CONSTRAINT-BASED SCHEDULING

Monte Zweben
Artificial Intelligence Research Branch
NASA Ames Research Center
Moffett Field, CA 94035
Mail Stop 244-17
zweben@ptolemy.arc.nasa.gov

ABSTRACT

The GERRY scheduling system developed by NASA Ames with assistance from the Lockheed Space Operations Company, and the Lockheed Artificial Intelligence Center, uses a method called constraint-based iterative repair. Using this technique, one encodes both hard rules and preference criteria into data structures called constraints. GERRY repeatedly attempts to improve schedules by seeking repairs for violated constraints. The system provides a general scheduling framework which is being tested on two NASA applications. The larger of the two is the Space Shuttle Ground Processing problem which entails the scheduling of all the inspection, repair, and maintenance tasks required to prepare the orbiter for flight. The other application involves power allocation for the NASA Ames wind tunnels. Here the system will be used to schedule wind tunnel tests with the goal of minimizing power costs. In this paper, we describe the GERRY system and its application to the Space Shuttle problem. We also speculate as to how the system would be used for manufacturing, transportation, and military problems.

1.0 INTRODUCTION

Efficient scheduling is crucial for manufacturing companies that must balance limited production resources against challenging order requests. Airlines and package delivery companies must schedule large fleets of vehicles coordinating transportation goals with maintenance goals but must also be adaptable to external forces such as weather and equipment failure. The DoD also faces daunting scheduling problems ranging from logistics transport problems to mission planning problems.

NASA also faces complex scheduling problems including telescope observation scheduling, spacecraft crew scheduling, and spacecraft mission planning. Our research is motivated by the Space Shuttle Ground Processing problem. Ground processing entails the inspection, testing, and repair activities required to prepare a Space Shuttle for launch at the Kennedy Space Center (KSC) in Florida.

This paper describes a scheduling algorithm that is being used to schedule shuttle ground processing but is also applicable to the other scheduling problems alluded to above. First we present our definition of a scheduling problem and then describe our scheduling method. After presenting the general approach we describe how it is used to solve the Space Shuttle problem and then briefly describe how it can be adapted to other real-world problems.

2.0 SCHEDULING

In this section we define the scheduling problem beginning with a simplified version and evolving to a more realistic definition.

2.1 General Problem

Generally scheduling systems are provided a set of activities, relationships between these activities (such as predecessor-successor requirements), resource requirements for each task (i.e., how much of what kind of resources are necessary), and a set of deadlines or milestones. With this input, scheduling systems determine start and end times as well as an assignment of resources to each activity such that: 1) the relationships between tasks are preserved, 2) no resource is over-allocated (i.e., at no time does the demand for a resource exceed its supply), and 3) all milestones are met.

For example, consider a Space Shuttle repair scenario where each Space Shuttle Main Engine needs to be inspected, removed, repaired, re-installed, and tested, in that order. The tasks associated with different engines are unrelated meaning that any task in support of one engine could simultaneously occur with the tasks in support of a different engine. Assume that each task requires 10 technicians, an

engineer, and a safety inspector. Suppose there were only 15 technicians on call for each shift. In this case, no two activities would be able to occur in parallel because together they require 20 technicians and there were only 15 available. If there were more technicians the system would place tasks in parallel in order to meet the milestone.

Consequently, a scheduler would determine activity start times that sequence the activities completely serially because any two activities' demand exceeds the supply of technicians.
In summary, scheduling systems search through the space of possible start times and resource assignments with the goal of finding an assignment that satisfies all domain constraints. These constraints include milestones, resource capacities, and temporal relationships.

2.2 Optimizing and Satisficing

Most scheduling systems simply find an acceptable schedule and then terminate. They are not necessarily concerned with finding the best schedule that satisfies the constraints. In many domains, there is great variability in the quality of schedules that satisfy constraints. For example, an organization might want to find the schedule that uses the minimal amount of overtime labor, or one that minimizes the overall flowtime of a schedule. Unfortunately, deriving the *optimal* schedule is a time consuming process that requires a great deal of combinatoric search. In most cases, near-optimal solutions are sufficient. The process of problem-solving with the goal of finding near-optimal solutions is called *satisficing"*[Simon]. The satisficing algorithm presented in the next section continues to search after finding a schedule that merely satisfies constraints, in order to find better quality schedules according to stated optimization criteria.

