

Oversubscribed Scheduling Problems

Laura V. Barbulescu

Fall 2002

1 Introduction

Scheduling problems arise in a variety of domains, such as manufacturing, military applications, and the space program. In general, scheduling problems specify a set of activities that share resources of limited capacity and need to be processed such that various constraints, primarily temporal, are satisfied. Fox (1994) defines scheduling as “assigning resources and times for each activity so that the assignments obey the temporal restrictions of activities and the capacity limitations of a set of shared resources”. However, for many real-world scheduling problems, a solution satisfying all the temporal and resource restrictions may not exist. In particular, problems for which there are typically more requests than can be accommodated with the available resources have been classified as *oversubscribed* [SP92]. Oversubscribed scheduling problems can also be found in the scheduling literature under the names of *overconstrained* problems [Pem00] or problems with *overloaded resources* [BPP98].

Many real-life applications, both military and civilian, involve scheduling resources that are heavily oversubscribed: the resources are scarce and the demand is high. Since increasing the amount of resources is very expensive, the requests are usually prioritized, and the scheduling process is guided such that the high priority requests are satisfied.

For many applications, scheduling is done manually, which is a time consuming process. Automation can significantly speed up scheduling, as well as improve the quality of the scheduling process. For example, in applications where the total rejection of a request is not an option, a negotiation process is started after the initial schedule is generated. The customers may be presented with various possibilities for modifying specific requests in conflict (for example, a customer might settle for less duration than initially specified). An automated scheduling tool can be designed to suggest such possible modifications; also, for any change in a request, a new schedule can be quickly generated.

The presence of oversubscribed resources can make the scheduling process difficult. While scheduling usually consists of assigning time slots and resources to the requests, for oversubscribed problems the solution identifies the best subset of the requests that can be satisfied given the resource and time constraints. Because not all of the requests can be scheduled, priority values or user preferences are typically specified for each request. An objective function is used to rate the subsets of requests to be scheduled: it could vary from a very simple “maximize the number of requests scheduled” to complex functions which use request priorities, user preferences, weighted attributes, etc. When solving an oversubscribed problem, two questions need to be answered:

which requests to schedule and what start times and resources to assign to them.

The need for automated scheduling is recognized for many oversubscribed applications: satellite scheduling, telescope scheduling, flight simulator scheduling, etc. In general, building an automated scheduler can be a challenging task, because it implies a great amount of interaction with human schedulers and it could sometimes be difficult to quantify the experience and domain specific knowledge used by human schedulers. For oversubscribed scheduling in particular, the task becomes more difficult. Sometimes the requests are not submitted with explicit priority values associated with them. However, during the scheduling process, human schedulers know that some of the requests are more important than others. Such knowledge needs to be quantified (in the form of priority values or preferences) in the automated scheduler. Furthermore, the quality of a solution needs to be expressed as an objective function to be used by the automated scheduler.

While the domains in which oversubscribed scheduling applications can be found vary widely, common characteristics of such applications can be identified. First, by definition the resources are oversubscribed. Second, the requests typically specify start times, durations, deadlines, and priorities. Alternative resources and setup times are sometimes specified as well. Finally, the objective function is usually defined using the request priorities. For example, scheduling observations for telescopes is typically oversubscribed. Astronomers spend a lot of time observing objects as they set in the west. The windows of opportunity to observe such objects are usually short, and there are more observation requests than can be accommodated by the telescope. The requests specify the beginning of the visibility window for the object as the start time. The end of the visibility window is the deadline, and priorities are also assigned to the observations. Setup times are usually needed to point the telescope from one object to another. An objective function is used to decide which observations to schedule. Another example (described by Rudova [RM99]) is searching for an optimal sequence of airplane departures from a runway. The start time and deadline for each departure are determined based on specific constraints on the time intervals between successive departures. To solve the problem, preferences are associated with the airplanes and their possible time slots; a possible objective function would maximize the sum of such preferences for the scheduled airplanes.

My research so far has examined an oversubscribed problem from the satellite scheduling domain: satellite range scheduling, which involves scheduling communications between users on the ground and more than 100 satellites in space. Similar to most oversubscribed scheduling problems, the requests specify start times, durations and deadlines. Alternative resources are specified for each request. However, no priorities

or preferences are specified. The objective function maximizes the number of scheduled requests. A description of my research in the satellite range scheduling domain is presented in section 3.

Although many oversubscribed scheduling applications have been presented in the literature, no one has yet studied it as a general class of scheduling problem. This research is a first step toward formally characterizing the unique features of the problem and identifying factors that affect algorithm performance on it.

Based on my current work, the proposed research will focus on answering two questions central to oversubscribed scheduling. First, I will investigate what features encountered in oversubscribed scheduling problems influence algorithm performance. The features I will consider are typically encountered in most oversubscribed scheduling applications. For example, an important feature is the contention for resources. Contention measures have been defined for satellite scheduling by various researchers ([FJMS01, Pem00]), but are general enough to be applied to other oversubscribed scheduling problems. Contention measures can be used to both characterize the problem instance and to investigate the change in algorithm performance when varying the contention. I will design a problem generator using parameters to model various problem features. By generating problems exhibiting certain attributes, I will be able to formulate and test hypotheses about the relationship between such problem attributes and algorithm performance. We have analyzed the relationship between problem features and algorithm performance before, for the well known flow-shop scheduling problem [WBHW99, WBWH02]. We designed a problem generator to produce problem instances with job correlation (the processing times of a given job are correlated) or with machine correlation (the processing times of all the jobs on a given machine are correlated). We showed that for these problems the performance of a state-of-the-art algorithm degrades rapidly: faster and less complex stochastic algorithms provide superior performance.

Second, I will investigate the impact of the choice of objective function on algorithm performance for oversubscribed scheduling problems. This is an important question, given that many real-world problems specify multiple and sometimes conflicting objectives to be optimized. The question can be posed for oversubscribed scheduling problems in general by referring to certain objective functions (such as maximizing the sum of request priorities, or minimizing the idle time) that are commonly encountered in various applications. I will perform empirical studies in order to determine which algorithms perform better given a certain objective function and how different are the solutions reported for similar objective functions.

The remainder of this document is organized as follows: An overview of oversubscribed

problems in the scheduling literature is presented in section 2. Section 3 presents my current research progress in the area of oversubscribed scheduling problems, focusing on satellite range scheduling. Finally, the future directions for the research I plan to complete for my dissertation are presented in section 4.

2 Background: Oversubscribed Scheduling Problems and Solution Methods

Oversubscribed scheduling problems are frequently encountered in the scheduling literature. This section is an overview of some of the most broadly researched oversubscribed scheduling problems. For each problem, I present a short problem description, the objective function and the solution methods. Both exact and approximate methods, varying from simple greedy heuristics, to complex, hybrid algorithms, have been employed to solve the problems presented here. A summary of the solution methods encountered is presented at the end of this section.

The emphasis in this overview is on the satellite scheduling domain as a source of oversubscribed scheduling problems for two reasons. First, a large amount of the work on oversubscribed scheduling problems has been done in the area. Second, I'm currently researching the problem of satellite range scheduling (I present a detailed study of satellite range scheduling in section 3); the overview of the satellite scheduling domain provides a context for my research. I also present two applications in the telescope scheduling domain; demands for observation time on the telescope are usually high and not all of them can be satisfied. Multiple similarities exist between satellite and telescope scheduling because both of them define visibilities for the targeted space objects. Finally, I present oversubscribed scheduling problem encountered in other domains, such as scheduling activities for space shuttle payloads, or scheduling communications for Heterogeneous Computer Systems.

2.1 Satellite Scheduling

In satellite scheduling, hundreds of requests compete for resources such as antennas at ground stations, instruments, recording devices, and transmitters on the satellites. Typically, more observations need to be scheduled than can be accommodated by the satellites and the ground stations. A general description of the satellite scheduling domain is provided by Jeremy Frank et al.(2001) [FJMS01]. Currently, scientific

observations from different satellites are independently scheduled, and thus a lot of manual coordination of the observations is needed. Automating the process of generating a schedule given a set of requests and the constraints associated with them can result in more efficient schedules. Automation also facilitates “what-if” scenarios, especially needed when negotiating with customers and re-scheduling are part of the scheduling process (as is the case of scheduling requests for military satellites, where outright rejection of a request is not an option).

A particular characteristic of satellite scheduling is the fact that the observations as well as the communication between the satellites and the ground stations depend on satellite visibility. Windows of visibility are defined based on the orbit of the satellite and the position of the ground station or of the object observed by the satellite. Other application specific features are the visibility conditions, the weather, the capability of the different instruments on board of the satellite, the transmission rates, the costs associated with using different communication devices, etc. All of these make satellite scheduling a very complex domain. Various studies referring to special cases of satellite scheduling have been published. I structure the presentation of the research done in this domain in a progression, starting with studies published for simple versions of the problem.

2.1.1 Previous Studies

Burrowbridge(1999) studies one of the most simple versions of satellite scheduling: scheduling the communications between ground stations and low-altitude satellites, with the objective of minimizing the number of scheduled tasks. The visibility windows for low altitude satellites have approximately the same duration as the duration of the requested communication; in other words, for this problem, each request has a fixed start time, a fixed end time, and a duration equal to the difference between the start and end time. The well-known *greedy activity-selector* algorithm [CLR90] is used to schedule the requests since it yields a solution with the optimal (maximal) number of scheduled tasks.

A simple one-resource satellite scheduling problem is solved by Pemberton(2000), where the requests have fixed start times and fixed durations; a priority value is associated with each request. The objective is to schedule the requests such that the sum of the priorities of the scheduled requests is maximized. Basically the difference between the scheduling problem solved by Pemberton and the low-altitude satellite scheduling investigated by Burrowbridge is the presence of priorities associated with the requests. Pemberton makes the point that applying constrained programming

techniques to this problem can result in poor performance given the large number of requests to be scheduled. Therefore he proposes a “priority segmentation algorithm” which is a hybrid algorithm combining a greedy approach with branch-and-bound. The requests sorted in the order of their priority are divided into groups of n requests. Starting with the highest priority request group, for each group of n requests, an optimal schedule is identified (using branch-and-bound and constraint propagation) where the previously scheduled requests are considered as locked (cannot be moved anymore). Pemberton applies his algorithm to one resource problems with 100 and 1000 tasks. However, for the 100 request problems, exact optimal solutions could be obtained using branch-and-bound algorithms such as the one described by Baptiste et al. [BPP98] (see Section 3 for a more detailed description of Baptiste et al.’s algorithm). Baptiste et al. report good algorithm performance when solving one machine scheduling problems with up to 100 requests; of course, differences in performance might exist when applying their algorithm to Pemberton’s problems. Also, in a review of Pemberton’s [Pem00] paper, Verfaillie [Ver00] mentions that a simple dynamic programming algorithm can outperform the priority segmentation algorithm by optimally solving Pemberton’s problems.

Wolfe et al.(2000) define a more complex one-resource problem, the “window-constrained packing problem” (WCP), which specifies for each request the earliest start time, latest final time and the minimum and maximum duration. The objective function for the WCP is also different, based on the idea that the middle of the time window in which the request can be scheduled is to be preferred over positions toward the edges. The objective function combines request priority with the position of the scheduled request in its required window and the number of requests scheduled. Two greedy heuristic approaches and a genetic algorithm are implemented to solve the window-constrained packing problem. The genetic algorithm uses job permutations as the population and is shown to improve on the results obtained with the two greedy heuristics for randomly generated problems.

A different scheduling approach is proposed for the more complex problem of scheduling communications between satellites and the Deep Space Network (DSN) 26-meter subnet [GC96] (which is a collection of antennas in California, Australia and Spain). For each satellite, a set of project requirements specifies the minimum and maximum number of communication events, and the minimum and maximum duration and time intervals between communications. Scheduling is done weekly; a problem defines the set of satellites to be scheduled, specific constraints on the satellites, and visibilities for each satellite and antenna. The solution should specify a maximum feasible set of time intervals for satellite-antenna communications such that all the project requirements are satisfied. The problem is usually oversubscribed; the goal is to obtain

good-enough solutions (or prove infeasibility) in a short time, such that “what if” reasoning is facilitated in the negotiation with the customers. Multiple satellites, multiple antennas, as well as various constraints make this problem much more complex than the ones previously presented. Gratch et al.[GC96] propose a heuristic scheduling approach (LR-26) using Lagrangian relaxation combined with constraint propagation search. Basically the constraints involving more than one antenna are relaxed, and each antenna is scheduled independently. The relaxed constraints are weighted and added into the objective function. Lagrangian relaxation attempts to find a global solution by adjusting the set of weights; if Lagrangian relaxation fails to identify a solution, constraint propagation search is used. An adaptive version of the scheduler is also described, where the scheduler is provided with a variety of heuristic methods and it chooses a particular combination that works well for a set of given problems. Experimental data show that the adaptive scheduler is selecting high performance strategies and results in improved problem solving performance.

