The Ames-Lockheed Orbiter Processing Scheduling System

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

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

Abstract

This paper describes a general purpose scheduling system and its application to Space Shuttle Orbiter Processing at the Kennedy Space Center. Orbiter processing entails all the inspection, testing, repair, and maintenance necessary to prepare the shuttle for launch and takes place within the Orbiter Processing Facility (OPF) at KSC, the Vehicle Assembly Building (VAB), and on the Launch Pad. The problem is extremely combinatoric in that there are thousands of tasks, resources, and other temporal conditions that must be coordinated. We are currently building a scheduling tool that we hope will be an integral part of automating the planning and scheduling process at KSC. Our scheduling engine is domain independent and is also being applied to Space Shuttle cargo processing problems as well as wind tunnel power scheduling problems. The significant technical contributions of our scheduling system are 1) the ability to handle dynamic rescheduling while considering the time it takes to reschedule, the optimization criteria in the domain, and the amount of perturbation to the original schedule; 2) the ability to represent arbitrary state conditions that change over time and the ability to declare the requirements and effects that activities have in relation to these conditions; and 3) the explicit representation and use of search control knowledge so that domain information can drive the scheduling process. Our scheduling engine is a constraint-based system implemented in CommonLISP that runs on a variety of platforms. We have tested our system with real orbiter processing data and have found the results promising. In the near future, we plan to deploy an early prototype of the system which will be used to shadow the current scheduling process at KSC.

1 Introduction

1.1 Description of Problem

Millions of people see or follow shuttle launches each year. They are familiar with the kinds of work performed by the shuttle crew. Most, however, are unaware of the amount of work that's involved in preparing a shuttle for launch. Preparing shuttles for launch requires successful and timely completion of many operations performed by many people.

Kennedy Space Center currently uses a three-tiered approach to developing schedules for shuttle flights. At the top level is the long range schedule. This schedule represents multiple shuttle flights over several years. The middle tier is developed about 60 days prior to the beginning of the flight. At this time, the planning person develops a flow that represents all of the activities that must be performed on the specific oribiter prior to launch. The granularity of activities developed at this tier is generally one OMI per activity. An OMI (Orbiter Maintenance Instruction) essentially describes a process that must be performed. This activity generally can be broken into about 10 primitive operations that must be performed on the floor. The third tier represents these primitive operations. Due to the large quantity of operations and the liklihood of change, the third tier schedule is generated each day for the next week.

The scheduling process works as follows. Approximately 60 days before the beginning of a flight, high level planners create the middle tier schedule. They are currently using a variant planning approach. That is, they are starting from a pre-existing flow, removing work that was unique to the previous flow, adding new work specific to this flow, and then rescheduling the activities. Once finished, they perform CPM analysis to develop a schedule. This schedule has many resource constraint violations that must be resolved. The planner person then uses the target start dates and resource balancing to make the schedule to work.

Once the flow has begun, the planning and scheduling people keep a detailed 72 hour schedule. This schedule shows all activities that are being performed. Scheduling at this level is primarily done based on past shuttle flights and 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 assurance peo-

ple weren't available. Other times, weather could cause a delay in certain work. These kinds of delays are part of the reality of executing the schedule. In this dynamic changing environment, it is all the more imperative that the people in charge be able to see potential impact of decisions being made. KSC managers do a super job currently, given the amount of information they have. It is desirable to recognize a problem and be able to work with the system to determine a solution.

1.2 Why Use A Heuristic Approach

On the surface, it appears that a good project management tool is all that is required to manage the scheduling of activities. KSC has been doing this at the second tier level and beginning to do this 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 configuration requirements. Existing project management tools can represent most of the temporal requirements and some of the resource requirements, however, no tool can represent the configuration requirements. Given this, the best any conventional tool can do is give you partial information.

For instance, there are activities (hazardous operations) that require the area to be cleared. When these activities occur, most other work can not occur. However, there may be no requirement saying that the hazardous operation must occur before or after other work. Conventional systems have no way of expressing this kind of temporal constraint.