2.3 State Conditions

Most scheduling systems reason about the changing availability of resources over time but few track the changes of arbitrary conditions. State conditions are attributes of the scheduling problem that change with time. The tasks of a scheduling problem are constrained by these conditions and occasionally the activities change the values of the conditions. Examples include the position of switches and other mechanical parts, the readings of sensors, and the location of objects. Schedulers that handle state conditions must provide a language to specify the additional *state constraints* and to specify the *effects* that tasks have on state conditions.

Examples of state conditions in the Space Shuttle scheduling problem include the position of the payload bay doors, the status of the orbiter's hydraulics system, and whether an area adjacent to the orbiter is hazardous. Examples of state constraints include the rule that no task may take place in a hazardous area. Additionally, some activities require orbiter hydraulics while others require the hydraulics to be off. Likewise, certain activities require the payload bay doors to be in one of their three main positions. Activities also change these conditions. Some activities result in opening or closing doors and turning hydraulics on or off. Similarly, hazardous operations cause the areas surrounding their respective work areas to be considered hazardous thus delaying any other operations that must share those areas.

Our system supports the modeling of state conditions and provides a language for state constraints and task state effects however, details of this language are beyond the scope of this paper. It suffices to say that the satisficing search mechanism presented below considers state constraints and state effects as it schedules.

2.4 Pre-emptive Scheduling

Pre-emption is the process of temporarily suspending activities and resuming them later. Pre-emption can be caused for a number of reasons. In a telescope observation scheduler, the system might interrupt an activity when a more important and rare astronomical event arises. Activities could also be suspended to allow more contentious activities to execute in their limited windows of opportunity. These are examples of *flexible pre-emption*. The Space Shuttle problem requires a more restricted type of pre-emption called *fixed pre-emption*.

Fixed pre-emption is the suspension and resumption of activities according to a strict calendar. In the Shuttle domain the calendar corresponds to work shifts. Some activities can be worked all shifts, every day, while others have certain restrictions such as no weekends, only third shift, or only first shift.

To handle this sort of pre-emption our system requires a calendar for each task that indicates how it is to be pre-empted or split into smaller pieces. For example, suppose a task that requires 12 hours work is assigned a first shift, no weekends calendar. If the task begins at 8:00 A.M. Monday, it will be suspended at 4:00 P.M. that day, then resumed Tuesday morning at 8:00 A.M., and then finally completed at noon. Thus the task spans two calendar days. Suppose however that the task began Friday.

It will then terminate Monday thus spanning four calendar days.

Pre-emption greatly complicates scheduling because of the way it interferes with resources and state conditions. Whenever a new time is considered for a task, the task must be split according to the calendar. However, it is sometimes inappropriate for the state and resource constraints to be valid for the entire period of the pre-empted tasks. For example, the resource needs that correspond to human labor should not be required during the suspended periods of the task's duration. In other words, it makes no sense for employees to be standing around idle. In these cases, the resource and state constraints must be *inherited* to the split tasks thus avoiding the idle periods. Other constraints can remain active throughout the duration of the task. An example of these include a resource request for a heavy piece of equipment that requires significant assembly. The equipment usually remains in the work area, unavailable to others because of the overhead required to set it up.

Our system allows the user to designate which resource requests and which state constraints and effects are to remain valid throughout the suspended period and which ones are valid only during active periods.

3.0 CONSTRAINT-BASED SCHEDULE REPAIR

In this section, we present the search method used by the GERRY scheduling system. The system allows the user to specify a set of *tasks*, a set of *state conditions*, and a set of *resources*.

Tasks have start and end times, resource requests, resource assignments, work durations, and calendars.

Resource pools are defined by the user and have a corresponding maximum capacity. For example, in the Space Shuttle domain there might be a pool of 20 technicians or three pools of 5 forklifts.

State conditions are also provided by the user along with the initial values for each condition. For example, the right-hand payload bay door with an initial value of *closed*.

3.1 Input

• Task Data - For each task, the following information is provided:
- work duration - amount of active work time required for the task to complete.
- calendar - the pre-emption times for the task.
- resource requests - the list of resource types and quantities necessary.

• Resource Data - For each resource pool, the following information is provided:
- type - the name of the resource category that the pool is classified as.
- capacity - the maximum amount of the pool that can be simultaneously assigned.

• State Condition Data - For each state condition, the following information is provided:
-initial value - the value for a condition which persists until a task changes it.

3.2 Output

For each task, the following information is determined:
• start time - the beginning of the task.
• end time - the finish of the task.
• resource assignments - the actual resources chosen.

The rules and preferences that schedules must observe are captured by *constraints*. Constraints are relationships that are desired by the user and are composed of the following items:

• Arguments - the tasks, resources, or state conditions that are related to each other.
Example: a task and a resource pool are arguments to a resource capacity constraint.

