View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Embry-Riddle Aeronautical University


The Space Congress® Proceedings

1991 (28th) Space Achievement: A Global Destiny

Apr 24th, 2:00 PM - 5:00 PM

# Paper Session II-B - Space Shuttle Processing: A Case Study in Artificial Intelligence

Cindy Mollakarimi
*Lockheed Space Operations Company, Titusville, FL*

Monte Zweben
*NASA, Ames Research Center, Moffett Field, CA*

Bob Gargan
*Lockheed Corporation, Palo Alto, CA*

Follow this and additional works at: https://commons.erau.edu/space-congress-proceedings

## Scholarly Commons Citation

Mollakarimi, Cindy; Zweben, Monte; and Gargan, Bob, "Paper Session II-B - Space Shuttle Processing: A Case Study in Artificial Intelligence" (1991). *The Space Congress® Proceedings*. 9.
https://commons.erau.edu/space-congress-proceedings/proceedings-1991-28th/april-24-1991/9

This Event is brought to you for free and open access by the Conferences at Scholarly Commons. It has been accepted for inclusion in The Space Congress® Proceedings by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.

EMBRY-RIDDLE Aeronautical University
SCHOLARLY COMMONS

# Space Shuttle Processing: A Case Study in Artificial Intelligence

**Cindy Mollikarimi**
Lockheed Space Operations Co.
1100 Lockheed Way
M.S. LSO-439
Titusville, Florida 32780
407-383-2200 Ext. 2852

**Robert Gargan**
Lockheed AI Center
3251 Hanover St. O/9620 B/259
Palo Alto, California 94304
gargan@laic.lockheed.com

**Monte Zweben**
NASA Ames Research Center
M.S. 244-17
Moffett Field, California 94035
zweben@pluto.arc.nasa.gov

## Abstract

Scheduling the ground processing functions of the Space Shuttle is an inherently difficult task. Most automated scheduling tools are oriented towards manufacturing problems which are very different from shuttle processing. The distinguishing factors between shuttle processing and manufacturing are:

1. Shuttle processing requires much unplanned to be added to the schedule while manufacturing process plans are typically determined well in advance.
2. Manufacturing activities are significantly more predictable than shuttle repair activities in terms of resource needs and durations.
3. Shuttle processing is fundamentally more complex than typical manufacturing concerns.
4. Shuttle processing requires reasoning about orbiter configuration as well as tasks and resources.

To address these discrepencies, we have developed a new scheduling system that adopts Artificial Intelligence techniques. This paper describes the unique capabilities of this scheduling system as well as some preliminary results of the system using the shuttle data.

## 1 Introduction

### 1.1 Problem Description

Space Shuttle Ground Processing is the labor intensive effort of repairing and maintaining Space Shuttles between flights. Kennedy Space Center currently uses a three-tiered approach to developing schedules for shuttle missions. At the top level is the long range schedule which represents multiple shuttle flights over several years. The middle tier schedule is developed about 100 days prior to the beginning of the flight. Activities at this tier are generally Orbiter Maintenance Instructions (OMI's). An OMI describes a process that could extend form one hour to a month. The third tier represents the primitive operations that define OMI's. Due to the large quantity of operations and constant schedule changes, the third tier schedule is only generated for a 11 day time period. Separate specific schedules for each OMI are also maintained for short periods of time.

The scheduling process works as follows. Approximately 100 days before the beginning of a flight, high level planners create the middle tier schedule. They start from a generic processing schedule, and add new work specific to the current mission. Once finished, they perform CPM analysis to develop a schedule. This schedule has many resource constraint violations that must be resolved. The planner then adjusts the schedule to balance resources.

During the execution of a schedule, the planning and scheduling staff maintain a detailed 72 hour schedule. This schedule shows all activities that are being performed. Scheduling at this level is primarily monitored via daily scheduling meetings. During the meetings, representativies of the various work groups discuss their resource requirements and target completion times. The person in charge of the meeting coordinates the dynamic rescheduling of the work to be performed. Unfortunately, delays still occur. For instance, on one occasion, work that was scheduled could not be performed because the necessary quality control inspectors were not available. Unavailable parts, new problems, broken equipment, and scare manpower all contribute to the uncertainty that results in schedule delays. In this dynamic environment, it is imperative that the schedule coordinators are capable of predicting the impact of decisions they intend to make. KSC managers perform superbly, given the amount of information they currently exploit. We expect that the system described below will greatly improve this decision-making process.

