Methodologies for Building Robust Schedules

John H. Dean
McDonnell Douglas Space Systems Company
16055 Space Center Boulevard
Houston, TX 77062
(713) 283-4008

Abstract: COMPASS is the name of a Computer Aided Scheduling System designed and built by McDonnell Douglas Space Systems Company for NASA. COMPASS can be used to develop schedule of activities based upon the temporal relationships of the activities and their resource requirements. COMPASS uses this information, and guided by the user, develops precise start and stop times for the activities. In actual practice however, it is impossible to know with complete certainty what the actual durations of the scheduled activities will really be. The best that one can hope for is knowledge of the probability distribution for the durations. This paper investigates methodologies for using a scheduling tool like COMPASS that is based upon definite values for the resource requirements, while building schedules that remain valid in the face of schedule execution perturbations. Representations for the schedules developed by these methodologies are presented, along with a discussion of the algorithm that could be used by a computer onboard a spacecraft to efficiently monitor and execute these schedules.

Introduction

The dictionary definition of robust is "strong and healthy." A robust schedule, therefore would be one which exhibits the characteristics we associate with strength and health. There are two interesting characteristics of schedule strength. The first is the ability of the schedule to accomplish useful work (how much is scheduled), and the second is the ability of the schedule to resist failure due to perturbations (the reliability of the schedule). Obviously these two characteristics are in competition with each other. A densely packed schedule will be more prone to failure if activities run long when actually executed. Alternatively, padding the scheduled durations of the activities with some extra "slack" time, in order to absorb any perturbations, reduces the number of activities that can fit into a fixed length schedule.

In order to examine the concept of robust schedules, we defined metrics that capture these two differing characteristics of schedule strength. Many metrics for measuring the amount of work that a schedule accomplish have been proposed before. In fact, it is these kinds of metrics that most schedule optimizers use as their objective function to maximize (or minimize). Examples of these kind of metrics include total make span time, summing the values of the activities placed on the schedule, mean or total tardiness, and mean time in process. This paper defines a schedule robustness metric that is a measure of the reliability of the schedule

Metrics for describing the reliability of hardware items is typically described as a Mean Time To Failure (MTTF). Furthermore, models exist which describe the expected reliability of systems built of component pieces for which the stochastic behavior is known, or can be derived. Similarly, our approach develops a notion of a MTTF for a schedule. To do this, we defined a concept of the failure of a schedule, and developed a model that describes how to calculate the MTTF of a schedule, given a description of the stochastic behavior of the activities that make up the schedule.

In order to define the concept of a schedule failure it is necessary to describe the overall schedule development and execution process. Schedule development begins with a set of tasks to be performed, along with there resource requirements. In addition, there may be some temporal relationships between tasks. For example, one task may require that a second task be completed before the first

task can begin. Based upon this information, along with other information such as the task priority or value, a schedule is created. Typically, the schedule that is created will be feasible. In other words, the schedule will contain no inconsistencies. All the given temporal constraints will be satisfied, and none of the resources will be oversubscribed. At execution time, the schedule is used to determine what resources should be assigned to what tasks, and when. As the schedule is executed, deviations from the a priori schedule will occur. If these deviations become too large, the schedule will no longer be valid, and a new schedule of the remaining tasks must be created. When this happens, the original schedule has suffered a failure.

Schedule development consists basically of assigning resources and times for the performance of activities in order to meet some deadline. It is well established that the Resource Constrained Scheduling decision problem (RCS) is NP-complete[1], and most scheduling decisions are NP-hard. This means that the length of time to develop a schedule is of exponential order relative to the number of tasks and/or resources. Since RCS is NP-complete, the time to verify a particular encoding of a solution to a RCS problem is of polynomial order relative to the number of tasks and resources, however. This provides the rationale for the definition of a schedule failure. When the perturbations become large enough that a polynomial bound algorithm can no longer accommodate the deviations, a schedule failure occurs, and the NP-hard problem must be solved again.

The remaining question is the representation of the schedule which can be verified in polynomial time. This paper will describe two schedule representations called the time constrained schedule representation and the order constrained schedule representation. These two representations can be merged into a single approach to allow the schedulers to use their choice of method.

