FINAL REPORT

ADVANCED TIMELINE SYSTEMS

Contract Number NAS8-39131

Mission Planning Division

Marshall Space Flight Center

Dr. R. L. Bulfin

Ms. C. A. Perdue

Department of Industrial Engineering

Auburn University

April 10, 1994

TIMELINE SCHEDULING

SPACE EXPERIMENT SCHEDULING

The Mission Planning Division of the Mission Operations Laboratory at NASA's Marshall Space Flight Center is responsible for scheduling experiment activities for space missions controlled at MSFC. In order to draw statistically relevant conclusions, all experiments must be scheduled at least once and may have repeated performances during the mission. An experiment consists of a series of steps which, when performed, provide results pertinent to the experiment's functional objective. Since these experiments require a set of resources such as crew and power, the task of creating a timeline of experiment activities for the mission is one of resource constrained scheduling.

For each experiment, a computer model with detailed information of the steps involved in running the experiment, including crew requirements, processing times, and resource requirements is created. These models are then loaded into the Experiment Scheduling Program (ESP) which attempts to create a schedule which satisfies all resource constraints. ESP uses a depth-first search technique to place each experiment into a time interval, and a scoring function to evaluate the schedule. The mission planners generate several schedules and choose one with a high value of the scoring function to send through the approval process.

The process of approving a mission timeline can take several months. Each timeline must meet the requirements of the scientists, the crew, and various

engineering departments as well as enforce all resource restrictions. No single objective is considered in creating a timeline.

The Experiment Scheduling Problem is:

Given a set of experiments, place each experiment along the mission timeline so that all resource requirements and temporal constraints are met and the timeline is acceptable to all who must approve it.

Specific characteristics of the problem are:

1. There is a limit on the available time for processing experiments, namely the mission duration.
2. Each model (experiment) may need to be run multiple times. Each execution of a model is called a performance.
3. Models require a set of resources of varying types. There are two sets of renewable resources, called nondepletables and equipment. Consumable resources and crew are also included in the set of resources.
4. The resource requirements of a model can vary over the processing time of the model. Each change in resource requirements constitutes a new step of the model.
5. The processing times of a model may vary. A minimum duration and a maximum duration are specified for each step of a model.
6. Some models allow (or require) delays between execution of the steps. Each model step has a minimum and maximum step delay associated with it. There is a requirement for some models with step delays that partial resource usage must be carried through the delay.

7. A model may require that its execution be performed during certain intervals in the mission. These are called performance windows and a model can have as many as ten windows specified.

8. A model may require that adjacent performances be separated by a certain length of time. Both a minimum and maximum performance separation can be specified.

9. The purpose or functional objective of some models require that the spacecraft be at certain positions in its orbit. This constitutes temporal constraints on the execution of the model. These available time windows are called orbit opportunities.

10. There can be three categories of orbit opportunities associated with a model. If a model requires a set of intersected orbit opportunities, all opportunities in the set must be available for the model to be scheduled, whereas selected orbit opportunities require that at least one be "open" or available for the performance. Avoided opportunities represent those which cannot be open during the execution of the performance.

11. A model may include the requirement that one or more of its steps must be run either simultaneously, before or after one or more steps of another model. These concurrency and sequencing relationships represent another type of temporal constraint on the execution of the model.

12. Concurrency can occur in one of three ways. Mandatory concurrency means that the model and its concurrent model cannot be scheduled unless the concurrency relationship is satisfied. Necessary concurrence is one-way. A

model requiring necessary concurrence cannot be scheduled without the concurrent model but this other model is not affected. The third type is desired concurrence; ESP attempts to satisfy this constraint but if that is not possible, the program schedules both models without concurrency.

13. Models can be designed with more than one way to execute the experiment. These alternatives, called scenarios, are defined by a sequence of steps. Each model scenario has a priority.

14. As an alternative to scenarios, a model may require a different set of steps for the first and last execution or performance of the model during the mission. These steps are called startup and shutdown steps.

15. Crew requirements for a model occur in several ways. Some models require specific crew members because, for example, they have special talents. These constraints are "rigid". Others specify a number of crew members from any of those available and are therefore "flexible" constraints. Any combination of rigid and flexible requirements is possible, such as a request for one crew member from a selection list of three and another crew member from a different selection list. In addition, some models have desired crew resources that are not mandatory for a performance to be scheduled.

Other constraints imposed on the mission timeline are flexible, or desired but not required. A feasible mission timeline is one that meets all rigid requirements or constraints but not necessarily these desired constraints. A good

schedule will meet many flexible constraints as well. Since approving a timeline involves different groups of people with different objectives, defining a good timeline is difficult. The following is a list of characteristics of a good mission timeline.

1. All requested performances of each model are scheduled.
2. The highest weight scenario is used for each performance of each model.
3. Soft constraints are met, such as desired concurrence, selected orbit opportunities, and crew monitoring.
4. One performance of each model is scheduled early in the mission.
5. There is a time separation between the first and second performances of each model to allow the scientist who created the experiment to perform validation of the results and make any necessary changes to the experiment's procedure.
6. The performances are ordered in such a way as to minimize the amount of crew information (eg performance procedure manuals) transferred between ground control and the spacecraft computer. The procedures are stored on the computer aboard the spacecraft, which must be purged periodically due to limited space. These transfers can be viewed as sequence dependent set-up times because no transfer is necessary if the procedure is already in the spacecraft's data base.
7. Resource usage is level.

The next section contains the results of an extensive literature search and review to investigate proposed formulations and solution methods that are

pertinent to the experiment scheduling problem. It begins with a general discussion of scheduling, followed by a discussion of relevant literature of a general nature. We conclude the section with a discussion of research on scheduling space related activities.

LITERATURE REVIEW

Background

Scheduling is allocating a set of limited resources to a set of tasks to be performed. The solution consists of a task sequence and a timetable of task start and completion times. Scheduling is a required task in a variety of environments including the construction, manufacturing and computer science industries. The primary resource in a scheduling problem can be time, as in project scheduling, or machines, as in processor scheduling.

Often the objective of processor scheduling is to determine the schedule or schedules that optimize the allocation of resources to tasks with regard to the completion times of the tasks. Examples of this type of objective include completing the processing of the set of jobs as quickly as possible or minimizing the time a processor is idle. Objectives which are dependent on the completion times of the scheduled tasks are called regular measures. Particularly characteristic of manufacturing applications is the assignment of release dates and

due dates to jobs, complicating the search for a feasible schedule and introducing other types of objectives such as minimizing job tardiness.

Although efficient algorithms exist for optimally solving several scheduling with regard to regular measures, in many cases small changes in the problem assumptions may make the problem very difficult. Resource-constrained scheduling has proven to be a difficult area of scheduling. In these problems the jobs to be processed have a set of additional resource requirements associated with them. In addition, there is a limit on the amount of each resource that is available. Resources can be divided into three categories. A renewable resource is one whose total usage is constrained at a given time. In this situation the resource is "returned" at the completion of the job requiring it. A resource is referred to as non-renewable if total consumption is constrained, i.e., if jobs consume amounts of the resource during processing and there exists a finite amount of the resource at the beginning of the schedule. Finally, a doubly constrained resource is one in which both total usage and consumption are limited.

A different type of scheduling problem is referred to as activity or project scheduling. These problems are encountered often in the construction industry where time is the primary resource and the tasks are the activities required to complete the project. The activities are related by a series of precedence constraints and this structure enables the project to be depicted as a network. Critical path techniques are used to define early and late start times for each activity to ensure completion of the project by a specified due date. Often these

problems are compounded by the addition of limited resources required by the activities. For this reason, these problems are also referred to as resource constrained scheduling problems.

As noted earlier, some scheduling problems include jobs with release date and due date parameters. For a particular job, these times describe an interval of in which processing for that job must take place. Some scheduling problems have a similar constraint for the machine or processor. In these problems there are "windows" of available processing time, ie., time intervals in which processing can take place. One example of this situation is a machine in a manufacturing facility that requires routine maintenance. Processing can take place when no maintenance tasks are scheduled. Another occurs in the air transportation industry where safety regulations prohibit the use of airplanes when a limit on the number of flight hours without maintenance is reached. Most procedures used to solve such problems treat these constraints implicitly. For example, in the machine maintenance problem, the task of maintenance can be treated as another job with a release date to insure that it is done after a certain number of processing time units.

An algorithm that can determine the optimal schedule in time bounded by a polynomial in the size of the problem, is called an efficient algorithm. There are many scheduling problems for which efficient algorithms are unknown; whether efficient algorithms exist for these problems is unknown. There is a class of problems, called *NP-hard*, that are provably the most difficult scheduling problems to solve. That is, an efficient algorithm for any *NP-hard* problem can be converted

into an efficient algorithm for any other scheduling problem. There are thousands of scheduling problems that are known to be *NP-hard*. For hundreds of years, many researchers have been trying to develop efficient algorithms for some NP-hard problems without success. This does not prove efficient algorithms do not exist for *NP-hard* problems, but the circumstantial evidence strongly suggests it is unlikely efficient algorithms will be found for *NP-hard* problems. If efficient algorithms cannot be found, non-polynomial (eg enumerative) algorithms must be used to find optimal schedules. The combinatorial nature of scheduling problems makes this approach too time consuming, even on the fastest computers, for all but very small problems. The other approach is to use heuristics to generate schedules that, while not optimal, perform well with respect to the measure of performance.

Resource Constrained and Related Scheduling Problems

Research conducted in the area of resource constrained scheduling can be divided project scheduling and processor scheduling. Project scheduling is characterized by a set of activities which are linked by precedence constraints. The project is often depicted as a network. Critical path methods are used to determine start times for the activities to insure project completion by the specified due date. This method is inadequate when the resources required to complete the project are constrained. For this reason, much research has been conducted in the area of resource constrained project scheduling.

Processor scheduling is characterized by a set of processors or machines which are the primary resources required to process a set of tasks. The machines may have identical or similar capabilities, as in the case of parallel processing, or they may vary in the type of processing of which they are capable, as in flow shop, job shop, or open shop environments. Processor scheduling with precedence constraints can be thought of as resource constrained project scheduling with the machines as resources. Some processor scheduling problems with additional resource constraints can be reformulated by viewing the resource as another machine.

Formulations

The task of processor or project scheduling under resource constraints can be accomplished by ignoring the resource constraints to obtain an initial schedule and then resolving resource conflicts where they occur by inserting idle time into the schedule. This method is adequate for obtaining a feasible schedule but essentially ignores the objective. Mathematical programming formulations of resource constrained scheduling problems explicitly consider an objective function. One of the most common methods of modeling scheduling problems mathematically is the use of zero-one decision variables to indicate placement of tasks in the schedule. These indicator variables can be of the form x_{ij} equals one if job i assigned to machine j, if job i is in process during time interval j, if job i starts at time j, or if job i completes in time interval j. Pritsker, Watters, and Wolfe (1969) introduce a zero-one programming formulation of the latter type for resource constrained project scheduling. Baker (1974) simplifies the notation in

his presentation of their model which allows constraints for resources, precedence relations, due dates, and concurrency. The resource constraints are of the form

$$\sum_{i=1}^{N} r_{ki} \sum_{u=t}^{t+p_j-1} x_{iu} \leq R_k$$

where r_{ki} is the amount of resource k required by job i, N is the number of jobs, R_k is the amount of resource k available, and p_i is the processing time for job i. This constraint uses the fact that

$$\sum_{u=t}^{t+p_j-1} x_{iu}$$

will equal 1 if job i is in process at time t. The formulation requires one constraint for every interval t from 1 to T_{max}, the last interval to be considered. The model of Christofides et al. (1987) uses a variable x_{ij} which equals one if job i starts at time j. This formulation is similar to that of Pritsker et al. except that the decision variable is summed over the interval $[(t-p_j+1),t]$ in the resource constraints.

Baker (1973) proposes another 0-1 formulation for processor scheduling where x_{ij} is equal to one if job i is assigned to processor j. Mazzola and Neebe (1986) formulate a resource constrained scheduling problem as an assignment problem with side constraints. Their formulation makes use of a 0-1 decision variable which can have a variety of interpretations, including assignment of job

to machine. The side constraints in this model represent a limit on the availabilities of depletable resources.

Other mathematical formulations of scheduling problems include the work of Manne (1960) (x_i is the start time of job i) and Baker (1974), Talbot and Patterson (1978), and Kasahara et al. (1988) (a decision variable is the completion time.) Baker's model is for job shop scheduling and includes an additional decision variable, y_{ipk}, which is equal to one if job i precedes job p on machine k. Blazewicz et al. (1993) uses the decision variable x_{ij} to represent the number of jobs of class i assigned to processor j in his model of a parallel processor environment with resource constraints. Here, classes represent groups of jobs with equivalent processing times and resource requirements. Patterson (1984) compares three different formulations including one developed by Davis and Heidorn (1971) in which jobs are divided into sub-jobs of unit duration "tied" together with precedence constraints. The other two are by Stinson (1976) and Talbot (1976). The focus of the article is on solution methods and therefore Patterson does not include these formulations.

As noted above, projects can be modeled using network diagrams. This method is used in the work of Davis and Patterson (1975) and Ohmae et al. (1992).

Exact Solution Methods

The mathematical models of Pritsker et.al (1969), Baker (1974) and Manne (1960) can be solved using any integer linear programming technique, but no specific algorithms were suggested in this literature. Mazzola and Neebe (1986)

use a Lagrangian Relaxation of the side constraints and solve the resulting assignment problem optimally to determine a lower bound on makespan for the original problem. This lower bound is used in a branch and bound enumeration scheme for determining the optimal solution to the original problem.

Two references use dynamic programming techniques to solve scheduling problems with resources. Blazewicz et al. (1993) presents an algorithm for solving the problem of parallel machine scheduling with resource constraints to minimize makespan. Jobs are divided into classes, each class having identical processing time and resource requirements. All feasible assignments of job class to machine are identified and backward pass dynamic programming is used to find the optimal assignments. The bounded enumeration procedure of Davis and Heidorn (1971) shown in Patterson (1984) uses dynamic programming to find the shortest route in a network of partial solutions, thus identifying the minimum length schedule.

Two of the algorithms discussed by Patterson (1984) are a branch and bound method (Stinson, 1976) and an implicit enumeration method (Talbot, 1976). They work by relaxing constraints to solve a related problem and then adding the constraints back to the problem step by step, forming a tree of partial solutions. Christofides et al. (1987) use a branch and bound method to build a feasible schedule. Nodes are fathomed by comparing schedule length to four lower bounds. One bound is created from the precedence constraints and two from relaxations of an integer programming formulation of the problem. The fourth bound is based on the idea of disjunctive arcs between jobs which cannot be processed

simultaneously due to resource conflicts. Each arc implies a different precedence constraint and therefore a different integer programming problem with relaxed resource constraints. Talbot and Patterson (1978) suggest an enumeration procedure using a project numbering scheme and compact arrays of resource requirements and availabilities. Their procedure uses less computer storage than other enumeration schemes because of the network cuts they use to fathom nodes representing partial schedules.

All of the example problems used to test the algorithms are limited to less than 100 jobs and three or fewer different resources. Computer time is generally less than one minute for problems of this size but grows exponentially as the number of jobs and/or the number of resources increase.