While all the satellite scheduling problems presented so far schedule communications from one ground station to satellites (the resource is represented by the ground station), Verfaillie et al. [VLS96] schedule requests for three instruments on a satellite (the three instruments are the resources here). Their version of the satellite scheduling problem emerged from scheduling requests for the *Spot5* satellite. The problem has been described in [VLS96, BVA⁺96]: scheduling a set of photographs, which are preference weighted and which can be taken from any one of the three instruments on the satellite. Constraints are imposed on the transition times between successive photographs on the same instrument, and on dataflow through the satellite telemetry. The goal is to select a subset of photographs such that the sum of the weights of the photographs is maximized. Verfaillie et al. [VLS96] introduce an algorithm called “Russian Doll Search”, which is a modified branch-and-bound algorithm based on the existence of a fixed variable ordering. The algorithm iteratively solves n subproblems: first the subproblem represented by the last variable, second the subproblem containing only the last two variables, and so on. For each subproblem solved, the result is recorded and used to bound the search when solving the next subproblem. The Russian Doll Search is compared to a depth first branch-and-bound and to two approximate methods: a greedy algorithm, and tabu search. Russian Doll Search provides optimal solutions, faster than the depth first algorithm and is therefore suited for small and medium size problems. It fails on large size problems, where more complex constraints are also present, and thus approximate methods are more appropriate for these problems.

A more recent, more complex version of Verfaillie et al.’s problem has been studied by Lemaître et al.(2000): scheduling the set of photographs for new Agile Earth observing

satellites. While satellites like Spot5 have only one degree of freedom, the Agile satellites have three degrees of freedom, and therefore more opportunities are present when scheduling the photographs. Again, each photograph specification includes a weight, the earliest start time, latest end time and duration. Additional constraints are present, such as: minimal time between two successive acquisitions, photographs that need to be taken twice from different time windows, and photographs that need to always be scheduled. Branch-and-bound using constraint propagation is combined with heuristics to control the combinatorial explosion (such as dropping the low weight photographs from the schedule from the start, or imposing a certain order on the sequence of the photographs to be scheduled). Local search is also used to solve the problem, in a next descent fashion, where a photograph is added to or removed from the current feasible sequence of photographs; several heuristics guide the choice of the photograph and its position in the sequence. Seven problems are used to compare the performances of the two algorithms. Local search outperforms the systematic search when a limit of 5 minutes of CPU time is imposed on the systematic search and local search is allowed to perform 100 executions of two minutes CPU time each. Obviously, the systematic method is expensive, and the approximate method seems to result in better performance here.

Going toward the goal of automated planning and scheduling for complex real-world satellite operations, researchers at NASA built a complex environment called ASPEN. The ASPEN (A Scheduling and Planning Environment) framework [CRK⁺00] is used to model and solve real-world applications involving Earth observing satellites. ASPEN implements features commonly encountered when modeling planning and scheduling problems; constructive and repair-based algorithms are implemented for finding a valid schedule given a set of goals. As a planning and scheduling framework, ASPEN is more focused on the optimization of the command sequence [CRK⁺00] (the planning aspect) than on the optimization of observation preferences. Two satellite applications in which ASPEN was used are the EO-1 and CX-1 missions. EO-1 [SGY⁺98] is an Earth imaging satellite; only four data takes per day can be performed. CX-1 [ECB⁺01] is a small Earth orbiting satellite. The goal of the CX-1 scheduling problems is to maximize the amount of atmospheric data downlinked. Several criteria are considered when computing the value of a schedule, such as: number of violated conflicts, amount of data downlinked, power usage, memory usage, etc. While Verfaillie et al. [VLS96, BVA⁺96] model as resources the three instruments on the satellite, both for EO-1 and for CX-1, ASPEN permits a much finer granularity in identifying the events to be scheduled and the resources involved.

Frank et al.(2001) propose a new approach to solving the complex satellite scheduling problem. It combines a constraint-based planner with a stochastic greedy search

algorithm: the greedy search selects the next task to be scheduled, and the planner propagates the constraints and adds to the schedule all the activities required by the scheduled task, such as setup and postprocessing. The greedy search is based on Bresina's HBSS algorithm [Bre96] (also see Section 2.2). A heuristic approach is used to choose the next task to be scheduled and the start time for the task, based on contention measures. The contention measures are used to define the variable ordering heuristic (basically the highest priority and highest contention tasks are scheduled first) and the value ordering (the task should be scheduled in the time period with the least contention). This approach is one of the most complex applied to the satellite scheduling problem; it models all the satellite resources and communication resources. However, at the date the paper was published, the authors were still implementing parts of the heuristic approach, and therefore no empirical data is available.

2.1.2 The Satellite Range Scheduling Problem

Satellite Range Scheduling has been the main focus of my research in the oversubscribed scheduling domain (a detailed study of the problem is presented in section 3). Therefore I chose to present it separately from the rest of the satellite scheduling applications described so far.

The U.S. Air Force Satellite Control Network (AFSCN) is responsible for coordinating communications between civilian and military organizations and more than 100 satellites; the generic problem of scheduling task requests for communication antennas is referred to as the Satellite Range Scheduling Problem [Sch93]. Space-ground communications are performed via 16 antennas located at 9 ground stations positioned around the globe. Customer organizations submit task requests to reserve an antenna at a ground station for a specific time period based on the windows of visibility between target satellites and available ground stations. Alternate time windows and ground stations may also be specified. As of 2000, over 500 task requests are received by the AFSCN scheduling center for a typical day. The communication antennas are oversubscribed, in that many more task requests are received than can be accommodated. Currently, human scheduling experts construct an initial schedule that typically leaves about 120 conflicts representing task requests that are unsatisfiable. However, satellites are extremely expensive resources, and the AFSCN is expected to maximize satellite utilization. Out-right rejection of task requests is not a viable option. Consequently, human schedulers must engage in a complex, time-consuming arbitration process between various organizations to eliminate all conflicts present in the initial schedule.

Most of the work previously done on this problem has been published by researchers from the Air Force Institute of Technology (AFIT). Gooley (1993) developed an algorithm based on a combination of mixed-integer programming (MIP) techniques and insertion heuristics, which achieved good overall performance: 91% – 93% of all requests scheduled. Schalck (1993) used MIP techniques based on Gooley’s formulation and scheduled more than 95% of the requests. Parish (1994) tackled the problem using a genetic algorithm called *Genitor*¹, which scheduled roughly 96% of all task requests, out-performing Schalck’s MIP approach. All three of these researchers tested their algorithms on a suite consisting of seven problem instances, representing actual AFSCN task request data and visibilities for seven consecutive days from October 12 to 18, 1992 and including about 300 requests each. Schalck and Gooley used different procedures in processing the data, resulting in different requests to be scheduled. Therefore a direct comparison of their results is not possible. It seems Schalck and Parish used similar procedures of data processing, and thus their results can be compared. Later, Jang (1996) introduced a problem generator employing a bootstrap mechanism to produce additional test problems that are qualitatively similar to the AFIT benchmark problems. Jang then used this generator to analyze the maximum capacity of the AFSCN, as measured by the aggregate number of task requests that can be satisfied in a single-day.

2.2 Telescope Scheduling

Aspects of telescope scheduling make this problem very similar to satellite scheduling. For example, in both telescope and satellite scheduling, windows of visibility are defined; also, setup times are associated with the viewing instruments (telescope) and antennas (satellite scheduling) and the weather impacts on the visibility conditions. In most cases, there are more requests than can be scheduled. I present three examples of oversubscribed telescope scheduling problems. Two refer to the Hubble Space Telescope (one is a simplified version of the problem, the other describes a complex scheduler built for the telescope). The third example refers to a telescope scheduling problem investigated by Bresina (1998).

When scheduling observations for the Hubble Space Telescope, demands for observations exceed by far the capabilities of the telescope. The candidate observations need to be scheduled on one of the six viewing instruments in a particular configuration. The observations have windows of visibility and durations. Setup times are associated with pointing the viewing instruments toward the object observed and with

¹For a description of *Genitor*, see section 3

reconfiguring the instruments. Power constraints impose a limit on the number of viewing instruments that can be active at the same time. Additionally, precedence constraints and priorities may be specified for the observations.

A simplified version of the problem is described by Smith et al.(1992). It is assumed that only one instrument can be active at any time; the setup time needed to reconfigure the instrument to be used is assumed to depend only on the previously used instrument. Two conflicting objectives are identified: maximizing the resource utilization (in this case telescope utilization) and rejecting as few candidate observations as possible. Two greedy scheduling heuristics are described, each addressing one of the two objectives. The “nearest neighbor look-ahead” heuristic minimizes the idle times on the telescope by choosing to schedule candidate observations with early start times; lookahead is performed to make sure that the candidate chosen is the least disruptive (resulting in the least number of observations that cannot be scheduled anymore). The “most-constrained first” heuristic maximizes the number of observations scheduled by choosing to schedule the observation with the fewest possible start times. In order to address the dual nature of the objective function, a composite strategy is then implemented. The composite strategy uses the most-constrained first heuristic to select the next observation scheduled; however, for the cases where the number of start times available is similar for all of the unscheduled candidates, the nearest neighbor look-ahead heuristic is used instead. On a set of seven randomly generated problems, the composite strategy manages to effectively combine the two heuristics.

Next, we present the complex scheduler built to solve the real problem of scheduling requests for the Hubble Space Telescope. SPIKE [JM94] is a scheduler that was initially built for the Hubble Space Telescope scheduling problem. SPIKE has been operational since October 1989; it can handle the broad complexity of telescope scheduling, where activities that need to be scheduled specify constraints of precedence, ordering, minimum and maximum time separation, preemption, and repetition. The constraints are classified into feasibility constraints (hard constraints) and preference constraints, which have a weight value associated. The temporal constraints have higher weights than resource constraints. The activities have priorities, and possible time assignments have preferences associated with them. Suitability functions are used to combine feasibility and preference constraints for the observations. In SPIKE, repair-based methods are used to build the schedule. Repair based methods identify conflicts in the schedule and try to repair them by moving tasks around. Such methods are very suitable for oversubscribed scheduling and have been widely used in scheduling space operations: ASPEN (described in the previous section) employs iterative repair, also Zweben et al.(1994) and Rabideau et al.(1999) used iterative

repair techniques to schedule space shuttle payloads (see next section). The repair-based method implemented in SPIKE is called “multistart stochastic repair”. Three main steps compose the multistart stochastic repair technique. The three steps are repeated until a time limit is reached and the best schedule is selected:

- First, an initial assignment is performed, by heuristically assigning the observations, with preference for the most constrained ones, at times for which the number of conflicts is minimized.
- Then, repair heuristics are applied: hill climbing is used to move the activities with most conflicts to a time where the number of conflicts is minimized.
- Finally, the conflicting activities are removed from the schedule; lower priority, higher number of violated constraints, lower preference time assignment activities are preferred for removal. If there are gaps in the schedule, the unscheduled tasks are used to fill the gaps (using a best-first approach).

When iterating through these three steps, the initial assignment and the repair heuristic can vary. The quality of the schedule is dictated by the number of observations scheduled, the total time scheduled, and the sum of preference values associated with the scheduled observations.

The third example of telescope scheduling presented in this section was described by Bresina(1998): the problem of scheduling observations for the “T3” automatic photoelectric telescope at Fairborn Observatory in Arizona (a telescope that gathers photon count information describing the brightness of different objects in the sky). The observations have priorities; also, some observations need to be repeated during the night. “Enablement intervals” are defined as the time intervals within which the observations can be executed and are determined by specific constraints such as the darkness and the distance to the moon. These enablement intervals are in fact the equivalent of the visibility windows encountered in satellite scheduling. The telescope is equipped with a telescope controller that uses dispatch heuristics to decide which observations to schedule next. The dispatch heuristics work well for nights with a balanced load of required observations. However, there are many nights for which not all the observations required can be executed; for such nights the problem of telescope scheduling is oversubscribed, and the dispatch heuristic performs poorly. The objective function is defined as a weighted sum of three attributes of the final schedule, with the most important attribute minimizing the sum of the priorities of the missed observations; the other two attributes minimize the airmass and the telescope slewing respectively.

Bresina proposes a heuristic to improve the quality of observation scheduling, based on the dispatch heuristics built into the telescope controller. The heuristic is embedded into what was a new search technique: HBSS (Heuristic-Biased Stochastic Sampling) (Bresina published an initial description of HBSS in 1996 [Bre96]). HBSS combines heuristic guidance with random sampling; it employs a bias function as a method of balancing the exploration of the search space and the exploitation of the states ranked best by a heuristic. Basically, HBSS incrementally builds a solution; at each step the next move is chosen with a probability based on the bias function. When a strong bias function is used (for example, exponential or high-degree polynomial), the search mostly follows the heuristic: high probability is associated with the moves ranked as the best by the heuristic (in the extreme greedy search behavior can be obtained). With a weak bias (logarithmic or linear), the search explores more of the space, in the extreme producing random-sampling behavior. Real scheduling problems were available for testing the new search algorithm; experimental results show that HBSS outperforms the dispatch heuristic on these problems.

Since its publication in 1996, HBSS has been widely referenced and used. In the scheduling domain, Frank(2001) proposes the use of the HBSS search technique in solving the problem of scheduling Earth observing satellites. Oddi et al.(1997) extend HBSS to define the bias dynamically, based on how well the heuristic can discriminate between the alternatives; they show the increased performance of the new algorithm when solving jobshop problems with non-relaxable deadlines and complex metric constraints. Also, we used HBSS to solve flowshop problems with various amounts of structure; HBSS was one of the best algorithms we implemented for these problems [WBHW99].

2.3 Scheduling Shuttle Payload Operations

Rabideau et al.(1999) describe the DATA-CHASER Automated Planner and Scheduler (DCAPS) used to schedule the DATA-CHASER space shuttle payload for a solar science experiment performed in 1997. The problem consists of finding a sequence of activities related to instrument observations, data storage, communication and power systems with the goal of increasing science return (number of measurements taken and downlinked) while satisfying all the spacecraft constraints. Both planning and scheduling are employed to solve the problem; planning determines the activities needed to be performed such that specific scientific goals are satisfied, while scheduling assigns start times to these activities such that the constraints (imposed by limited availability of the resources, but also the temporal constraints) are satisfied.