Time Constrained Schedule Representation

The standard definition of a RCS problem is a follows: Given a set T of tasks t_i, for $1 \le i \le n$, with durations defined by a function $l: T \rightarrow \mathbf{Z}^+$, resource requirements $R_i: T \rightarrow \mathbf{R}_0^+$, and resource bounds B_i for $1 \le i \le k$, and an overall deadline $D \in \mathbf{Z}^+$; find (does there exist) a schedule $\sigma: T \rightarrow \mathbf{Z}_0^+$ such that

$$\sigma(t) + l(t) < D \quad \text{for all } t \in T \tag{1}$$

$$\sum_{\{t \in T \mid \sigma(t) \le j \le \sigma(t) + l(t)\}} R(t) \le B_i \quad \text{for all } 0 \le i \le n \text{ and } 0 \le j < D \tag{2}$$

where $\mathbf{Z}^+$ is the set of positive integers and $\mathbf{R}_0^+$ is the set of reals ≥ 0.

Under this notation, the set T defines the tasks that need to be scheduled. The tasks can be scheduled to start at any integral value of time between zero and the overall schedule deadline D. The resource requirements are defined by the functions R_i, which associate a real value with each task for each resource i. The resource bounds B_i defines the capacity of each resource. The function σ defines a schedule by assigning to each task an integral start time. Equations 1 and 2 guarantee that this schedule satisfies the overall deadline and the resource capacity bounds, respectively. However, this representation does not provide any mechanism for handling perturbations in the task durations, since only a single integer length is defined for each task by the function l.

The time constrained schedule representation extends this notation to the probabilistic case by assuming that the task length function returns an assumed duration of the activity. In general, one can define a family of mappings from the probability distribution for the task durations to an assumed duration for scheduling by

$$l_p(t) = \min \{z \in \mathbf{Z} \mid Pr(X_t \le z) \ge p\} \text{ for } 0 \le p \le 1 \tag{3}$$

where X_t is a random variable equal to the duration of task t. This formula defines the assumed duration of a task t, with respect to a probability p, to be the minimum duration for which the probability of completing the task is at least p. P is called the probability threshold.

This approach accommodates random variation in the task duration by defining a window in which the task can execute. The size of this window is controlled by the parameter p. When $p = 1.0$, the window is set to the worst case execution time for each task. A value of $p = 0.5$ would set the window for each task to the median value of the duration probability distribution. When this schedule representation is used by an onboard executive, a task would never begin before its assigned start time, as defined the

function σ. If the actual duration of any task exceeded the window defined for that task, we can no longer guarantee that the resource and deadline constraints are satisfied without resolving a NP-hard problem. Therefore, at this time a schedule failure has occurred. Since the boundary conditions by which a schedule failure is determined by the fixed time windows, this approach to accommodating variable duration tasks is called the time constrained representation.

The time constrained approach provides a simple mechanism for a real time schedule executive to be able to determine when to initiate tasks, while determining if the schedule remains valid in light of the actual durations seen so far. However, the time constrained representation is fairly fragile in terms of its resistance to failure. It is easy to see that the probability of a task successfully completing within its window is just p. If we assume that the durations for the tasks are stochastically independent, the probability that all n tasks will complete within their windows is p^n. As $n \rightarrow \infty$, $p^n \rightarrow 0$.

Order Constrained Schedules

The fragility of the time constrained approach is due to the fact that the schedule is successful if and only if all the windows completely surround the actual duration of their tasks. There is no capability in this approach for the random variations to "average out." Even if all but one task use less than their allotted time, but the one task exceeds its window, a schedule failure will occur. In trying to develop an alternative representation which allows for increased flexibility by allowing the random variations to accumulate and average out, the technique of pert charting naturally comes to mind.

In a pert chart, the schedule is represented as a directed acyclic graph (DAG). The DAG is a graphical representation of the predecessor - successor partial ordering. There are two commonly used representation of the DAG, called "activity on node" and "activity on edge." This paper will use the "activity on edge" representation. In the "activity on edge" representation of a pert chart, the nodes or vertices of this graph are called events, and the edges are the tasks or activities. If the edges *e1* and *e2* are part of a directed path through the DAG, in that order, then the task associated with *e1* is a predecessor of the task associated with *e2*. Occasionally dummy tasks need to be added to the DAG to accurately depict all the predecessor/successor relationships. These are usually drawn as dashed edges. The earliest possible start of a task is maximum length of all the paths that lead up to the start node for the task, where the length of an edge in the path is just the corresponding task duration. As the schedule executive executes the schedule, the actual durations can be substituted for the assumed durations for each task. This has the effect that a task can start only when all of its predecessors are finished.