Heuristics

It is important to consider the complexity of problems and algorithms when searching for efficient solution methods for the resource constrained scheduling problem. Some specific resource constrained scheduling problems have been shown to be solvable in polynomial time whereas their generalizations are NP-Hard. For example, Blazewicz et al. (1993) present an algorithm by Garey and Johnson (1975) for optimally solving the two-parallel-machine problem with unit processing times and arbitrary resource constraints. This algorithm consists of creating a graph with nodes representing jobs and arcs representing resource feasibility between pairs of jobs. A maximum matching identifies the pairs of jobs which should be processed simultaneously in the optimal schedule. Efficient algorithms exist for solving maximal matching problems. However, if processing

times are not all the same, the problem is *NP-hard* (Jeffcoat and Bulfin, 1992). Because the resource constrained scheduling problem is *NP-hard*, much of the research in this area focuses on efficient heuristics. The enumeration schemes suggested above all employ methods of reducing the search space so that the optimal solution can be found in a reasonable amount of time. Still, as noted by Talbot and Patterson (1978), "these optimizing procedures can be terminated prior to optimality and still provide a feasible schedule". They state that this termination is imperative when the number of jobs is greater than fifty. Baker (1973) notes that enumeration schemes can determine optimal solutions for small problems quickly but these procedures do nothing to identify the characteristics of the jobs which account for the optimal behavior.

List scheduling is a general heuristic approach that involves prioritizing jobs by their characteristics to determine the sequence of the jobs in the schedule. These dispatching rules are very easy to implement, allowing practitioners to try several different priority rules to find the one that works best for the objective, and therefore identify general characteristics that are important for the objective.

In addition to their branch and bound algorithm, Mazzola and Neebe (1986) also suggest a construction and improvement heuristic procedure for the assignment problem with side constraints. Kasahara et al. (1988) suggest a depth-first implicit-heuristic search that combines branch and bound techniques with those of list scheduling. Typical depth-first search involves calculating a bound for each of the maximum depth nodes of the tree of partial solutions and choosing the node with the best value. Their method eliminates these calculations

by choosing the node according to a list schedule. Davis and Patterson (1975) compare the performance of several dispatching heuristics to the optimal solutions for 83 multiple resource project scheduling problems. Job parameters determined by critical path techniques such as late finish times and slack provided half of the priority rules and measures of resource utilization were used in the others. They determined that sorting the activities in non decreasing order of slack times performed well for a variety of the problems. Ohmae et al. (1992) investigate combining several job characteristics found from critical path analysis into one priority measurement to determine dispatching order.

Precedence Constraints

Project scheduling implies precedence constraints but processor scheduling problems exist in which jobs are independent. Additional constraints added to mathematical programming formulations of scheduling problems insure that precedence requirements are met. Such constraints are of the form $s_j + p_j \leq s_k$, where s_j and s_k represent start times for jobs j and k respectively and p_j represents the processing time of job j. This constraint ensures that job j is completed before job k starts. Note that these start times could represent decision variables or a summation of decision variables, as in the case of zero-one decision variables.

If the problem environment requires that one job be immediately preceded by another job, the inequality could be changed to an equality. This change can be used to model a job with variable resource requirements over the processing time interval. The job can be divided into a set of sub-jobs with constant resource

utilization that must be executed in order with no idle time between sub-jobs (Willis, 1985).

In general, the complexity of a scheduling problem is increased when job dependencies are added to it. These dependencies add more constraints to the mathematical formulation of the problem. However, Jeffcoat (1990) notes that when a model is time-based, as in the case of x_{it} equals one if job i assigned to time interval t, these precedence constraints eliminate many partial solutions from consideration, and may actually make the problem easier to solve.

Release Times and Deadlines

Another constraint on start times of jobs is imposed by release times and deadlines. As in precedence relationships, these constraints increase the complexity of scheduling problems. Even scheduling problems with two machines and very simple resource requirements become *NP-hard* when different release times are imposed (Blazewicz et al., 1993). Jeffcoat and Bulfin (1992) present a simulated annealing algorithm for resource constrained scheduling of parallel processors and jobs with release dates and due dates.

Related Scheduling Problems

Scheduling independent jobs with multiple resources can be formulated as a generalized bin packing problem. Csirik and Vliet (1993) present an efficient bin packing algorithm. Cutting stock and pallet loading problems are variations of the bin packing problem. Many routing problems, which arise in physical distribution, include time window constraints, resource restrictions and precedence constraints.

Space Experiment Scheduling References

Previous research on scheduling scientific experiments for execution in space includes the scheduling of observations in the unmanned satellite missions of Voyager and Ulysses and Spacelab telescope observations. NASA has also considered related scheduling problems such as prelaunch operations scheduling. Many solution methods have been investigated including mathematical programming techniques, dispatching rules and search methods such as simulated annealing, artificial intelligence and expert systems. We present this research in chronological order. Note that the work on ESP (e. g. Jaap and Davis, 1988, Stacy and Jaap, 1988) is not discussed in this section.

Mathis (1981) uses the model of Pritsker, Watters and Wolfe (1969) to formulate the Spacelab crew activity and experiment scheduling problem. His decision variable is x_{ijkt} which is equal to one if step k of performance j of model i starts in time period t. The algorithm works by creating an initial schedule which is feasible with respect to the timeline constraints and then checking resource feasibility. Performances are moved if the algorithm detects a resource conflict. The algorithm uses a dispatching rule to determine which of the offending performances to move. The creation of a feasible schedule for the problem requires multiple passes through the algorithm because moving performances may result in new resource violations.

Mathis's algorithm uses only a formulation rather than any optimization techniques implied by the formulation. The selection of the performance to move is the only part of the algorithm which employs an optimization technique. His

tests of the algorithm's performance results in the conclusion that the algorithm requires more time and produces comparable schedules to the program which existed at MSFC at that time. This program focused on creating a feasible schedule in the shortest amount of time by using dispatching procedures.

Grone (1982) proposes an algorithm to solve the problem of scheduling telescope observations for a spacelab mission. The objective of the problem is to maximize viewing time. Because there are three telescopes which must all move together on a single platform, the problem is formulated as a single machine problem. Each telescope views a particular type of target and each observance is considered the processing of a job. Slew time between targets is modeled as sequence-dependent set-up times and the entire problem is equivalent to a traveling salesman problem.

Temporal constraints for the problem include overall maximum time (the end of the mission) and unavailable time windows throughout. The algorithm uses job priorities, minimum duration times, and requested number of observations for each target, ie. requested number of performances of each job as input. The suggested algorithm evaluates a weight for each job based on priority, duration, and number of performances as well as a measure of ability to "fit" the job into the current time window and consecutive windows. These weights are used to determine the jobs to be scheduled in the applicable time window. Part of the selection process includes a reselection subroutine to check if inserting idle time and performing another job would yield a better score or weight than the job currently in the position. The algorithm creates several partial schedules for each

time window and chooses the best for each to combine into one schedule. Once a time has been found in which a particular job can be run, the algorithm schedules the job consecutively as many times as possible. Although the author recognizes that there should be a balance between the number of performances of different types of jobs, this goal is not included in the algorithm. Grone suggests manual editing.

Guffin, Roberts and Williamson (1985) present the ASTRON algorithm for scheduling crew and experiment activities for shuttle astronomy missions. ASTRON chooses the next target to schedule by means of a criteria function which is based on target window duration, observation time, slew time, target availability time and target priority. The objective of ASTRON is to maintain high utilization of resources.

Deuermeyer, Shannon and Underbrink (1986) devise a method of producing dispatch lists for use in the present experiment scheduler, ESP. They first divide all of the models into classes based on sequencing and precedence relationships. The first two classes contain models that must be performed before other models and these classes are ranked higher so that they will be scheduled first. They use resource similarities to cluster the remaining models and then use a schedule and repair algorithm to sequence the jobs in this class.

Pierce (1987) addresses the problem of scheduling horizontal payloads for space shuttle flights. The solution consists of utilizing an "expert system" called EMPRESS (Expert Mission Planning and Replanning Scheduling System). A "job" consists of assembling and installing the payload into the carrier structure,

performing trial runs of the payload/experiment, and installing the structure into the craft. The manual system used before EMPRESS was considered to be too slow and inflexible. Planners also wanted to perform "what-if" type scenarios.

EMPRESS is a modular program which works by building an initial schedule that consists of a list of all activities to be performed, then rescheduling jobs as required to meet resource constraints. EMPRESS solves to feasibility. No mention is made of an objective function. There are no time windows in the problem.

Three references by Kurtzman (1988, 1989, 1990) address the problem of scheduling crew activities on the space station with emphasis on producing real-time schedules by the crew members themselves. The system is called the MFIVE system. The objective is to create a schedule which will complete all activities in the shortest time possible.

Constraints such as early and late start times are written for each activity and combined in a method called "active constraint propagation" to shrink the time windows available for each job. This reduces the search space. Several heuristics are examined to determine a dispatching order of the jobs. Some of these heuristics utilize a "maximum compatibility matrix" to determine pairs of jobs to be scheduled simultaneously.

These heuristics produce initial schedules. The method of "intelligent perturbation" was used to find iteratively better schedules. This method involves increasing the priorities of activities that were not scheduled successfully so that the next schedule will be more likely to include them. Best results were obtained

when a heuristic using the maximum compatibility method was used in conjunction with the intelligent perturbation iteration technique.

The problem of scheduling experiments for unmanned spacecraft is addressed in Scherer et al. (1990). The scheduling process is constrained by resources, time windows and activity interdependencies. The objective function for the problem is to maximize the value of science while minimizing constraint violations. The value of science is dependent on the value of the included experiments, the resource utilization and the schedule feasibility. Optimal schedules are deemed impossible to obtain because of the subjectivity of this measure and the changing of objectives over time.

The research was conducted to suggest heuristics for finding near-optimal schedules which could be included in JPL's PLAN-IT II scheduling system. Random hill climb and simulated annealing are the two heuristics examined. In addition, two metaheuristics are evaluated with each heuristic. They are tabu search and strategic oscillation. The random hill climb heuristic with tabu search performed the best for this problem.

Scheduling unmanned activities is addressed in Thalman et al. (1991). The problem is to schedule the activities of the SOLar-STellar Irradiance Comparison Experiment instrument, referred to as SOLSTICE which flew on board the Upper Atmosphere Research Satellite (UARS) in 1991. The objective is to maximize observation time while producing minimum impact on other instruments. More generally, it is desired to maximize contribution to science within the resources constraints.

Space application scheduling is characterized by a larger number of activities, resources and interactions than manufacturing applications. UARS SOLSTICE scheduling involves using data for each possible target (star) to determine if it should be the next to be viewed. This decision is based on slew time to the star, accessibility, and previous success rate of viewing that star with the instrument.

Thalman et al. evaluate three common artificial intelligence techniques, but do not considered them further. They claim neural networks perform well in pattern recognition but not in optimization and simulated annealing and genetic algorithms require questionable assumptions and are computationally intensive. Also, the degree of randomness that they require deems them unsuitable for the space scheduling problem.

The AI method of tabu search was deemed to be the best because of its flexibility and ease of implementation. Thalman et al. do not consider tabu search to be a metaheuristic as do Scherer et al.. This version of tabu search is comparable to Scherer's random hill climb with tabu search.

Tabu search works by beginning with an initial schedule and generating a set of candidate "moves", each of which would create a new schedule. Each move is evaluated and the best one is chosen as the "current best". Moves previously visited and rejected are kept in a tabu list to avoid revisiting them. However, if the score for the tabu moves is high, the algorithm will override the tabu status in hopes of a much higher score at a later move.

A knowledge-based approach was also considered for solving the problem which consists of writing all of the constraints and objectives as rules that the program uses to build a schedule. The technique is not iterative, so "searching" is not possible. Although faster than Tabu Search, the results were not as good.

Zoch et al. (1991) address the activity scheduling component of mission planning and scheduling. This context is very broad; it includes planning and scheduling at the network level, the platform and payload levels and the customer level which includes instrument activities, spacecraft activities and ground activities.

The suggested solution procedure involves using a language called FERN and a system called ROSE. Both planners and scientists write requests for activities in the FERN language and transfer them electronically to the ROSE program which schedules the requests and confirms the times of execution to the requester. ROSE selects activities to schedule next based on priority, resource consumption, and a component of time restriction. Activities are then placed in the schedule according to preferences and where they will fit. The selection and placement heuristics can be specified by the user. ROSE contains a reselection module to change the initial schedule to a feasible schedule with respect to resource limitations. The end result from ROSE is usually a conflict-free schedule. Manual scheduling is possible to make improvements to the resulting schedule. No mention is made of an objective; the focus is on feasibility rather than optimality, although some measure or score is used to determine the next job to schedule.

Constraint-Based Scheduling is the focus of research conducted by Zweben (1991) for the problem of Space Shuttle Ground Processing. Zweben describes the GERRY system which contains an iterative algorithm for schedule and repair.

Summary and Conclusions

The following table is a taxonomy of the references pertaining to space experiment scheduling. The characteristics are divided into groups. There are characteristics for the jobs scheduled, the objective function used by the algorithm, the type of algorithm suggested, the capability of the algorithm, and the environment for which the algorithm was created. Columns labeled (1) through (11) represent previous research in chronological order, except the first column refers to ESP (Jaap and Davis, 1988) which is currently in use at MSFC. For brevity, we will not discuss ESP.

Only the algorithms of Mathis and Zoch et al. address multiple steps and step delays. The environment addressed by Zoch et al. is one of ground processing for Spacelab missions (denoted by a G in the Spacelab row in the Environment section of the characteristics). The ROSE system is an expert system whose objective is only to create a feasible schedule. For these reasons, we feel this system holds no advantages over ESP for the space experiment scheduling problem. Mathis admits that his proposed algorithm is slow and produced comparable schedules to a pure dispatching algorithm. Although computer advances would surely speed this process, the quality of the schedules created would still compare to those of ESP.

	Characteristic	1	2	3	4	5	6	7	8	9	10	11
JOB	Res. Const.	X	X			X	X	X	X	X	X	X
	Res. Types	X				X			X		X	
	Windows	X	X	X	X	X	X	X	X	X	X	X
	Steps	X	X								X	
	Step Delays	X	X								X	
	Precedence	X	X			X	X	X	X	X	X	X
	Performances	X	X	X	X	X		X	X		X	
	Variable Times	X							X		X	
OBJ	Score	X		X	X				X	X		
	Slew Time			X	X					X		
	Makespan		X			X		X				X
	Feasibility						X				X	
ALG	Expert System						X				X	
	Dispatch	X			X	X						
	Search		X	X				X	X	X		X
CAP	Real Time							X				
	Manual Capability	X					X	X			X	X
ENV	Satellite			X					X	X		
	Space Lab	X	X			X					G	G
	Space Station	X			X			X				

1. Jaap and Davis (1988)
2. Mathis (1981)
3. Grone (1982)
4. Guffin et al. (1985)
5. Deuermeyer et al. (1986)
6. Pierce (1987)
7. Kurtzman (1989)
8. Scherer and Rotman (1990)
9. Thalman et al. (1991)
10. Zoch et al. (1991)
11. Zweben (1991)

Taxonomy of Space Applications Scheduling Research

Only the models of Scherer et al. and Zoch et al. allow variable processing times. Scherer et al. treat the duration of each job as a decision variable. This has the disadvantage of increasing the number of decision variables and constraints by an amount equal to the number of models. The computations reported in his paper did not allow this situation. The ROSE system allows the user to specify a range of values for processing times.