• Penalty - a score of how poor the arguments are with respect to the constraint.
Example: if the resource pool argument were overallocated during the time of the task, then the penalty of the resource capacity constraint would be high.

• Weight - a number reflecting the importance of the constraint.
Example: resource capacity constraints for scarce resources such as expensive equipment would have higher weights.

• Repairs - suggested schedule modifications that are intended to improve the penalty.
Example: move tasks that are involved in an overallocation to a time where more of the resource is available.

Loosely speaking, the penalty is analogous to the amount of money one would have to pay with respect to the current assignment of times and resources. The weight of the constraint reflects its importance when compared to other constraints. Repairs are methods for changing the schedule, either by substituting resources, or by moving, adding, or deleting activities.

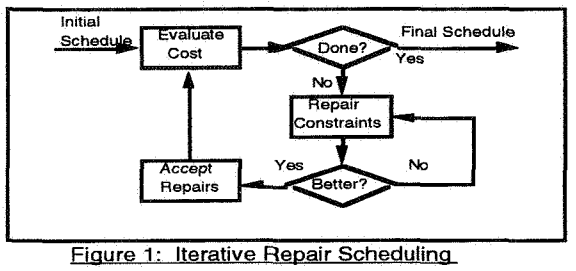

Figure 1: Iterative Repair Scheduling

Figure 1. presents our simple iterative repair algorithm. See [Zweben et. al.][Zweben1 et. al.] for details. The system begins with an initial schedule and then initiates a repair loop.[1] If the problem posed to the system is a rescheduling problem then the initial schedule is the schedule with changes imposed by the user. If the system is scheduling from scratch, then all tasks are placed at their earliest possible start times while preserving temporal constraints. This is accomplished with a well known polynomial (i.e., efficient) algorithm [Davis, Waltz].

In the repair loop, the system calculates the cost of the solution. This calculation is simply the sum of each constraint's penalty multiplied by its weight. If the cost is below a threshold set by the user, the search terminates.[2] Otherwise, a cross-section of the highly penalized constraints are repaired. We often refer to these constraints as the violated constraints because their penalties exceed a certain threshold.

In short, the system simply starts with a schedule and isolates the violated constraints. Then it moves tasks around and substitutes resources as suggested by the repairs embodied in the violated constraints. It accepts the new schedule if the new cost is lower than the previous cost. If the repaired schedule is worse than previous one, it is rejected and new repairs are attempted on the previous schedule.[3] The system continues this process until the cost of the solution is acceptable to the user, or the system is terminated by the user. The system also terminates if a certain number of iterations have been tried.

[1]Similar repair techniques are used in OMP [Biefeld, et. al.] and in the work on the MIN-CONFLICTS heuristic [Minton, et.al.].

[2]The search also terminates if the system exceeds the iteration bound imposed by the user.

[3]Actually the system sometimes accepts worse schedules in order to break out of situations called *local minima*.. In a local minimum, any repair leads to a worse schedule, but subsequent repairs could improve the schedule so that it is eventually superior. This technique is called simulated-annealing and was originally reported in [Kirkpatrick].

4.0 SYSTEM FUNCTIONALITY

4.1 User Interface Overview

GERRY allows both manual and automatic scheduling thus requiring a sophisticated user interface. The user instructs the interface to display a chart. The chart library is extensible and allows the user to define different views of the schedule. For example, the user could ask for a time-line (i.e., Gantt Chart) and a resource profile (i.e., a histogram of resource usage over time) to be displayed for every task. Alternatively, the user could require a time-line and state condition profile (i.e., a histogram of state conditions over time) for a specific set of tasks. Figure 2. is part of a Space Shuttle schedule with a time-line and resource profile. Figure 3. shows the same schedule (with hourly units) at a coarser level of resolution. Zooming in and out of different levels of resolution is accomplished by clicking in the upper right hand boxes of the chart. This convention was adopted from the COMPASS scheduling system of Barry Fox at McDonnell Douglass in Houston, Texas.

When the chart is displayed each of the activities and histograms are mouse-sensitive. One can drag a task to a new location, modify its status as pending, active, or complete, and ask for a the list of resources that the task uses. The status of an activity is reflected by the shading of the displayed bar. If the activity is shaded black, it is complete. Ongoing activities are outlined in black (but not shaded). Unshaded activities that do not have an outline are pending.

In addition to status, shading is also used to reflect the danger level of the task. If an activity is shaded with a cross-hatch pattern then it is considered hazardous.