With this idea as the basis for the order constrained approach, two questions need to be answered. How is the original resource constrained scheduling solution converted into a DAG, and how does the executive determine if the schedule is still valid based upon the DAG and the actual durations so far?

To illustrate the problems associated with creating a DAG from the resource constrained scheduling problem, consider the allocation of resource i as shown in FIGURE 1. In this figure, the horizontal axis represents

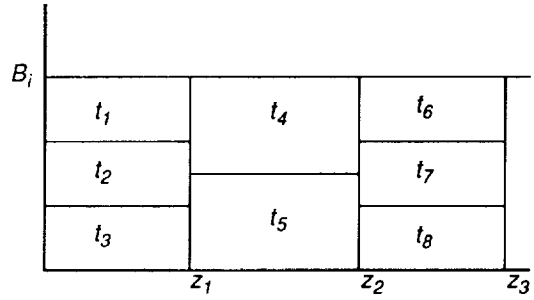

FIGURE 1 Resource timeline

time, and the vertical axis represents the allocation of resource i to the various tasks. In this example, t_1, t_2, t_3, t_6, t_7, and t_8 each have a resource requirement of 0.33 B_i. Tasks $t_4$and t_5 each have a resource requirement of 0.5 B_i. The time constrained approach guarantees that the sum of the resource requirements of all simultaneously executing tasks does not exceed the resource bound. For example, the executive would never allow tasks 1, 2 and 5 to execute simultaneously by ensuring that the windows for tasks 1 and 2 end before the window for task 5 begins. The problem for the order constrained approach is to define a partial ordering, implemented as a DAG, which accomplishes the same goal.

One straightforward way of accomplishing this goal is to define the partial order relation <• by

$$t_i <\bullet\, t_j \quad \text{iff} \quad \sigma(t_i) + l(t_i) \leq \sigma(t_j) .$$

In other words, this means that task t_1 precedes task t_2 if, and only if, t_1 is scheduled to finish at or before the scheduled start of task t_2. This in general will create more predecessor / successor relationships than are necessary, but it is a simple matter to go through and remove the redundant relations. FIGURE 2 shows the pert chart DAG which results from applying this procedure to the schedule in FIGURE 1.

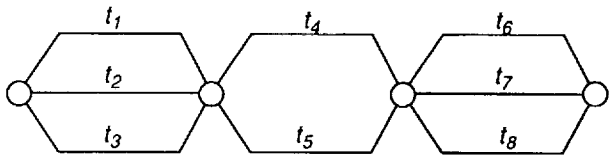

FIGURE 2 Pert DAG induced by resource constraints

While it can readily be seen that this partial ordering of the tasks will ensure that the resource capacity constraints are not exceeded, it can also be seen that it is overly constraining. For example, once tasks 1 and 2 complete, task 4 can be safely initiated since the resources required by tasks 1 and 2 are more than enough to satisfy task 4's requirement. Repeated application of this logic will eventually reduce the pert graph in FIGURE 2 to the graph shown in FIGURE 3.

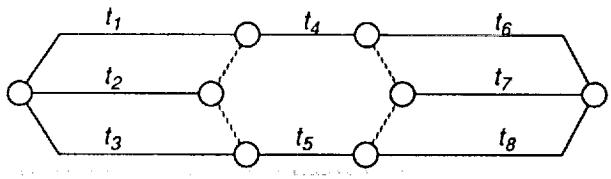

FIGURE 3 Reduced Pert DAG

Formally, then an order constrained schedule as a solution to an RCS problem is defined to be a partial ordering <• of the tasks in T such that:

$$\text{length}(\lambda) \leq D \quad \text{for all paths } \lambda \text{ in } <\bullet \tag{4}$$

$$\sum_{t \in S} R_i(t) \leq B_i \tag{5}$$

for all i, and for all $S \subseteq T$ such that
$t_1, t_2 \in S \Rightarrow (t_1, t_2) \notin <\bullet$ and $(t_2, t_1) \notin <\bullet$

Equation 4 is the revised constraint that guarantees that the partial ordering satisfies the overall deadline requirement. Equation 5 ensures that any set of tasks that might execute at the same time does not exceed the capacity of any resource.

