scheduling Algorithms Applied to the Earth Observing Systems Domain*, Management Science Volume 46 , Issue 1.

APPENDIX A: ABBREVIATIONS AND ACRONYMS

BPP	Bin Packing Problem
CNES	Centre National d'Etudes Spatiales
CRISP	Centre for Remote Imaging, Sensing and Processing, National University of Singapore
CVRP	Capacitated Vehicle Routing Problem
DMC	Disaster Monitoring Constellation
EO	Earth Observation
EOS	Earth Observation Satellite
LEO	Low Earth Orbit
NASA	National Aeronautics and Space Administration
NP	Nondeterministic polynomial-time
SAR	Synthetic Aperture Radar
SCoTWOP	Satellite Constellation Time-Window Optimization Problem
SPOT	Système Pour l'Observation de la Terre
SSP	Satellite Scheduling Problem
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem

VRPMTW	Vehicle Routing Problem with Multiple Time-windows
VRPTW	Vehicle Routing Problem with Time-windows

APPENDIX B: SAMPLE TEST DATA SET

Table B-1 below contains the test data used for experiments relating to the activation of 250 image windows for a constellation of six satellites subject to the full set of constraints. The various parameters are described in Paragraph 4.2.4.

Table B-1: Sample Test Data Set

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
1	0	600	0	0	0	-	-	-
2	3000	3042	-17	15	150	1	2	2010

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
3	3021	3062	5	15	150	2	2	1560
4	3090	3141	26	6	60	3	3	1158
5	3159	3217	-20	14	140	4	5	2016
6	3222	3268	-12	7	70	5	2	833
7	3252	3292	-28	14	140	6	1	1946
8	3291	3324	3	6	60	7	5	618
9	3314	3359	-2	10	100	8	2	1330
10	3377	3412	-11	14	140	9	5	2072
11	3382	3430	-3	6	60	10	1	1032
12	3424	3466	17	11	110	11	6	1991
13	3462	3508	9	15	150	12	3	2010
14	3500	3530	-10	9	90	13	5	1260
15	3500	3548	-14	13	130	14	2	2431
16	3551	3608	-16	6	60	15	3	1068
17	3611	3647	15	15	150	16	1	1860
18	3661	3703	-20	14	140	17	2	2184
19	3732	3792	6	9	90	18	5	1485
20	3789	3842	-17	14	140	19	3	1932
21	3832	3864	-4	11	110	20	6	1232
22	3883	3936	-25	12	120	21	4	1500
23	3941	3972	15	6	60	22	5	684
24	3989	4044	-22	5	50	23	1	840
25	4070	4115	-14	5	50	24	3	570
26	4091	4133	30	11	110	25	1	2145
27	4104	4139	-16	7	70	26	4	833

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
28	4133	4189	24	9	90	27	3	1341
29	4200	4253	26	12	120	28	1	2376
30	4257	4310	-27	7	70	29	2	1085
31	4310	4348	-15	8	80	30	5	1280
32	4357	4393	-27	13	130	31	1	2093
33	4370	4412	-13	6	60	32	1	840
34	4397	4444	7	13	130	33	6	1430
35	4446	4486	-18	10	100	34	5	1590
36	4482	4521	12	9	90	35	6	1296
37	4501	4556	11	7	70	36	4	1001
38	4535	4577	9	15	150	37	6	2610
39	4591	4643	-5	15	150	38	5	2235
40	4620	4662	-28	6	60	39	5	1044
41	4681	4724	-9	10	100	11	1	1030
42	4746	4792	29	15	150	41	6	2550
43	4817	4861	-6	12	120	42	1	1380
44	4855	4901	2	10	100	43	5	1890
45	4908	4939	-8	12	120	44	1	1692
46	4909	4950	-19	12	120	45	2	1788
47	4971	5015	9	11	110	46	2	1903
48	5005	5046	20	12	120	47	1	2244
49	5075	5107	3	12	120	48	3	2040
50	5134	5184	-26	5	50	49	5	810
51	5155	5197	10	7	70	50	2	784
52	5206	5240	0	8	80	51	5	1304

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
53	5220	5258	-8	14	140	52	5	2366
54	5282	5321	-3	12	120	53	4	1524
55	5337	5370	-29	7	70	54	1	1316
56	5367	5424	-22	14	140	55	6	1400
57	5441	5488	3	9	90	56	3	900
58	5500	5549	-16	14	140	57	5	1414
59	5579	5617	-19	5	50	58	5	895
60	5647	5695	-24	10	100	59	3	1230
61	5716	5764	4	14	140	60	1	1820
62	5750	5801	-8	8	80	61	3	1464
63	5813	5860	22	7	70	62	3	1064
64	5842	5893	27	12	120	63	2	2256
65	5901	5956	-5	6	60	64	5	882
66	5943	5977	22	5	50	65	2	655
67	5974	6019	-8	8	80	66	2	928
68	5999	6059	-13	14	140	67	1	2072
69	6046	6079	19	13	130	68	3	1833
70	6066	6113	-6	10	100	69	3	1760
71	6132	6175	-5	15	150	70	3	1710
72	6192	6223	6	11	110	71	6	1100
73	6250	6305	-20	9	90	72	5	1278
74	6329	6374	-19	15	150	73	2	2760
75	6347	6404	4	13	130	74	3	1963
76	6401	6439	-14	13	130	75	2	1300
77	6448	6491	-12	5	50	76	2	800

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
78	6461	6521	19	11	110	77	5	1232
79	6534	6579	9	14	140	78	4	1582
80	6559	6618	-8	15	150	79	6	1695
81	6592	6625	13	6	60	80	6	1110
82	6642	6702	29	5	50	81	3	665
83	6717	6766	-2	8	80	82	2	960
84	6740	6788	-2	15	150	83	6	1875
85	6763	6797	10	8	80	84	3	816
86	6812	6871	-4	5	50	85	1	675
87	6872	6902	-1	7	70	86	1	1169
88	6927	6963	-4	12	120	87	5	1800
89	6979	7031	-28	9	90	88	3	990
90	7036	7096	13	13	130	89	5	1885
91	7110	7156	-1	14	140	90	2	1722
92	7128	7178	-10	5	50	91	6	810
93	7203	7255	-12	15	150	92	4	2670
94	7281	7319	20	8	80	93	2	1224
95	7300	7332	-13	12	120	94	3	1596
96	7321	7362	10	11	110	95	3	1232
97	7357	7393	-13	10	100	96	2	1630
98	7409	7459	-28	13	130	97	5	1651
99	7484	7536	-27	5	50	98	3	815
100	7506	7545	-7	10	100	99	6	1510
101	7526	7582	13	15	150	100	6	1530
102	7552	7583	28	11	110	101	5	1331