### 1.2 AI vs. Project Management Tools

On the surface, it appears that project management tools are sufficient to solve the Space Shuttle Ground Processing problem. In fact KSC has performed in this manner for the second tier level and beginning to do so at the third tier as well. Unfortunately, the project management tools can only address part of the problem. Each activity has temporal requirements, resource requirements, and orbiter configuration requirements. Existing project management tools can represent most of the temporal requirements and some of the resource re-

quirements, but, no tool can represent the configuration requirements. Given this, the best any conventional tool can do is give you partial information.

Much of the work being performed on the orbiter requires that the orbiter be in a specific configuration. In most cases there may not be any technical sequencing requirement connecting several activities, but they are related through orbiter configuration. For example, certain tile work might require the payload bay doors to be in a closed position, certain tests require the payload bay doors partially open (to gain access to other parts of the orbiter), and certain tests such as deploying an antenna require the doors completely open. All three types of activities are independent of each other but require conflicting configurations of the orbiter. An effective scheduling system must consider these interactions.

Another important contribution of AI is that it allows a scheduling system to consider far more alternatives than a project management system. We search the space of possible schedules using heuristics (i.e., rules of thumb) to drive the scheduler toward more desireable solutions. Project management systems typically determine the earliest and lates possible times for activities and then simply resolves resource allocations by delaying activities later until resources are available.

AI approaches to scheduling are not new. ISIS [Fox, M. S., 1983] and then OPIS [Ow, P.S. and S.F. Smith, 1988] focused on developing a constraint based job shop scheduling system. KSC has also performed scheduling work previously. Empress [Hankins, G.B., et. al, 1985] and Phits [Gargan Jr., 1987] both focused on aspects of the planning and scheduling cargo processing.

### 1.3 Operational Concept

For each mission, we download a network of activities, constraints, and resources, from the Computer-Aided Planning and Scheduling system (CAPSS) that is based upon the Artemis project scheduling product. We then begin either with the schedule downloaded or schedule the network again within our tool. Our system can then evaluate the goodness of the schedule based upon the constraints and automatically resolve the constraint conflicts. Through a graphical user-interface, the planner or manager can enter schedule changes and will be informed of the ramifications of the schedule changes. This entails a before and after report of any changes to tasks and a graphical depiction of the violated constraints. Then the user can continue to manually modify the schedule or ask the rescheduler to *deconflict* the schedule automatically.

The major contribution of the work reported here is in rescheduling and therefore the remaining portion of this paper will concentrate on rescheduling. First, we formulate the rescheduling problem and then describe our rescheduling algorithm. Finally, we demonstrate results of using our algorithm on the Shuttle data.

## 2 Fixed Preemptive Scheduling

Scheduling is the process of assigning times and resources to the activities of a plan. Fixed preemptive scheduling is a specialization of classical scheduling, where each activity is preempted when it intersects an illegal time interval specified by an activity work calendar. An activity work calendar designates when work is prohibited. Holidays and overtime shifts are typical examples. Fixed preemptive schedulers split activities into the shortest sequence of contiguous subtasks, such that each subtask is legal with respect to the parent's work calendar. Fixed preemptive scheduling is distinguished from flexible preemptive scheduling because of the shortest contiguous sequence restriction. In other words, fixed preemptive problems prohibit idle time between the split subtasks of an activity, if that idle time is legal with respect to the task calendar.

Scheduling assignments must satisfy a set of domain constraints. Generally, these include temporal constraints, milestone constraints, and resource requirements. Temporal constraints relate tasks[1] to other activities (e.g., $end(T1) \leq start(T2)$) and milestone constraints relate tasks to fixed metric times (e.g., $end(T1) \leq$ 11/23/90 12 00 00). A resource requirement consists of a type and quantity of a resource (e.g., 4 mechanical technicians, 3 cranes). Each resource requirement has a corresponding capacity constraint. The constraint asserts that the resource must not be overallocated.

We also model the state requirements and effects for each activity A state requirement asserts that a state variable must have a certain value over a period of time (e.g., the payload bay doors must be open during an activity, the power and the hydraulics must be off during a task, or an area must be clear during an activity). Each state requirement has a corresponding constraint that forces the state variable to have the correct value over the specified time interval. State effects model how activities change state variables (e.g., an activity opens the payload bay doors from the end of an activity and persists until something else closes them, or an activity makes an area hazardous during the activity, etc.). Figure 1. summarizes the definition of fixed preemptive scheduling problems.