The interface supports many task look-up methods. For example, one can scroll to a point in a chart where a particular task begins, one can scroll to an over-allocation, or one could simply use scroll bars.

Also included in the interface are a form editor and a temporal constraint grapher. The form editor, shown in Figure 4., is simply a mechanism to enter new activities and constraints. The grapher, shown in Figure 5., allows the user to inspect the complex temporal relationships between tasks. The grapher works in a demand-driven manner instead of cluttering the display with the entire schedule's graph. One clicks on a task and the graph expands from that point on.

4.2 Schedule Monitoring and Rescheduling

While GERRY can be used as a planning tool for future schedules, its strength is in its ability to monitor schedule execution and adapt to the schedule changes imposed by elements outside the system's control. Users modify tasks by changing their status, dragging them around, changing their constraints, and by adjusting task durations. Users can also add and delete tasks.

One of the most important functions of the user interface is the ability to alert the user to the ramifications of their changes. There are three main charts used to inform the user of what they have done: 1) a before-and-after chart, 2) a constraint violation summary chart, and 3) a constraint violation problem report. The before-and-after chart, shown in Figure 6., reports all the tasks that have been changed by indicating their new position and their old position. The constraint violation summary chart, shown in Figure 7., is a list of the current constraint violations. By clicking on one of the violations in the constraint violation summary chart, a constraint violation problem report appears that explains the conflict. For example, the constraint violation problem report shown in Figure 8. explains why a particular resource capacity constraint is violated. Only the tasks that request the resource during the interval of the violation are displayed and the interval is shaded in color.

After changes are given to the system, the user can manually resolve the outstanding violations or ask the system to use the iterative repair method. When the automated method terminates (or is interrupted) it reports a before-and-after chart. If there are outstanding violations, then a constraint violation summary chart is also displayed.

5.0 PROBLEM DOMAINS

5.1 Space Shuttle Ground Processing

In the Space Shuttle Ground Processing domain we start with schedules provided by an existing project management tool used at the Kennedy Space Center. It uses the critical path method (CPM) to schedule activities at the earliest possible times. These schedules are used by Flow Managers who have the responsibility to prepare the orbiter in time for the designated launch date. The data sets start with about 300-400 activities that expand into thousands of split tasks and constraints. Our project team attends KSC schedule meetings and updates the schedule accordingly. Currently, we are in the process of delivering new schedules to the flow managers and

beginning to use the constraint-based repair method to optimize the schedule. Below we enumerate the constraints used for this application.

The constraints for this application include:

5.1.1. Resource Capacity Constraints

• Arguments: - Start of a task
- End of a task
- Resource Pool

• Penalty: - Constraint is violated when the pool is over-allocated during the task.
Example: 3 tasks in parallel all need a technician but only 2 are on call.

• Repair: - Strategy 1: Substitute
Assign a resource pool of the same type that is not over-allocated.

- Strategy 2: Move
Move one of the tasks contending for the resource to the next time when there is a sufficient amount of the resource.

To decide which task to move the following heuristics are used:

-Heuristic 1: Fitness
Prefer to move tasks that use an amount of the resource that is close to the amount over-allocated.

-Heuristic 2: Slack
Avoid moving tasks that have little slack between their earliest and latest start times.[4]

-Heuristic 3: Dependents
Avoid moving tasks with temporal dependents (e.g., postrequisites).

-Heuristic 4: Priority
Avoid moving high priority tasks.

-Heuristic 5: In Process
Avoid moving tasks that have begun.

-Heuristic 6: Proximity
Avoid moving tasks that are to begin soon.

[4]Slack time indicates the amount of time a task could slip before it affects the milestone. This measure is calculated from the CPM algorithm mentioned earlier.

5.1.2. State Constraints

Arguments: - Start of a task
- End of a task
- State Condition
- Required State

Penalty: - Constraint is violated when the condition does not reflect the required state during the task.
Example: The payload bay doors are closed for a task that requires them to be 160 degrees open.

Repair: - Strategy 1: Move
Move the task to the next time where the state condition reflects the desired state.

5.1.3. Milestone Constraints

Arguments: - End of a task
- Due Date

Penalty: - Constraint is violated when the end of the task is completed later than the given date.

Repair: - Strategy 1: Move
Move the task back earlier, before the given time.

Currently we lack the domain knowledge that would distinguish between the importance of these constraints so they all have the same weight. The system uses these constraints (and their corresponding repairs) to minimize missed launch dates (via milestone constraints) and to minimize over-allocation of KSC personnel (via resource constraints) while maintaining the correct orbiter configurations (via state constraints).