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
103	7602	7644	21	5	50	102	1	660
104	7627	7683	27	9	90	103	3	1539
105	7694	7752	-25	13	130	104	3	1859
106	7737	7788	-25	5	50	105	1	675
107	7809	7849	-25	15	150	106	4	1665
108	7847	7889	-2	14	140	107	6	2520
109	7879	7917	27	14	140	108	6	2198
110	7902	7943	5	6	60	109	1	1164
111	7951	7996	-29	6	60	110	3	780
112	7993	8040	9	12	120	111	2	2076
113	8019	8058	8	12	120	112	1	2124
114	8080	8128	-8	5	50	113	6	865
115	8102	8140	5	13	130	114	3	2509
116	8121	8171	-25	15	150	42	1	1665
117	8177	8226	-26	15	150	116	2	2685
118	8227	8266	-7	12	120	117	2	1956
119	8283	8330	21	10	100	118	5	1950
120	8357	8398	-5	13	130	7	3	2041
121	8375	8417	6	5	50	120	5	630
122	8419	8456	26	8	80	121	4	936
123	8468	8516	6	15	150	122	2	1995
124	8524	8582	7	11	110	123	3	1958
125	8571	8628	-14	11	110	124	1	1507
126	8625	8669	20	11	110	125	2	2024
127	8663	8718	-15	7	70	126	2	1393

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
128	8717	8747	-21	12	120	127	3	1236
129	8769	8814	-12	9	90	128	2	1224
130	8786	8841	0	11	110	129	5	1540
131	8864	8897	3	6	60	130	1	918
132	8871	8905	-19	8	80	131	6	1568
133	8883	8916	30	8	80	132	6	944
134	8926	8962	-16	5	50	133	5	685
135	8983	9038	-6	13	130	134	4	1664
136	9036	9077	12	5	50	135	3	990
137	9098	9157	24	13	130	37	1	2054
138	9142	9182	-16	11	110	137	4	1397
139	9177	9232	21	7	70	138	4	784
140	9212	9248	-2	12	120	139	5	1932
141	9250	9299	-6	15	150	140	4	2775
142	9305	9350	17	12	120	141	1	1920
143	9344	9374	-15	6	60	142	2	1200
144	9357	9388	-25	11	110	143	5	1705
145	9394	9439	15	9	90	144	3	1584
146	9450	9484	-26	15	150	145	5	1995
147	9506	9536	-12	11	110	146	4	1342
148	9536	9566	-17	13	130	28	6	2405
149	9555	9601	-12	15	150	148	3	2310
150	9631	9675	6	6	60	149	1	876
151	9700	9742	-28	14	140	150	1	1750
152	9748	9802	26	14	140	151	2	2142

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
153	9803	9851	-8	13	130	152	6	1924
154	9857	9901	-29	11	110	153	6	1210
155	9928	9980	6	7	70	154	1	1288
156	9961	9994	12	6	60	155	1	834
157	10015	10056	-28	13	130	156	1	2275
158	10065	10104	14	5	50	157	2	540
159	10086	10126	16	10	100	158	1	1740
160	10102	10144	-10	13	130	159	2	1443
161	10124	10156	-22	15	150	160	3	2445
162	10177	10229	-27	13	130	15	4	1482
163	10252	10305	7	15	150	162	6	1995
164	10316	10368	7	15	150	163	5	2325
165	10363	10417	-11	5	50	164	2	540
166	10439	10493	15	8	80	165	3	1312
167	10512	10554	18	11	110	166	1	1496
168	10580	10634	1	7	70	167	6	1057
169	10633	10672	-1	6	60	168	4	714
170	10661	10721	17	8	80	169	4	1408
171	10724	10784	5	9	90	170	3	1377
172	10798	10830	-23	10	100	171	2	1750
173	10842	10872	-7	9	90	172	2	1494
174	10868	10924	-12	11	110	173	3	1914
175	10922	10973	-3	7	70	174	4	1302
176	10967	11001	0	10	100	175	5	1520
177	11026	11057	-28	10	100	176	2	1250

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
178	11041	11073	23	6	60	177	4	1170
179	11046	11106	24	5	50	178	4	855
180	11111	11149	-18	7	70	179	4	980
181	11158	11188	-22	12	120	180	5	2376
182	11192	11223	13	6	60	181	2	1140
183	11206	11257	-9	13	130	182	3	1950
184	11276	11333	14	12	120	183	5	2340
185	11333	11388	-5	5	50	184	4	800
186	11359	11405	26	8	80	185	4	936
187	11427	11462	-1	6	60	186	1	1128
188	11435	11472	29	5	50	187	5	750
189	11488	11538	-25	8	80	188	2	1400
190	11542	11588	-20	15	150	189	6	1740
191	11582	11622	30	8	80	190	3	1176
192	11615	11669	-6	11	110	191	2	2134
193	11679	11728	13	5	50	192	5	795
194	11747	11801	-21	11	110	193	2	1375
195	11787	11842	5	9	90	194	5	1584
196	11859	11890	-10	8	80	195	4	1024
197	11903	11960	-21	12	120	196	1	1812
198	11938	11992	5	6	60	197	3	990
199	11991	12047	5	14	140	198	6	1568
200	12064	12122	2	14	140	199	6	2156
201	12102	12157	-1	15	150	200	1	2490
202	12173	12203	16	7	70	201	1	728

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
203	12199	12253	-22	5	50	202	5	560
204	12237	12283	-9	9	90	203	2	1683
205	12258	12290	-8	10	100	204	6	1100
206	12273	12303	0	10	100	205	4	1100
207	12320	12367	-9	9	90	206	2	1431
208	12371	12413	30	5	50	207	3	970
209	12420	12467	-1	13	130	208	2	1729
210	12454	12492	-5	11	110	209	2	1364
211	12464	12519	26	5	50	210	3	580
212	12525	12576	-5	9	90	211	3	1782
213	12573	12619	-29	8	80	212	6	848
214	12604	12652	-7	10	100	213	3	1840
215	12629	12670	11	5	50	214	6	815
216	12676	12720	-9	5	50	215	2	920
217	12698	12738	0	12	120	216	1	1848
218	12716	12764	10	7	70	217	4	749
219	12748	12798	12	8	80	218	3	1344
220	12815	12861	22	8	80	219	2	856
221	12845	12904	-2	5	50	220	2	660
222	12908	12958	-13	12	120	221	1	2040
223	12960	13013	-21	6	60	222	6	1158
224	13026	13060	-20	9	90	223	1	1575
225	13079	13139	8	13	130	224	6	1560
226	13141	13172	14	12	120	225	1	1800
227	13200	13235	-22	6	60	226	5	816