Three of the more applicable references (Mathis, Deuermeyer et al., Kurtzman) tried to minimize makespan. Many of the characteristics of a good mission timeline are met if makespan is minimized. However, there are other objectives to consider as well. A model that weights the early time intervals more heavily than later ones would create better schedules than one which attempts only to minimize makespan.

Because of the complexity of the problem and the enormity of the problem size, we feel that measures should be made to eliminate the less important characteristics of the problem before scheduling occurs. Deuermeyer et al. and Kurtzman do this by creating temporal constraints to reduce the placement possibilities of each performance. Other clustering methods and constraint propagation techniques can be created to simplify the detailed scheduling task.

MULTICRITERIA SCHEDULING

Much work has been done on multicriteria decision making (MCDM). We will discuss basic concepts of MCDM as they relate to multicriteria scheduling. Complete details on MCDM can be found in Goicoechea et al. (1982) or Steuer (1986). For ease of discussion, we will restrict ourselves to two criteria, although the results can be easily extended to more than two criteria. As we discuss schedules, we will assume that the schedules are feasible, i. e. they satisfy crew, resource, equipment, orbit opportunity and performance window restrictions.

When there are two criteria, schedules which perform well with respect to one criterion will often perform poorly with respect to the other. One schedule dominates another if it performs strictly better on one criterion, and no worse on the other. A schedule not dominated by any other schedule is called nondominated, efficient or Pareto-optimal. Most "real" scheduling problems have many nondominated schedules. Clearly, dominated schedules are undesirable.

The three schedules in the following table illustrate. Let percent of performances scheduled and overtime costs be two criteria to evaluate a schedule. Schedule S2 is preferred to schedule S1 since it schedules a greater percentage of performances and requires less overtime. We say that S2 dominates S1. Alternatively, we may say S1 is dominated by or inferior to S2, or S1 is inefficient. Since S1 is dominated we can ignore it. Note that S3 does not dominate S1 and, if S2 is unknown, S1 can not be discarded.

Criterion	Schedule		
	S1	S2	S3
Percent performances scheduled	79	83	78
Overtime cost ($100,000)	10.75	10.10	8.15

Examining S2 and S3 shows that S2 schedules a greater percentage of performances, but S3 has lower overtime cost. S2 does not dominate S3, nor does S3 dominate S2. Based on the three schedules given in the table, S2 and S3 comprise the nondominated set.

When choosing a schedule, it is clear that we prefer a nondominated schedule. How do we generate a nondominated schedule? We must solve some sort of optimization problem; there are two general approaches we may take. The first is a hierarchical approach while the second requires optimizing a weighting or scoring function.

Hierarchical Approach

The hierarchical approach is to order the criteria by importance, and solve a single criterion problem with the more important criterion as the objective. Then a single criterion problem with the secondary criterion is solved, but with the constraint that the primary criterion does not get worse than the value obtained in solving the first problem. With two criteria, this is called a secondary criterion

problem, and is essentially finding the best possible solution for the secondary criterion among all optimal solutions for the primary criterion.

For our example, we might choose percentage of performances scheduled as the primary criterion and overtime cost as the secondary criterion. First we would find a schedule that maximizes the percentage of performances scheduled. Then we would find a schedule that minimizes overtime cost while keeping the percentage of performances scheduled at the maximal value. If there were more than two criteria, we would try to find the best schedule for the third criterion while maintaining the levels of the first and second criteria.

The hierarchical approach works well when there is a clear ordering of the criteria, there are many alternative optima for the primary criterion, and it is relatively easy to find the best schedule for a secondary criterion when holding the value of a primary criterion fixed. Discussion of previous work using this approach can be found in Smith (1956), Heck and Roberts (1972), Emmons (1975a, 1975b), Burns (1976), Bansal (1980), Shanthikumar and Buzzacott (1982), Shanthikumar (1983), Potts and Van Wassenhove (1983), Posner (1985), Bagchi and Ahmadi (1987), and Chen and Bulfin (1994). None of this work seems extendible to the NASA timelining problem, since it only considers two criteria, a single "machine" and no resources. As the number of criteria considered increase, the problem quickly becomes intractable. Due to the number of criteria and the fact that solving a timeline problem for one criterion is difficult, this approach will not be considered further.

Tradeoff Approach

Another approach is to have the decision maker (DM) expresses a tradeoff between criteria through a utility function, which, once specified, allows the problem to be treated as a single criterion problem. The DM must express weights for each criteria so they can be combined into a single value. If the criteria are truly conflicting, finding these "conversion" units is very hard. Typical utility functions are additive, multiplicative, or exponential; other forms are possible. Additive utility functions are the most prevalent, since, as Morris (1977) points out, the increased effort to develop a non-additive utility function is rarely worth the effort. A schedule which optimizes the value of the utility function, is nondominated.

We demonstrate an additive utility function using S1, S2 and S3 from the previous example. First we determine the relative importance of the percent of performances scheduled versus overtime. For exposition, suppose we feel that performances scheduled are four times more important than overtime cost. Since the two criteria are in different units, we must somehow normalize them. The easiest way is to somehow make overtime costs a percentage, but a percentage of what? Suppose we can estimate a maximum acceptable overtime cost, for the example, say 15. Then the ratio of actual overtime for a given timeline to the maximum desirable overtime is a percentage. For S2, it would be .67 and for S3, .54. This percentage is large for a "bad" schedule, while the percentage of performances scheduled is large for a "good" schedule. To make them compatible, we use the complement of the overtime ratio, i. e., one minus the percentage. This

gives a value of .33 for S2 and .46 for S3. Using weights of 4 and 1 for the two criteria, the utility of S2 is

$$4(.83) + 1(.33) = 3.65$$

while S3 has utility

$$4(.78) + 1(.46) = 3.58$$

and hence S2 is preferred over S3 given the stated utility function.

Note that if the actual overtime is larger than the maximum desirable, the resulting percentage is negative. Since we are assuming an additive function, this will decrease the utility value, making the schedule less desirable.

Little work has been done in scheduling with utility functions. Except for two or three special situations, these problems are all *NP-hard* (Chen & Bulfin, 1993a, 1993b). Huckert et al. (1980) and Kao (1980) have proposed general solution schemes using this approach. These papers make an important contribution to the philosophy and modeling of multicriteria scheduling problems. However, both authors point out that it appears unlikely that problems with more than about twenty jobs could be solved using their algorithms. For most scheduling problems this approach has not been attractive for two reasons; it is difficult for the DM to state an explicit utility function, and there are no efficient scheduling algorithms available for general objective functions.

A utility function must possess certain properties; see the classic work of Keeney and Raiffa (1976). It is usually difficult for a DM to develop a utility function that truly reflects the appropriate tradeoffs. Often, a scoring function is used as a proxy for a utility function. All utility functions are scoring functions,

but not all scoring functions are utility functions. Optimizing a utility function guarantees a nondominated solution under certain convexity assumptions, while even if convexity assumptions hold, a scoring function may not produce a nondominated solution. However, for discrete problems, such as scheduling, convexity does not hold, so even if we optimize a utility function it is possible the schedule generated is not nondominated. Therefore, ensuring a scoring function is a utility function is not as critical.

Using either of these two approaches generates a (hopefully) nondominated schedule. If the DM is happy with the hierarchical ordering of criteria or the utility/scoring function used, the schedule can be implemented. However, it is often better to let the DM explicitly tradeoff between several nondominated schedules.

We could generate **all** nondominated (efficient) schedules for the problem, and allow the DM to choose one of them. This approach has been used on scheduling problems by Van Wassenhove and Gelders (1978), John and Sadowski (1984), Sen and Gupta (1983), Nelson et al. (1986) and Van Wassenhove and Baker (1982). Chen and Bulfin (1993a, 1993b) have shown that except for maximum tardiness and flowtime on a single machine, generating the nondominated set of schedules is *NP-hard*. One drawback is the effort required to generate a single nondominated schedule may be immense. For the timelining problem, generating a single nondominated solution is *NP-hard*. Even if each nondominated schedule could be generated efficiently, there may be a very large number of nondominated schedules. As the number of criteria increase, the

number of nondominated schedules will also increase, likely nonpolynomially. For the NASA timelining problem, the number of nondominated schedules is vast. Also, generating a single guaranteed nondominated schedule is difficult of itself.

As one part of the multicriteria facet of timelining, we recommend the continued use of a weighted additive scoring function with a heuristic scheduler. If the weights are nonnegative, this scoring function is a utility function, and the probability of generating a nondominated schedule is higher. Also, it is much less difficult for the DM to develop an additive function than multiplicative or exponential functions. We also propose generating more than one solution. Rather than generate the entire nondominated set, we recommend generating a small subset of schedules which are, hopefully, nondominated. The problem is that unless all schedules are evaluated (either implicitly or explicitly) we do not know if a schedule is truly nondominated. Of course since we cannot generate one guaranteed nondominated schedule in polynomial time, this is not so critical. What we suggest is to generate a relatively small number of "good" schedules and use star plots to choose one to implement. Star plots will be discussed more fully in the evaluation section.

EVALUATION OF A TIMELINE

Performance Measures

The following is a list of possible evaluation measures for the space experiment scheduling problem. The measures are used in two evaluation techniques: scoring functions and star plots. These measures are divided into

four categories: scientist measures, crew measures, resource measures, and cost measures. For the purpose of consistency, we define each measurement so that its value is between zero and one and we desire to maximize it. Each evaluation measure is divided into two performance measures. The first is an extreme or worst case measurement, a minimum in our case, and the second is an average measurement.

The following notation is used to define the performance measures:

Let

n be the number of models

P_i be the number of performances requested for model i

Π_i be the number of performances scheduled for model i

D be the mission duration

$T1_i$ be the time of the first performance of model i

$T2_i$ be the time of the second performance of model i

V be the validation time required between the first two performances of a model

U_q be the desired percent utilization of time for crew member q

CT_q be the scheduled percent utilization of time for crew member q

CY_q be the average time between performances of the same model executed by crew member q

Y_q be the desired time between performances of the same model executed by crew member q

O_k be total orbit opportunity time available for orbit opportunity k

Θ_k be total orbit opportunity time scheduled for orbit opportunity k

g be the number of orbit opportunities available

R_ℓ be the amount of resource ℓ available (nondepletable and equipment)

Φ_ℓ be the amount of resource ℓ scheduled (nondepletable and equipment)

r be the number of resources, and

c be the number of crew members.

Scientist Measures

These measurements evaluate the schedule's performance to the jobs requested. The scientists responsible for designing the timeline experiments are the customers.

The most important measurement of a schedule's performance to a scientist's requirements is the number of times the scientist's experiment is executed. Each scientist requests a number of performances for his or her model so that he or she can obtain valuable results. The first scientist measure is therefore the percent of performances requested that were scheduled.

1. Performances requested that were scheduled.

$$S1_{\min} = \min_{i=1,n}\left\{\frac{\pi_i}{P_i}\right\}$$

$$S1_{avg} = \sum_{i=1}^{n} \frac{\pi_i}{P_i}.$$

Other important characteristics of a good schedule from a scientist's point of view are how early in the schedule an experiment is run and whether there is the correct amount of time between performance one and two of an experiment. The latter is important because of the validation that the scientist performs to the experiment's procedure after the first performance. We evaluate these two criteria separately and combine them to create a measurement of the placement of initial performances.

- Percentage of schedule duration remaining after performance 1 of job i begins.

$$x_1 = \min_{i=1,n}\left\{\frac{(D-T1_i)}{D}\right\}$$

$$y_1 = \frac{\sum_{i=1}^{n}\frac{(D-T1_i)}{D}}{n}$$

- Deviation of the time between the first and second performances of each model to the desired spacing. These measurements are normalized with a linear function over the interval [0,D].

$$x_2 = \min_{i=1,n}\left\{1-\frac{|T2_i-T1_i-V|}{D}\right\}$$

$$y_2 = \frac{\sum_{i=1}^{n}\left(1-\frac{|T2_i-T1_i-V|}{D}\right)}{n}$$

These four measurements are combined to form a minimum and an average measurement as follows.

2. Placement of initial performances of each model.

$$S2_{\min} = \min\{x_1, x_2\}$$

$$S2_{avg} = \frac{y_1+y_2}{2}$$

Crew Measures

These measures pertain to the time spent by spacecraft crew members. The crew members are the customers for these measures. The first measurement models the satisfaction of the crew members in regard to the amount of work required. It is based on the assumption that each crew member has an ideal percent utilization of his or her time in a schedule.

1. Deviation from ideal crew member utilization.

$$P1_{\min} = \min_{q=1,c}\left\{1-\left|U_q-\frac{CT_q}{D}\right|\right\}$$

$$P1_{avg} = \frac{\sum_{q=1}^{c}\left(1-\left|U_q-\frac{CT_q}{D}\right|\right)}{c}$$

Another measure of crew member satisfaction pertains to the variety of tasks that the person is scheduled to perform.

2. Deviation from ideal cycle time between repeated models for the crew members' schedules. (This is an attempt to determine the variety of the crew members' tasks.)

$$P2_{\min} = \min_{q=1,8} \left\{ \frac{|Y_q - CY_q|}{Y_q} \right\}$$

$$P2_{avg} = \frac{\sum_{q=1}^{8} \left\{ \frac{|Y_q - CY_q|}{Y_q} \right\}}{8}$$

Resource Measures

These measures are designed to show how effectively the resources are being utilized. These are system performance measures and have no obvious customer.

1. Available orbit opportunities that were scheduled.

$$R1_{\min} = \min_{k=1,g} \left\{ \frac{\Theta_k}{O_k} \right\}$$

$$R1_{avg} = \sum_{k=1}^{g} \frac{\Theta_k}{O_k}$$

2. Available resources that were used, including equipment resources.

$$R2_{\min} = \min_{k=1,r} \left\{ \frac{\Phi_k}{R_k} \right\}$$

$$R1_{avg} = \sum_{k=1}^{r} \frac{\Phi_k}{R_k}$$

Cost Measures

Another type of system performance measurements is cost measurements. We are able to assign a dollar value to each. To normalize each of the two measurements, we use the best in set of schedules that are generated (i.e., the one

with the lowest cost for the measurement) as the denominator in a ratio of best to actual. Note that this means that one schedule will have a perfect score of one and the other schedules generated will have a measurement greater than zero and less than or equal to one. Also note that because we have defined cost measures in this way, there is no minimum cost measurement for each schedule generated. Suppose z schedules are generated.

1. The cost of ground crew overtime.

$$C1_{\min} = \min_{s=1,z}\left\{\frac{LC1}{OT_s}\right\}$$

$$C1_{avg} = \frac{\sum_{s=1}^{z} \frac{LC1}{OT_s}}{z}$$

2. The cost of data dumps required.

$$C2_{\min} = \min_{s=1,z}\left\{\frac{LC2}{DD_s}\right\}$$

$$C2_{avg} = \frac{\sum_{s=1}^{z} \frac{LC2}{DD_s}}{z}$$

Scoring Functions

The decision makers must use all of the measurements selected for explicit consideration in choosing the best timeline. One way to view the measurements is to combine them into one number by means of a scoring function.