Scheduling is performed in three steps:

1. An initial schedule is generated using domain specific knowledge.
2. The schedule is repaired to remove conflicts; frequently the resources are over-subscribed but other conflicts may also be present. The scheduler uses iterative repair to remove the conflicts in the schedule. Iterative repair has been described by Zweben et al.(1994); it iteratively selects a conflict in the schedule and it adds, moves or deletes an activity in an attempt to solve the conflict. For example, suppose a resource is oversubscribed at time t : if an activity exists which increases the amount of resource available, such an activity can be added to the schedule. Also, the scheduler can move one of the activities requiring the resource at that particular time t to a different time in the schedule; in the worst case, an activity can be removed.
3. The oversubscribed schedules are optimized using a special “packing” algorithm, based on the doubleback algorithm described by Crawford(1996). The packing algorithm performs a sweep over the scheduled activities starting with the first one and moves them to their earliest possible start time; the idea is that by doing so, more room will be available toward the end of the schedule for the activities for which the resource requirements could not be satisfied in the initial schedule.

These three steps are identical (conceptually) to the three steps of the multistart stochastic repair technique implemented by SPIKE (for scheduling requests for the Hubble Space Telescope). SPIKE repeats the three steps multiple times; however it is not clear whether the scheduler for DATA-CHASER also repeats them. On the other hand, the activities scheduled for DATA-CHASER are determined by the planner; for the telescope, the requests are part of the problem specification. Rabideau et al. note that the use of the DATA-CHASER Automated Planner and Scheduler resulted in “significant quantitative improvements in operations efficiency” [RCWM99] over the manual generation of the schedules.

2.4 Scheduling F-14 Flight Simulators

Syswerda [Sys91] describes the problem of scheduling access to flight simulator resources in a laboratory: F-14 jet fighters, together with the radar and the support equipment are the resources to be scheduled. The tasks require extensive setup of

the equipment; the optimization of the setup is important since it can take from one to two hours. Precedence relations specify that some tasks cannot be started before the completion of other tasks. Preference and priority are also specified for the tasks. When not all the tasks can be scheduled, the ones with lower priority are dropped from the schedule. The objective function is the sum of the weighted priorities for the scheduled tasks. The priorities are weighted by a 1.5 factor if for the corresponding task preference constraints are violated; otherwise the weight factor is 2. A genetic algorithm is used to solve problems with 90 tasks and up to 30 resources.

2.5 Heterogeneous Computing Systems

Heterogeneous computing systems are ensembles of interconnected computers of various computational capabilities [Bra01]. Applications defined as collections of communicating tasks with various computational requirements need to be executed on these computers; obviously the goal is to achieve increased application performance. The tasks to be scheduled have priority values; the tasks also have arrival and due dates. Multiple versions are defined for the tasks, where only one version needs to be executed (this is similar to having alternative resources specified, as in the case of Satellite Range Scheduling, previously described). User preferences are defined for each version. Lower preference versions of a task have reduced requirements; meaning that by allowing lower preference versions to be scheduled, more tasks can be accommodated, given the oversubscribed character of the problem. Precedence constraints can also be imposed on tasks (modeling the idea that a task cannot start until it receives all its input data, which is possibly produced by other tasks). The tasks need to be assigned to the machines, and then on each machine, the tasks need to be ordered. The objective function is defined as the sum of the product of preference and priority for each of the tasks completed before the deadline.

Braun [Bra01] defines two greedy heuristics for scheduling the tasks. A generational genetic algorithm (GA) as well as a steady-state GA (*Genitor* [Dav91]) are also implemented using a special structure of the chromosome and crossover and mutation operators designed for this chromosome structure. *Genitor* performs the best, however one of the greedy heuristics also results in good performance (the genetic algorithms were seeded with the solution obtained by the greedy heuristic).

2.6 Data Staging

Data staging [TBS⁺00] is described as a data management problem for a heterogeneous network of locations. Limited data storage is available at each location, and communication links are available at specified times between the locations. Data items become available in certain data locations, and requests for data items are made from specified locations in the network. The requests include priorities and deadlines. The requests are satisfied by transferring the data from the locations where it is generated to the locations where it is requested; the transfer is made using the communication links available. The communication links have a start time and end time; the duration for transmitting a data item from one location to another through a specified communication link is also specified.

Theys et al.[TBS⁺00] define three greedy heuristics for scheduling data communications from multiple sources to their destination, based on Dijkstra's multiple source shortest-path algorithm for a weighted, directed graph. The objective function is represented by the sum of the priorities of the satisfied requests. The authors state that no other research group has published work related to this problem; therefore the heuristics are only compared to each other.

2.7 Solution Methods

As presented so far, a variety of algorithms are employed when solving oversubscribed scheduling problems. In Table 1, I summarize the algorithms developed for the problems in this section. Exact algorithms (first two columns in Table 1) always find the optimal solution of the problem; examples of such algorithms are various versions of tree search, such as the Russian Doll Search algorithm used by Verfaillie (1996) to schedule a set of photographs to be taken from a satellite (see subsection 2.1). I also included in this category the greedy activity selector algorithm [CLR90]. The applicability of this algorithm to oversubscribed scheduling problems is reduced (it can only solve very simple one-resource problems with fixed start times and end times); however, Burrowbridge(1999) used it to optimally schedule low-altitude satellite requests. I include the algorithm in the table for completeness. Most of the solutions for oversubscribed scheduling problems use approximate algorithms: greedy and search algorithms, such as hill climbing, genetic algorithms, tabu search, iterative repair or HBSS. Greedy algorithms (the fourth column in Table 1) are very frequently used. Obviously if domain specific knowledge can be translated into a greedy algorithm, solutions for the problems can be obtained very quickly, and many times such solutions

Problem	Exact Algs.		Approximate Algs.						Hybrid Algs.	
	Act. Sel.	Tree Search	Greedy	GA	Tabu Search	Hill climb.	Iter. repair	HBSS	Tree Search & Lagrangian Relaxation	Tree Search & Greedy Heur.
Burrowbridge(1999)	X									
Pemberton(2000)										X
Wolfe(2000)			X	X						
Verfaillie(1996)		X	X		X					
Lemaître(2000)						X				X
ASPEN(2000)							X			
Gratch(1996)									X	
Frank(2001)								X		
Range sched(AFIT)				X						X
Smith(1992)			X							
SPIKE(1994)							X			
Bresina(1998)								X		
Rabideau(1999)							X			
Syswerda(1991)				X						
Braun(2001)			X	X						
Theys(2000)			X							

Table 1: Summary of the algorithms used to solve the oversubscribed scheduling problems described in this section.

prove to be competitive with the ones produced by more time-consuming methods (for example, see [Bra01]). Genetic algorithms (the fifth column in the table) have been successfully used on a variety of scheduling applications; many researchers report great success in using genetic algorithms for oversubscribed problems as well. Iterative repair (the eighth column in the table) is also successfully used to solve oversubscribed scheduling problems: note though that the research for both satellite scheduling (using ASPEN) and for space shuttle payload scheduling has been carried out by researchers from the same Artificial Intelligence Group at JPL. Finally, hybrid algorithms (last two columns in the table) combine exact and approximate techniques (for example, tree search can be used to solve subproblems); like the exact algorithms, the hybrid ones are not very frequently used.

Most of the research overviewed in this section seems to suggest that approximate algorithms are better suited to solve complex, large size oversubscribed scheduling problems than the exact or hybrid algorithms. Thus, for large size problems, Verfaillie(1996) and Lemaître (2000) observed a decline in the performance of the Russian Doll Search and the hybrid heuristic combined with a tree search algorithm respec-

tively, noting that approximate methods (such as hill climbing) perform better for such problems. Also, research at AFIT [Par94] has shown a genetic algorithm to outperform a hybrid of exact methods and heuristic search when solving the satellite range scheduling problem. Pemberton(2000) successfully uses a hybrid algorithm, but he solves very simple problems. Burrowbridge also solves a very simple version of satellite scheduling. The hybrid algorithms, while not widely used, show some promising results when scheduling communications between satellites and the DSN (Deep Space Network) 26-meter subnet [GC96].

Tree search algorithms using constraint propagation have been proposed for overconstrained problems (overconstrained problems are problems for which not all the constraints can be satisfied) [FW92]. Oversubscribed scheduling problems can be considered a subclass of overconstrained problems. In particular, for oversubscribed scheduling problems, the constraints which cannot be satisfied are always resource constraints: the resources available are insufficient and cannot satisfy all the requests. Freuder et al.(1992) propose branch-and-bound algorithms to solve what they define as “partial constraint satisfaction problems”: a solution satisfies only a subset of the constraints because the problem is overconstrained and not all the constraints can be satisfied. For oversubscribed scheduling problems, the constraints that are conflicting are related to the capacity of the resources in the problem and the solution cannot drop constraints on resource capacity. Therefore some of the tasks will not appear in the final schedule, which in terms of partial constraint satisfaction translates to dropping all the constraints corresponding to these tasks. Baptiste et al.(1998) describe a branch-and-bound algorithm based on constraint propagation and use it to maximize the number of scheduled jobs for an overloaded² one-machine scheduling problem (a description of the algorithm is given in section 3). However, real-world oversubscribed scheduling problems are usually more complex; the general opinion seems to be that for such problems exact branch-and-bound methods are very inefficient, given the explosion of the search space [Pem00, FJMS01, BVA⁺96].

3 Current State of My Research

In his Ph.D. thesis, Bresina(1998) identifies certain attributes of the telescope scheduling problem that will result in one algorithm performing well or performing poorly. He describes an evaluation method called “Expected Solution Quality” to characterize particular instances of the problem and the quality of the solutions; the method

²Baptiste uses the word “overloaded” instead of oversubscribed; we decided to use the “oversubscribed” term because it appears more frequently in the scheduling literature.

is based on iteratively sampling the space of solutions. However, most of the studies presented in Section 2 propose new heuristics and performance results for the various problems studied without investigating the reasons for the observed performance. For problems like data staging, for which no previous work has been published, an exploratory study comparing algorithm performance is appropriate. However, for problems which have been intensively researched, a study of problem characteristics could provide important insight into reasons for algorithm performance. For Satellite Range Scheduling (which has been the focus of my current research), various algorithms have also been proposed [Sch93, Goo93, Par94, Bur99]. At this stage, we need to investigate the attributes of current problems and solution methods in order to improve the quality of the schedules obtained.

This section presents a formal and empirical study of Satellite Range Scheduling. I structured the study as a progression. I start with a simple version of the problem, where only one resource is present. I follow by looking at a version of the problem studied at AFIT. Finally, for planning and experimental control purposes, I built a new problem generator that generalizes features found in the initial two versions of the problem studied and introduces new realistic features.

By studying the simple (one resource) versions of the problem, we were able to identify the equivalence to a well known one machine scheduling problem; to the best of our knowledge, this connection was never before made. Also, NP-completeness can be proven as a result of this equivalence: while previous studies ([Goo93, Par94, WS00]) mentioned the NP-completeness, no formal proof was offered before. We showed that for the one-resource version of the problem, algorithms from the machine scheduling domain outperform the algorithms specifically designed for Satellite Range Scheduling. Next, we investigated if these performance results generalize for the problem with multiple resources. We used old data from 1992 from the U.S. Air Force Satellite Control Network (AFSCN); we also built a problem generator to produce problems similar to the ones currently solved by AFSCN. Three main results emerged from our empirical study of algorithm performance for multiple-resource problems. First, the performance results obtained for the single-resource version of the problem did not generalize: the algorithms from the machine scheduling domain performed poorly for the multiple-resource problems. Second, a simple heuristic was shown to perform well on the old problems from 1992; however it failed to scale to larger, more complex problems. Finally, a genetic algorithm was found to yield the best overall performance on the larger, more difficult problems produced by our generator. Part of the material presented in this section was published as a PPSN conference paper [BHWW02]. Also, this study has been submitted for publication to the Journal of Scheduling [BWWH02].

3.1 Problem Description

Satellite Range Scheduling is an important application for both the government (e.g., military and NASA) and civilian applications (access to commercial satellites). For purposes of analysis, we consider two abstractions of the Satellite Range Scheduling problem. In both cases, a problem instance consists of n task requests where the objective is to minimize the number of unscheduled tasks. A task request T_i , $1 \leq i \leq n$, specifies both a required processing duration T_i^{Dur} and a time window T_i^{Win} within which the duration must be allocated; we denote the lower and upper bounds of the time window by $T_i^{Win}(LB)$ and $T_i^{Win}(UB)$, respectively. We assume that time windows do not overlap day boundaries; consequently, all times are constrained to the interval $[1, 1440]$, where 1440 is the number of minutes in a 24-hour period. Tasks cannot be preempted once processing is initiated, and concurrency is not allowed, i.e., the resources are unit-capacity. In the first abstraction, there are n task requests to be scheduled on a *single* resource (i.e., communications antenna). Clearly, if alternative resources for task requests are not specified, there are no resource interactions, and the tasks can be scheduled on each resource independently; we refer to this abstraction as Single-Resource Range Scheduling (SiRRS), which we investigate in Section 3.3. In the second abstraction, each task request T_i additionally specifies a resource $R_i \in [1..m]$, where m is the total number of resources available. Further, T_i may optionally specify $j \geq 0$ additional (R_i, T_i^{Win}) pairs, each identifying a particular alternative resource and time window for the task; we refer to this formulation as Multi-Resource Range Scheduling (MuRRS).