One final question that needs to be addressed is how to calculate the length of a path through the pert network. Obviously, the length of the path should be the sum of the lengths of the individual tasks, but what value do we use for the length of the tasks, since we are assuming that these values vary? The a priori assumption, at schedule build time, is a task length based upon a probability threshold p, just as in the time constrained case. After the completion of the schedule, the a posteriori value of the task lengths is just the observed actuals. But what about during the execution of the schedule, when there are some actuals, and some unknowns? One could just use the a priori assumed lengths for the unknown durations. However a more general approach is to define a second probability threshold q, with $0 \leq q \leq p$. This defines a new length function l_q. The parameter q controls the amount of pessimism about the ability to recover when the actual execution is behind the a priori schedule. When $q = 0$, the executive will not declare a failure as long as there is some possibility of completing the schedule within its overall deadline by assuming that all remaining tasks will complete in their best case, or minimum durations. When $q = p$, the assumption is that the remaining tasks will complete in no less time than the a priori assumed durations. In either case, when the decision is made that the tasks will no longer complete by the overall deadline according to the current schedule, a failure is declared.

For a given partial order over the tasks of T, it is possible to calculate the length of the longest path, based upon the l_q length, and starting at the end node of each task t. If the actual end time of task t is later than $D - \max(l_q(\lambda))$, where λ is any path starting at t, then at least one path through task t will have a path length greater than D.

Therefore it is possible to precompute a deadline for each task by which time it must the task must complete in order for the schedule to meet the overall deadline in light of the actuals so far.

This suggests that it is possible to combine the two schedule representations into one. The combined representation consists of a partial ordering of the tasks of T, the window start times defined by the function $\alpha(t)$, and the window end time defined by $\Omega(t)$. For a time constrained approach, the partial order is empty, and the window start and end functions are defined by:

$$\alpha(t) = \sigma(t) \tag{6}$$

$$\Omega(t) = \sigma(t) + l(t) \tag{7}$$

For an order constrained approach, the partial is determined as described above, and the window start and end times are defined by:

$$\alpha(t) = 0 \tag{8}$$

$$\Omega(t) = D - \max_{\lambda \in P}(l_q(\lambda)) \tag{9}$$

where P is the set of all paths starting at the end node of t.

The job of the onboard schedule executive is to find all tasks that have no unfinished predecessors. Once the start window has been reached for these tasks, they are initiated. If any currently executing task fails to finish by its window end time, the schedule has failed and must be repaired by reinvoking the scheduler. It is fairly easy to see that the job of this onboard executive is tractable in the sense that it can be completed in a polynomial order of the number of tasks.

Development of Robust Schedules

Armed with this model of a flexible schedule representation than can accommodate some measure of perturbations during its execution, it is possible to define a method for using a deterministic scheduling system like COMPASS to build and manage robust schedules.

Since resource constrained scheduling is a NP-hard problem, COMPASS uses a mixed initiative dialog to generate feasible schedules that satisfy the user defined requirements.[2,3] Extending COMPASS to handle uncertain requirements, in particular probabilistic task duration, should therefore consist of adding commands to allow the user to interactively control the risk and uncertainty inherent in a particular schedule. Specifically, the user must be able to view, analyze and modify the risk and uncertainty inherent in a particular schedule. Analysis of a given schedule can be performed by performing Monte Carlo simulation of a large number of possible schedule executions to determine the MTTF of the schedule. If either the MTTF or the number of tasks the fit in the schedule is unacceptable, the user can adjust the a priori duration probability threshold and reschedule the tasks.

Conclusions

By combining fixed time windows with a pert style precedence graph, it is possible to build a schedule representation that can be executed and monitored by an automatic schedule executive in tractable way. Given that the durations of the scheduled tasks are not deterministic, but instead are represented by probability distributions, it is possible to identify probability threshold to control the a priori durations to use for scheduling and a posteriori limits to be monitored against. Given the probability distributions of the task durations, it is possible to perform a Monte Carlo analysis to determine the MTTF of a given schedule.

Acknowledgments

This research was supported in part by NASA under NAS9-17885.

References

1. Garey, M. R. and Johnson, D. S., *Computers and Intractability*, W. H. Freeman & Co., San Francisco, 1979.

2. Fox, B. R., *Mixed Initiative Scheduling*, AAAI - Spring Symposium on AI in Scheduling, Stanford, CA, 1989.

3. Fox, B. R., *Non-Chronological Scheduling*, AAAI - Spring Symposium on AI in Scheduling, Stanford, CA, 1989.