Time-window #	Time-window Start (sec)	Time-window End (sec)	Lookangle (deg)	Imaging Duration (sec)	Memory Requirement (MB)	View #	Satellite #	Image Value ($)
228	13241	13280	-1	8	80	227	5	1232
229	13265	13296	11	13	130	228	1	2314
230	13282	13314	17	7	70	229	6	875
231	13302	13339	-19	14	140	230	4	1582
232	13327	13359	7	12	120	231	1	1356
233	13384	13433	8	7	70	232	2	889
234	13451	13505	8	8	80	233	4	816
235	13499	13552	12	9	90	234	3	1368
236	13543	13577	13	11	110	235	5	1628
237	13600	13648	-1	12	120	236	3	2064
238	13669	13705	-13	15	150	237	5	1575
239	13732	13788	7	11	110	238	5	2123
240	13779	13832	-25	9	90	239	1	1710
241	13827	13875	-11	12	120	240	5	1452
242	13905	13964	-29	9	90	241	5	1602
243	13974	14009	15	9	90	242	1	1782
244	13985	14015	26	10	100	243	2	1410
245	14027	14064	-17	13	130	244	6	1859
246	14094	14136	13	13	130	245	4	2132
247	14110	14151	23	5	50	246	3	925
248	14131	14164	-28	15	150	247	3	2220
249	14148	14181	17	13	130	248	6	2288
250	14181	14236	-4	8	80	249	3	1008
251	14219	14253	0	0	0	-	-	-

APPENDIX C: MATLAB® COMPUTER CODE

Code for DataReader.m

```
clear all
global n TW q Q value C cap twin st satno viewno nosats
n = xlsread('Bdata6','n')
lookangle = xlsread('Bdata6','lookangle')
ld = xlsread('Bdata6','ld')
TW=xlsread('Bdata6','TW')
st = []
q = xlsread('Bdata6','q')
Q = xlsread('Bdata6','Memory')
value = xlsread('Bdata6','Value')
satno = xlsread('Bdata6', 'satno')
nosats = xlsread('Bdata6', 'nosats')
viewno = xlsread('Bdata6', 'viewno')
maxTC = 100000
Early = TW(:,1)
Late = TW(:,2)
for i=1:n;
    for j = 2:n
        if i == 1
            setuptime(i,j)=abs(lookangle(i)-lookangle(j));
        elseif  satno(j)~= satno(i)
            setuptime(i,j)=9999999;
        else
        setuptime(i,j)=abs(lookangle(i)-lookangle(j));
        end
    end
end

setuptime
for i=1:n;
    for j = 2:n
        if Early(j) < Early (i) & Late(j) < Early(i);
           traveltime (i,j)=999999;
        else
            traveltime (i,j)=0;
        end
    end
end
traveltime;
C = max(setuptime, traveltime)
TW
cap = {q,Q}
twin = {ld,TW,st}
```

Code for SequenceBuilder.m

```
function Population = SequenceBuilder(GenomeLength, FitnessFcn, options)

global  n TW q Q value C cap twin st satno viewno nosats
Pop = cell(20,1);
tmprte = cell(20,2)
for k=1:20
    XFlg = -2;
    while any (XFlg ~=1)
    % Create a binary vector that has n-2 ones or zeros
    X=[];
    %Generate a random binary vector
    for i=1:n-2
        if rand <= 0.5
           v = 0;
        else
           v=1;
        end
        X=[X v];
    end

    % Fix binary vector if image is allocated to more than one time-window
    for i=1:n-2
      for j=i:n-2
            if j~=i & viewno(j)==viewno(i)
                if X(j)==1 & X(i)==1
                    if rand >= 0.5
                    X(j)=0
                    else
                    X(i)=0
                    end
                end
            end
      end
    end

    % Split the binary vector in accordance with the satellite number of
    % the respective time-window
    a=[2:n-1]
    b=satno(2:n-1,1)'
    c=X.*b
    for r=1:nosats
        for i=1:n-2;
            if c(i)==r
               d(i,r)=1;
            else
               d(i,r)=0;
            end
        end
    end

    for r=1:nosats
        e=d(:,r)';
        tmprte{k,r}=nonzeros([1 e.*a 1])'
    end
    pp=[1 X];
    [TC,XFlg,out] = rteTC(tmprte{k,r},C,cap,twin,[]);
    XFlg
```

```
    if any (XFlg ~=1)
       continue
       else
          Pop{k}=pp;
       end
    end

    tmprte;
    Population = cell2mat(Pop);
    viewno;
end
```