Suppose we choose a simple weighted additive scoring function to evaluate a timeline. First we place a weight on each of the four types of measurements; scientist, crew, resources and cost. Then we combine the minimums and the averages in each category into one measurement for the minimum and one for the average. This is done as follows.

$$S_{min} = \min\{S1_{min}, S2_{min}\} \qquad S_{avg} = \frac{(S1_{avg} + S2_{avg})}{2}$$

$$P_{min} = \min\{P1_{min}, P2_{min}\} \qquad P_{avg} = \frac{(P1_{avg} + P2_{avg})}{2}$$

$$R_{min} = \min\{R1_{min}, R2_{min}\} \qquad R_{avg} = \frac{(R1_{avg} + R2_{avg})}{2}$$

$$C_{min} = \min\{C1_{min}, C2_{min}\} \qquad C_{avg} = \frac{(C1_{avg} + C2_{avg})}{2}$$

Finally we combine each measurement in each category into two timeline measurements for the minimum and average value of a schedule.

$$MIN = \{ (w_s \times S_{min}), (w_p \times P_{min}), (w_r \times R_{min}), (w_c \times C_{min}) \}$$

$$Avg = \{ (w_s \times S_{avg}) + (w_p \times P_{avg}) + (w_r \times R_{avg}) + (w_c \times C_{avg}) \}$$

The weighted additive model is easy to compute and understand and therefore we consider only this type of scoring function in our evaluation techniques.

Star Plots

Scoring functions are appealing because they condense all of the performance measures into one number so that there is no ambiguity in the comparison of two or more timelines. However, we lose visibility when we evaluate a timeline with one number. For this reason, we propose another evaluation technique that maintains measurement information by category and displays the measures graphically. These graphs are called star plots.

A star plot consists of an axis for each measurement in each category. Our definition of timeline performance measures requires eight axes, with no need for negative numbers. Each axis starts at the center (0) and ends at one, the highest possible value. The minimum and the average values of each measurement are plotted on the axis of that measurement. All average points and all minimum points are connected to form two concentric polygons. Since the cost measurements do not have minimums, the same point is used twice in the star plot. We illustrate both scoring functions and star plots with an example.

Example

The table below illustrates results from three hypothetical timelines. We generate scores and star plots for these timelines.

Table 1. Timeline Measurements

		Timeline		
		I	II	III
Scientist	$S1_{min}$	.63	.45	.41
	$S1_{avg}$	.75	.51	.70
	$S2_{min}$	.32	.78	.28
	$S2_{avg}$	.55	.83	.48
Crew	$P1_{min}$	.17	.10	.12
	$P1_{avg}$	.23	.11	.19
	$P2_{min}$	.04	.28	.01
	$P2_{avg}$	.11	.34	.08
Resource	$R1_{min}$	.21	.45	.21
	$R1_{avg}$	.30	.65	.25
	$R2_{min}$	.27	.43	.21
	$R2_{avg}$	.67	.47	.45
Cost	$C1_{avg}$	1	.78	.62
	$C2_{avg}$	.54	1	.53

This table contains the minimums and averages of the four categories of measurements for each timeline.

	Timeline		
Measure	I	II	III
S_{min}	.32	.45	.28
S_{avg}	.65	.67	.59
P_{min}	.04	.10	.01
P_{avg}	.17	.23	.14
R_{min}	.21	.43	.21
R_{avg}	.49	.56	.35
C_{avg}	.77	.89	.58

These aggregate measurements can be combined with weights to give a single value for the timeline. To illustrate, we will use three different sets of weights on each of our three example timelines. The weights are

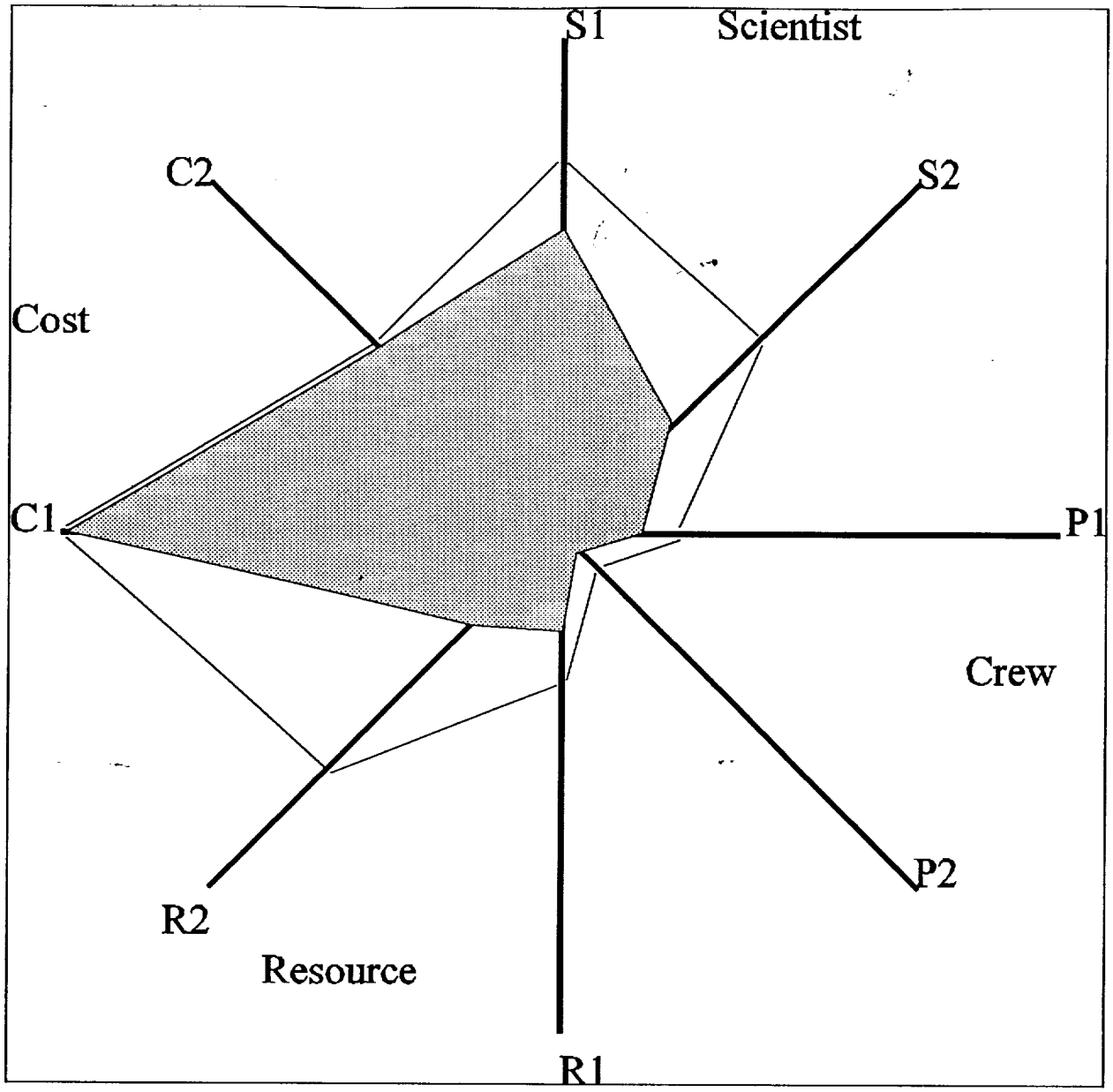
Category	Weight Set 1	Weight Set 2	Weight Set 3
Scientist	.25	.45	.35
Crew	.25	.30	.15
Resource	.25	.15	.35
Cost	.25	.10	.15

Using these weights and the scoring function based on average previously discussed, we can calculate scores for each timeline. They are

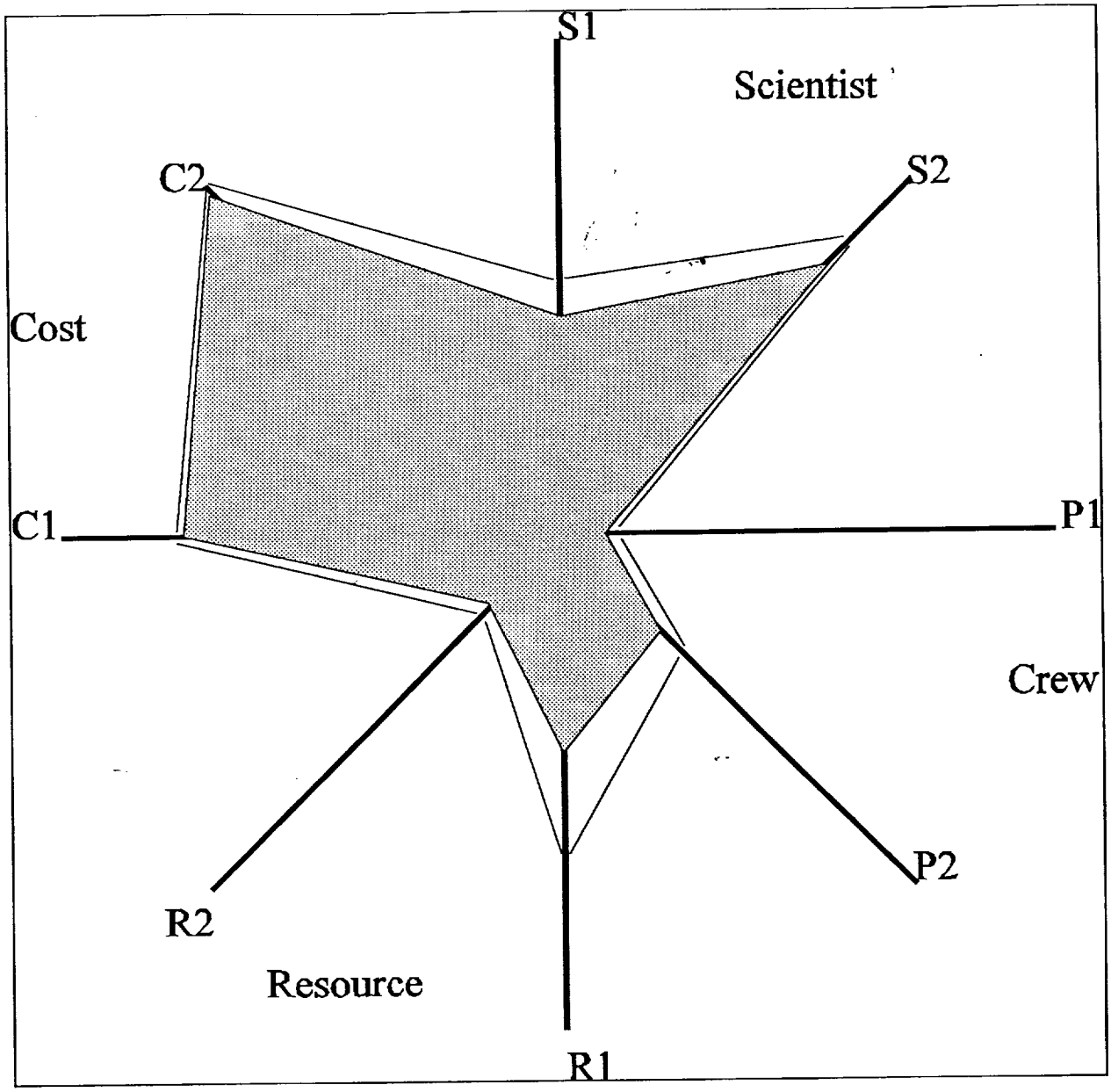
		Timeline	
Weight Set	I	II	III
1	.52	.59	.42
2	.49	.54	.42
3	.54	.60	.44

For these weights, Timeline II is always preferred ever the other two. Different weights might have resulted in timeline I being preferred, but timeline III is dominated and would never be preferred. Different scoring functions, such as the minimum rather than average could have been used, or a weighted sum of both averages and minimums would also be possible.

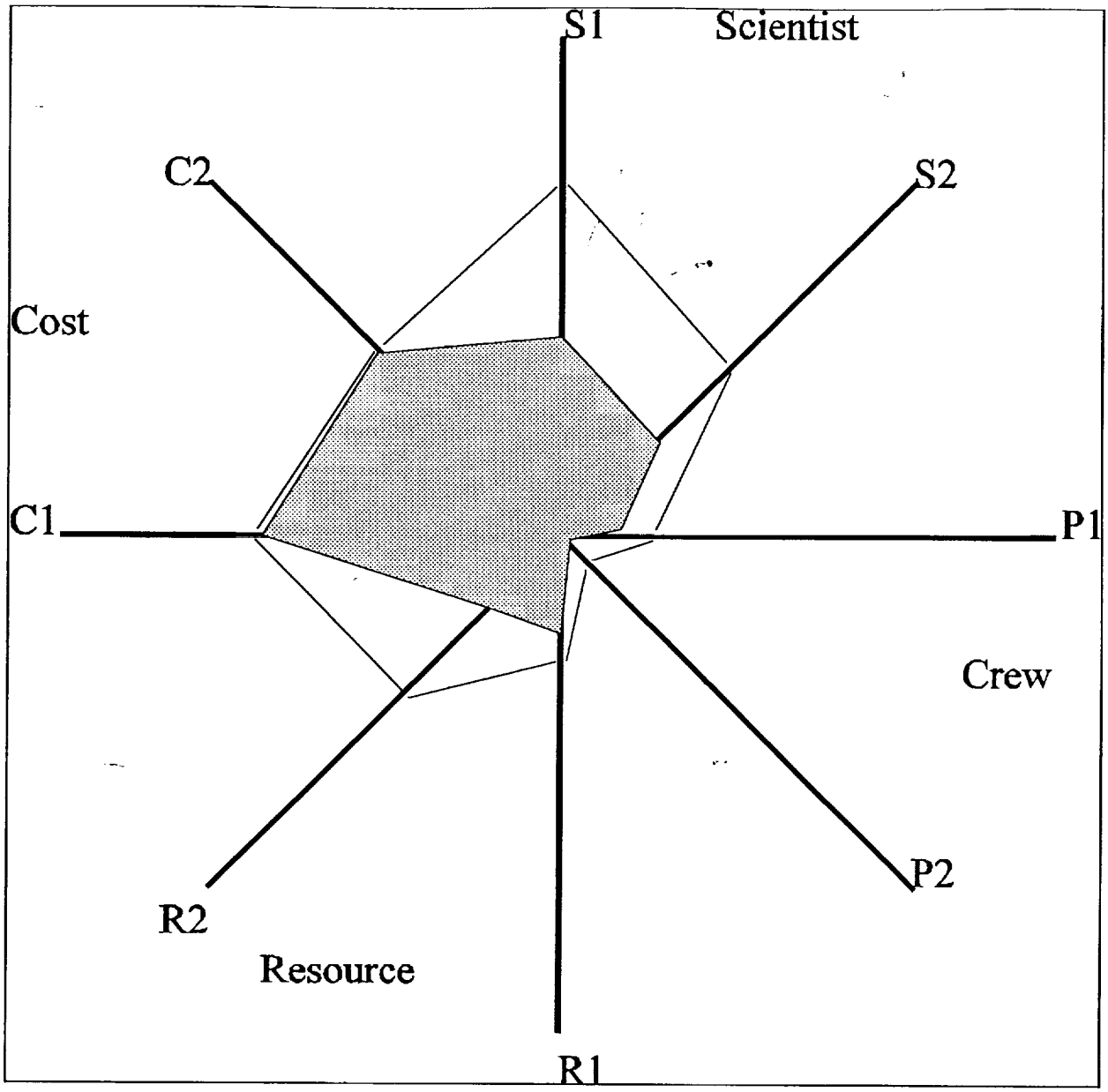
Star plots can be used to show the relative differences in these three example timelines. We give the plots below.