Additionally, much of the work being performed on the orbiter requires that the orbiter be in a specific configuration. Again there may not be any hard temporal requirement connecting several activities, however, one activity may change the state of the orbiter and later activities may require that configuration. A simple example of this is requiring that the orbiter bay doors be closed to do certain types of tile work.

Heuristic approaches to scheduling are not new. ISIS [Fox83] and then OPIS [SFO86] focused on developing a constraint based job shop scheduling system. KSC has also had scheduling work done previously. Empress [HJK*85] and Phits [Gar87] both focused on different aspects of planning and scheduling of cargo processing.

1.3 Our Approach

We have viewed KSC scheduling as a Constraint Satisfaction Problem (CSP). In our system, we represent most information using variables. Each variable can take on a range of values. Constraints are used to filter the values of the given variables. For example, Figure 1.3 shows two activities and some of the variables associated with them. We use constraints maintain the relationship between the start time, the end time, and the duration of an activity. Additionally, using constraints we can express a requirement that one activity must start before the other one can start (or similarly with the end time) or that one must precede the other. During the scheduling process, the list of possible values for a given variable

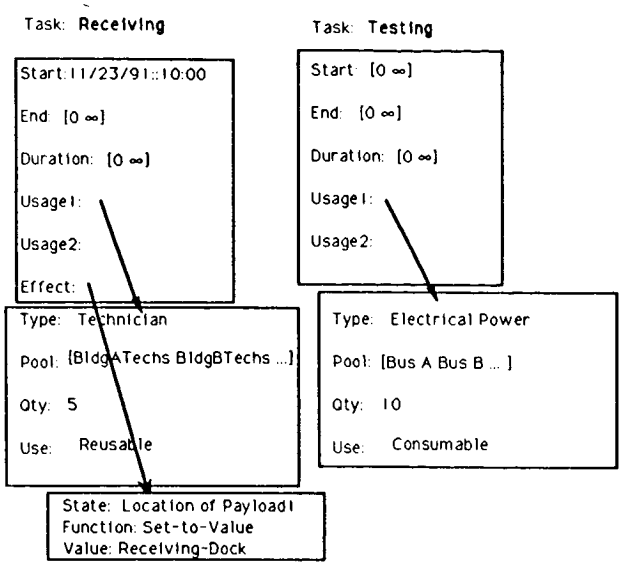

Figure 1: Task Representation and the Use of Constraints

is filtered based on the various constraints. The scheduling system searches the space of possible schedules for a time when all variables can be fixed and all constraints satisfied.

The remainder of this paper describes the process in more detail. We first introduce the knowledge representation we have utilized. This description will be of the various types of constraints we are using. Next we describe a rule system we have recently added that allows the user to encapsulate the search control knowledge explicitly rather than implicitly as is usually done. Finally, we describe the process of rescheduling. Rescheduling is especially critical to KSC since activity status is in a constant state of flux.

2 Knowledge Representation

Scheduling knowledge is being represented via constraints. Constraints are applied to the various variables that are a part of the objects of the system. For instance, our major object in the system is the activity. Activities contain many status slots. They also contain slots represented as variables such as the start time, end time, and duration. Constraints can be tied between multiple variables to maintain some consistency between various objects. For instance, a constraint would be used to state that a resource that is needed for a specific activity must be available during the time of allocation. The remainder of this section gives a brief overview of the types of constraints we represent.

2.1 Resource Requirements

Constraints can be used to require specific resources for an activity. Resource classes can be represented hierarchically. Each resource class can have one or more resource pools associated with it. Each resource pool can have a capacity of 1 or more. A resource pool of one is used to represent a specific individual or piece of

equipment. Alternatively, if uniqueness is not of concern the pools could contain many values. Using this type of resource, would result in allocating one from the pool to the activity, however, it wouldn't matter which one.

2.2 State Requirements

An important distinction between our system and others is our representation of state information and the use of constraints to maintain the proper state. For instance, most of the work on the shuttle requires that the orbiter be in some particular state (for instance, the orbiter doors being closed). As was mentioned before, current systems cannot represent this information. In our system, the activity representation has been extended to support task requirements and task affects in addition to the representation of states.