Code for rteTC.m

```
function [TC,XFlg,out] = rteTC(rte,varargin)
global n TW q Q value C cap twin st
%RTETC Calculate total cost of route with time-windows.
%            TC = rteTC(rte,C)
% [TC,XFlg,out] = rteTC(rte,C,cap,twin,rtefeas)
%               = rteTC(rte),  use arguments C,twin,... from previous call
%                              and do not check input for errors
%    rte = vector of single-route vertices
%        = m-element cell array of m routes
%          (to get sum(TC) as output, use vector rte = [rte{:}] as input)
%      C = n x n matrix of costs between n vertices
%     TC = m-element vector of route costs, where
%          TC(i) = Inf if route i is infeasible
%
% Optional input and output arguments used to determine route feasibility:
%    cap = {q,Q} = cell array of capacity arguments, where
%              q = n-element vector of vertex demands, with depot q(1) = 0
%              Q = maximum route load
%   twin = {ld,TW,st} = cell array of time-window arguments, where
%             ld = n or (n+1)-element vector of loading/unloading
%                  timespans, where
%                     ld(rte(1))   = load at depot
%                     ld(n+1)      = unload at depot, if rte(1) == rte(end)
%                = scalar of constant values "ld" for rte(2) ... rte(end)
%                  and 0 for rte(1); or rte(2) ... rte(end-1) and 0 for
%                   rte(end), if rte(1) == rte(end)
%                = 0, default
%             TW = n or (n+1) x 2 matrix of time-windows, where
%                     TW(i,1)      = start of time-window for vertex i
%                     TW(i,2)      = end of time-window
%                     TW(rte(1),:) = start time-window at depot
%                     TW(n+1,:)    = finish time-window at depot,
%                                    if rte(1) = rte(end)
%                = (n+1)-element cell array, if multiple windows, where
%                     TW{i}        = (2 x p)-element vector of p window
%                                    (start,end) pairs
%             st = (optional) m-element vector of starting times at depot
%                = TW(1,1) or min(TW{1}), default (earliest starting time)
%rtefeas = {'rtefeasfun',P1,P2,...} = cell array specifying user-defined
%          function to test the feasibility of a single route (in addition
%          to time-windows, capacity, and maximum cost), where RTETC
%          argument out(i) along with user-specified arguments P1,P2,...
%          are passed to function and a logical value should be returned:
%                isfeas = RTEFEASFUN(out(i),P1,P2,...)
%        = {'maxTCfeas',maxTC} is a predefined route feasibility function
%          to test if the total cost of a route (including loading/
%          unloading times "ld") exceeds the maximum waiting time "maxTC"
%          (see below for code)
%XFlg(i) = exitflag
%        =  1, if route is feasible
%        = -1, if infeasible due to capacity
%        = -2, if infeasible due to time-windows
%        = -3, if infeasible due to user-defined feasibility function
%    out = m-element struct array of outputs
% out(i) = output structure with fields:
%        .Rte     = route indices, rte{i}
%        .Cost    = cost from vertex j-1 to j,
%                   Cost(j) = C(r{i}(j-1),r{i}(j)) and Cost(1) = 0
%                 = drive timespan from vertex j-1 to j
%        .Demand  = demands of vertices on route, q(rte{i})
%        .Arrive  = time of arrival
```

```
%        .Wait    = wait timespan if arrival prior to beginning of window
%        .Start   = time loading/unloading started (starting time for
%                   route is "Start(1)")
%        .LD      = loading/unloading timespan, ld(rte{i})
%        .Depart  = time of departure (finishing time is "Depart(end)")
%        .Total   = total timespan from departing vtx j-1 to depart. vtx j
%                   (= drive + wait + loading/unloading timespan)
%        .EarlySF = earliest starting and finishing times (default starting
%                   time is "st" and default finish. time is "EarlySF(2)")
%        .LateSF  = latest starting and finishing times
%
% For each route rte{i}, feasibility is determined in the following order:
%    1. Capacity feasibility: SUM(q(rte{i})) <= Q if feasible
%    2. Time-window feasiblity: [TCi,ignore,outi] = RTETC(rte{i},C,twin);
%                               TCi < Inf if feasible
%    3. User defined feasibility: isfeas = RTEFEASFUN(outi,P1,P2,...);
%                                 isfeas == true if feasible
%

% Copyright (c) 1994-2004 by Michael G. Kay
% Matlog Version 8 22-Nov-2004

% Input Error Checking ************************************************

% Set to empty

if nargin < 2
   if isempty(C)
      error('Additional input arguments required for first call.')
   else
      isfirstcall = 0;
   end
else
   isfirstcall = 1;
   if length(varargin) < 4
      [varargin{length(varargin)+1:4}] = deal([]);
   end
   [C,cap,twin,rtefeas] = deal(varargin{:});
   [q,Q,ld,TW,st] = deal([]);
end

m = 1;
if iscell(rte), m = length(rte); end
TC = Inf * ones(m,1);               % All routes initialized to infeasible
XFlg = ones(m,1);                   % All flags initialized to feasible

[n,nC] = size(C);

% Route
if isfirstcall & ~isempty(rte) & (~(isreal(rte) | iscell(rte)) | ...
      (~iscell(rte) & (min(size(rte)) ~= 1 | ...
      any(rte(:) < 1 | rte(:) > n))) | ...
      (iscell(rte) & (any(cellfun('prodofsize',rte) ~= ...
      cellfun('length',rte)) | any([rte{:}] < 1 | [rte{:}] > n))))
   error('"rte" not a valid route.')
end

% Cost
if isfirstcall
   if n ~= nC
      error('C must be a square matrix.')
   elseif any(any(C<0))
      error('C must be a non-negative matrix.')
   elseif any(diag(C) ~= 0)
```

```
      error('C must have zeros along its diagonal.')
   end
end

% Capacity
if isfirstcall && ~isempty(cap)
   if ~iscell(cap) || length(cap(:)) ~= 2
      error('"cap" must be a two element cell array.')
   end
   q = cap{1}; Q = cap{2};
   if length(q(:)) ~= n
      error(['"q" must be an ',num2str(n),'-element vector.'])
   elseif q(1) ~= 0
      error('Depot''s demand, q(1), should equal 0.');
   elseif length(Q(:)) ~= 1 || Q < 0
      error('Q must be a nonnegative scalar.')
   elseif any(q > Q)
      error('Elements of "q" can not be greater than Q.')
   end
end

% Time-window
if isfirstcall && ~isempty(twin)
   if ~iscell(twin) || length(twin(:)) < 1 || length(twin(:)) > 3
      error('"twin" must be a one or three element cell array.')
   end
   ld = twin{1};
   if ~isempty(ld), ld = ld(:)'; end
   if length(twin) > 1, TW = twin{2}; else TW = []; end
   if ~isempty(TW) && ~iscell(TW), TW = padmat2cell(TW); end
   if length(twin) > 2, st = twin{3}; st = st(:); else st = []; end

   if ~isempty(ld) && all(length(ld) ~= [1 n n+1])
      error('Length of "ld" must equal 1, n, or n + 1.')
   elseif ~isempty(TW)
      if iscell(TW), TW = cell2padmat(TW); end
      if all(size(TW,1) ~= [n n+1]) || mod(size(TW,2),2) ~= 0 || ...
            any(all(isnan(TW'))) || ...
            any(any(TW(:,1:2:end-1) > TW(:,2:2:end))) || ...
            any(any(xor(isnan(TW(:,1:2:end-1)),isnan(TW(:,2:2:end)))))
         error('TW not valid time-windows.')
      end
   elseif ~isempty(st) && ~isempty(rte) && length(st(:)) ~= m
      error('Starting time "st" must be an m-element vector.')
   end
end

% Route feasibilty function
if ~isempty(rtefeas) &&  strcmp(rtefeas{1},'maxTCfeas') && ...
      all(isinf(rtefeas{2}))
   rtefeas = [];  % maxTC = Inf => always feasible
end
if isfirstcall && ~isempty(rtefeas)
   if length(rtefeas(:)) < 1
      error('"rtefeas" must be at least a one element cell array.')
   end
   if ~ischar(rtefeas{1})
      error('First element of "rtefeas" must be a string.')
   elseif ~strcmp(rtefeas{1},'maxTCfeas') &&
~exist(lower(rtefeas{1}),'file')
      error(['Function "' rtefeas{1} '" not found.'])
   end
end

% Empty "rte" used for error checking and to store input arguments
```

```
if isempty(rte)
   if nargout > 0, TC = []; XFlg = []; out = []; end
   return
end
% End (Input Error Checking) ****************************************

% Initial timing output structure
if nargout > 2 || ~isempty(rtefeas)
   out = struct('Rte',[],'Cost',[],'Demand',[],'Arrive',[],'Wait',[],...
      'Start',[],'LD',[],'Depart',[],'Total',[],'EarlySF',[],'LateSF',[]);
   out(1:m) = out;
end

% Evaluate each route
for i = 1:m

   if iscell(rte), r = rte{i}; else r = rte(:)'; end

   % 1. Capacity feasibility
   if ~isempty(q) && ~isinf(Q)
      if nargout > 2 || ~isempty(rtefeas), out(i).Demand = q(r); end
      if sum(q(r)) > Q; XFlg(i) = -1; continue, end
   end

   % Calculate route cost
   c = diag(C(r(1:end-1),r(2:end)))';
   if isempty(ld) || all(ld == 0)
      ldi = zeros(1,length(r));
   else
      if length(ld) == 1
         ld = [0 ld*ones(1,n-1)];
         if r(1) == r(end), ld = [ld 0]; end
      end
      ldi = ld(r);
      if r(1) == r(end)
         if length(ld) ~= n+1
            error('Length of "ld" must equal n + 1.')
         end
         ldi(end) = ld(n+1);
      end
   end
   TC(i) = sum(c) + sum(ldi);
   if nargout > 2 || ~isempty(rtefeas)
      out(i).Rte = r;
      out(i).Cost = [0 c];
      if ~isempty(ld), out(i).LD = ldi; end
      out(i).Total = [0 c] + ldi;
   end

   % 2. Time-window feasibility
   if ~isempty(TW)
      B = TW(:,1:2:end-1);
      E = TW(:,2:2:end);
      Br = B(r,:); Er = E(r,:);
      if r(1) == r(end)
         if size(B,1) ~= n+1, error('Length of TW must equal n + 1.'), end
         Br(end,:) = B(n+1,:); Er(end,:) = E(n+1,:);
      end
      if ~isempty(st), sti = st(i); else sti = []; end
      if nargout < 3 && isempty(rtefeas)
         TC(i) = rteTW(c,ldi,Br,Er,sti);
      else
         [TC(i),s,w] = rteTW(c,ldi,Br,Er,sti);
```

```
            if ~isinf(TC(i))
               s_late = latestart(c,ldi,Br,Er);
            else
               s_late = NaN;
            end
            out(i).Arrive = [0 s(2:end)-w(2:end)];
            out(i).Wait = w;
            out(i).Start = s;
            out(i).Depart = s + ldi;
            out(i).Total = [0 c] + w + ldi;
            out(i).EarlySF = [s(1) s(end) + ldi(end)];
            out(i).LateSF = [s_late(1) s_late(1) + TC(i)];
         end
         if isinf(TC(i)), XFlg(i) = -2; continue, end
      end

      % 3. User defined feasibility function
      if ~isempty(rtefeas)
         isfeas = feval(rtefeas{1},out(i),rtefeas{2:end});
         if length(isfeas(:)) ~= 1 % | ~islogical(isfeas)
            error('Output argument "isfeas" must be a scalar logical value.')
         end
         if isfeas == 0,
            TC(i) = Inf; XFlg(i) = -3; continue, end
      end

   end % FOR loop

   % ************************************************************************
   function [TC,s,w] = rteTW(t,ld,B,E,st)
   %RTETW Single route time-window.

   tol = 1e-8;
   n = size(B,1);

   if isempty(st), s = min(B(1,:)); else s = st; end

   s = s + ld(1);
   for i = 2:n  % Forward scan to determine earliest finish time
      s = s + t(i-1) + ld(i);
      Bi = B(i,:) + ld(i);
      if ~any(s + tol >= Bi && s - tol <= E(i,:))
         s = min(Bi(Bi >= s));
         if isempty(s)
            TC = Inf; s = NaN; w = NaN; return
         end
      end
   end
   f = s;

   s = f - ld(n);
   for i = n-1:-1:1  % Reverse scan to determine latest start time for the
      % earliest finish
      s = s - t(i) - ld(i);
      Ei = E(i,:) - ld(i);
      if ~any(s + tol >= B(i,:) & s - tol <= Ei)
         s = max(Ei(Ei <= s));
      end
   end
   TC = f - s;
   if isnan(TC), TC = sum(t) + sum(ld); end % If all Br == -Inf and all Er =
   Inf

   if nargout > 1
```

```
   s = [s zeros(1,n-1)];
   w = zeros(1,n);
   for i = 2:n  % Second forward scan to delay waits as much as possible
      % to the end of the route in case unexpected events occur
      s(i) = s(i-1) + ld(i-1) + t(i-1);
      Bi = B(i,:);
      if ~any(s(i) + tol >= Bi & s(i) + ld(i) - tol <= E(i,:))
         w(i) = s(i);
         s(i) = min(Bi(Bi >= s(i)));
         w(i) = s(i) - w(i);
      end
   end
end

% ************************************************************************
function s = latestart(t,ld,B,E)
%LATESTART Determine latest start time.

tol = 1e-8;
n = size(B,1);

s = max(E(end,:)) - ld(n);
for i = n-1:-1:1  % Reverse scan to determine latest start time
   s = s - t(i) - ld(i);
   Ei = E(i,:) - ld(i);
   if ~any(s + tol >= B(i,:) && s - tol <= Ei)
      s = max(Ei(Ei <= s));
   end
end


% ************************************************************************
function isfeas = maxTCfeas(outi,maxTC)
%MAXTCFEAS Maximum total cost route feasibility function.
% isfeas = maxwaitfeas(outi,maxTC)
%   outi = struct array of outputs from RTETC for single route i
%          (automatically passed to function)
%  maxTC = scalar maximum total cost (including un/loading times) of route
%
% Route is feasible if sum(outi.Total) <= maxTC
%
% This function can be used as a template for developing other
% route feasibility functions.

% Input error check
if ~isnumeric(maxTC) || length(maxTC(:)) ~= 1 || maxTC < 0
   error('"maxTC" must be a nonnegative scalar.')
end

% Feasibility test
if sum(outi.Total) <= maxTC
   isfeas = true;
else
   isfeas = false;
end
```