Timeline I



Timeline II



Timeline III

From these plots we can see where the timelines differ. For example, Timeline II is better for placement of the early performances and crew variability, but Timeline I requires less ground crew overtime and has a higher percent performances scheduled.

HEURISTICS

LIST ALGORITHMS

We now discuss possible methods for creating spacecraft timelines. Perhaps the simplest type of scheduling heuristic to create and execute is a list algorithm. A list algorithm sorts the tasks to be scheduled in some order, thus creating a list of tasks in the suggested order of execution. A list algorithm does not consider start and stop times of the tasks. We examine five list algorithms.

A common list algorithm creates a list schedule by sorting the tasks in order of increasing processing time. This is the Shortest Processing Time (SPT) algorithm. SPT guarantees optimal solutions for the single machine problem to minimize flow time. We use it as a heuristic to solve our space experiment scheduling problem. Reversing the order of SPT gives us a Longest Processing Time (LPT) list which we try as well.

Our third list algorithm sorts the models by Earliest Due Date (EDD). The due date of performance i of a model is defined as follows:

$$DD_i = a_i + i\left(\frac{b_i - a_i}{p_i}\right)$$

where the interval $[a_i,b_i]$ is the performance time window of performance i of the model and p_i is the number of performances requested in the performance window of performance i.

The fourth algorithm (SLK) sorts the models by slack time which is defined as the difference between late start (LS) and early start (ES) of a model. The late start of a model performance is equal to the due date of the performance minus the minimum processing time (tmin) required to execute the performance. We define the early start of a model performance to be equal to the interval start time of the performance's time window if the performance is the first in the interval and the due date of the previous performance if the performance is the second or a consecutive performance. Mathematically we have

$$
\begin{aligned}
ES_i &= a_i+(i-1)\left(\frac{b_i-a_i}{p_i}\right) \\
LS_i &= DD_i-tmin_i \\
SLK_i &= LS_i-ES_i \\
&= DD_i-tmin_i-ES_i \\
&= a_i+i\left(\frac{b_i-a_i}{p_i}\right)-\left[a_i+(i-1)\left(\frac{b_i-a_i}{p_i}\right)\right]-tmin_i \\
&= \left(\frac{b_i-a_i}{p_i}\right)-tmin_i
\end{aligned}
$$

Notice that the slack of a performance is independent of the performance if the model has only one performance time window. In this case, all of the model's requested performances should be executed in the same interval so that $[a_i,b_i]$ is independent of i.

Our fifth list algorithm is the only one that does not consider the processing times of the models. It is called the Resource Price list algorithm (RPR). We define the price of a resource to be the Lagrangian multiplier of the following knapsack constraint: $rx \leq R$ where r is a job-dimensional vector which contains the jobs' requirements for the resource, R is the amount of the resource available, and x is a job-dimensional vector of decision variables. This is a 0-1 vector where a 1 means that the corresponding job is in process.

If a resource is highly constrained, its x vector will be sparse and the Lagrangian multiplier used for the price of the resource will be high. The opposite is true for resources that are not tightly constrained.

We use these resource prices to obtain a cost of one performance of each model. Our resource prices are in units of cost per unit resource-time. We multiply this cost by the number of units of the resource required by a model times the processing time of the model (we use maximum processing times). We do this for each resource required for the model and then sum the costs. This gives us a cost of executing one performance of the model. We sort in order of increasing costs.

We perform all but one of these list algorithms in two ways. First, we consider each performance of each model to be a job and we sort all of the performances to create a list. Because each performance has identical resource requirements and processing times, the sort key value will be identical for all performances of a model except for the EDD and SLK algorithms. In addition,

since the majority of the models in our data cases have only one performance time window, the slack times will usually be identical as well. This creates a list in which all performances of a model occur consecutively. We would expect the schedule created from such a list to exclude some models altogether because they do not occur sooner in the list. For this reason we developed another category of list algorithms.

The other method of performing the algorithms views each model as a job and creates a job-dimension vector containing the number of performances requested of each model. We sort the set of jobs iteratively and decrement this vector until all performances have been scheduled. We call these "Round-Robin" list schedules. A round robin schedule contains one performance of each model at the beginning of the list. This should create schedules that exclude fewer models than the performance list counterpart.

ROC List Algorithm

Another simple type of algorithm for solving scheduling problems is a greedy algorithm which schedules tasks based on current information only and never changes the task's position. One disadvantage of greedy algorithms is their inability to view the entire scheduling horizon at one time and choose the best position for each task. Another is that there is no iterative search and improvement method.

We now introduce a greedy algorithm that generates a list of model performances. We call it the Resource and Opportunity Compatible List

Algorithm, or ROC. ROC decomposes the mission timeline into disjoint intervals and creates a list of model performances for each interval. It attempts to place compatible models consecutively so that they will be scheduled simultaneously when we create a detailed schedule from the list.

Pre-Processing

The first step in our procedure is crude pre-processing. The following methods are used:

1. Scenario Aggregation. We eliminate the lower weighted scenarios of each model and consider only one scenario per model.
2. Time Window Transformation. This creates a list of jobs each with one performance window.
3. Performance Elimination. We eliminate the performances of each model that we know will not schedule due to time constraints.
4. Model Sequencing Transformation. We change the time windows of two models so that they are mutually exclusive.
5. Step Delay Aggregation. We add the minimum delay between two consecutive steps to the step duration of the first step.
6. Step Duration Aggregation. We consider processing times to be fixed at the maximum step durations.
7. Step Detail Aggregation. We view the requirement of a resource by a model to be the maximum requirement for that resource over all steps of the model.

We choose to aggregate step durations to maximum values. If all performances do not fit in the mission timeline, we can aggregate using minimum

or average processing times. In addition to these formal pre-processing procedures, we ignore the characteristic of model concurrency.

In the next step we divide the mission timeline into a set of mutually exclusive intervals created by the start and stop times of the performance windows of each model. Figure 4 is a pictorial representation of this procedure. The four models have performance window start and stop times that create eight endpoints on the mission timeline. These endpoints create seven intervals. We then view the performance window of each model as the combination of one or more sequential mutually exclusive intervals. Each interval has a set of models which can be performed in that interval. In the example of Figure 4, these sets are shown in Table VI.

Table VI: Assignment of Models to Intervals

Interval	1	2	3	4	5	6	7
Models	3	2,3	1,2,3	1,2,3,4	1,3,4	1,4	1

The next step of the procedure is to divide the number of performances of the model in proportion to the interval length. Note that a model with a large performance window can be divided into a group of jobs with smaller windows and fewer performances. If the processing time of job k, call it $tmax_k$, is greater than the interval length, the number of performances assigned to the interval for job k is less than one.

As shown in Table VI, these intervals partition the jobs into sets. ROC is performed iteratively on each set of jobs, i.e., each timeline interval. The algorithm chooses the next job to schedule based on resource and opportunity compatibility, and a job priority.

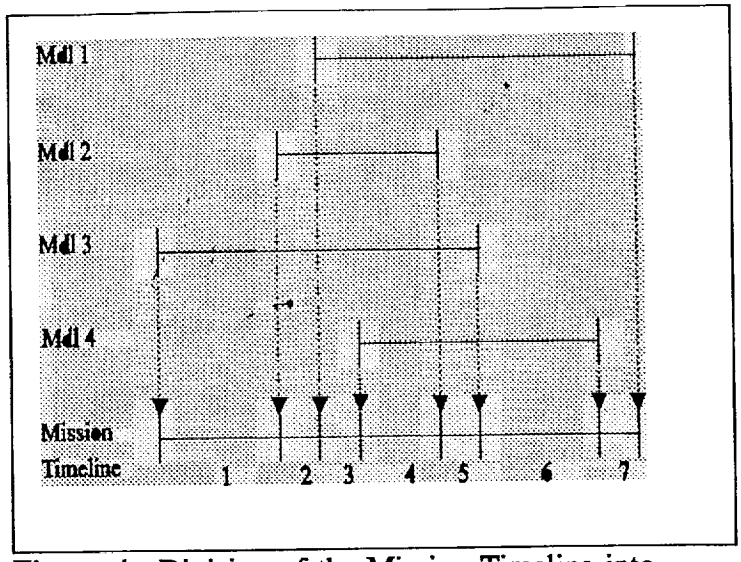


Figure 4. Division of the Mission Timeline into Mutually Exclusive Intervals

Opportunity compatibility means that the jobs involved do not require conflicting orbit opportunities and resource compatibility means that the resources available are not over-extended when the jobs are run simultaneously. ROC places these compatible jobs consecutively in the list. When we create a detailed schedule from the list, however, we desire that these compatible jobs be scheduled simultaneously rather than consecutively. The algorithm begins a new interval when no models are left to schedule in the present interval or when none of the models can be scheduled due to lack of resources.

Algorithm Statement

We now describe the list routine. Let m be the number of mutually exclusive intervals. Let S_{ij}, 0<i,...,m be the sets of jobs to be scheduled simultaneously in interval i. The subscript j represents the set number and is greater than or equal to one. The total number of sets that can be scheduled in an interval depends on the number of models that can be scheduled in the interval as

well as the resource requirements of the models. One obvious upper bound on j is the total number of models to schedule. The sequence $(S_{i1},S_{i2}, \ldots)$, forms the dispatch list for interval i. The set of sequences $\{(S_{i1},S_{i2},\ldots) \mid 0<i,\ldots,m\}$ forms the dispatch list for the entire mission timeline. Let C be the current set. Let P be the vector containing the total number of unscheduled performances of each model. The algorithm is:

1. For each pair of jobs, determine the ability to schedule the jobs together based on orbit opportunities. Two jobs that do not have conflicting opportunities are called "opportunity compatible". The measure of this compatibility is inversely proportional to the Hamming distance between the pair of opportunity vectors, as described below.
2. Begin a new interval.
3. Begin a new current set, C.
4. Score all remaining jobs to be scheduled in the current interval.
5. If C is empty, pick the job p with the highest score to schedule next. Go to 9.
6. If C is not empty let G be the set of jobs that are orbit opportunity compatible and resource compatible with all jobs in C.
7. If G is empty, go to 10.
8. Choose the job from G with the highest score. Call this job p.
9. Add job p to C. Reduce P(p). If any element of P is greater than zero, go to 4.
10. Let j=j+1 and set S_{ij}=C. If more performances will fit in the interval, go to 3.
11. If there are more intervals in the timeline, go to 2.
12. Stop.

Orbit Opportunity Compatibility

The list algorithm does not generate starting times for the jobs. The experiment scheduling problem includes many complicated time constraints and interdependencies and an algorithm that ignores these will surely fail to produce good results. For this reason, we divide the mission timeline into intervals and perform the algorithm on each interval separately. Orbit opportunities are another set of time constraints that we cannot ignore. Orbit opportunities are aggregated to model detail by combining all requirements of the steps, as in step detail aggregation. The result is a vector of 0's, 1's and -1's where each element corresponds to a unique orbit opportunity. Call this vector OP. A one in a position in OP means that the model requires the associated opportunity and a negative one means that the model must avoid the associated opportunity. Some example vectors are shown in Table VII.

Table VII: Orbit Opportunity Vector for Four Models.

	DATA	SOLAR1	TDRS	ATM2
OP(1)=	(0,	1,	0,	0)
OP(2)=	(1,	0,	-1,	1)
OP(3)=	(1,	0,	1,	1)
OP(4)=	(1,	1,	0,	1)

Models 2 and 3 cannot be scheduled together because of the conflicting orbit opportunity requirement for TDRS. Their opportunity compatibility score is zero. Both Model 1 and Model 4 can be scheduled with 3, as well as with each other but 3 and 4 are more compatible than 3 and 1 because they have more common elements. Simultaneous scheduling of 3 and 4 results in a better utilization of orbit opportunity resources. The Hamming distance is a comparison between two binary vectors. The Hamming distance

between vectors x and y is denoted by H(x,y) and is equal to the number of coordinates, i, for which x_i and y_i are different (Tremblay, 1975). Our definition for Hamming distance does not require that the vectors be binary, since our orbit opportunity vectors are not binary. Note that H(OP(1),OP(3))=4 and H(OP(3),OP(4))=2.

The Hamming distance measures the difference between two vectors. We desire a measure of the similarity of two vectors. The obvious choice is (n-H(x,y)) where n is the dimension of the vectors x and y. This is simply the number of coordinates, i, for which x_i and y_i are alike. We call this the Hamming score, or HS(x,y).

The Hamming score is helpful but does not provide desired results in all cases. For example, recall that OP(1)=(1,0,1,1) and OP(3)=(0,1,0,0). These two models are opportunity compatible but the Hamming score of their opportunity vectors is zero. Therefore, our opportunity compatibility score is defined as follows:

$$OC(x,y) = \begin{cases} H(x,y)+1, & \text{if } x \text{ and } y \text{ do not contain conflicting opportunities} \\ 0, & \text{otherwise} \end{cases}$$

Table VIII contains a summary of the three orbit vector comparisons.

The OC values are stored in a two dimensional, symmetric matrix. The algorithm uses the matrix to determine the set of jobs that are opportunity compatible with the jobs previously scheduled in the current set. The selection process favors the jobs that have higher compatibility values.

Resource Feasibility

Table VIII: H(x,y), HS(x,y) and OC(x,y) for Four Opportunity Vectors

	OP(1)			OP(2)			OP(3)			OP(4)		
	H	HS	OC	H	HS	OC	H	HS	OC	H	HS	OC
OP(1)	-	-	-	4	0	1	4	0	1	2	2	3
OP(2)	4	0	1	-	-	-	1	3	0	2	2	3
OP(3)	4	0	1	1	3	0	-	-	-	2	2	3
OP(4)	2	2	3	2	2	3	2	2	3	-	-	-

The algorithm attempts to maintain resource feasibility by examining a resource usage array called REQ. This is an (n×r)-dimensional array where n is the number of models and r is the number of resources. REQ(i,j) is equal to the amount of resource j required by model i. Step 4 of the algorithm eliminates a job from the selection set if adding it to the current set would over-utilize a resource. It does so by maintaining an r-dimensional vector R where R(j) contains the amount of resource j used by the jobs in the current set. One method to determine resource compatibility is to reduce R by the row of REQ associated with the candidate model. In this case, when a new job k is added to the current set, the algorithm makes the following assignment:

$$R(j) = R(j) - REQ(k,j) \quad \forall\, j\ ,\ 1 \le j \le r$$

Note that the accuracy of the resource feasibility measure depends on the aggregation scheme used to represent the models. We aggregate step detail to model detail by using the maximum resource requirements over the steps of the model. This overstates the resource requirements which means that the ROC algorithm may fail to place a model in the current set when it would actually fit. After ROC fills a current set and saves it, it re-initializes the available resources vector, R and empties the current set. ROC does not consider the

processing times of the jobs when it makes this assignment. It does, however, consider the processing times of the jobs to determine when to begin a new interval.