In the near future we intend to include another constraint that demonstrates the flexibility of our system. This new constraint will inform the system to minimize labor costs by avoiding overtime labor on the weekend.

5.1.4. Weekend Constraints

Arguments: - Global constraint.

Penalty: - Constraint is violated when a large number of tasks intersect the weekend.

Repair: - Strategy 1: Move

Move the tasks with sufficient slack time off the weekend.

5.2 Manufacturing Problems

In job-shops, there are resources such as machines and human operators. Similar to pre-determined launch dates in the NASA domain, job shops have order due dates. In job shops, each machine has to be set up correctly depending upon the task at hand. Typically jobs follow a process plan that is fairly well known in advance. There are very similar optimization criteria in this domain as there are in the Space Shuttle domain. In fact, the constraints described above are usually applicable. Additional constraints would also be written that would modify the schedule to minimize the number of machine set-ups required thus minimizing flow time. Additionally, constraints that minimize the amount of work-in-process inventory would be incorporated. We claim that a knowledge engineer could easily do this without writing another program but rather simply writing new constraints.

5.3 Airline, Trucking, and Parcel Service Problems

In the transportation sector, large fleets of vehicles must be scheduled on a daily basis. These operations are stricken with unexpected events such as unpredicted malfunctions and malevolent weather. When these events occur, it is crucial to get back on track minimizing impact to the original schedule.

In transportation problems there are additional decision variables that constrain the schedule which include the start and end locations of any task and the speed that one will travel between those locations. Constraints would be added that relate the locations, speed, and duration of the task. Additionally the quantity of certain resource requests must be constrained by the duration. For example, the amount of fuel required by an aircraft is dependent upon how long the plane will travel. Constraints that serve to minimize fuel and delays, while observing safety constraints would be added to the constraints discussed above.

5.4 Military Problems

Many military problems resemble transportation problems but with targeting and probability of success factors added. The tasks are generally trips from one's bases to the enemy's targets (and hopefully back home again). In addition to the transportation constraints discussed above, constraints that model the appropriateness of various aircraft and ordnances for targets would be required.

5.5 Power Utilization

Ames Research Center is also deploying GERRY to minimize power consumption of the Ames wind tunnels. The rates that local utilities charge NASA are based upon the season, time of day, and quantity of power used. Therefore the wind tunnel test schedule can greatly affect energy costs. Ames will use GERRY to adjust the wind tunnel test schedule to minimize its power costs but maintaining the deadlines imposed by those who need the tunnels. Constraints are used to penalize schedules of high cost while repairs move tasks to lower these costs.

6.0 SUMMARY

We have developed a framework for scheduling called constraint-based iterative repair. This framework supports complex scheduling problems where satisficing is required. GERRY, the system based upon this framework, is operational and is being deployed at the Kennedy Space Center in Florida in support of Space Shuttle Ground Processing. The system uses the optimization criteria encoded as constraints to find near-optimal schedules. We claim that our approach is amenable to other problems faced within industry and government and welcome others to apply it.

ACKNOWLEDGEMENTS

Thanks to the entire GERRY project team at NASA Ames, the Lockheed Space Operations Company, and the Lockheed Artificial Intelligence Center. Also thanks to Martin Cohen for his careful review of this paper.

BIBLIOGRAPHY

[Biefeld, et. al.] Biefeld, E., and Cooper, L., Bottleneck Identification Using Process Chronologies, In *Proceedings of IJCAI-91*, 1991.

[Davis] Davis, E., Constraint Propagation with Interval Labels, *Artificial Intelligence*, 32(4), 1987.

[Kirkpatrick] Kirkpatrick, S., Gelatt Jr., C., Vecchi, M., Optimization by Simulated Annealing, *Science*, 220, 1983

[Minton, et. al.] Minton, S., Philips, A., Johnston, M., Laird, P., Solving Large Scale CSP

and Scheduling Problems with a Heuristic Repair Method, In *Proceedings of AAAI-90.*

[Simon] Simon, H., *The Sciences of the Artificial,* MIT Press, 1969

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

[Zweben, et.al.] Zweben, M., Deale, M., and Gargan, B., Anytime Rescheduling, In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning and Scheduling,* 1990.

[Zweben1 et.al.] Zweben, M., Deale, M., and Gargan, B., An Empirical Study of Rescheduling Using Constraint-Based Simulated Annealing, In *Proceedings of the IJCAI-91 Workshop on Production Planning and Scheduling,* 1991.