Code for TabuSat.m

```
%Find an initial feasible route of n-1 vertices.

tic
global  n TW q Q value C cap twin st satno viewno nosats

XFlg = -2
while XFlg ~=1
    X=[]
    for i=1:n-1
        g=rand;
        if g<=0.5
            v = 0;
            X=[X v];
        elseif g> 0.5
            v=1;
            X=[X v];
        end
    end
    % Fix bit string if image is allocated to more than one time-window
    for i=1:n-2
      for s=i:n-2
            if s~=i & viewno(s)==viewno(i)
                if X(s)==1 & X(i)==1
                    h=rand
                    if h >= 0.5
                    X(s)=0
                    else
                    X(i)=0
                    end
                end
            end
      end
    end

    % Determine if the bit string represent a feasible set of image
    % sequences
    a=[2:n]
    b=satno(2:n,1)'
    c=X.*b
    for r=1:nosats
        for i=1:n-1;
            if c(i)==r
                d(i,r)=1;
            else
                d(i,r)=0;
            end
        end
    end
    f=[1 X]
    for r=1:nosats
        e=d(:,r)';
        rte{r}=nonzeros([1 e.*a 1])'
    end
    celldisp (rte)
    [TC,XFlg,out] = rteTC(rte,C,cap,twin,[])
    if XFlg ~=1
        continue
    else
        optc=-sum(f*value)
    end
```

```
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Start the Tabu Search routine
k_max = 100 %was 100
L=5%was 5
% Initialise number of iterations

k=1;

% Initialise logging vectors

E=zeros(1,k_max);
optE=zeros(1,k_max);

% Initialise tabu list

tL=zeros(1,n-1);

% Initially, let the optimal X be equal to the initial estimate of x

optX=X

% Loop

while(k<=k_max)

% Update tabu list

tL=tL-(tL>0);

% Go through neighbourhood looking for
% best allowed solution

currc=optc;
for j=1:n-1
   if(tL(j)==0)
    m=n-1
    XFlg = -2
        while XFlg ~=1
            indeks = round(rand*m)
            if indeks==0
                rand('state',sum(100*clock))
                indeks = round(rand*m)+1
                if indeks==m+1
                    indeks=m
                end
            end
            if X(indeks)==1
                X(indeks)=0
            elseif X(indeks)==0
                X(indeks)=1
            end
             % Fix bit string if image is allocated to more than one time-
window
            for i=1:n-2
              for s=i:n-2
                    if s~=i & viewno(s)==viewno(i)
                        if X(s)==1 & X(i)==1
                            h=rand
                            if h >= 0.5
                            X(s)=0
                            else
```

```
                        X(i)=0
                        end
                    end
                end
              end
            end

            % Determine if the bit string represent a feasible set of image
             % sequences
            a=[2:n]
            b=satno(2:n,1)'
            c=X.*b
            for r=1:nosats
                for i=1:n-1;
                    if c(i)==r
                       d(i,r)=1;
                    else
                       d(i,r)=0;
                    end
                end
            end
            f=[1 X]
            for r=1:nosats
                e=d(:,r)';
                rte{r}=nonzeros([1 e.*a 1])'
            end
            celldisp (rte)
            [TC,XFlg,out] = rteTC(rte,C,cap,twin,[])
            if XFlg ~=1
               continue
            else
               tmpc=-sum(f.*value')
               tmpX =X
            end
        end;
        if(tmpc<currc)
            currc=tmpc;
            tmpj=j;
        end
    end
end

% Update X with best solution from its neighbourhood

X=tmpX

% Log current status

E(k)=currc;
optE(k)=optc;

% Indicate that this permutation is invalid for
% L iterations

tL(tmpj)=L;

% If a globally better solution is found,
% update the corresponding variables

    if(currc<=optc)
      optc=currc;
      optX=tmpX;
    end
    k=k+1;
```

```
end
  E
  optE
  optc
  optX
  celldisp (rte)
  toc
```