A state is an object in our system that can take on multiple values. We will eventually support finite state machines, although we currently support known state changes. For instance, the orbiter contains two bay doors. Each door can take values of opened, closed, or half-open. Other state information is represented similarly. Activities now can require that before the activity can start the object (in this case the shuttles bay door) must be open. Conversly, an activity can specify that as a result of executing the given activity the following object state will be changed. In the above example, an activity might specify that the pay doors are moved from the open to closed position.

The task requirements and affects described above are encoded as constraints on the given activity. These constraints must be maintained by the system in the same manner as the other constraints. By representing this type of information explicitly, the scheduler can take advantage of this type of knowledge when sequencing operations. Scheduling systems that have been developed previously would have represented this information implicitly as precedence constraints. This would reduce the overall flexibility of the system as well as make it more difficult to reschedule activities when problems arise. By using the state information to constrain the activities, it allows a more flexable schedule.

2.3 Temporal Relationships

Even though some information should be represented via state constraints, other information still should be represented via temporal constraints. Some of the most often used constraints are the precedence constraints. One type of precedence relation states that activity-1 must be completed before activity-2 can start. All standard off the shelf project management tools allow this form of representation. Additionally, some tools will also allow the user to specify that one activity must begin before the other one can begin (or end before the second one ends). The standard tools, however, do not generally allow the user to place delays on all the precedence relationships. In addition, they don't allow the user to express that two activities can occur in any order, however, they can not happen at the same time. This form of mutual exclusion is necessary for constraining hazardous operations on the space shuttle. This type of work must be done without other work being done in the area.

2.4 Calendars

Finally, it is necessary to represent when the work can actually be performed. We represent this type of information in a calendar. A calendar specifies when work can be performed. It takes into consideration holidays and daylight savings time. Each activity specifies a calendar that should be used to determine when it should be scheduled. This is consistent with the current way in which work at KSC is scheduled. There are a variety of calendars ranging from one 8-hour shift 5 days a week to a 7 day 24 hour calendar.

2.5 Representing Control Knowledge

One of our goals in developing this system was not to tie the scheduling process too closely to the KSC domain. Different applications should be scheduled in different manners. There is always domain knowledge that can be utilized to more effectively schedule operations. There is a basic conflict here between adding in this domain specific control knowledge and still maintaining generality.

Our desire would be to use a formal language to specify the control knowledge necessary to guide the scheduler through the search process. This would allow the applications developer to add specific control knowledge to the system to customize it for their application. Currently, most scheduling systems use LISP code to encode the search strategy. The problem with this is that it implicitly requires the search process to be the same for all applications of the scheduling system. Even if a single application area is all that was intended, this approach also fails because it restricts the end-user from being able to customize the search process at some later date should the need arise.

As a result of these issues, we are integrating a rule language into our scheduler. The user will encode all search control knowledge into the rule system. The system presently contains general knowledge encoded in rules. For instance, one rule that plays a role in determining the task to schedule states that if there is a same start or same end time relation between two tasks and the second task is not scheduled, then strongly prefer scheduling this task next.

As we continue to develop this portion of the system, we will be adding more KSC specific rules into the system. For example, there are certain types of work that are prone to identifying unforseen problems, so it is useful to do this work as early as possible so the problems can be identified. This type of domain specific information will eventually be represented as a rule to the system.

3 Dynamic Rescheduling

One of the most critical needs of KSC is the ability to reschedule the activities because work schedules are constantly changing. There's a variety of reasons ranging from unanticipated training schedules to bad weather. Whenever a problem arises, it is necessary for the planning people at KSC to adjust schedules to handle the problems proposing the appropriate work around. While

this is by far their worst problem, it is also one which is extremely difficult to meet using the existing tools. Because of the lack of support for the various kinds of constraints described above, planning personnel have used precedence relationships to force the schedule to take place in a certain order. Doing this, however, reduces future flexibility in the schedule by arbitrarily using precedence relationships where they are not really necessary Additionally, the planning personnel force the start time for many activities in the schedule to be fixed rather than relying on the constraints to determine new start times. This often results in the work not being performed when scheduled because the orbiter is not in the appropriate configuration or the necessary resources are not available. The resolution of these problems are often the subject of the daily scheduling meetings involving many KSC and Lockheed personnel.