Starting a New Interval

The total amount of a resource available in an interval is equal to the product of the initial availability value for that resource and the length of the interval. Similarly, the total amount of a resource required by a model is equal to the product of the requirement value and the processing time of the job. We refer to these measures as resource "areas" because of the way in which each are depicted on a Gantt chart (see Figure 5). ROC accumulates the resource requirement areas as it schedules jobs. It begins a new interval when a resource area is filled. If the processing time of a scheduled job is greater than the interval length, ROC uses the interval length to compute the required resource areas for all of the applicable resources.

The conditions that ROC uses to determine when to begin a new current set and when to begin a new interval do not ensure that a feasible schedule can be created from the results. ROC re-initializes R when it begins a new current set. This translates to moving to time x in the Gantt chart of Figure 5. ROC creates new current sets repeatedly until there is no more resource area, at which time it begins a new interval. This implies that the jobs in consecutive current sets may overlap, as shown in Figure 6. The resource compatibility measures in the algorithm do not consider these overlaps and therefore it can place too many jobs in an interval. Since the aggregation scheme can create the opposite problem, and the true processing times considered by the detailed scheduler are not fixed, we would hope that these problems will cancel each other.

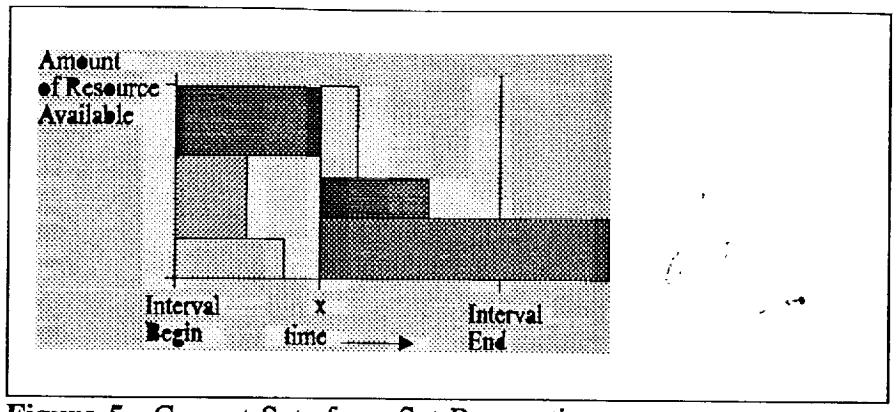


Figure 5. Current Sets from Set Perspective

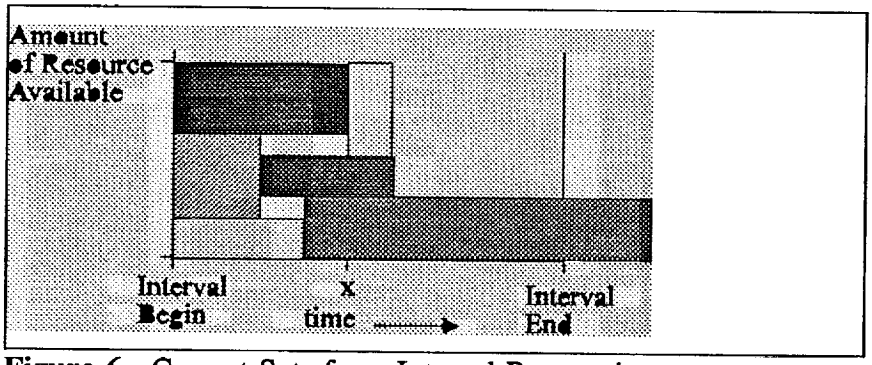


Figure 6. Current Sets from Interval Perspective

Job Priorities

The essence of a list algorithm to solve general scheduling problems is the intelligent choice of which job to schedule next. This decision is based on a weight or priority calculated for each job. The job with the highest priority is the next job scheduled. The list algorithm for the experiment scheduling problem uses such a measure. However, its effect is somewhat diminished because of the opportunity compatibility measures and the resource feasibility which the algorithm considers in the selection process as well. We include these latter two measures to maintain feasibility. The job priority is the only measure we consider with the purpose of finding a good schedule. We consider two priorities. They are (1) the number of performances remaining to schedule of a model and (2) the due dates of the model performances remaining to schedule.

Performances. One characteristic of a good schedule is equitable amount of mission time to each of the experiments on the job list. A very simple job priority is the number of performances that are left to schedule. This can be the number of performances remaining to schedule in the current interval or for the remainder of the mission timeline. When the algorithm chooses a job performance and adds it to the current set, it reevaluates the priority for that job before it chooses the next job to add to the current set.

Due Dates. Scheduling practitioners in manufacturing often sequence jobs in order of earliest due date. The EDD method minimizes the maximum tardiness of the jobs. We define a due date for space experiments using the performance window of the experiment and the number of performances required. If [a,b] is the performance window of a model and p is the number of performances required, we create a due date for the i^{th} performance as follows:

$$DD_i = a + i\left(\frac{b-a}{p}\right)$$

This definition of due date is identical to that used in our EDD list algorithm. We incorporate the due date measure into the existing ROC algorithm so that compatible models are scheduled together. The algorithm always chooses the model from the set, G, of candidate models whose next unscheduled performance has the earliest due date.

Because of the intervals created by ROC, some models that can be scheduled in an interval have no due dates that occur within that interval. We still view the due date of the next model performance as the criteria for scheduling even if it occurs outside of the interval. We do not, however, eliminate that due date and calculate the date of the next performance unless it does occur in the interval.

When we use either of these scores to decide on the best job to schedule next, we only consider those jobs that can feasibly be run simultaneously with the jobs in the current set. In the case of orbit opportunities, if a candidate job has a compatibility score of zero with any of the other jobs in the current set, it is removed from further consideration. By failing to reconsider the compatibility score when choosing the best job, we are ignoring the fact that some jobs are more opportunity compatible than others. We resolve this by choosing the job with the highest product of score and compatibility score (OC) with the last job scheduled.

Clustering Algorithm

A good schedule of space experiments contains all requested performances of all models. Such a schedule maintains a high level of resource utilization for the constrained

resources. This means that model performances will overlap, i.e., will run simultaneously. Model clustering is determining which models can be performed at the same time and combining these to form a new task. We significantly reduce the number of tasks to schedule if we predetermine model clusters. These clusters are resource compatible and if they are good clusters then they use close to the maximum amount available of the more constrained resources. A cluster with this property is said to be "full". No two full clusters are performed simultaneously. By clustering, we transform the space experiment scheduling problem into a scheduling problem with no resources, a problem which is much easier to solve.

Our clustering procedure is a suboptimal algorithm which solves a linear program to create the clusters and uses the clusters to create a list of model performances. As in the ROC algorithm, the clustering algorithm places clustered model performances on the list consecutively. This is done to increase the likelihood that they will be scheduled simultaneously in the detailed schedule.

Before we create clusters, we perform pre-processing on the models similar to that done before ROC. We use the scenario, step delay, step duration, and step detail aggregation methods as in ROC. We ignore performance time windows, sequencing, and concurrency.

Generalized Knapsack Problem with Multiple Choice Constraints

ROC creates model clusters when it schedules compatible jobs consecutively but it does little to create good clusters. The problem of assigning models to clusters can be formulated as an integer programming problem and solved with a heuristic to find good clusters. Our objective in developing clusters is to place all models in the fewest number of clusters possible. Schedules created with full clusters have high resource utilization and

complete many model performances. The model-to-cluster assignment is done so as to maintain resource feasibility. We state the problem formally as:

$$Max \sum_{i=1}^{I} \sum_{j=1}^{J} P_j x_{ij}$$

$$s.t. \sum_{i=1}^{I} r_{ik} x_{ij} \leq R_k \ \forall j=1,\ldots J \ \forall k=1,\ldots,K$$

$$\sum_{j=1}^{J} x_{ij} \leq 1 \ \forall i=1,\ldots,I$$

$$x \in \{0,1\}$$

where

P_j = the value of placing a model in cluster j

$x_{ij} = \begin{cases} 1 & , \text{if model } i \text{ is assigned to cluster } j \\ 0 & , \text{otherwise} \end{cases}$

r_{ik} = the requirement of model i for resource k

I = the number of models

J = the number of clusters

K = the number of resources

The objective function coefficients are defined to ensure that as many models as possible are placed in the fewest clusters. We guarantee a feasible solution that includes all models by letting I=J.

This problem is a Generalized Knapsack Problem with Multiple Choice Constraints (GMCKP). If we obtain a solution to the problem, we have clusters of models that can be run simultaneously without overextending resources. From here we must examine the number of performances requested for each of the models in a cluster to determine how many times the cluster should be run in a mission schedule. But there is little chance that all models in a cluster have the same number of performances. Another problem with this formulation is the size. Our largest data case contains 174 models. The formulation of this problem requires 174^2 variables, or 30,276. This problem also contains 74 resources which equates to $174 \times 74 = 12,876$ resource constraints.

Generalized Knapsack Problem

Because of the formidable size of the clustering formulation as a GKPMC, we reformulate the problem to assign models to one cluster. We then remove one performance of the assigned models from the problem and solve it again to assign models to the second cluster. We solve the problem iteratively until it assigns all model performances to a cluster. Because we no longer consider multiple clusters at one time, we change our objective function. We still desire full clusters. Our choice for an objective function coefficient for model i is the processing time of i. The interpretation is to complete as much processing in the current cluster as the resource constraints allow. We no longer have a set of mutually exclusive constraints and therefore our problem is a Generalized Knapsack Problem (GKP). The new formulation is

$$Max \sum_{i=1}^{I} (tmax)_i x_i$$

$$s.t. \sum_{i=1}^{I} r_{ik} x_i \leq R_k \quad \forall k=1,\ldots,K$$

$$x \in \{0,1\}$$

where

$tmax_i$ = the maximum processing time of model i

$x_i = \begin{cases} 1 & \text{, if model } i \text{ is assigned to the cluster} \\ 0 & \text{, otherwise} \end{cases}$

r_{ik} = the requirement of model i for resource k

I = the number of models

K = the number of resources

Cluster Lengths. A full cluster is one that uses close to the maximum available amount of the constrained resource. Another definition of a full cluster is one that does not contain much wasted time. Large differences in the processing times of clustered models causes wasted time. This point is best made with the illustration in Figure 7.

Cluster 1 and 2 use the same units of the resource and yet cluster 2 is clearly a fuller cluster. Our formulation does not consider this two-dimensional nature of the cluster that it creates. Suppose we add one more constraint to the problem of the form

$\sum_{i=1} tmax_i x_i \leq T$, where T changes with each execution of the problem. It starts as a

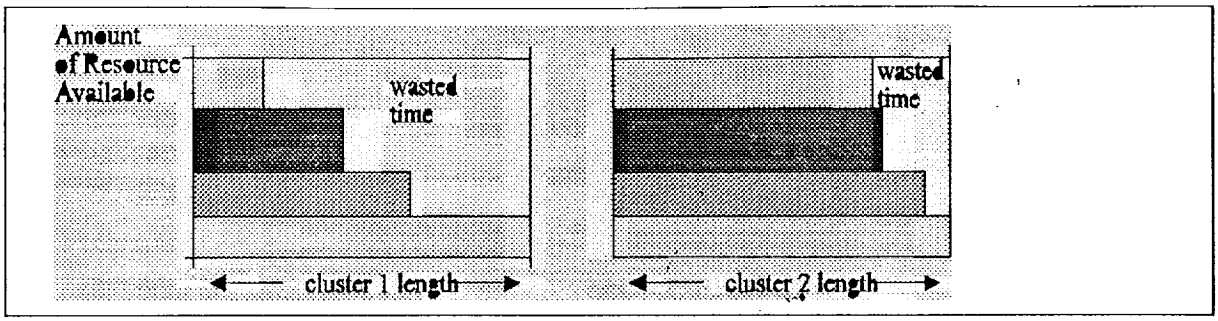


Figure 7: The Effect of Cluster Length on Cluster Fullness

small number and gradually increases until it is equal to the mission duration. Including this constraint forces the algorithm to search for jobs close to the same length.

Cluster List. Once we create the clusters we view each as a job to be scheduled during the mission. The processing time of each "job" is the processing time of the longest model performance in the cluster. We create a dispatch list for the mission in two ways. We use the Shortest Processing Time (SPT) and the Longest Processing Time (LPT) methods to sort the clusters. We do not concern ourselves with the order of the models within each cluster when we write the sorted clusters to the list.

COMPUTATIONAL RESULTS AND CONCLUSIONS

Results

The list, ROC and clustering algorithms were programmed using FORTRAN 77 on a VAX platform. We used a program by Bulfin and Liu (1985) to solve the Generalized Knapsack Problem with Mutually Exclusive Constraints in the clustering algorithm. We created ASCII output files for both programs containing a list of model performances in the suggested order of execution. These files were subsequently used by ESP to create a detailed timeline for the mission.

We use data from two previously flown Spacelab missions; ATLAS2 and SLJ. ATLAS2 was an astronomy mission and SLJ's experiments focused on life sciences and microgravity. Table IX shows pertinent characteristics of these two missions.

Table IX: Mission Parameters

	ATLAS2	SLJ
# of Models	107	174
Total # of Performances	1844	992
# of Nondepletables	7	15
# of Equipment	39	51
# of Crew	4	8
# of Orbit Opportunities	36	11
Max Experiment Time (hrs)	2235	3694
Mission Duration (hrs)	199	192

Maximum experiment time is defined as the sum of the maximum processing time of all model performances. We use four ratios to evaluate the algorithms. The first two are the ratio of the number of performances scheduled and the number of models scheduled to that requested for these parameters. We call these ratios PR and MR respectively. Their values will always be less than or equal to 100 and we desire the highest possible ratio. The last two are the amount of crew time (CR) and the amount of experiment time (ER) scheduled relative to the amount of each requested. ESP calculates the requested times based on the minimum processing times of the models. The values for these ratios are greater than 100 if many model performances are scheduled at their maximum processing times. We desire high values for these measurements but not at the expense of much smaller values for PR and MR,

since they are more important than the time ratios. We compare our results for each mission to those obtained from a randomized list of the model performances in that mission.

All runs were made using a version of ESP which reads a list and schedules it automatically. Hand scheduling with ESP using the same list gave consistently better results. However, for ease in generating schedules the list version was used in all tests. The relative differences between algorithms should be the same either way.

We first view the results from the list algorithms. We distinguish the round-robin version of each algorithm by prefixing the algorithm name with the letter "R". The random list is identified by "RAN". Table X contains results of all the list algorithms.

As we would expect, the round robin schedules have higher MR values that the performance-list schedules. The differences do not appear significant in the SLJ mission. The RSPT list creates a better schedule than random but the RLPT algorithm does not consistently perform worse as we would expect since it is the reverse of the RSPT list. We notice similar results with SPT and LPT.