### 2.1 Rescheduling

In real-world applications, schedules rarely execute as planned because of the inherent uncertainty of operational environments. This uncertainty is typically manifested as:

1. modifications to the start and end of activities,
2. modifications to the quantity of resources required,
3. modifications to the work durations of activities,
4. unavailable or defective resources,
5. unexpected state conditions,
6. the addition of new activities that have become relevant, and
7. the removal of existing activities that have become obsolete.

[1] We use the terms task and activity interchangeably.

**Given a set of tasks, each with:**

1. a work duration
2. a work calendar
3. a set of temporal constraints
4. set of resource requirements
5. a set of state requirements
6. a set of state effects

**Find:**

1. a splitting of each task into subtasks,
2. a metric start and end time for each subtask, and
3. an assignment of a specific resource pool for each resource request,

**Such that:**

1. the subtasks of each task are the shortest set of contiguous tasks legal according to each activity work calendar,
2. the aggregate duration of subtasks sums to the corresponding task's work duration,
3. all temporal constraints are satisfied,
4. all state requirements are satisfied, and
5. no resource is overallocated or prematurely depleted (i.e, all resource capacity constraints are satisfied).

Figure 1: Fixed Preemptive Scheduling

As originally described in [Ow, P., Smith S., Thiriez, A., 1988], any rescheduling algorithm must be sensitive to the speed of rescheduling, the domain optimization criteria, and the amount of perturbation to the original schedule. Typical optimization criteria include the minimization of flow time (work-in-process time), the minimization of labor overtime, and the minimization of deadline tardiness. In our presentation of constraint-based simulated annealing below, we will discuss how our heuristics address optimization criteria.

## 3 Constraint-Based Simulated Annealing

We have extended traditional simulated annealing with a constraint framework that is used to both evaluate solutions and to improve solutions. In the following sections we describe our constraint language, give examples of the specific constraints used in our experiments, and finally present the role of constraints during search.

### 3.1 Constraints

Constraints are defined by the functions depicted in Figure 2. Every constraint has a penalty function, a weight, and a repair function. The penalty function measures the degree of violation for the constraint. The constraint weight is a measure of utility or importance for the constraint. Both the weight and penalty contribute to the *goodness* evaluation of a schedule called the cost function (see Figure 2.). The repair function modifies a schedule with the intention of improving the constraint's penalty. Repairs usually improve the penalties, but occasionally they inflict further constraint violations (that get repaired in later iterations of the search). Repair functions either replace resource assignments or reassign activity times. Tasks that are temporally reassigned must also be re-split according to their work calendars. Before discussing the repair functions in more detail, we present the MOVE operator that performs temporal reassignment.

#### 3.1.1 MOVE Operator

The MOVE operator places the given task at a given time, and then if necessary, moves other tasks to satisfy temporal constraints. It takes a state, a task, a time, and a direction and then finds a new state: $MOVE : S \times Task \times time \times direction \rightarrow S'$. A state is an assignment of values to all variables constituting a schedule. If direction is one, then the start of the task is placed at the given time, otherwise, the end of the task is placed at that time. The MOVE operator is implemented as a Waltz constraint propagation algorithm over time intervals [Waltz, D., 1975, Davis, E., 1987]. In constraint satisfaction terminology, this algorithm enforces arc-consistency [Mackworth, A.K., 1977, Freuder, E. C., 1982]. The algorithm recursively enforces temporal constraints until there are no outstanding violations.

After every schedule modification, the MOVE operator is employed to preserve temporal constraints. For example, if a task is delayed by the user, the Waltz algorithm will reassign each postrequisite that has violated temporal constraints. This in turn causes further violations that are recursively resolved until temporal quiescence.

Repair strategies also rely on the MOVE operator and are described in the next sections. It is important to note that the penalty, weight, and repair functions for temporal constraints are unnecessary because these constraints are preserved by the MOVE operator.