Although it hasn't been previously studied, the SiRRS is equivalent to a well-known problem in the machine scheduling literature, denoted $1|r_j|\sum U_j$, in the three-field notation widely used by the scheduling community [GLJK79], where:

$\mathbf{1}$ denotes that tasks (jobs) are to be processed on a single machine,
 r_j indicates that a release time is associated with job j , and
 $\sum U_j$ denotes the number of tasks not scheduled by their due date.

Using a more detailed notation, each task T_j , $1 \leq j \leq n$ has (1) a release date T_j^{Rel} , (2) a due date T_j^{Due} , and (3) a processing duration T_j^{Dur} . A job is on time if it is scheduled between its release and due date; in this case, $U_j = 0$. Otherwise the job is late, and $U_j = 1$. The objective is to minimize the number of late jobs, $\sum_{j=1}^n U_j$. Concurrency and preemption are not allowed.

$1|r_j|\sum U_j$ scheduling problems are often formulated as decision problems, where the objective is to determine whether a solution exists with L or fewer tasks (i.e., con-

flicts) completing after their due dates, without violating any other task or problem constraints. The $1|r_j|\sum U_j$ scheduling problem is \mathcal{NP} -complete [GJ77] [GJ79]. This result can formally be used to show that Satellite Range Scheduling is also \mathcal{NP} -complete [BWWH02].

3.2 Algorithms for the Satellite Range Scheduling Problem

In this section, we document the various heuristic algorithms that we used in our analysis of both SiRRS and MuRRS. We begin by describing a branch-and-bound algorithm designed by Baptiste et al. (1998) for the $1|r_j|\sum U_j$ machine scheduling problem. Next we briefly discuss two greedy heuristic algorithms designed specifically for the $1|r_j|\sum U_j$ problem. We introduce the methods for both solution encoding and evaluation in local search algorithms for both formulations of the Satellite Range Scheduling, and define the core algorithms used in our analysis: the *Genitor* genetic algorithm, a hill-climbing algorithm, and random sampling. Finally, we conclude by discussing our decision to omit some algorithms based on well-known scheduling technologies.

3.2.1 Branch-and-Bound Algorithm for Single-Resource Range Scheduling

Baptiste et al. [BPP98] introduce a branch-and-bound algorithm to solve the $1|r_j|\sum U_j$ problem. The algorithm starts by computing a lower bound on the number of late tasks, v . Branch-and-bound is then applied to the decision problem of finding a schedule with v late tasks. If no such schedule can be found, v is incremented and the process is repeated. When solving the decision problem, at each node in the search tree, the branching scheme selects an unscheduled task and attempts to schedule the task. The choice of the task to be scheduled is made based on a heuristic that prefers small tasks with large time windows over large tasks with tight time windows. Dominance properties and constraint propagation are applied; then the feasibility of the new one-machine schedule is checked. If the schedule is infeasible, the algorithm backtracks, and the task is considered late. Determining the feasibility of the one-machine schedule at each node in the search tree is also \mathcal{NP} -hard. However, Baptiste et al. note that for most of the cases a simple heuristic can decide feasibility; if not, a branch and bound algorithm is applied.

3.2.2 Constructive Heuristic Algorithms

Constructive heuristics begin with an empty schedule and iteratively add jobs to the schedule using local, myopic decision rules. These heuristics can generate solutions to even large problem instances in sub-second CPU time, but because they typically employ no backtracking, the resulting solutions are generally sub-optimal. Machine scheduling researchers have introduced two such greedy constructive heuristics for $1|r_j|\sum U_j$.

Dauzère-Pérès (1995) introduced a greedy heuristic to compute upper bounds for a branch-and-bound algorithm for $1|r_j|\sum U_j$; we denote this heuristic by *Greedy_{DP}*. The principle underlying *Greedy_{DP}* is to schedule the jobs with little remaining slack immediately, while simultaneously minimizing the length of the partial schedule at each step. Thus, at each step, the job with the earliest due date is scheduled; if the completion time of the job is after its due date, then either one of the previously scheduled jobs or this last job is dropped from the schedule. Also, at each step, one of the jobs dropped from the schedule can replace the last job scheduled if by doing so the length of the partial schedule is minimized. By compressing the partial schedule, more jobs in later stages of the schedule can be accommodated.

Bar-Noy et al. (2002) introduced a greedy heuristic, which we denote by *Greedy_{BN}*, based on a slightly different principle: schedule the job that *finishes* earliest at each step, independent of the job due date. Consequently, *Greedy_{BN}* can schedule small jobs with significant slack before larger jobs with very little slack. Both *Greedy_{DP}* and *Greedy_{BN}* are directly applicable only to SiRRS; simple extensions of these heuristics to MuRRS are discussed in Section 3.6.

3.2.3 Local Search Algorithms

In contrast to constructive heuristic algorithms, local search algorithms begin with one or more complete solutions; search proceeds via successive modification to a series of complete solution(s). All local search algorithms we consider encode solutions using a permutation π of the n task request IDs (i.e., $[1..n]$); a *schedule builder* is used to generate solutions from a permutation of request IDs. The schedule builder considers task requests in the order that they appear in π . For SiRRS, the schedule builder attempts to schedule the task request within the specified time window. For the MuRRS, each task request is assigned to the first available resource (from its list of alternatives) and at the earliest possible starting time. If the request cannot be scheduled on any of the alternative resources, it is dropped from the schedule (i.e.,

bumped). The evaluation of a schedule is then defined as the total number of requests that are scheduled (for maximization) or inversely, the number of requests bumped from the schedule (for minimizing).

A Next-Descent Hill-Climbing Algorithm

Perhaps the simplest local search algorithm is a hill-climber, which starts from a randomly generated solution and iteratively moves toward the best neighboring solution. A key component of any hill-climbing algorithm is the move operator. Because little problem-specific knowledge is available for Satellite Range Scheduling, we have selected a domain-independent move operator known as the *shift* operator. Local search algorithms based on the shift operator have been successfully applied to a number of well-known scheduling problems, for example the permutation flow-shop scheduling problem [Tai90]. From a current solution π , a neighborhood under the shift operator is defined by considering all $(N - 1)^2$ pairs (x, y) of task request ID positions in π , subject to the restriction that $y \neq x - 1$. The neighbor π' corresponding to the position pair (x, y) is produced by *shifting* the job at position x into the position y , while leaving all other relative job orders unchanged. If $x < y$, then $\pi' = (\pi(1), \dots, \pi(x - 1), \pi(x + 1), \dots, \pi(y), \pi(x), \pi(y + 1), \dots, \pi(n))$. If $x > y$, then $\pi' = (\pi(1), \dots, \pi(y - 1), \pi(x), \pi(y), \dots, \pi(x - 1), \pi(x + 1), \dots, \pi(n))$.

Given the relatively large neighborhood size, we use the shift operator in conjunction with next-descent (as opposed to steepest-descent) hill-climbing. The neighbors of the current solution are examined in a random order, and the first neighbor with either a lower or equal fitness (i.e., number of bumps) is accepted. Search is initiated from a random permutation and terminates when a pre-specified number of solution evaluations is exceeded.

In practice, neighborhood search algorithms fail to be competitive because the neighborhood size is so large. With 500 task requests, the number of neighbors under both shift and other well-known problem-independent move operators is approximately 500^2 . The best algorithms for this problem produce solutions using fewer than 10,000 evaluations.

The *Genitor* Genetic Algorithm

Previous studies of Satellite Range Scheduling by AFIT researchers indicate that the *Genitor* genetic algorithm [Dav91] provides superior overall performance [Par94]. *Genitor* has also been successfully used by Braun (2001) to schedule tasks for heterogeneous computing systems (see Section 2). As mentioned in Section 2, Wolfe and Sorensen(2000) also obtained good performance results with a genetic algorithm for

the Window Constrained Packing (WCP) Problem; the WCP problem differs from the SiRRS solely in the choice of the objective function.

As in the hill-climbing algorithm, solutions are encoded as permutations of the task request IDs. Like all genetic algorithms, *Genitor* maintains a population of solutions. In each step of the algorithm, a pair of parent solutions is selected, and a crossover operator is used to generate a single child solution, which then replaces the worst solution in the population. The result is a form of elitism, in which the best individual produced during the search is always maintained in the population. Selection of parent solutions is based on the rank of their fitness, relative to other solutions in the population. A linear bias is used such that individuals that are above the median fitness have a rank-fitness greater than one and those below the median fitness have a rank-fitness of less than one [Whi89].

Typically, genetic algorithms encode solutions using bit-strings, which enable the use of “standard” crossover operators such as one-point and two-point crossover [Gol89]. Because solutions in *Genitor* are encoded as permutations, a special crossover operator is required to ensure that the recombination of two parent permutations results in a child that (1) inherits good characteristics of both parents and (2) is still a permutation of the n task request IDs. Following Parish (1994), we use Syswerda’s (relative) order crossover operator, which preserves the relative order of the elements in the parent solutions in the child solution. Syswerda’s crossover operator has been successfully applied in a variety of scheduling applications [Sys91], [WRWH99], [SP91].

The version of the *Genitor* Genetic Algorithm used here was originally developed for a warehouse scheduling application [RHWM96, WHR⁺98, WRWH99], but it has also been applied to problems such as job shop scheduling [VW00].

3.2.4 A Baseline: Random Sampling

Random sampling produces schedules by generating a random permutation of the task request IDs and evaluating the resulting permutation using the scheduler builder introduced earlier in this section. Randomly sampling a large number of permutations provides information about the distribution of solutions in the search space, as well as a baseline measure of problem difficulty for heuristic algorithms.

3.2.5 Other Scheduling Algorithms

We also considered straightforward implementations of tabu search [GL97] for MuRRS, but the resulting algorithms were not competitive with *Genitor*. With 500 task requests, the number of neighbors under both shift and other well-known problem-independent move operators is approximately 500^2 . With such a large neighborhood, tabu search and other forms of neighborhood local search are simply not practical. We have explored methods for reducing the neighborhood size, but performance was extremely poor.

Additionally, we developed constructive search algorithms for SiRRS based on texture-based [BDSF97] and slack-based [SC93] constraint-based scheduling heuristics. We found that texture-based heuristics are effective when the total number of task requests is small (e.g., $n \leq 100$) for SiRRS. We considered similar algorithms for MuRRS, but were largely unsuccessful; straightforward extensions of constraint-based technologies for multiple resources with alternatives did not appear to be effective.

3.3 Problem Difficulty and Single-Resource Range Scheduling

Given the equivalence of the SiRRS with the $1|r_j|\sum U_j$ machine scheduling problem, our first goal was to analyze the performance of heuristic algorithms designed specifically for $1|r_j|\sum U_j$; if these algorithms are competitive, extensions may provide good performance on the more complex MuRRS problem. Secondly, SiRRS is much easier to analyze, giving us a starting point for understanding the more complex version of the problem.

The question we investigated in the context of the SiRRS was: do heuristics originally developed in the context of $1|r_j|\sum U_j$ out-perform heuristics explicitly developed for Satellite Range Scheduling? We developed a problem generator based on the characteristics of the AFSCN application. The AFSCN schedules task requests on a per-day basis; consequently, we restricted the lower and upper bounds of the task request time windows T_i^{Win} to the interval $[1, 1440]$, or the number of minutes in a 24-hour period. In the AFIT benchmark problems, the overwhelming majority (more than 90%) of task request durations T_i^{Dur} fall in the interval $[20, 60]$. We denote the *slack* T_i^{Slack} associated with a request T_i by $T_i^{Win}(UB) - T_i^{Dur} - T_i^{Win}(LB)$; the slacks of task requests in the AFIT benchmark problems generally range from 0 to 100. Finally, no communications antennas in any AFIT benchmark problem is assigned more than 50

task requests, and typically far fewer.

Based on the above observations, we generated random instances of SiRRS problems using the following procedure for each of the n task requests, which takes as input the maximum slack value $MAXSLACK$ allowed for any task request:

1. Sample the processing duration T_i^{Dur} uniformly from the interval $[20, 60]$.
2. Sample the slack T_i^{Slack} uniformly from the interval $[0, MAXSLACK]$.
3. Sample the window start time $T_i^{Win}(LB)$ uniformly from the interval $[1, 1440 - T_i^{Slack} - T_i^{Dur}]$.
4. Let $T_i^{Win}(UB) = T_i^{Win}(LB) + T_i^{Slack} + T_i^{Dur}$.

We generated 100 random instances for each combination of $n \in [30, 40, 50, 60]$ and $MAXSLACK \in [0, 25, 50, 75, 100, 125, 150, 175, 200]$. In the context of the AFSCN scheduling problem, problem sets with small-to-moderate n and $MAXSLACK$ correspond to realistic problem instances; instances with larger values of n and $MAXSLACK$ are generally unrealistic, but were included to bracket performance.