Our goal with our scheduling tool is to provide the rescheduling capabilities to alleviate the above problem. In order to be successful, we must provide a tool that efficiently determines the new schedule. Our approach has been to investigate the use of iterative improvement scheduling algorithms. This approach differs from traditional "AI" scheduling approaches in that they incrementally repair complete solutions to the scheduling problem rather than systematically extending the partial solution to the problem. Our approach has led us to develop a framework [Zwe90] that converges on a solution by making local repairs to the violated constraints of some approximately correct schedule. This approach has two advantages over the other approaches. First, our approach is significantly faster than conventional heuristic based scheduling techniques. Second, because of the nature of our algorithm, a solution can be returned at any point in the algorithm, with the solution improving the longer the algorithm is given to execute.

Our algorithm is implemented in two phases. The first phase is the systematic repair of all temporal constraints. The result is a schedule that is consistent with respect to temporal constraints, but is likely to contain resource and state varable constraint violations. This schedule is the input to the second phase – constraint-based simulated annealing. During this phase, the scheduler incrementally repairs violated resource and state-variable constraints. The remainder of this section describes the two phases in more detail.

3.1 Temporal Shift

The temporal shift, which is the first phase of our rescheduling algorithm, takes a desired change in start and end times for a given activity and creates a schedule without any temporal constraint violations. We originally achieved a consistent schedule by systematically shifting all activities with temporal constraint violations in a fashion similar to those used by OPIS [OST88]. We later discovered that this approach by itself would not fill our needs because of constraints tying the end time of one activity to the end time of another (or start time to start time constraints). These constraints, in conjunction with the more conventional constraints (end time to start time) could lead the system back to the original task that was moved, so the approach of taking the earliest unscheduled task would no longer apply.

We decided to use Waltz's algorithm [Dav87] to address this anomaly. The algorithm is based on changing the intervals for each activity when shifted. Each change causes the interval to be filtered so that each interval continues to represent the range of times when an activity can begin (or end). The algorithm begins by rescheduling the changed task. It then collects the activities that have temporal constraint violations. Those activities' times are then filtered by a similar amount. This algorithm has the advantage that it quickly determines plausable schedules with minimum amounts of change and works for the general class of constraints used by KSC.

The algorithm is not guaranteed to be successful. If an activity has been marked as permanent and an attempt is made to move it, then the algorithm will return unsuccessfully. This could be useful for addressing milestones as well as activities dependent on some natural event (i.e. sunrise).

3.2 Constraint Based Simulated Annealing

The second phase is based on simulated annealing [KGV83]. It begins with the scheduling assignment resulting from phase one of rescheduling and then evaluates a "cost" of the assignment. The cost function for our experiments is the number of constraints violated for the given assignment. Then, by repairing 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 with some probability. This last step allows the algorithm to escape local minima. We have customized this general approach to constraint satisfaction problems which is described in more detail elsewhere [Zwe90]. The basic algorithm is as follows (where T is a set of tasks with assignments made in phase one):

```
Solve(T){
   Old = Cost(T);
   Repeat until Old <= *THRESHOLD* {
      Next = Find_New_Solution(T);
      New = Cost(Next);
      If New < Old Then { Old = New; T = Next;}
                   Else { With probability P do
                                { Old = New;
                                  T = Next;} };
      SaveBestSolutionIfNecessary;
   }
}
```

3.2.1 Systematic Repairs: Finding a New Schedule

In our previous work, we concentrated on simple local repairs in order to investigate the utility of the simulated annealing search framework. Here, we focus on fast rescheduling, with a heuristic bias against schedules with excessive work-in-process (WIP) time and against schedules that require radical perturbations to the original schedule. This bias is enforced by the repair strategies themselves. First, only those tasks involved in con-

straint violations are modified, and second, when tasks are moved they are not moved drastically.