Code for SimulanSat.m

```
%Find an initial feasible route of n-1 vertices.(X,optc,M,k_max,alpha,T_0)

tic
global  n TW q Q value C cap twin st satno viewno nosats
k_max=10000
alpha=0.9999
T_0=100
newc=0
optc=0
tmpX=zeros(1,n-1)
XFlg = -2
while any (XFlg ~=1)
    rand('state',sum(100*clock))
    X=[]
    for i=1:n-1
        g=rand;
        if g<=0.5
            v = 0;
            X=[X v];
        elseif g> 0.5
            v=1;
            X=[X v];
        end
    end
     % Fix bit string if image is allocated to more than one time-window
    for i=1:n-2
      for s=i:n-2
            if s~=i & viewno(s)==viewno(i)
                if X(s)==1 & X(i)==1
                    rand('state',sum(100*clock))
                    h=rand
                    if h >= 0.5
                    X(s)=0
                    else
                    X(i)=0
                    end
                end
            end
      end
    end


    a=[2:n]
    b=satno(2:n,1)'
    c=X.*b
    for r=1:nosats
        for i=1:n-1;
            if c(i)==r
                d(i,r)=1;
            else
                d(i,r)=0;
            end
        end
    end
    f=[1 X]
    for r=1:nosats
        e=d(:,r)';
        rte{r}=nonzeros([1 e.*a 1])'
    end
    celldisp (rte)
```

```
    [TC,XFlg,out] = rteTC(rte,C,cap,twin,[])
    if any (XFlg ~=1)
        continue
    else
        optc=-sum(f*value)
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Initialise temperature

  T=T_0;

  % Initialise logging vectors

  E=zeros(1,k_max);
  optE=zeros(1,k_max);
  Cost=zeros(1,k_max);


  % Initially, let the optimal X be equal to the
  % initial estimate of x

  OptX=X;

  % Now calculate the cost of the initial tour.  This is
  % our initial old cost and our initial optimal cost.

  oldc=optc;


  % Initialise number of iterations

  k=1;

  % Loop

  while(k<=k_max)

    % Find a solution in the neighbourhood


    m=n-1
    XFlg = -2
        while any (XFlg ~=1)
            indeks = round(rand*m)
            if indeks==0
                rand('state',sum(100*clock))
                indeks = round(rand*m)+1
                if indeks==m+1
                    indeks=m
                end
            end
            if X(indeks)==1
                X(indeks)=0
            elseif X(indeks)==0
                X(indeks)=1
            end
             % Fix bit string if image is allocated to more than one time-
window
        for i=1:n-2
          for s=i:n-2
```

```
            if s~=i & viewno(s)==viewno(i)
                if X(s)==1 & X(i)==1
                    rand('state',sum(100*clock))
                    h=rand
                    if h >= 0.5
                    X(s)=0
                    else
                    X(i)=0
                    end
                end
            end
      end
    end


a=[2:n]
b=satno(2:n,1)'
c=X.*b
for r=1:nosats
    for i=1:n-1;
        if c(i)==r
            d(i,r)=1;
        else
            d(i,r)=0;
        end
    end
end
f=[1 X]
for r=1:nosats
    e=d(:,r)';
    rte{r}=nonzeros([1 e.*a 1])'
end


celldisp (rte)
[TC,XFlg,out] = rteTC(rte,C,cap,twin,[])
if any (XFlg ~=1)
   continue
   else
   newc=-sum(f.*value')
   tmpX=X
   end
end;
% If this is a better solution, select it

if(newc<=oldc)
    oldc=newc;
    X=tmpX;

     % If in addition this is the optimal solution found so far

    if(newc<=optc)
        optc=newc;
        OptX=tmpX;
    end

% If this is an inferior solution, select it with a certain
% probability

else
  r=rand;
  if(r<exp((oldc-newc)/T))
      oldc=newc;
      X=tmpX;
```

```
        Cost(k)=1;
      end
    end

    % Log current status

    E(k)=oldc;
    optE(k)=optc;

    % Update temperatur

    T=alpha*T;

    % Now increment k

    k=k+1;


  end
E
  optE
  optc
  OptX
  celldisp (rte)
  toc
```