We used a branch and bound algorithm developed by Baptiste et al. (1998)³ to compute optimal solutions for our test instances. We ran random sampling, the two greedy constructive heuristic algorithms introduced in Section 3.2.2, as well as hill climbing and *Genitor* for each problem instance. We compared the results to the optimal solutions computed using branch and bound. The results showed that the two heuristics originally developed in the context of $1|r_j|\sum U_j$ out-perform, on average, heuristics explicitly developed for Satellite Range Scheduling. The equivalence of SiRRS to $1|r_j|\sum U_j$ scheduling enabled us to leverage algorithms for computing optimal solutions to problem instances. Our results demonstrate that SiRRS problems with characteristics found in real-world AFSCN problems can often be solved to optimality, and in general, near-optimal solutions can be found.

3.4 Scheduling the Low-Altitude Requests

As shown in Section 2, Burrowbridge (1999) considered a simplified version of SiRRS, where only low-altitude satellites are present and the objective is to maximize the

³We thank Dr. Philippe Baptiste for providing executable versions of their branch-and-bound algorithm.

number of scheduled tasks. The well-known *greedy activity-selector* algorithm [CLR90] is used to schedule the requests since it yields a solution with the maximal number of scheduled tasks.

We proved that Satellite Range Scheduling for low altitude satellites continues to have polynomial time complexity even if multiple resources are options. In particular, this occurs in the case of scheduling low-altitude satellite requests on one of the k antennas present at a particular ground station. For our proof, the k antennas must represent equivalent resources. We viewed the problem as one of scheduling multiple, but identical resources.

We modified the greedy activity-selector algorithm for multiple resource problems: the algorithm still schedules the requests in increasing order of their due date, however it specifies that each request is scheduled on the resource for which the idle time before its start time is the minimum. Minimizing this idle time is critical to proving the optimality of the greedy solution. We called this algorithm *Greedy_{IS}* (where *IS* stands for Interval Scheduling) and showed that it is optimal for scheduling the low-altitude requests. The problem of scheduling the low-altitude requests is equivalent to an interval scheduling problem with k identical machines (for more on interval scheduling, see Bar-Noy et al. (2002), Spieksma (1999), Arkin et al. (1987)). It has been proven that for the interval scheduling problem the extension of the greedy activity-selector algorithm is optimal; the proofs are based on the equivalence of the interval scheduling problem to the k -colorability of an interval graph [CL95]. We devised a new proof, similar to the one in [CLR90] for the one-machine case (the proof was the result of a group effort and is given here for completeness reasons).

Theorem 1 *The Greedy_{IS} algorithm is optimal for scheduling the low-altitude satellite requests for MuRRS with identical resources and no slack.*

Proof: Let S^* be an optimal schedule. Let G be the greedy schedule. Assume S^* differs from G . We show that we can replace all the non-greedy choices in S^* with the greedy tasks scheduled in G ; during the transformation, the schedule remains feasible and the number of scheduled requests does not change and therefore remains optimal. This transformation also proves that G is optimal.

The transformation from S^* to G deals with one request at a time. We examine S^* starting from time 0 and compare it with G . Let t_D be the first point in time where G differs from S^* . Let Y be the next request picked to be scheduled in G after time t_D . This means that Y is the next job scheduled after time t_D in G with the earliest finish time. Let the resource where request Y is scheduled in G be R_Y . Let $P(Y)$ be the

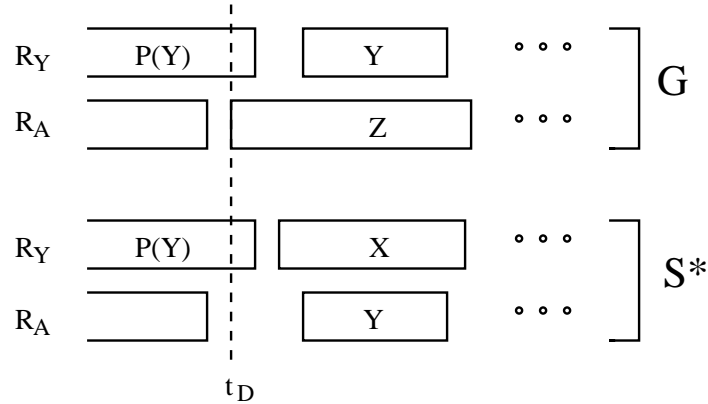


Figure 1: The optimal and greedy schedule are identical before time t_D . Note that Y was the next task scheduled in the greedy schedule G . In this case, Y appears on some alternative resource, R_A , in the optimal schedule S^* , and X appears on resource R_Y . Note that the start time of request Z is irrelevant, since the transformation is performed on schedule S^* .

job preceding Y in the greedy schedule G . When constructing the greedy schedule G , Y is chosen to have the earliest due date *and* is placed on the resource that results in minimum idle time. If Y is not scheduled the same in S^* and G , then either Y does not appear at all in S^* (case 1), or it appears on an alternative resource R_A in S^* (case 2, see Figure 1).

Case 1: The greedy choice Y is not present in S^* . Instead, some request X appears on R_Y . Then Y can replace X in the optimal schedule because the due date of Y is earlier than the due date of X . Also, there is no conflict in the start time, since in both schedules the requests X and Y follow $P(Y)$ in S and G respectfully. The number of scheduled tasks does not change.

Case 2: The greedy choice Y is present in S^* on resource R_A and some other request X follows $P(Y)$ on resource R_Y in S^* . We can then swap *all of the subsequent requests* scheduled after the finish time of $P(Y)$ on resources R_A and R_Y in schedule S^* . Again, there is no conflict in the start times, since in the two schedules the requests X and Y follow $P(Y)$ in S and G respectfully. This places Y on the same resource on which it appears in G . The number of scheduled tasks does not change.

The transformation continues until S^* is transformed into G . One of the two cases always applies. At no point does the number of scheduled tasks change. Therefore, G is also optimal. \square

3.5 The Multi-Resource Range Scheduling with Alternatives: Algorithm Analysis and the AFIT Benchmark Problems

This section presents an analysis of heuristic algorithm performance for MuRRS, which more accurately models the real AFSCN scheduling problem. We investigated three questions: Can the results of SiRRS be leveraged for MuRRS? What is the best performing algorithm for this problem? Why? To address these questions, we tested the same algorithms as for SiRRS (modified as possible to fit the demands of the new problem) on MuRRS problems. We then hypothesized and tested an explanation for the observed performance.

As discussed in Section 2, heuristics for MuRRS have previously been evaluated using only the seven problem instances in the AFIT benchmark. Although the set includes both high and low-altitude satellite requests along with alternatives, the low-altitude requests in these problems can be scheduled only at one ground station (by assigning it to one of the antennas present at that ground station). The number of requests to be scheduled for the seven problems are 322, 302, 300, 316, 305, 298, and 297 respectively. Since 1992, the number of requests received during a typical day has increased substantially (to more than 500 each day), while the resources have remained more or less constant.

In our experimental setup, we replicated the conditions and the reported results from Parish’s study [Par94]. We ran *Genitor* on each of the seven problems in the benchmark, using the same parameters: population size 200, selective pressure 1.5, order-based crossover, and 8000 evaluations⁴ for each run. We also ran random sampling and hill-climbing on each AFIT problem, with a limit of 8000 evaluations per run. For each algorithm, we performed a total of 30 independent runs on each problem.

Currently, no complete algorithm is available for computing optimal solutions for MuRRS. Consequently, all comparisons of heuristic algorithms for MuRRS are necessarily relative; even if one algorithm consistently outperforms another, there is no assurance that the algorithm is generating optimal, or even near-optimal, solutions. Consequently, to baseline the performance of the search algorithms, we also implemented a greedy constructive heuristic based on the *Greedy_{DP}* as follows: First, we used *Greedy_{DP}* to schedule the primary requests (ignoring the alternative resources) for each of the specified resources; the result is an initial schedule corresponding to the primary requests. Then we considered the unscheduled requests and attempted to insert them in the schedule using the specified alternative resources. The unsched-

⁴An increase in the number of evaluations to 50k and of the population size to 400 did not improve the best solutions found for each problem.

Day	<i>Genitor</i>			Hill Climbing			Random Sampling			Greedy _{DP}	Greedy _{DP}	MIP
	Min	Mean	Stdev	Min	Mean	Stdev	Min	Mean	Stdev	-LB		
1	8	8.6	0.49	15	18.16	2.54	21	22.7	0.87	20	20	10
2	4	4	0	6	10.96	2.04	11	13.83	1.08	18	19	6
3	3	3.03	0.18	11	15.4	2.73	16	17.76	0.77	22	24	7
4	2	2.06	0.25	12	17.43	2.76	16	20.20	1.29	18	20	7
5	4	4.1	0.3	12	16.16	1.78	15	17.86	1.16	15	18	6
6	6	6.03	0.18	15	18.16	2.05	19	20.73	0.94	25	27	7
7	6	6	0	10	14.1	2.53	16	16.96	0.66	22	22	6

Table 2: Performance of *Genitor*, hill climbing, and random sampling on the AFIT benchmark problems, in terms of the best and mean number of bumped requests. All statistics are taken over 30 independent runs. The results of running the greedy heuristic *Greedy_{DP}* are also included. The last column reports the performance of Schalck’s [Sch93] Mixed-Integer Programming algorithm.

uled requests were considered in an arbitrary order; each request was scheduled at the earliest time on the first alternative resource available and bumped if none of the alternative resources were available.

The results are summarized in Table 2. Included in the table are the results obtained by Schalck using Mixed Integer Programming [Sch93]. As previously reported, *Genitor* yielded the best overall performance. For the *Greedy_{DP}*, we also computed for each problem a lower bound on the number of bumped requests, equal to the number of requests which cannot be scheduled at any of their alternative resources after scheduling the primary requests. We denoted this lower bound *Greedy_{DP} – LB*. Independent of the heuristic used to insert the unscheduled requests in the schedule corresponding to the primary requests, the final number of bumped requests is greater than or at most equal to the lower bound. *Greedy_{DP}* resulted in the worst performance and was often outperformed by random sampling. Scheduling the primary requests first drastically narrows the opportunities of scheduling the rest of the requests using the alternative resources.

3.5.1 Explaining the Performance on the Benchmarks

To exploit the differences in scheduling slack and the number of alternatives between low and high-altitude requests, we designed a simple greedy heuristic (which we call the “split heuristic”) that first schedules all the low-altitude requests in the order given by the permutation, followed by the high-altitude requests. We showed that: (1) for

more than 90% of the best known schedules found by *Genitor*, the split heuristic does not increase the number of conflicts in the schedule, and (2) the split heuristic typically produces good (and often best-known) schedules.

We hypothesized that *Genitor* may be learning to schedule the low-altitude requests before the high-altitude requests, leading to the strong overall performance. If true, the evaluation of high-quality schedules should, on average, remain unchanged when the split heuristic is applied. To test this hypothesis, we ran 1000 trials of *Genitor* on each AFIT problem. The resulting permutations found by *Genitor* were then interpreted using the split heuristic: all of the low altitude requests were scheduled first, but both the low and high altitude requests were still scheduled based on the order in which they appear in the permutation.

The results are summarized in Table 3. The second column (labeled “Total Number of Best Known Found”) records the number of schedules (out of 1000) with an evaluation equal to the best found by *Genitor* in any run. We then applied the split heuristic to each such schedule. The schedules resulting from the split heuristic fall into three categories. First, the conflicts are identical to those found by *Genitor* when interpreted without the split heuristic; the number of schedules in this category is given in the third column (“Same Evaluation Same Conflicts”). Second, the number of bumps is the same but the bumped tasks differ from those found by *Genitor* without the split heuristic. The number of schedules in this category is given in column “Same Evaluation Different Conflicts”. Third, the evaluation of the schedules changes when the split heuristic is applied. The last column reports the number of schedules in this category. By separating the requests from the schedules produced by *Genitor* into low and high-altitude requests, the evaluation of more than 80% of the schedules remains unchanged. The numbers in the last column of the table also warn that when using the split heuristic only a subspace of the permutations is considered (the permutations that are separated into low and high-altitude requests); this subspace does not contain all the best-known solutions, and, in fact, for different instances of the problem this subspace could be suboptimal. But more than 90% of the time (i.e., in $(4553 + 1120)/6182$ cases), the same evaluation results when the same permutation is 1) directly mapped to a schedule (first come, first served, based on the order of the permutation) to obtain a solution, or when 2) all of the low altitude requests from the permutation are filled first and then all of the high altitude requests are scheduled. This suggests that *Genitor* is indeed scheduling low altitude requests with high priority most of the time.