Our repair strategy also exploits the knowledge that any task move is likely to violate temporal constraints. Thus, after any constraint repair causes a task to move, temporal constraint violations are resolved first by executing the temporal shift algorithm given above. Because these repairs explicitly exploit the knowledge of how repairs interact, they are no longer local.

The following are two of the repair strategies employed by the rescheduler:

```
capacity(?start ?end ?resource):
1. Deallocate this current resource.
2. Try to find a pool that is available
       from ?start to ?end.
3. If one exists, change ?resource to
       be that pool and reallocate.
4. Otherwise task = the task associated
       with this constraint;
      new-start = ?start + random(1 .. 10)
                  * c * d;
      new-end = new-start + duration(task);
      TemporalShift(?task, new-start, new-end);
```

The constant c is a small,fixed time unit (a day in the payload processing domain) and d is a direction (1 or -1) that is set by the change that the user makes. The strategy attempts to substitute a new resource pool, but if that is impossible, it moves the requesting task back or forward in time. After the task is moved, the temporal shift algorithm of phase one is executed – this systematically propagates the change caused by the repair to all temporal dependents.

```
temporal-equals(?t1 ?t2 ?a ?v):

First strategy:

1. supporter = the first task after ?t1
               that sets ?a = ?v;
2. task = the task associated with this
          constraint;
3. new-end = start(task) - c;
4. new-start = new-end - duration(supporter);
5. TemporalShift(supporter, new-start, new-end);

If unsuccessful:

1. task = the task associated with
          this constraint;
2. new-start = the first time of a state
               transition, t,
      (away from ?t1 in the direction of d)
      where ?a is set to ?v;
3. new-end = new-start + duration(task);
4. TemporalShift(task, new-start, new-end);
```

This repair is analogous to the modal truth criterion of non-linear planning [Cha87] but without the flexibility of adding actions. The preferred repair is to move a task that sets the state-variable appropriately, to a time interval before the task that has the requirement (i.e., to use Chapman's terminology, moving a white knight). If this is impossible, the task with the requirement is moved to a point in time when the state variable is set appropriately. This will move the task directly after the closest white knight.

In either case, to perform a move, the temporal shift of phase one is employed which results in a temporally consistent schedule.

4 Development Status

The project to apply the scheduler to the KSC shuttle processing problem has been underway for about a year. Since February, we have been working with actual data from a completed shuttle flight. While this data did not provide us with the data to utilize our state variable representation, it did provide us the ability to test our algorithms on realistic amounts of data.

Our plan is to shadow the STS-37 flight later this summer. The initial purpose of this first test is to collect the necessary scheduling information to put into the system. Currently, no information exists in computer form stating various configuration requirements of the orbiter for the various activities. Additionally, the resource information that is presently stored must be compared with the floor supervisors for accuracy as well as adding new resources that are not presently being stored. 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.

5 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 (not described here) at KSC are automated.

References

[Cha87] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(4), 1987.

[Dav87] Ernest Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(4):281–331, 1987.

[Fox83] Mark S. Fox. *Constraint-Directed Search: A Case Study of Job Shop Scheduling.* PhD thesis, Carnegie-Mellon University, 1983.

[Gar87] R.A. Gargan Jr. Mission planning and simulation via intelligent agents. In *Proceedings of Space Station Automation III*, November 1987.

[HJK*85] G. B. Hankins, J. W. Jordan, J. L. Katz, A. M. Mulviehill, J. N. Dumoulin, and J. Ragusa. Empress: expert mission planning and replanning scheduling system. In *Proceedings of Expert Systems in Government Symposium*, 1985.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, 1983.

[OST88] P. Ow, S. Smith, and A. Thiriez. Reactive plan revision. In *Proceedings of AAAI-88*, 1988.

[SFO86] Steven F. Smith, Mark S. Fox, and Peng Si Ow. Construction and maintaining detailed production plans: investigations into the development of knowledge-based factory scheduling systems. *AI Magazine*, 7(4), Fall 1986.

[Zwe90] Monte Zweben. A framework for iterative improvement search algorithms for constraint satisfaction problems. In *AAAI-90 Workshop on Constraint-Directed Reasoning*, 1990.