#### 3.1.2 Resource Capacity Constraints

The resource capacity constraint is a relation among the start time, end time, and a resource pool variable of a task. It states that the resource assigned to the request must not exceed its capacity during the task. For example, the first resource request of a task has the corresponding constraint:

$$holds(ST(?T), ET(?T), \\ TU(Pool(ResourceRequest(?T, 1))) \leq \\ Capacity(Pool(ResourceRequest(?T, 1))))$$

where ST stands for the start time, ET for the end time, and TU for total aggregate usage of the resource pool assigned to the request The penalty of the constraint is boolean – it is one if the condition is violated and zero otherwise. The weight of the constraint is one.

The repair for a resource capacity constraint initially attempts to substitute a new resource pool. If this does

not satisfy the constraint, it selects an activity contributing to the overallocation and reassigns it to another time. This reassignment exploits the MOVE operator to preserve temporal constraints.

The computational complexity of this repair is proportional to the cost of selecting a task to move. One viable strategy is to move the task associated with the constraint which yields a constant time selection. Another strategy is to move a different task that is simultaneously using the resource. Any heuristic used for this choice should consider the following criteria:

**Fitness:** Move the task that is using an amount closest to the amount that is overallocated. A task using a smaller amount is not likely to have a large enough impact and a task using a far greater amount is likely to be in violation wherever it is moved.

**Temporal Slack:** Any task that is highly constrained (i.e., few legal times) temporally is likely to cause temporal constraint violations and therefore could result in large perturbations to the schedule.

**Temporal Dependents:** Similar to temporal slack, a task with many dependents is likely to cause temporal constraint violations, if moved.

**Severity of Bottleneck:** Prefer tasks that do not need to be moved drastically to avoid extending flow time and to mimize perturbation.

**Priority:** The system should avoid delaying important tasks, but prefer moving them earlier.

**In-Process:** A task that has already begun should be completed as soon as possible, rather than temporarily stopping it, and then continuing later.

**Chronological Proximity:** It is better to move activities that start later in the schedule than those that are about to begin.

**Cycles:** It is better to avoid moving tasks that have been moved frequently in previous iterations because the iterative improvement algorithm can potentially cycle.

We address the speed of scheduling with the above criteria by considering only the next available time for each move, rather than by exploring many possible times. This same criteria also avoids extending flow time because later available times are not immediately considered

In our current implementation, we consider only fitness, temporal dependents, severity of bottleneck, in-process, and chronological proximity. Let T be the set of tasks that are using the resource during the time corresponding to a constraint. We calculate two probabilities for each member of T: the probability of moving the activity to the next later available time and the probability of moving the activity to its next earlier time. These probabilities are calculated by combining scores based on the criteria above. We disregard scores for criteria that are not very discriminatory with the hope of improving the effectiveness of this scoring. For example, if all the culprits have a comparable number of temporal dependents, then the scores for this criterion are discarded when calculating the move probabilities. The repair then chooses the move randomly with respect to the probabilities calculated.

The use of probabilistic repairs that are biased by heuristic knowledge is an important attribute of this technique because it circumvents infinite cycling and myopic side effects. For example, suppose the system resolves a resource constraint by delaying the *best* activity according to its heuristics. Then suppose a milestone is violated and the activity is returned to its initial time in the next iteration. Without probabilities, this would infinitely recur. Myopic side effects are also avoided because sometimes the system will disregard its local heuristic and result in better schedules.

### 3.1.3 State Constraints

The state constraint is a relation among a time interval, a state variable and a state. The constraint indicates that the state variable must be in the given state over the given interval. For example, a task requiring that the main landing gear of the space shuttle be deployed during the activity would be:

$$holds(ST(?T), ET(?T),$$
$$MainLandingGear(Atlantis) = DOWN)$$

The penalty of this constraint is boolean with a weight of one. To repair this constraint, the task with the requirement is reassigned to the next point in time when the state variable is assigned the desired value. Again the MOVE operator is used to shift a task and to preserve temporal constraints. In the future, we plan on extending this repair with options resembling the modal truth criterion of non-linear planners [Chapman, D., 1987]. One option is to introduce a new activity that satisfies the state requirement. Another is to move a task that sets the state variable appropriately, before the task with the requirement[2]. The final option is to move an activity that *clobbers* the required state to another time where it does not interfere. In future work, we intend to tackle planning problems with a probabilistic decision function analogous to the resource constraint repair. While this approach sacrifices the completeness properties that many non-linear planners enjoy, we believe that the anytime characteristics (see Section 4) of our search will be appealing.