Our second hypothesis was that using the split heuristic results in solutions with a small number of conflicts. Figure 2 presents a summary of the results obtained when

Day	Total Number of Best Known Found	Same Evaluation Same Conflicts	Same Evaluation Different Conflicts	Worse Evaluation
1	420	38	373	9
2	1000	726	106	168
3	996	825	115	56
4	937	733	50	154
5	862	800	12	50
6	967	843	56	68
7	1000	588	408	4
TOTAL	6182	4553	1120	509

Table 3: The effect of applying the split heuristic when evaluating best known schedules produced by *Genitor*

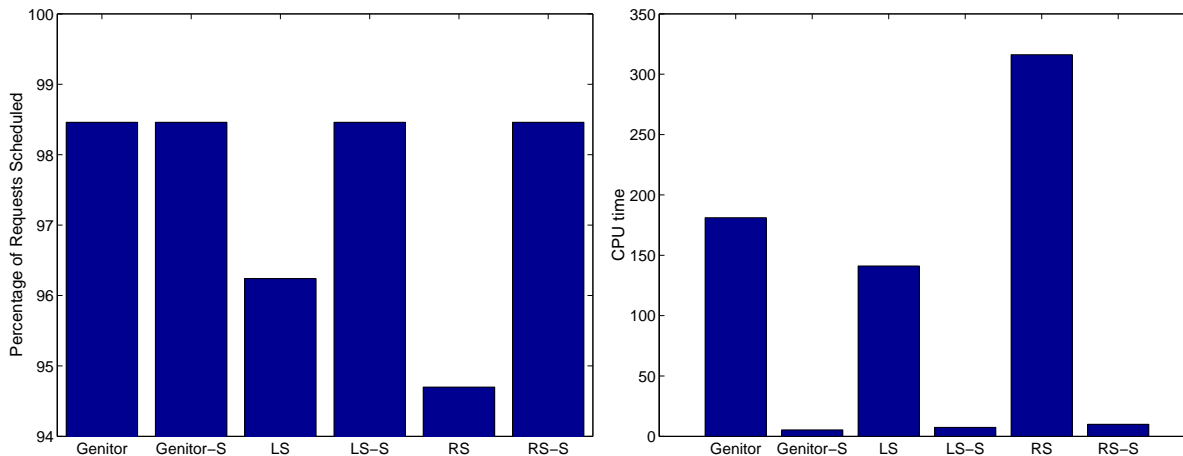


Figure 2: Algorithm performance for the seven AFIT benchmark problems

using *Genitor*, hill climbing, and Random Sampling without the split heuristic (30 experiments, 8000 evaluations per experiment), as well as the split versions denoted by *Genitor-S*, Hill Climbing-S and Random Sampling-S. The split versions of the three algorithms were run in 30 experiments with 100 evaluations per experiment. The minimum number of bumps in 30 experiments is recorded for each problem as the percentage of requests scheduled. The left half of Figure 2 presents the average percentage of requests scheduled for the seven problems by each algorithm. The corresponding average CPU times (in seconds) appear in the right half of the figure. For all the problems, even the simplest algorithm, Random Sampling-S, finds the best known solutions, as illustrated in Table 4.

Day	Best Known	Random Sampling-S		
		Min	Mean	Stdev
1	8	8	8.2	0.41
2	4	4	4	0
3	3	3	3.3	0.46
4	2	2	2.43	0.51
5	4	4	4.66	0.48
6	6	6	6.5	0.51
7	6	6	6	0

Table 4: Results of running random sampling in 30 experiments, by generating 100 random permutations per experiment. A problem-specific heuristic is used in the evaluation function, where the low-altitude requests are evaluated first.

We can prove that, in fact, *the low altitude requests are scheduled optimally*. Since the split heuristic divides the low and high altitude requests, the low altitude request, which have no slack and no alternative windows for scheduling, correspond to multiple instances of SiRRS problems with no slack and no time windows; in other words, the low altitude requests are optimally solved by the *Greedy_{IS}* algorithm. We implemented *Greedy_{IS}* and used it to schedule the low altitude requests. In every case, the overall solution found by *Genitor* generates subsolutions over the subset of low altitude requests that are identical in evaluation to those found by *Greedy_{IS}*.

This also suggests that another way to solve these problems is to use *Greedy_{IS}* to schedule the low altitude requests, and then use *Genitor* to schedule the remaining high altitude requests. This fact, coupled with the fact that the best known solutions can also be obtained by randomly sampling a small number of permutations, indicates that the AFIT benchmarks are easy to solve by first splitting the data into low altitude and high altitude requests.

When does the “split heuristic” fail? We can build a simple problem instance for which the optimal solution cannot be found using the split heuristic. Consider the problem represented in Figure 3. There are two ground stations with two antennas each. There are two high-altitude requests, *R3* and *R4*, with durations three and seven, respectively. *R3* can be scheduled between start time 4 and end time 13; *R4* can be scheduled between 0 and 9. Both *R3* and *R4* can be scheduled at either of the two ground stations. The rest of the requests are low-altitude requests. *R1* and *R2* request the first ground station, while *R5*, *R6*, *R7*, and *R8* request the second ground

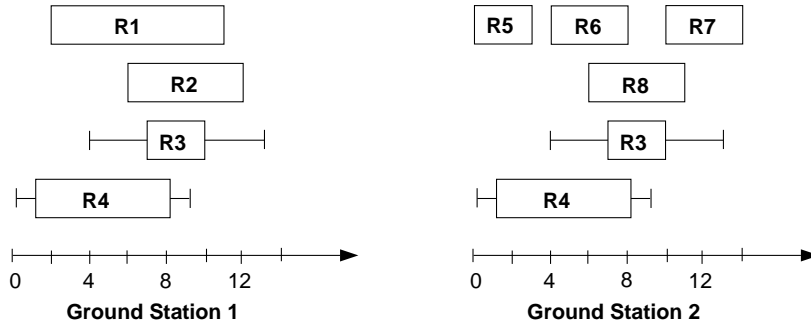


Figure 3: Example of a problem for which the split heuristic cannot result in an optimal solution. Each ground station has two antennas; the only high-altitude requests are $R3$ and $R4$.

station. This problem fits the description of the Satellite Range Scheduling problems in the AFIT benchmark: the low-altitude requests can be scheduled only at a specific ground station, with a fixed start and end time, while the high-altitude requests have alternative resources and a time window specified. For all the permutation schedules, if the split heuristic is used, $R3$ and $R4$ cannot be scheduled. However, it is possible to find schedules where both $R3$ and $R4$ get scheduled, and only one request ($R1$, $R2$, or $R8$) gets bumped. The subspace containing the permutations with all the low-altitude requests before the high-altitude requests is suboptimal – the global optimum is not necessarily contained in this subspace. The example shows the potential for failure to generate optimal solutions using the split heuristic.

3.6 Generalizing the AFIT Problems

Does the algorithm performance obtained for the AFIT benchmark transfer to contemporary problems (e.g., larger number of tasks)? Do the results obtained by Parish showing that *Genitor* outperforms other algorithms also hold on other, larger problems? To explore these questions, we built a problem generator that produces problems similar to the AFIT benchmark by modeling features encountered in the real-world problems. Then we compared the results of running *Genitor*, hill climbing, and random sampling on problems produced by the problem generator to the results reported for the AFIT problems. We showed that: (1) the performance of the split heuristic on the seven AFIT problems does not transfer to the problems produced by our generator and (2) *Genitor* consistently results in the smallest number of unscheduled requests.

Customer Type	Predictability	Fract. of Reqs. for Each Request Type			
		H/S	P/D	P/C	Mu
Operations and Maintenance	0.9	0.85	0.0	0.1	0.05
Image Intelligence	0.3	0.0	0.5	0.5	0.0
Signal Intelligence	0.7	0.05	0.1	0.85	0.0

Table 5: Customer types

3.6.1 New Problem Generator

We have designed a new, more realistic problem generator, which preserves some of the features of both SiRRS problems as well as the AFIT benchmark problems, while introducing new characteristics of the contemporary application.

The SiRRS problem generator described in section 3.3 requires parameters to control the request duration and the size of the time window. In the new generator, we use these parameters to model different types of requests encountered in the real-world satellite scheduling problem, such as downloading data from a satellite, transmitting information or commands from a ground station to a satellite, and checking the health and status of a satellite. We classify the requests into four possible types:

- State of health (H/S): short (5-15 minutes), flexible, common
- Maneuver (Mu), for example changing the orbit of a satellite: longer (20-60 minutes), inflexible, rare
- Payload download (P/D), for example downloading data from the satellite: long (10-30 minutes), more flexible, common
- Payload commanding (P/C): variable length (5-60 minutes), flexible, common

Our taxonomy is based on the fact that requests of a certain type share characteristics of duration, flexibility (the size of the time window versus the duration of the request) and frequency⁵. For each request type, we define the upper and lower bound on its duration T_i^{Dur} and the maximum slack $MAXSLACK$ (to determine the window size). The actual requests are generated using these parameters as described in section 3.3.

⁵For now, we do not model periodic requests and therefore do not use the frequency information.

As a second feature, the new generator introduces models for customer behavior. We define three customer types (see Table 5), based on the fact that, for example, some customers will mostly generate health and status requests or payload command requests. For each customer, the number of requests to be generated is determined as a fraction of the total number of requests. We define customer patterns by specifying a time window midpoint, a resource, and the type of request. Customer patterns model the previous behavior of a customer (which could have been collected from past schedules). A database of customer patterns is used to produce the requests. For each customer type, the fraction of the requests generated for each request type and the *Predictability* are defined. The *Predictability* for each customer represents the fraction of requests that will be generated for this customer using customer patterns from the database. The rest of the requests corresponding to each customer will be randomly generated. The duration and window size for each request are determined using the parameters associated with the request type.

The AFIT problems separate the requests into low altitude and high altitude. The low altitude requests tend to be shorter and have a duration equal to the corresponding visibility window. The high altitude requests are more flexible, with the size of the visibility window larger than the requested duration. The AFIT problems assume that all the low altitude requests have to be scheduled at the same ground station. In our new generator, we specify more than one ground station as possible alternatives for both low and high altitude requests due to the higher contention for resources now. Also, the time windows specified for alternative ground stations are generated based on an offset in time. The offsets represent the time needed for a satellite in its orbit to become visible from one ground station to another. We use two databases (low altitude and high altitude) of offsets, which were compiled⁶ from data collected on the Web (<http://earthobservatory.nasa.gov/MissionControl/overpass.html>) about the visibilities of various satellites from the locations of the nine ground stations. We preserve the request duration for all the alternatives. The visibility window size is equal to the request duration for all the low altitude requests, and it can vary for the high altitude requests.

The process of generating the requests in the new generator is described in Figure 4. With a 0.5 probability, we determine for each request if it is a low altitude or high altitude one.

⁶Thanks to Ester Gubbrud, a senior undergraduate who spent part of the summer of 2001 working with Adele Howe, for helping us compile the databases.

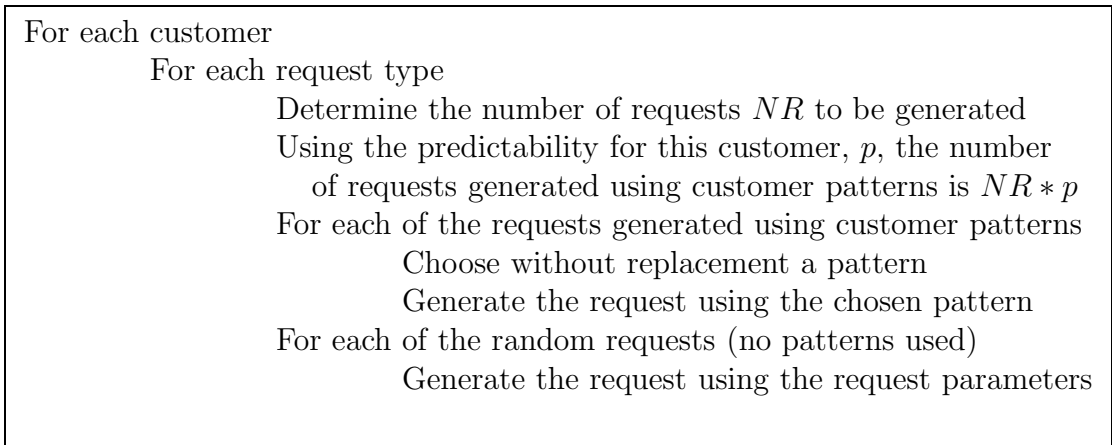


Figure 4: Pseudocode for the new generator

3.6.2 Algorithm Performance

We repeated the experiments described for the AFIT problems (section 3.5) by running Genitor, hill climbing, and random sampling, for problems produced by the new generator. To baseline the performance of the search algorithms, we also ran the greedy constructive heuristic *Greedy_{DP}* extended for MuRRS as described in section 3.5.

To compare our results to the ones reported for the AFIT problems, but also to generate realistic problems, we ran the experiments for problem sizes 300, 350, 400, 450, and 500. For each size, we generated 30 problem instances. For each algorithm, we performed 30 runs with 8000 evaluations per run for each problem. Increasing the number of evaluations to 50k and of the population size to 400 did not improve the best solutions found for each problem. We recorded the number of unscheduled requests for each run.

Figure 5 shows that *Genitor* on average outperforms *Genitor-S* and both versions of hill climbing and random sampling, as well as the greedy constructive heuristic. In fact, as Table 6 shows, *Genitor* (without the split heuristic) always outperforms all the other algorithms. In Tables 6 and 7, we first subtract the minimum number of bumped requests for each problem from the minimum number of bumped requests reported by each of the algorithms for that problem in 30 runs. Then we average these differences over the 30 instances generated for each size; we also compute the variance. From both Figure 5 and Table 6, it is clear that the split heuristic always results in an average decrease in performance. The *Greedy_{DP}* results in the worst

Size	<i>Genitor</i>		Hill Climbing		Random Sampling	
	Mean	Variance	Mean	Variance	Mean	Variance
300	0.000	0.000	0.000	0.000	0.167	0.213
350	0.000	0.000	0.333	0.368	1.067	1.099
400	0.000	0.000	1.233	1.702	2.833	3.523
450	0.000	0.000	3.667	3.678	5.967	6.240
500	0.000	0.000	8.300	3.941	11.767	7.840
Size	<i>Genitor-S</i>		Hill Climbing-S		Random Sampling-S	
	Mean	Variance	Mean	Variance	Mean	Variance
300	0.767	0.737	0.767	0.737	0.867	0.671
350	0.667	0.851	0.967	1.551	1.367	2.033
400	1.100	1.128	2.167	2.626	2.933	3.168
450	1.467	1.223	3.967	4.309	5.200	6.717
500	2.200	2.097	8.700	8.907	10.667	10.161

Table 6: The difference between the minimum number of bumps reported by an algorithm and the minimum number of bumps found by any of the six algorithms (with or without the split heuristic) is averaged over the 30 instances for each problem size

performance; as with the AFIT benchmarks, scheduling the primary requests first drastically narrows the opportunities of scheduling the rest of the requests using the alternative resources.