The list algorithms do not appear to be robust to changes in mission characteristics as can be seen by the lack of consistent results over the two missions. One exception is the resource price algorithms. In all but one case, the ATLAS2 and SLJ measurements are consistently higher or lower than the results from the random lists. Note also that the resource price algorithms outperform the random lists in all but two cases but these differences seem significant only for the ATLAS2 mission.

We look next at the ROC algorithm. Recall that we use two priorities to create two different lists for each mission. They are the number of performances left (P) and the earliest due date (DD). The results follow in Table XI.

Table X: List Algorithms vs. Random List

	RAN	SPT	RSPT	LPT	RLPT
	ATLAS2				
PR	36.6	41.2	40.9	37.2	32.8
MR	72.5	84.9	91.9	65.7	75.8
CR	98.2	78.2	123.3	75.7	129.2
ER	212.4	212.1	222.0	199.7	225.2
	SLJ				
PR	56.1	56.7	56.7	58.0	55.2
MR	73.2	77.5	79.8	75.7	72.3
CR	32.6	33.5	32.5	32.9	32.5
ER	93.8	107.1	114.4	101.9	95.9

	RAN	EDD	SLK	RSLK	RPR	RRPR
	ATLAS2					
PR	36.6	37.2	36.2	36.1	41.5	41.2
MR	72.5	71.7	70.7	85.9	74.8	90.9
CR	98.2	96.2	98.2	124.7	91.6	122.4
ER	212.4	212.7	184.9	223.7	209.3	221.6
	SLJ					
PR	56.1	56.6	58.3	55.0	56.2	56.7
MR	73.2	75.7	70.5	71.7	76.3	79.2
CR	32.6	31.2	32.5	32.1	32.9	33.9
ER	93.8	94.4	94.5	95.4	90.5	108.3

The ROC(P) algorithm consistently outperforms the ROC(DD) algorithm although the differences are only significant in the SLJ mission. A randomly generated list outperforms the ROC(DD) algorithm for the SLJ mission but does not for the ATLAS2 mission. ROC(P) consistently outperforms the randomly generated list in all but the PR ratio but none of the differences are significant.

Now consider the results from the clustering algorithm. Recall that our formulation was extended to include a constraint on the length of the cluster to force the algorithm to find models of the same length to cluster together. Our initial results with this formulation

Table XI: ROC Algorithm vs. Random List

	RAND	ROC(P)	ROC(DD)
	ATL		
PR	36.6	35.5	35.3
MR	72.5	76.7	75.8
CR	98.2	122.2	121.5
ER	212.4	222.6	192.9
	SLJ		
PR	56.1	55.5	45.6
MR	73.2	76.3	20.2
CR	32.6	32.1	19.0
ER	93.8	97.7	8.1

showed that the variability in the processing times of the models was too great to achieve the desired results. The effect of this extra constraint was to create more clusters containing fewer models. We eliminated the constraint and used the original formulation. We tried the Shortest Processing Time rule (SPT) and the Longest Processing Time rule (LPT). The results appear in Table XII.

The results for the clustering algorithm differ between mission more than those for the ROC algorithm. For ATLAS2, CL(SPT) performs better than CL(LPT) and the random list for the PR and MR measures and CL(LPT) outperforms CL(SPT) and the random list for the CR and ER measures. For SLJ, CL(SPT) and the random list are consistently better than CL(LPT) and CL(SPT) performs better than the random list in all but the PR measure. The numbers for SLJ exhibit less variability than those for ATLAS2. The differences between CL(LPT) and CL(SPT) in ATLAS2 seem significant for all but the ER measure.

The ROC(P) algorithm consistently provides better results than the CL(LPT) algorithm but outperforms the CL(SPT) algorithm in only some cases for some measures. The

Table XII: Clustering Algorithm vs. Random List

	RAND	CL(SPT)	CL(LPT)
	ATL		
PR	36.6	40.2	29.1
MR	72.5	82.8	62.6
CR	98.2	95.6	122.0
ER	212.4	210.4	213.7
	SLJ		
PR	56.1	55.8	54.2
MR	73.2	75.7	70.5
CR	32.6	33.0	29.8
ER	93.8	101.2	88.9

differences only seem significant for the ATLAS2 mission. Both clustering algorithm lists outperform the ROC(DD) algorithm for the SLJ mission but exhibit mixed results for the ATLAS2 mission. Figures 8, 9, 10 and 11 show the results of each mission separately in two bar graphs.

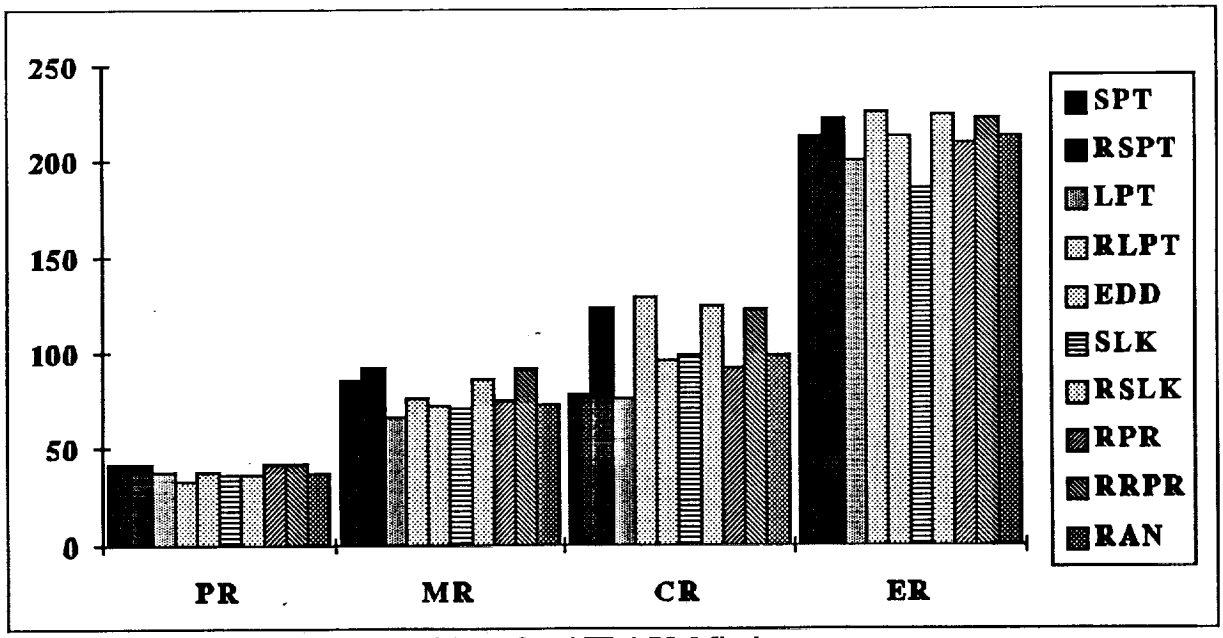


Figure 8. Results of List Algorithms for ATLAS2 Mission.

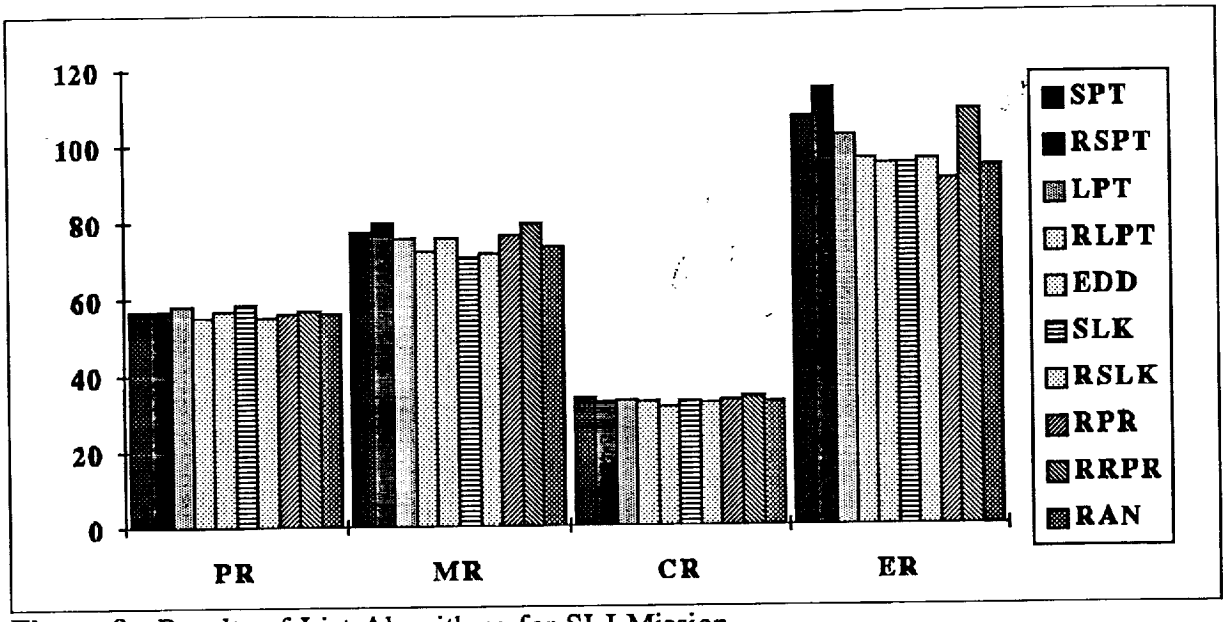


Figure 9. Results of List Algorithms for SLJ Mission.

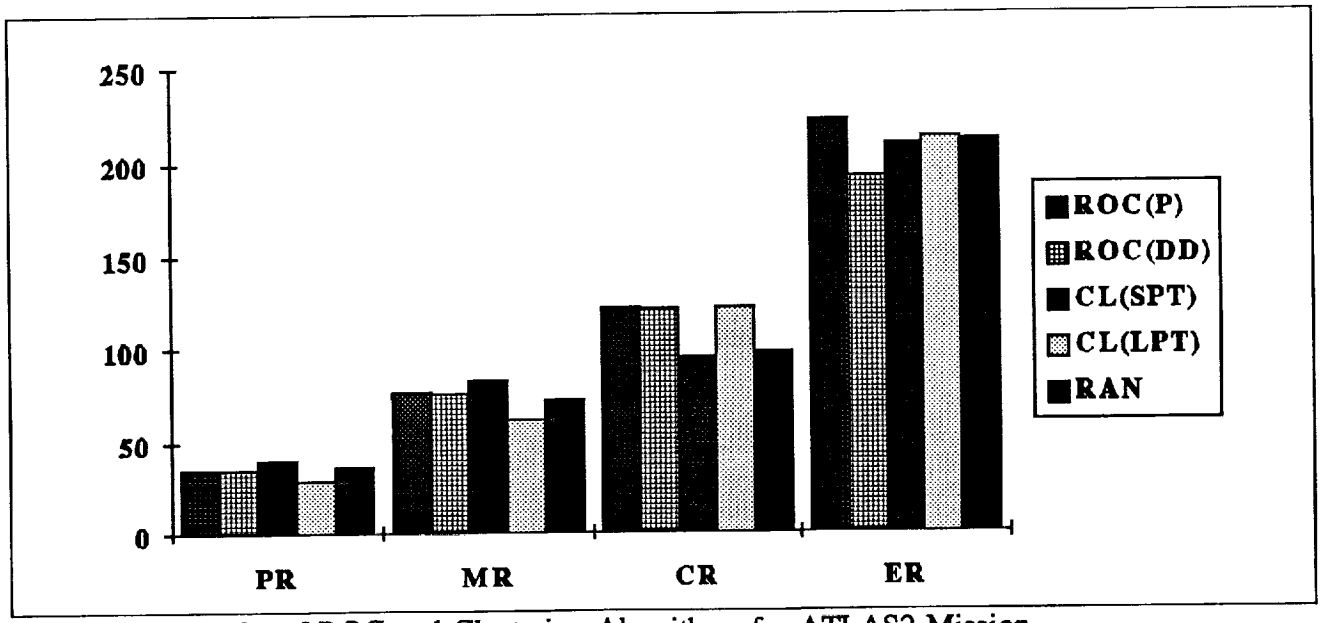


Figure 10: Results of ROC and Clustering Algorithms for ATLAS2 Mission.

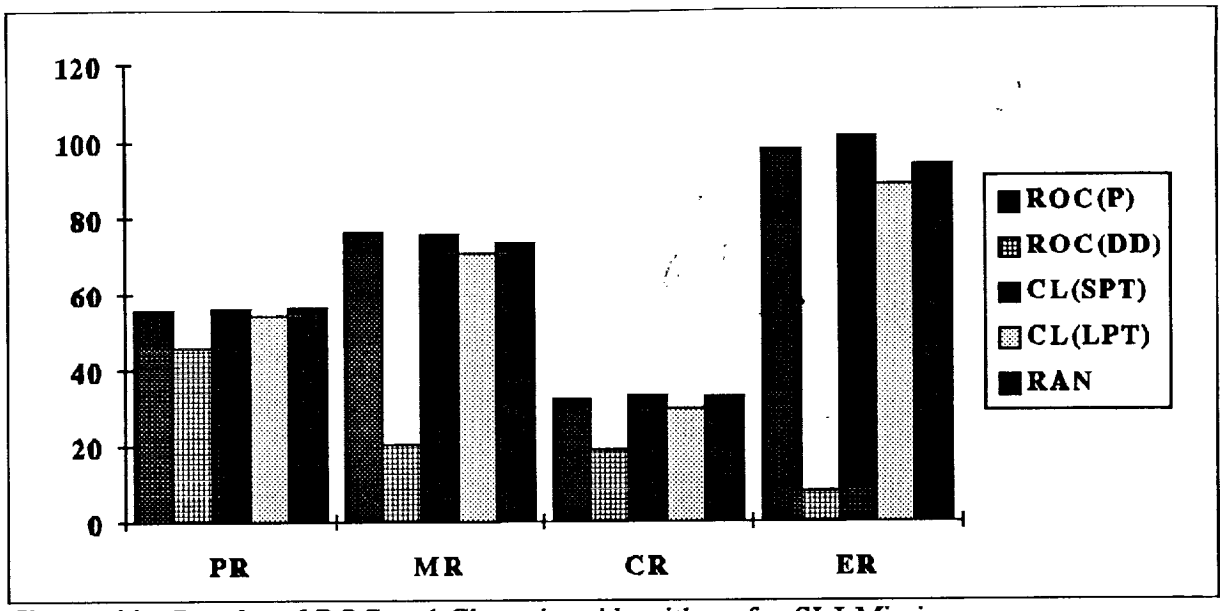


Figure 11. Results of ROC and Clustering Algorithms for SLJ Mission

Conclusions

The majority of the list algorithms perform better than the random list for at least half of the performance measures. Although many of these improvements are small, because the list algorithms are easy to execute, we suggest that the scheduler use some or all of these algorithms and choose the one which creates the best schedule. The round robin version of the Resource Price Algorithm is the only one that creates a schedule that is consistently better than the randomly generated schedule for both missions. We therefore suggest that it be used.