### 3.1.4 Milestone Constraints

The milestone constraint enforces a relationship between a task and a metric time. For example, $holds(end(?T) \leq 11\ 23\ 90\ 12\ 00\ 00)$. The penalty of the constraint is boolean and the weight of the constraint is one. The repair uses the MOVE operator to shift the violated activity to satisfy the milestone.

## 3.2 Search Algorithm

Rescheduling begins when a user enters schedule modifications via a graphical user interface. Then, for each modification, the MOVE operator is enforced. This provides the initial scheduling assignment for annealing.

[2] This option was included in an earlier prototype of the system but it is not used in these experiments.

- Let $S \cong \{s_1, s_2, ..., s_n\}$ be the set of possible states where each $s_i$ is a unique assignment of values to all variables (i.e., a schedule).
- Let $C = \{c_1, c_2, ..., c_n\}$ be the set of constraints.
- $Penalty_{c_i} : S \rightarrow [0, 1]$ is a function defining the cost of a single constraint violation given a state.
- $Weight_{c_i} :\rightarrow [0, 1]$ is a function defining the importance or utility of a constraint.
- $Repair_{c_i} : S \rightarrow S'$ is a function that modifies a state to improve a constraint violation.

Figure 2: Constraints: A representation of generation and test knowledge.

The goodness of this assignment is calculated by the cost function. The specific cost function for our experiments is simply the number of constraints violated for the given assignment. Then, by repairing penalized constraints, it suggests a new solution and evaluates its cost. If the new cost is an improvement, it adopts the new assignment and continues. If the new solution is worse, the algorithm adopts it according to the escape probability. This last step allows the algorithm to escape local minima – situations where any move will result in a worse state. The basic algorithm is as follows (where S is a full schedule):

```
Solve(S){
   Old = Cost(S);
   Repeat until Old <= *THRESHOLD* {
          S' = New(S);
          NewC = Cost(S');
          If NewC < Old
             Then  Old = NewC; S = Next;
             Else { With probability Escape do
                          Old = NewC; S = Next;
                  };
          SaveBestSolutionIfNecessary;
   }
}
```

During each iteration, a subset of the outstanding violations is retrieved and then repaired. Currently, we repair the ten earliest availability constraints, and all the violated state-variable constraints. We plan to experiment with these parameters to determine how they affect the convergence to a solution. We bound the search by a maximum number of iterations and a maximum cumulative time. Generally, we use a very large time bound and a limit of 40 iterations per run.

## 4 Anytime Characteristics

When searching for a solution, the annealing algorithm saves its best solution to date and returns it when the algorithm is interrupted. This approach meets the criteria put forth in [Dean, T., and Boddy, M., 1988] to be classified as an anytime algorithm. Their criteria classifies anytime algorithms as those that:

1. can be interrupted and restarted

- $$Cost(s) = \sum_{c_i \in C} Penalty_{c_i}(s) * Weight_{c_i}$$

  is a function indicating the *goodness* of a state.
- $New : S \rightarrow S'$ is a function that transforms a state into a new state by a sequence of repairs: $Repair_{c_i}(s) \circ Repair_{c_j}(s) \circ ... Repair_{c_n}(s)$
- $Escape(s, s', T) = e^{-|Cost(s) - Cost(s')|/T}$ is the probability that the system will transition into a *worse* state in order to escape a local minimum.

Figure 3: The basic functions of constraint-based simulated annealing.

2. can be terminated at any time and will output an answer
3. return answers that improve in a well-behaved manner over time.

An additional consideration is that the solution output must be useful to the user. It makes no sense to be *anytime* if the solution can not be utilized effectively.

Our algorithm is interruptible, restartable, and outputs a solution when terminated. The solution quality increases as a step-function of time. Figure 4. is an actual run of our algorithm that demonstrates the relationship between the cost and best cost over time. Interim solutions are useful in our application domains because human schedulers can manually resolve conflicts in the schedule, especially when there are few conflicts that tend to be over-allocations of resources. Usually, the remaining conflicts can be resolved by allowing proximate activities to share resources. Our system is not capable of modeling this sharing capacity at this time.

## 5 Preliminary Results

We have modeled the Space Shuttle processing environment with about 500 activities that are split into a approximately 4000 subtasks. There are 1600 temporal constraints, 8000 resource constraints, and 3900 state requirements. The orbiter processing environment is rife with uncertainty and reactive decisions are made quickly. Suppose a scheduling change causes many conflicts for a particular schedule. It would be unacceptable for technicians and other personnel to remain idle while the system resolves every conflict. An anytime solution must be adopted, at least for the activities slated for immediate execution.