4 Future Work

When solving an optimization problem, the algorithm chosen should fit the problem, and therefore it should be tailored to the problem features and the objective function. My future work will be focused on answering two general questions related to the impact of problem features and the objective function on algorithm performance for oversubscribed scheduling problems.

The first general research question I will investigate is: “What problem features encountered in oversubscribed scheduling problems influence algorithm performance?”. I intend to consider problem features that are present in most of the oversubscribed scheduling problems. Examples of such problem features are: the request duration

Size	Greedy _{DP} – LB		Greedy _{DP}	
	Mean	Variance	Mean	Variance
300	4.533	3.361	5.167	3.523
350	6.967	9.344	8.067	9.513
400	8.467	8.602	10.667	12.437
450	11.600	9.007	15.500	13.017
500	15.633	13.413	22.400	22.179

Table 7: The difference between the minimum number of bumps reported by an algorithm and the minimum number of bumps found by the greedy heuristic is averaged over the 30 instances for each problem size

versus its time window, characteristics of the alternative resources, such as number and type, the contention for the resources, request priorities, setup times, downtimes etc. By definition, a start time and deadline specify for each request a time window within which the request duration needs to be scheduled; also, contention for the resources is always present. Setup times are encountered in many oversubscribed scheduling applications, such as satellite scheduling, telescope scheduling, the flight simulator. Depending on the problem considered, alternative resources for the requests can also be encountered. Constraints on the availability of the resources (where some resources are available only for certain time intervals) also appear in many oversubscribed scheduling problems; downtimes are an example of this type of constraints.

The second general question I will investigate is: “How does the choice of the objective function impact on algorithm performance for oversubscribed scheduling problems?”. In general, real-world scheduling problems specify multiple (sometimes conflicting) objectives to be optimized. For example, an algorithm reporting good performance in minimizing the idle times on a resource probably will not be suited to maximize the sum of the priorities of the scheduled requests. Smith et al.(1992) have answered a similar question when optimizing two conflicting objectives for the problem of scheduling observations for the Hubble Space Telescope (see Section 2). I will define new objective functions in the context of Satellite Range Scheduling problem and perform empirical studies in order to determine which algorithms perform better given a certain objective function and how different are the solutions reported for similar objective functions.

While the main research focus will be to finalize the study of the Satellite Range Scheduling problem, the problem features and objective functions I will define are

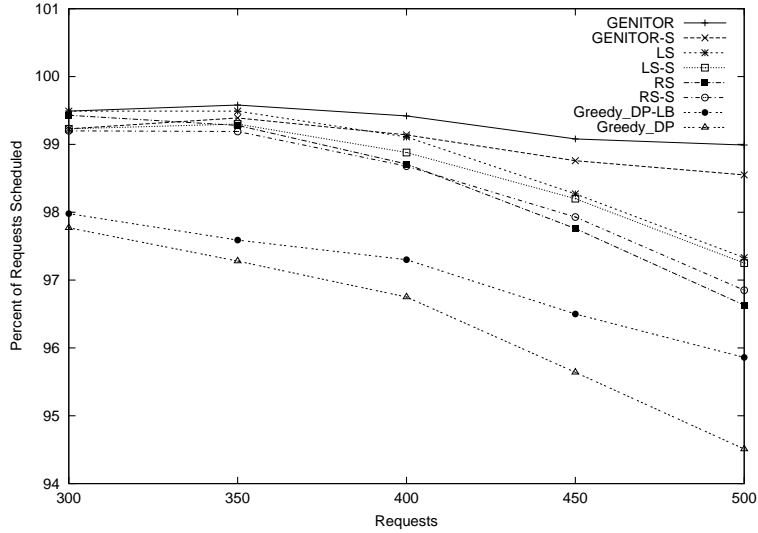


Figure 5: Average percent of requests scheduled by the no-split and split versions of each of the algorithms.

general enough for the results to transfer to other oversubscribed scheduling problems. Therefore, I will investigate how the results generalize for a similar problem in the oversubscribed scheduling domain.

4.1 Problem Features for Satellite Range Scheduling

As mentioned above, the problem features I will consider are: the request duration versus its time window, characteristics of the alternative resources, such as number and type, the contention for the resources, request priorities, setup times, downtimes. Knowledge about problem features can be used to pick an algorithm to solve the problem or to design specific heuristics. For example, we have shown in Section 3 that a greedy algorithm finds optimal solutions for Satellite Range Scheduling problems where only low altitude requests are present and all the alternatives for a request are antennas at the same ground station. For such problems, the request duration is equal to the time window. The problem can be optimally solved no matter how many alternative resources are specified for each request; however, the alternative resources are specified in a certain way. Thus, the requests can be partitioned into subsets based on the alternative resources they require such that all the requests in a subset specify the same alternative resources, and no resource appears as alternative in more than one subset. For such a problem, the optimal solution can be found

independently of the contention for the resources; setup times and downtimes are not present.

Some of these problem features are in fact also considered by Gooley [Goo93] when defining an insertion heuristic for Satellite Range Scheduling. His heuristic iteratively selects the next request to be scheduled, based on the value of the request duration divided by the length of the time window. The requests with a greater value for this fraction are scheduled first (the assumption is that the greater the value, the less flexibility there is in scheduling such a request). His insertion heuristic also considers the number of alternative resources for each request, based on the idea that requests with a smaller number of alternative resources should be scheduled first. While the contention for the resources is not explicitly considered, the heuristic does rank the alternative resources for a request based on the free time available. I will implement Gooley's heuristic and investigate the question: how do various problem features impact the performance of the heuristic? I will also compare this heuristic to other algorithms I implemented. I expect that Genitor without the split heuristic will outperform this new algorithm as well. Gooley's heuristic starts with a schedule containing only low altitude requests and inserts the high altitude requests in this schedule. The low altitude requests are not moved from their initially assigned positions; therefore this new algorithm is somewhat similar to our split heuristic. For the same reasons as the split heuristic, the new algorithm will fail to optimally solve problems such as the one depicted in Figure 3.

A general contention measure for resources is defined by Frank (2001); a similar (but simpler) contention measure is mentioned by Pemberton (2000). Pemberton defines contention as the sum of resource capacity required by all the requests divided by the total capacity available. In addition to these, Frank's formula is based on request priority and number of possible start times for each request. Note that the number of possible start times for a request depends on the request duration and time window (it is given by the difference between the time window and the request duration incremented by one). Obviously, the number of alternative resources available for each request should also be built into the contention measure; Frank and Pemberton do not consider alternative resources. I will therefore incorporate the number of alternative resources into the definition given by Frank for resource contention. I will use the new measure to characterize problem instances and investigate how algorithm performance changes when varying the contention for resources.

Many oversubscribed scheduling problems also specify setup times as intervals of time between the end of a request and the beginning of the next one. Sometimes the duration of the setup times can be slightly decreased, in order to improve the

final schedule. For example, in Satellite Range Scheduling, turn-around times are scheduled before each request. The human schedulers developing schedules for the Satellite Range Scheduling problem have some flexibility in accommodating requests in the schedule. The turn-around times (which need to be scheduled before the actual requests) have predefined values for each antenna; however, these turn-around times can be modified (decreased) such that some requests that initially could not be scheduled will actually make it into the schedule. We have recently obtained data for a schedule generated by human schedulers. By examining the values scheduled for the turn-around times, I will define a set of rules to be used by the algorithms in order to decide if less turn-around time should be scheduled. I will examine the changes in algorithm performance when scheduling less turn-around time.

Downtimes are usually intervals of times when the various resources in the problem are scheduled for maintenance. When a downtime is scheduled, the access of the requests to that resource is blocked; scheduling downtimes will result in less time available to schedule the requests. Obviously scheduling downtimes will impact on the algorithm performance. I will modify the algorithms we currently have to also schedule the downtimes for Satellite Range Scheduling, and analyze the impact of this problem feature on algorithm performance.

4.1.1 New Problem Generator for Satellite Range Scheduling

To support the analysis of the impact of problem features on algorithm performance in the context of Satellite Range Scheduling, I will define a new problem generator.

The need for problem generators in the scheduling domain is motivated by at least two reasons. First, a problem generator offers support for experimental control: by generating problems exhibiting certain attributes, we can formulate and test hypotheses about the relationship between such problem attributes and algorithm performance. As specified before, request length versus time window, number of alternatives for the requests, the contention for the resources, setup times, and downtimes are the problem features that will be modeled using the generator. Second, while evaluating algorithm performance on real problems might be desirable, real problems are usually difficult to obtain and use. It was difficult to obtain real instances of the Satellite Range Scheduling problem; when we obtained the problems, we realized we needed documentation to interpret the data, which was not in a “human-readable” format. The subsequently obtained documentation still does not clearly explain the file format of the problems, leaving us to make certain assumptions about the data. Even though we have to make assumptions about the data, we can identify certain features

present in the real problem and model them in the problem generator.

We have recently obtained real data for scheduling two days⁷, and we hope to get more data. Using the real data, we can reverse engineer certain problem characteristics. We can model the request lengths, the contention for the resources, the characteristics of the alternative resources, and the turn-around times; downtimes can also be identified in the real problems and modeled in the problem generator. I intend to define parameters to control for these problem features. Thus I will generate realistic problem instances, where these parameters have values similar to the ones for the real data. I will also generate problem instances using parameters values that are different from the “realistic” ones, in order to bracket algorithm performance.

4.2 Objective Functions for Satellite Range Scheduling

For Satellite Range Scheduling, the goal of minimizing the number of conflicts in the schedule was chosen because operational schedulers stated that “no conflict or group of conflicts is considered worse than any other conflict” [Goo93]. However, objective functions based on priority values are more common in the oversubscribed scheduling domain [Pem00, VLS96, BVA⁺96, LVJ00, Sys91, Bra01] than an objective function maximizing the number of scheduled requests. Most of the oversubscribed scheduling problems have some priority or preference associated with the requests; these priority values are used to decide which requests to schedule. By defining priority values for the requests, the research on algorithm performance for Satellite Range Scheduling can be brought closer to research on other oversubscribed scheduling applications.

While no explicit priority values are associated with the requests in Satellite Range Scheduling, the human schedulers have additional knowledge about requests that are considered “mission critical”; such requests are given scheduling priority [Goo93]. We can define request priorities, to reflect, for example, the flexibility in scheduling a request. Flexibility could be defined as the number of positions available to schedule a request. Low flexibility requests will be assigned high priority. The assumption behind the way we defined priorities here is that by scheduling the most constrained requests first, more requests can make it into the final schedule. This assumption is similar to classic variable ordering heuristics (which are heuristics deciding which variable to instantiate first); for example, Frank (2001) defines the variable ordering heuristic: “Schedule the observation of highest priority and highest overall contention”. Also,

⁷We thank William Szary and Brian Bayless from the Schriever Air Force Base for providing the real scheduling data

a similar assumption can be identified in Gooley’s insertion heuristic, which defines rules for variable ordering based on request flexibility. The first objective function I will define will maximize the sum of the priorities for the scheduled requests.

Assigning request priorities based on the flexibility in scheduling the request (as described above) is likely to bias the outcome, by giving preference to scheduling low altitude requests first. To test this, I will also use an arbitrary scheme of assigning priorities. Such a scheme will also allow for high altitude requests with a higher priority than low altitude requests and it will disconnect the priority values from the type of the request (low versus high altitude).

A slightly different version of the objective function maximizing the sum of request priorities is defined for some oversubscribed scheduling applications [Bre98, Bra01]. This second objective function also includes a penalty based on the priorities of the **unscheduled** requests.

For these two objective functions based on the priority of the requests, I will design experiments to answer several questions. A first question is: “What algorithm is better suited to optimize the priorities of the requests in the schedule?” To answer this question, I will implement extensions for algorithms specifically designed to optimize the sum of the request priorities for one-machine scheduling problems. I will compare such extensions to genetic algorithms and local search. I will design an experimental study similar to the one described in Section 3. I will investigate how the algorithm performance changes when specifying a penalty for the requests that are not scheduled, as opposed to ignoring these requests when evaluating the schedule. I will also compare the solutions generated by the two objective functions to answer the question: “How different are the solutions generated by using the two objective functions for the same problem?”.

I will also define a third objective function that can be used instead of the one maximizing the number of scheduled requests. Sometimes the request durations and setup times can be slightly modified such that more requests can be scheduled (this usually implies interaction with the customers who generated those requests or with the operators at the ground stations who can modify the setup times). To investigate such opportunities, the scheduling process can be modified such that all the requests are inserted in the schedule. Of course, in the new schedule, the capacity of the resources available is exceeded where the scheduled requests overlap. By defining an objective function minimizing the sum of the overlaps, the resulting schedule will contain information about minimal changes in request durations and/or setup times such that more requests can be scheduled. This objective function can also aid the human schedulers in solving the conflicts present in the schedule. I will implement

algorithms to schedule the requests with overlaps. A first simple approach would be to build an initial schedule using any of the available algorithms and then to heuristically insert in this schedule the unscheduled requests such that the overlaps are minimized. Different approaches could build the schedule with overlaps in one pass. I will compare solutions generated by various approaches for the same problem. I will also compare these solutions to solutions obtained by minimizing the number of bumped requests for the same problem.