Although none of the greedy algorithms dominate the others in both missions for all performance measures, there are some differences which we feel are significant. In the ATLAS2 mission results, note the inverse relationship between the PR and MR and the CR and ER performance measures when the clustering algorithms are used. The performance and model ratios are consistently better than the crew and experiment time ratios for the CL(SPT)

vs. random list and the CL(SPT) vs. CL(LPT) comparisons. CL(SPT) places short clusters at the beginning of the list. ESP schedules many of these short clusters before it reaches those with the longer processing times. These short clusters contribute much to the number of performances scheduled and to the number of models scheduled. The SLJ mission results are not consistent with this observation. Perhaps this is due to the fact that SLJ has fewer performances of models with longer processing times. There is less variability in the processing times of the clusters in this mission.

Similarly, the CL(LPT) algorithm results in higher ratios for the CR and ER measures in the ATLAS2 mission. In the SLJ mission, CL(SPT) is higher for all ratios. Since the performance and model ratios are more important than the time ratios, the clustering algorithm with SPT performs better than LPT.

Using the ROC algorithm with the priority of the number of performances remaining to schedule provides better results than using ROC with due dates, especially for the SLJ mission. The ROC(DD) algorithm with SLJ performed very poorly with all measures. Perhaps this is due to the longer models in SLJ and the method of calculating and decrementing due dates in ROC(DD). ROC(P) seems more robust to differences between missions and we therefore suggest it over ROC(DD).

Although the RRPR, ROC(P) and the CL(SPT) algorithms provide better results than the random list in some cases, these differences are small. From this we conclude that an algorithm which creates a list for ESP does not provide enough detail to ESP. We must give a start time and a stop time for each model performance to ensure that ESP follows the order correctly. This is due to the fact that ESP places a performance in the earliest possible position. In addition, we cannot rely on the aggregation of the models' characteristics to

determine these start and stop times. We feel that we must look at specific intervals in the mission that are critical and models that require these critical intervals in order to create a mission schedule that contains all models and as many of their performances as possible. We feel that a better problem formulation would include more detail of orbit opportunities and performance time windows.

Future Research

Future research should focus on aggregate timelining algorithms which generate clusters of models to be scheduled together. These clusters satisfy, in aggregate, the resource restrictions imposed on experiment scheduling. Given a cluster, a detailed timeline could be generated in a variety of ways. The simplest way would be to use ESP. It might also be possible to allow a human to schedule jobs within the cluster. Discussion of these steps follows.

Preprocessing

Preprocessing is examining the mission data to aggregate information (or, in some cases, possibly eliminate it) in order to make the timeline decision more tractable. The key is to keep critical information so that the resulting timeline is easily transformed into a feasible detailed timeline. This requires being able to identify critical resources for a particular mission. Since missions are often quite different, this is a dynamic task. We will develop procedures to analyze mission data and determine appropriate aggregations. For different models different

aggregations result. For example, some models may aggregate steps, while for others step integrity must be maintained. The procedure must be capable of determining good aggregations for vastly different missions.

Assignment/Clustering

Given an aggregation scheme, models must be grouped so that the group satisfies resource restrictions. Two approaches will be investigated, assignment and clustering. In assignment, time is divided into a number of "buckets" and models are assigned to a bucket. There is a "cost" to assign a particular model (or more precisely, performance of a model) to a particular bucket. This cost would depend on the resource usage of the model, resource availability of the bucket, the relative time in the mission of the bucket, the number of performances of the model etc. Clustering is similar to assignment, except the models are grouped together without specifying a particular bucket.

Sequencing

To transform the assignment/cluster to something more like a timeline, we must consider sequencing. For assignment, sequencing within the bucket is important, while for clustering, the sequence of the clusters themselves are important. Depending on the length of the bucket, the assignment algorithm could be used recursively to break buckets down into smaller time slices. For sequencing clusters, standard single machine scheduling results may prove helpful.

Testing

Once a heuristic for timelining is developed it must be thoroughly tested. This will ensure it is correct and effective. There are four subtasks.

Validation.

Validation ensures the heuristics are operating as designed. Internal validation is a logical test. External validation will consist of solving small test problems and checking the results.

Retrospective Testing.

The heuristic will be tested on several missions that have already flown. The aggregate schedule will be turned into a detailed schedule using ESP and results will be compared to the timelines proposed for those missions. If possible, comparisons to actual timelines will also be made.

Random Problem Testing.

Problems will be randomly generated so that they are similar to actual MSFC missions. This will allow many, say several hundred different problems to be generated. Each problem will be solved by the heuristic and the solution evaluated with respect to several measures of performance. Also, bounds on the best possible solution will be calculated, and compared to the heuristic solutions. This will verify the quality and efficacy of the heuristic.

Predictive Testing.

The heuristic, in conjunction with ESP, will be used to generate a timeline for a mission that has yet to be flown. This schedule will be compared to the timeline proposed for the mission.

BIBLIOGRAPHY

Bagchi, U., and Ahmadi, R. H., "An Improved Lower Bound for Minimizing Weighted Completion Times with Deadlines," *Operations Research*, 35, 311-313 (1987).

Baker, K. R., *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York (1974).

Baker, K. R., "Procedures for Sequencing Tasks with One Resource Type," *International Journal of Production Research*, 11, 125-138 (1973).

Bansal, S. P., "Single Machine Scheduling to Minimize the Weighted Sum of Completion Times with Secondary Criterion, A Branch and Bound Approach," *European Journal of Operations Research*, 5, 177-181 (1980).

Blazewicz, J., Ecker, K., Schmidt, G. and Weglarz, J., *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, Berlin (1993).

Blazewicz, J., "Complexity of Computer Scheduling Algorithms Under Resource Constraints," in: Proceedings, First Meeting AFCET-SMF on Applied Mathematics, Palaiseau, Poland, 169-178 (1978).

Burns, R. N., "Scheduling to Minimize the Weighted Completion Times with Secondary Criteria," *Naval Research Logistics Quarterly*, 23, 125-129 (1976).

Chen, C. L. and R. L. Bulfin, "Complexity of Single Machine Multi-Criteria Scheduling Problems," *European Journal of Operational Research*, 70, 115-1 25, 1993.

Chen, C. L. and R. L. Bulfin, "Scheduling a Single Machine To Minimize Two Criteria: Maximum Tardiness and Number of Tardy Jobs," to appear, *IIE Transactions*, 1994.

Chen, C. L. and R. L. Bulfin, "Complexity of Multiple Machine Multi-Criteria Scheduling Problems," *Working paper*, 1993.

Chen, C. L. and R. L. Bulfin, "Scheduling Unit Processing Time Jobs on a Single Machine with Multiple Criteria," *Computers and Operations Research*, 17, 1-7, 1990.

Christofides, N., Alvarez-Valdes, R. and Tamarit, J.M., "Project Scheduling with Resource Constraints: A Branch and Bound Approach," *European Journal of Operational Research*, 29, 262-273 (1987).

Csirik, J. and Van Vliet, A., "An On-Line Algorithm for Multidimensional Bin Packing," *Operations Research Letters*, 13, 149-158 (1993).

Davis, E. W. and Patterson, J. H., "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling," *Management Science*, 21, 944-955 (1975).

Davis, E. W. and Heidorn, G. E., "Optimal Project Scheduling Under Multiple Resource Constraints," *Management Science*, 17, B803-B816 (1971).

Deuermeyer, B. L., Shannon, R. E. and Underbrink, A. J., "Creation of the Selection List for the Experiment Scheduling Program (ESP)," Technical Report, MSFC Contract No. NAS8-35972 (1986).

Emmons, H., "One Machine Sequencing to Minimize Mean Flow Time with Minimum Number Tardy," *Naval Research Logistics Quarterly*, 22, 585-592 (1975b).

Emmons, H., "A Note on a Scheduling Problem with Dual Criteria," *Naval Research Logistics Quarterly*, 22, 615-616 (1975a).

Garey, M. R. and Johnson, D. S., "Complexity Results for Multiprocessor Scheduling Under Resource Constraints," *SIAM Journal on Computing*, 4, 187-200 ((1975).

Goicoechea, A., Hansen, D. R., and Duckstein L., *Multiobjective Decision Analysis with Engineering and Business Applications*, John Wiley & Sons Inc., New York (1982).

Grone, B., "A Scheduling Algorithm for Spacelab Telescope Observations," The 1982 NASA/ASEE Summer Faculty Fellowship Program, August (1982).

Guffin, O. T. , Roberts, B. H. and Williamson, P. L. , "A Practical Scheduling Algorithm for Shuttle-Based Astronomy Missions," American Institute of Aeronautics and Astronautics, 23rd Aerospace Sciences Meeting, Reno, Nevada, 11 pages, January (1985).

Heck, H., and Roberts, R. "A Note on the Extension of a Result on Scheduling with Secondary Criteria," *Naval Research Logistics Quarterly*, 19, 403-405 (1972).

Huckert, H., Rhode, R., Roglin, O., and Weber, R., "On the Interactive Solution to a Multicriteria Scheduling Problem," *Zeitschrift fur Operations Research*, 24, 47-60 (1980).

Jaap, J. P. and Davis, E. K., "The Scheduling Techniques of ESP2," Second Annual Workshop on Space Automation and Robotics, Dayton, Ohio, 6 pages, (1988).

Jeffcoat, D. E., "Algorithms for Resource-Constrained Project Scheduling," unpublished Ph.D. Dissertation, Auburn University (1990).

Jeffcoat, D. E. and Bulfin, R. L., "Simulated Annealing for Resource-Constrained Scheduling," *European Journal of Operational Research*, 70, 43-51, (1992).

John, T. C., and Sadowski, R. P., "On a Bicriteria Scheduling Problem," presented at *ORSA/TIMS Meeting*, Dallas, Texas, November (1984).

Kao, E. P., "A Multiple Objective Decision Theoretic Approach to One-Machine Scheduling Problems," *Computers and Operations Research*, 7, 251-259 (1980).

Kasahara, H., Kai, M., Narita, S. and Wada, H., "Application of DF/IHS to Minimum Total Weighted Flow Time Multiprocessor Scheduling Problems," *Systems and Computers in Japan*, 18, 25-34 (1988).

Keeney, P. L. and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, John Wiley & Sons, New York (1976).

Kurtzman, C. R., "Experiment Scheduling for the Industrial Space Facility," 7th AIAA Computers in Aerospace Conference, Technical Papers: Part 2, Washington, DC, American Institute of Aeronautics and Astronautics, 1088-1094 (1989).

Kurtzman, C. R. and AKIN D. L., "The MFIVE Space Station Crew Activity Scheduler and Stowage Logistics Clerk," 7th AIAA Computers in Aerospace Conference, Technical Papers. Part 2 Washington, DC, American Institute of Aeronautics and Astronautics, 947-957 (1989).

Kurtzman, C. R., "Intelligent Perturbation Algorithms to Space Scheduling Optimization," Goddard Space Flight Center: Space Network Control Conference on Resource Allocation Concepts and Approaches, 295-298, September (1991).

Kurtzman, C. R., "Time and Resource Constrained Scheduling, with Applications to Space Station Planning," unpublished Ph.D. Dissertation, Massachusetts Institute of Technology, (1988).

Manne, A. S., "On the Job-Shop Scheduling Problem," *Operations Research*, 8, 219-223 (1960).

Mathis, F. H., "Mathematical Programming Techniques for Scheduling Spacelab Crew Activities and Experiment Operations," NASA/ASEE Summer Faculty Fellowship Program, August (1981).

Mazzola, J. B. and Neebe, A. W., "Resource Constrained Assignment Scheduling," *Operations Research*, 34, 560-572 (1986).

Morris, W. T., *Decision Analysis*, Grid Publishing, Columbus, Ohio, 1977.

Nelson, R. T., Sarin, R. K., and Daniels, R. L., "Scheduling with Multiple Performance Measures: The One-Machine Case," *Management Science*, 32, 464-480 (1986).

Ohmae, Y., Hasegawa, S. and Okuda, M., "Multi-Project Scheduling," *Journal of Information Processing*, 15, 267-279 (1992).

Patterson, J. H., "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem," *Management Science*, 30, 854-867 (1984).

Pierce, R., "Expert Mission Planning and Replanning Scheduling System for NASA KSC Payload Operations," First Annual Workshop on Space Operations Automation and Robotics (SOAR 87), 299-306, October (1987).

Posner, M. E. "Minimizing Weighted Completion Times with Deadlines," *Operations Research*, 33, 562-574 (1985).

Potts, C. N., and Van Wassenhove, L. N., "An Algorithm for Single Machine Sequencing with Deadlines to Minimize Total Weighted Completion Time," *European Journal of Operations Research*, 12, 379-387 (1983).

Pritsker, A. A. B., Watters, L. J. and Wolfe, P. M., "Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach," *Management Science*, 14, 93-108 (1969).

Scherer, W. T. and Rotman, F., "Combinatorial Optimization Techniques for Spacecraft Scheduling Automation," *Annals of Operations Research*, NEED INFORMATION HERE (1990).

Sen, T. and Gupta, S. K., "A Branch and Bound Procedure to Solve a Bicriteria Scheduling Problem," *IIE Transactions*, 15, 84-88 (1983).

Shanthikumar, J. G., "Scheduling n jobs on One Machine to Minimize the Maximum Tardiness with Minimum Number Tardy," *Computers and Operations Research*, 10, 255-266 (1983).

Shanthikumar, J. G., and Buzzacott, J. A., "On the Use of Decomposition Approaches in a Single Machine Scheduling Problem," *Journal of the Operations Research Society of Japan*, 25, 29-47, (1982).

Smith, W. E., "Various Optimizers for Single Stage Production," *Naval Research Logistics Quarterly*, 3, 59-66 (1956).

Stacy, K. L. and Jaap, J. P., "Space Station Payload Operations Scheduling with ESP2," Second Annual Workshop on Space Automation and Robotics, Dayton, Ohio, 7 pages, (1988).

Steuer, R. E., *Multiple Criteria Optimization: Theory, Computation, and Applications*, John Wiley & Sons Inc., New York (1986).

Stinson, J. P., "A Branch and Bound Algorithm for a General Class of Resource-Constrained Project Scheduling Problem," unpublished Ph. D. Dissertation, Pennsylvania State University, 1976.

Talbot, B. F., Patterson, J. H., "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems," *Management Science*, 24, 1163-1174 (1978).

Thalman, N., Sparn, T., Jaffres, L., Gablehouse, D., Judd, D. and Russell, C., "AI Techniques for a Space Application Scheduling Problem," The 1991 Goddard Conference on Space Applications of Artificial Intelligence, 83-93, May (1991).

Van Wassenhove, L. N., and Baker, K.R., "A Bicriteria Approach to Time/Cost Trade-Offs in Sequencing," *European Journal of Operations Research*, 11, 48-54 (1982).

Van Wassenhove, L. N., and Gelders, L. F., "Solving a Bicriteria Scheduling Problem," *European Journal of Operations Research*, 4, 42-48 (1978).

Willis, R. J., "Critical Path Analysis and Resource Constrained Project Scheduling - Theory and Practice," *European Journal of Operational Research*, 21, 149-155 (1985).

Zoch, D. R., Lavallee, D., Weinstein, S. and Tong, G. M., "A Planning Language for Activity Scheduling," Space Network Control Conference on Resource Allocation Concepts and Approaches, 285-294, September (1991).

Zweben, M., "Constraint-Based Scheduling," Technical Report, NASA Ames Research Center, 19 pages, September (1991).