Figure 5. presents the results of simulating rescheduling scenarios using actual Space Shuttle processing data. We modify a random number of activities and then initiate rescheduling. The graph plots best cost against cumulative time. These graphs indicate that the algorithm scales to very large problems and maintains its anytime characteristics.

## 6 Development Status

The project to apply the scheduler to the KSC shuttle processing problem has been underway for about a year. Since early 1990, we have been working with actual data from a completed shuttle flight. While this data did not provide us with orbiter configuration data, it did provide us the ability to test our algorithms on realistic data sets.

Our plan is to shadow the STS-37 flight this winter. The main goal of this experiment is to collect the necessary scheduling information to enter into the system. To date, resource and configuration information for ground processing is either unavailable in an electronic medium or is significantly inaccurate. During the testing period, we will add in the changes to the work as they occur providing new schedules in a timely manner. As the quality of the information being stored in the knowledge base increases, our system will produce better schedules. The schedules we produce will then be compared to existing work schedule providing us some insight into new information to add to our system. Our hope is that even at this early phase of testing, we will be able to provide the KSC personnel some insight into alternative schedules that might not have been considered in the past.

## 7 Conclusion

In this paper, we described a research scheduling tool that is being applied to scheduling ground processing activities for the space shuttle. Research in this area has been on-going for several years and is at a state where an application of this magnitude can be attempted. We provided a brief overview of the scheduling system providing examples of the use of the various pieces to the KSC application. Experimentation with the repair strategies will continue as we use the rescheduling component of the system with the real data. It is generally felt, that there is a tremendous potential for savings to the shuttle program if this effort and the other phases of the scheduling process at KSC are automated.

## References


[Chapman, D., 1987] Chapman, D. Planning for Conjunctive Goals. *Artificial Intelligence*, 32(4), 1987.

[Davis, E., 1987] Davis, E. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32(3), 1987.

[Dean, T., and Boddy, M., 1988] Dean, T., and Boddy, M. An Analysis of Time-Dependent Planning. In *Proceedings of AAAI-88*, 1988.

[Fox, M. S., 1983] Fox, M. S. *Constraint-Directed Search: A Case Study of Job Shop Scheduling.* PhD thesis, Carnegie Mellon University, 1983.

[Freuder, E. C., 1982] Freuder, E. C. A Sufficient Condition for Backtrack-Free Search. *J. ACM*, 29(1), 1982.

[Gargan Jr., 1987] R.A. Gargan Jr. Mission planning and simulation via intelligent agents. In *Proceedings of Space Station Automation III*, November 1987.

[Hankins, G.B., et. al, 1985] Hankins, G.B., Jordan, J.W., Katz, J.L., Mulvehill, A.M., Dumoulin, J.N., Ragusa, J. EMPRESS:Expert Mission Planning and RE-planning Scheduling System. In *Expert Systems in Government Symposium*, 1985.

[Mackworth, A.K., 1977] Mackworth, A.K. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1), 1977.

[Ow, P., Smith S., Thiriez, A., 1988] Ow, P., Smith S., Thiriez, A. Reactive Plan Revision. In *Proceedings AAAI-88*, 1988.

[Ow, P.S. and S.F. Smith, 1988] Ow, P.S. and S.F. Smith. "Viewing Scheduling as an Opportunistic Problem Solving Process. *Annals of Operation Research*, 12, 1988.

[Waltz, D., 1975] Waltz, D. Understanding Line Drawings of Scenes with Shadows. In P. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.

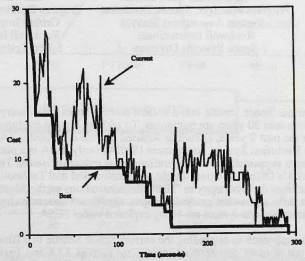


Figure 4: Best Cost and Current Cost

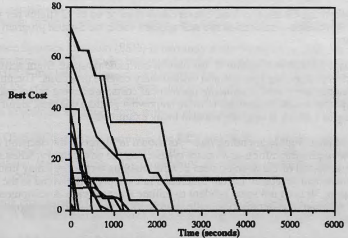


Figure 5: Space Shuttle Rescheduling Problems