4.3 Generalizing the Results

I'm currently looking for a new oversubscribed scheduling application. This application should exhibit certain similarities to Satellite Range Scheduling: for example, the requests should specify release times and deadlines, a duration to be scheduled after the release time and before the deadline, and multiple/alternative resources. Additionally, the requests should include priorities. While no priorities are included for requests in Satellite Range Scheduling problems, most of the problems researched in the oversubscribed scheduling domain specify request priorities.

Given that it may be difficult to obtain real data for a scheduling problem, alternatively, I will define a "generic" oversubscribed scheduling problem. The requests will include release times and deadlines, a duration, multiple/alternative resources, and priorities. Also, setup times (which might depend on the previously scheduled request) as well as downtimes will be specified. I will use contention measures for resources to ensure that the problems are oversubscribed. Such a "generic" scheduling problem is general enough to be a simplified model for Satellite Range Scheduling, as well as telescope scheduling, flight simulator problems, or heterogeneous computing systems. I will also perform a validation study for the "generic" scheduling problem, showing how data for various oversubscribed scheduling problems can be obtained by modifying certain parameter values for the "generic" scheduling problem.

I will build a problem generator for the new application (which will be either a real-world application or the "generic" problem), specifying parameters to control the problem features identified for Satellite Range Scheduling: request length versus time window, number of alternatives for the requests, the contention for the resources, setup times, and downtime. I will then investigate the two main research questions in the context of the new application, by repeating the experiments performed for Satellite Range Scheduling. This experimental study is intended to investigate if the results obtained for Satellite Range Scheduling generalize for other oversubscribed scheduling applications.

5 Schedule

- End of Fall 2002 - new problem generator for Satellite Range Scheduling. I will reverse engineer the new data we obtained to model the load on resources, resource durations, alternatives available for the requests, windows of visibilities. I will also model downtimes and request priorities in the new generator.
- End of Summer 2003 - finalize the research on the impact of problem features and objective functions on algorithm performance for Satellite Range Scheduling. I will design and run experiments, obtain data, analyze the results.
- End of Spring 2004 - generalize the results for a different oversubscribed scheduling application (or for a “generic” oversubscribed scheduling problem). Perform empirical studies similar to the ones ran for Satellite Range Scheduling.
- Summer 2004 - write the thesis, dissertation defense.

6 Expected Contribution

I expect to contribute to the current state of the art in oversubscribed scheduling along several directions. First, I will formally characterize the features of this important class of scheduling problem. While examples of applications are frequently described in the literature, to the best of my knowledge no one has published a study dedicated to oversubscribed scheduling as a class of scheduling problem.

I will implement problem generators based on the description of the two applications and reverse engineering the available data. The scheduling community is interested in new problem generators for real-world problems. Providing better tools for generating synthetic data enables the researchers to hypothesize and test the effects of various problem characteristics. By designing the new problem generators I will also formalize and operationalize the definition of oversubscribed scheduling.

I also expect to provide guidance to the developers in choosing the best algorithm to solve an oversubscribed scheduling problem, based on the problem features and the objective function. Understanding the impact of various problem features and objective functions on algorithm performance should result in better optimization algorithms for solving oversubscribed scheduling applications.

References

- [AS87] E.M. Arkin and E.B. Silverberg. On the k-coloring of intervals. *Discrete Applied Mathematics*, 18:1–8, 1987.
- [BDSF97] J. C. Beck, A. J. Davenport, E. M. Sitarski, and M. S. Fox. Texture-based heuristic for scheduling revisited. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 241–248, Providence, RI, 1997. AAAI Press / MIT Press.
- [BHW02] L. Barbulescu, A.E. Howe, J.P. Watson, and L.D. Whitley. Satellite Range Scheduling: A comparison of genetic, heuristic and local search. In *Parallel Problem Solving from Nature (PPSN)*, 2002.
- [BNGNS02] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing*, 31(2):331–352, 2002.
- [BPP98] P. Baptiste, C. Le Pape, and L. Peridy. Global constraints for partial CSPs: A case-study of resource and due date constraints. In Michael Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming - CP98*, pages 87–101. Springer, 1998.
- [Bra01] T.D. Braun. Heterogeneous distributed computing: Off-line mapping heuristics for independent tasks and for tasks with dependencies, priorities, deadlines, and multiple versions. In *Ph.D. Thesis*. School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, 2001.
- [Bre96] J.L. Bresina. Heuristic-Biased Stochastic Sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 271–278, Portland, OR, 1996.
- [Bre98] J.L. Bresina. Stochastic heuristic search and evaluation methods for constrained optimization. In *Ph.D. Thesis*. Graduate School- New Brunswick, Rutgers, The State University of New Jersey, 1998.
- [Bur99] S.E. Burrowbridge. Optimal allocation of satellite network resources. In *Master's Thesis*. Virginia Polytechnic Institute and State University, 1999.

- [BVA⁺96] E. Bensana, G. Verfaillie, J. Agnese, N. Bataille, and D. Blumstein. Exact and inexact methods for the daily management of an Earth observation satellite. In *Proceedings of the Fourth International Symposium on Space Mission Operations and Ground Data*, Munich, Germany, 1996.
- [BWWH02] L. Barbulescu, J.P. Watson, L.D. Whitley, and A.E. Howe. Scheduling space-ground communications for the Air Force satellite control network. In *Submitted for publication*, 2002.
- [CL95] M.C. Carlisle and E.L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59:225–235, 1995.
- [CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT press, Cambridge, MA, 1990.
- [Cra96] J.M. Crawford. An approach to resource constrained project scheduling. In G.F. Luger, editor, *Proceedings of the 1996 Artificial Intelligence and Manufacturing Research Planning Workshop*. The AAAI Press, Albuquerque, NM, 1996.
- [CRK⁺00] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. ASPEN - automating space mission operations using automated planning and scheduling. In *6th International SpaceOps Symposium (Space Operations)*, Toulouse (France), 2000.
- [Dav91] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
- [DP95] S. Dauzère-Pérès. Minimizing late jobs in the general one machine scheduling problem. *European Journal of Operational Research*, 81:131–142, 1995.
- [ECB⁺01] B. Engelhardt, S. Chien, A. Barrett, J. Willis, and C. Wilklow. The DATA-CHASER and Citizen Explorer benchmark problem sets. In *European Conference on Planning*, Toledo (Spain), 2001.
- [FJMS01] J. Frank, A. Jonsson, R. Morris, and D.E. Smith. Planning and scheduling for fleets of Earth observing satellites. *International Symposium on Artificial Intelligence, Robotics, Automation and Space*, 2001.
- [Fox94] M.S. Fox. ISIS: a retrospective. In Michael B. Morgan, editor, *Intelligent Scheduling*, pages 391–422. Morgan Kaufmann Publishers, 1994.

- [FW92] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. In *Artificial Intelligence*, volume 58, pages 97–109, 1992.
- [GC96] J. Gratch and S. Chien. Adaptive problem-solving for large-scale scheduling problems: A case study. *Journal of Artificial Intelligence Research*, 4:365–396, 1996.
- [GJ77] M.S. Garey and D.S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM Journal on Computing*, 6:416–426, 1977.
- [GJ79] M.S. Garey and D.S. Johnson. *Computers And Intractability: A Guide To The Theory Of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
- [GL97] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, MA, 1997.
- [GLJK79] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
- [Gol89] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [Goo93] T.D. Gooley. Automating the Satellite Range Scheduling process. In *Master's Thesis*. Air Force Institute of Technology, 1993.
- [Jan96] K. Jang. The capacity of the Air Force Satellite Control Network. In *Master's Thesis*. Air Force Institute of Technology, 1996.
- [JM94] M.D. Johnston and G.E. Miller. Spike: Intelligent scheduling of Hubble space telescope observations. In Michael B. Morgan, editor, *Intelligent Scheduling*, pages 391–422. Morgan Kaufmann Publishers, 1994.
- [LVJ00] M. Lemaître, G. Verfaillie, and F. Jouhaud. How to manage the new generation of Agile Earth Observation Satellites. In *6th International SpaceOps Symposium (Space Operations)*, Toulouse (France), 2000.
- [OS97] A. Oddi and S.F. Smith. Stochastic procedures for generating feasible schedules. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-93)*, pages 308–314, Providence, RI, 1997. AAAI Press / MIT Press.

- [Par94] D.A. Parish. A genetic algorithm approach to automating Satellite Range Scheduling. In *Master's Thesis*. Air Force Institute of Technology, 1994.
- [Pem00] J.C. Pemberton. Toward scheduling over-constrained remote-sensing satellites. In *Proceedings of the Second NASA International Workshop on Planning and Scheduling for Space*, pages 84–89, San Francisco, CA, 2000.
- [RCWM99] G. Rabideau, S. Chien, J. Willis, and T. Mann. Using iterative repair to automate planning and scheduling of shuttle payload operations. In *Innovative Applications of Artificial Intelligence (IAAI 99)*, Orlando, FL, 1999.
- [RHWM96] S. Rana, A.E. Howe, L.D. Whitley, and K. Mathias. Comparing heuristic, evolutionary and local search approaches to scheduling. In B. Drabble, editor, *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 174–181. AAAI Press, 1996.
- [RM99] H. Rudova and L. Matyska. Uniform framework for solving over-constrained and optimization problems. In *In CP'99 Post-Conference Workshop on Modeling and Solving Soft Constraints*, Alexandria, VA, 1999.
- [SC93] S. Smith and C.C. Cheng. Slack-based heuristics for constraint satisfaction problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 139–144, Washington, DC, 1993. AAAI Press / MIT Press.
- [Sch93] S.M. Schalck. Automating Satellite Range Scheduling. In *Master's Thesis*. Air Force Institute of Technology, 1993.
- [SGY⁺98] R. Sherwood, A. Govindjee, D. Yan, G. Rabideau, S. Chien, and A. Fukunaga. Using ASPEN to automate EO-1 activity planning. In *Proceedings of the 1998 IEEE Aerospace Conference*, Aspen, CO, 1998.
- [SP91] G. Syswerda and J. Palmucci. The Application of Genetic Algorithms to Resource Scheduling. In L. Booker and R. Belew, editors, *Proc. of the 4th Int'l. Conf. on GAs*. Morgan Kaufmann, 1991.
- [SP92] S.F. Smith and D.K. Pathak. Balancing antagonistic time and resource utilization constraints in over-subscribed scheduling problems. In *The*

Eighth IEEE Conference on Applications of Artificial Intelligence, Monterey, CA, 1992.

- [Spi99] F.C.R. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2:215–227, 1999.
- [Sys91] G. Syswerda. Schedule Optimization Using Genetic Algorithms. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 21. Van Nostrand Reinhold, NY, 1991.
- [Tai90] E.D. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47:65–74, 1990.
- [TBS⁺00] M.D. Theys, N. Beck, H.J. Siegel, M. Jurczyk, and M. Tan. Scheduling heuristics for data requests in an oversubscribed network with priorities and deadlines. In *The 20th International Conference on Distributed Computing Systems*, pages 97–109, Taipei, Taiwan, 2000.
- [Ver00] G. Veraille. Commentary on the paper entitled: Toward scheduling over-constrained remote-sensing satellites. In *Proceedings of the Second NASA International Workshop on Planning and Scheduling for Space*, pages 90–91, San Francisco, CA, 2000.
- [VLS96] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search for solving constraint optimization problems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 181–187, Portland, OR, 1996.
- [VW00] M. Vazquez and D. Whitley. A Comparison of Genetic Algorithms for the Static Job Shop Scheduling Problem. In Schoenauer, Deb, Rudolph, Lutton, Merelo, and Schwefel, editors, *Parallel Problem Solving from Nature*, 6, pages 303–312. Springer, 2000.
- [WBHW99] J.P. Watson, L. Barbulescu, A. E. Howe, and L. D. Whitley. Algorithm performance and problem structure for flow-shop scheduling. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-93)*, pages 688–695, Orlando, FL, 1999. AAAI Press / MIT Press.
- [WBWH02] J.P. Watson, L. Barbulescu, L.D. Whitley, and A.E. Howe. Contrast-ing structured and random permutation flow-shop scheduling problems: Search space topology and algorithm performance. *INFORMS Journal on Computing*, 14(2), 2002.

- [Whi89] L.D. Whitley. The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best. In J. D. Schaffer, editor, *Proc. of the 3rd Int'l. Conf. on GAs*, pages 116–121. Morgan Kaufmann, 1989.
- [WHR⁺98] L.D. Whitley, A.E. Howe, S. Rana, J.P. Watson, and L. Barbulescu. Comparing heuristic search methods and genetic algorithms for warehouse scheduling. *Systems, Man and Cybernetics*, 1998.
- [WRWH99] J.P. Watson, S. Rana, L.D. Whitley, and A.E. Howe. The impact of approximate evaluation on the performance of search algorithms for warehouse scheduling. In *Journal of Scheduling*, 1999.
- [WS00] W.J. Wolfe and S.E. Sorensen. Three scheduling algorithms applied to the Earth observing systems domain. In *Management Science*, volume 46(1), pages 148–168, 2000.
- [ZDDD94] M. Zweben, B. Daun, E. Davis, and M. Deale. Scheduling and rescheduling with iterative repair. In Michael B. Morgan, editor, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.