Code for GeneticSat.m

```
function [X,FVAL,REASON,OUTPUT,POPULATION,SCORES] =  GeneticSat
%%   This is an auto generated M file to do optimization with the Genetic
Algorithm and
%    Direct Search Toolbox. Use GAOPTIMSET for default GA options
structure.
tic
%%Fitness function
fitnessFunction = @Satfitness;
%%Number of Variables
nvars = 249;
%Start with default options
options = gaoptimset;
%%Modify some parameters
options = gaoptimset(options,'PopulationType' ,'bitString');
options = gaoptimset(options,'Generations' ,100);
options = gaoptimset(options,'StallTimeLimit' ,Inf);
options = gaoptimset(options,'CrossoverFcn' ,@crossoverspecial2);
options = gaoptimset(options,'MutationFcn' ,{ @mutationgaussian 1  1  });
options = gaoptimset(options,'Display' ,'off');
options = gaoptimset(options,'PlotFcns' ,{ @gaplotbestf });
options = gaoptimset(options,'CreationFcn' ,@SequenceBuilder);
%%Run GA
[X,FVAL,REASON,OUTPUT,POPULATION,SCORES] =
ga(fitnessFunction,nvars,options);
toc
```

Code for Satfitness.m

```
function z = Satfitness(x)
global  n TW q Q value C cap twin st satno viewno nosats
x
Optz=0;
Optx=x;
Viewfeas= zeros(n-1);
% If the same image is taken more than once the sequence is infeasible -
% return
for i=1:n-2
    for j=1:n-2
        if j~=i & viewno(j)==viewno(i)
           if x(i)==1 & x(j)==1
              Viewfeas(i,j)= 1
           end
        end
    end
end
Viewfeas;
if any (Viewfeas == 1)
   z=Inf;
   return
% Determine if the proposed sequences are feasible and calculate the
% fitness value
else
    a=[2:n];
    b=satno(2:n,1)';
    c=x.*b;
    for k=1:nosats
        for i=1:n-1;
            if c(i)==k
                d(i,k)=1;
            else
                d(i,k)=0;
            end
        end
    end
    h=[1 x];
    for k=1:nosats
        e=d(:,k)';
        rte{k}=nonzeros([1 e.*a 1])'
    end
    celldisp (rte)
    [TC,XFlg,out] = rteTC(rte,C,cap,twin,[])
    if any (XFlg ~=1)
       z=Inf;
    else
        z=-sum(h.*value');
    end
end
```

Code for crossoverspecial2.m

```
function xoverKids  =
crossoverspecial2(parents,options,GenomeLength,FitnessFcn,unused,thisPopula
tion)
%CROSSOVERSCATTERED Position independent crossover function.
%   XOVERKIDS = CROSSOVERSCATTERED(PARENTS,OPTIONS,GENOMELENGTH, ...
%   FITNESSFCN,SCORES,THISPOPULATION) creates the crossover children
XOVERKIDS
%   of the given population THISPOPULATION using the available PARENTS.
%   In single or double point crossover, genomes that are near each other
tend
%   to survive together, whereas genomes that are far apart tend to be
%   separated. The technique used here eliminates that effect. Each gene
has an
%   equal chance of coming from either parent. This is sometimes called
uniform
%   or random crossover.
%
%   Example:
%    Create an options structure using CROSSOVERSCATTERED as the crossover
%    function
%     options = gaoptimset('CrossoverFcn' ,@crossoverscattered);

%   Copyright 2003-2004 The MathWorks, Inc.
%   $Revision: 1.9.4.1 $  $Date: 2004/08/20 19:48:08 $
global  n TW q Q value C cap twin st satno viewno


% How many children to produce?
nKids = length(parents)/2;

% Allocate space for the kids
xoverKids = zeros(nKids,GenomeLength);

% To move through the parents twice as fast as thekids are
% being produced, a separate index for the parents is needed
index = 1;

% for each kid...
for i=1:nKids
    % get parents
    r1 = parents(index);
    index = index + 1;
    r2 = parents(index);
    index = index + 1;

    % Randomly select half of the genes from each parent
    % This loop may seem like brute force, but it is twice as fast as the
    % vectorized version, because it does no allocation.
    for j = 1:GenomeLength
        if(rand > 0.5)
            xoverKids(i,j) = thisPopulation(r1,j);
        else
            xoverKids(i,j) = thisPopulation(r2,j);
        end
    end
    %Customization to repair children that violate singularity constraint
    for j=1:n-2
        for k=j:n-2
            if k~=j & viewno(k)==viewno(j)
                if xoverKids(k)==1 & xoverKids(j)==1
```

```
                    h=rand
                    if h >= 0.5
                    xoverKids(k)=0
                    else
                    xoverKids(j)=0
                    end
                end
            end
        end
    end
end
```

APPENDIX D: EXAMPLE OF GENETIC ALGORITHM OUTPUT

Figure D-1 below is a typical example of the results obtained for each of the test cases. The particular test case is for a constellation of six satellites subject only to the minimum set of constraints (constraint Group A). The figure shows the best and average value obtained for the objective function for each of the 100 generations consisting of a population of 20 solutions.

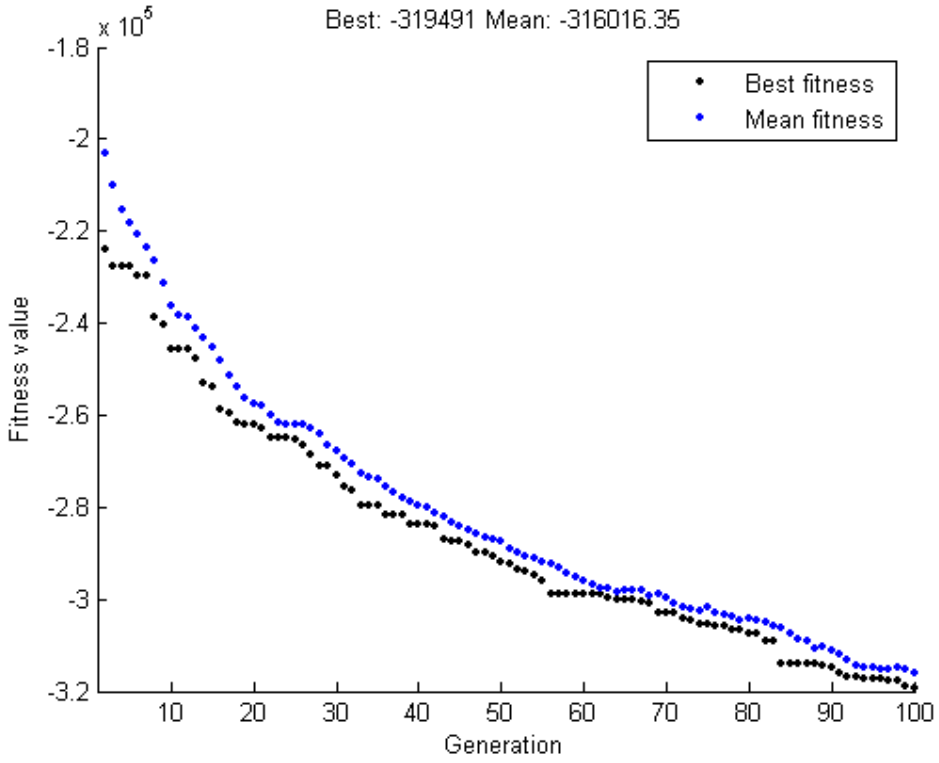

Figure D- 1: Sample of the graphic results for a typical computer run showing best and mean fitness values obtained for each of the 100 generations in a solution population comprising 20 individuals.

Note that the fitness value improves steadily over the generations with no sense of stagnation. This is typical of the results obtained.

The table below contains an example of a typical listing of the results for the same test case shown in Figure D-1. It shows six "routes", `rte {1}` though `rte {6}`, as a vector of time-window numbers that makes up the imaging sequence for the particular satellite. Next it shows six values for the `XFlg` parameter, the six ones denoting that each of the six sequences or "routes" are feasible. The elapsed time for the 100 generations is displayed next before the final genotype of the solution is displayed as a binary vector of size 249

```
rte{1} =

Columns 1 through 12

    1    2   22   32   58   59   62   64   71   74   75   76

 Columns 13 through 24

   83   94  101  104  109  110  115  138  140  142  144  166

 Columns 25 through 32

  175  183  196  203  210  213  231    1


rte{2} =

 Columns 1 through 12

    1    4    5    7   11   13   24   25   41   43   46   47

 Columns 13 through 24

   51   53   70   73  100  105  107  117  125  126  128  135

 Columns 25 through 36

  146  152  158  163  164  165  176  178  205  212  220  223

 Columns 37 through 41

  226  233  235  238    1


rte{3} =

 Columns 1 through 12

    1   20   39   48   52   56   57   61   77   82   85  103

 Columns 13 through 24

  108  111  113  114  121  123  127  132  137  145  149  150

 Columns 25 through 36

  153  156  159  161  167  172  177  184  187  190  194  201

 Columns 37 through 39

  224  241    1


rte{4} =

 Columns 1 through 12

    1    9   12   18   21   35   38   40   50   67   79   87

 Columns 13 through 24

   88   93   95   99  112  116  131  133  134  147  160  169

 Columns 25 through 34

  170  181  193  199  221  232  234  242  243    1
```

```
rte{5} =

  Columns 1 through 12

    1   15   16   19   30   33   54   55   63   72   78   90

  Columns 13 through 24

  118  122  124  129  136  139  148  151  168  173  180  185

  Columns 25 through 36

  191  197  204  206  208  217  219  222  227  228  229  236

  Columns 37 through 40

  240  245  248    1


rte{6} =

  Columns 1 through 12

    1    3    8   14   23   28   29   31   36   37   42   49

  Columns 13 through 24

   60   65   66   69   91   92   98  120  130  157  162  174

  Columns 25 through 36

  182  186  188  189  192  195  198  200  202  214  218  237

  Columns 37 through 38

  247    1


XFlg =

     1

     1

     1

     1

     1

     1


Elapsed time is 149.000000 seconds.


ans =


  Columns 1 through 12

    1    1    1    1    0    1    1    1    0    1    1    1
```

Columns 13 through 24

1 1 1 0 1 1 1 1 1 1 1 1

Columns 25 through 36

1 0 1 1 1 1 1 1 0 1 1 1

Columns 37 through 48

1 1 1 1 1 1 0 0 1 1 1 1

Columns 49 through 60

1 1 1 1 1 1 1 1 1 1 1 1

Columns 61 through 72

1 1 1 1 1 1 0 1 1 1 1 1

Columns 73 through 84

1 1 1 1 1 1 0 1 1 1 0 1

Columns 85 through 96

0 1 1 0 1 1 1 1 1 1 0 0

Columns 97 through 108

1 1 1 1 0 1 1 1 0 1 1 1

Columns 109 through 120

1 1 1 1 1 1 1 1 1 0 1 1

Columns 121 through 132

1 1 1 1 1 1 1 1 1 1 1 1

Columns 133 through 144

1 1 1 1 1 1 1 0 1 0 1 1

Columns 145 through 156

1 1 1 1 1 1 1 1 0 0 1 1

Columns 157 through 168

1 1 1 1 1 1 1 1 1 1 1 1

Columns 169 through 180

1 0 1 1 1 1 1 1 1 0 1 1

Columns 181 through 192

1 1 1 1 1 1 1 1 1 1 1 1

Columns 193 through 204

1 1 1 1 1 1 1 1 1 1 1 1

Columns 205 through 216

1 0 1 0 1 0 1 1 1 0 1 1

Columns 217 through 228

```
  1   1   1   1   1   1   1   0   1   1   1   1

Columns 229 through 240

  0   1   1   1   1   1   1   1   1   0   1   1

Columns 241 through 249

  1   1   0   1   0   1   1   0   0
```