Iterative Repair for Scheduling and Rescheduling

MONTE ZWEBEN
AI RESEARCH BRANCH, MAIL STOP 269-2
NASA AMES RESEARCH CENTER
MOFFETT FIELD, CA 94035
(415) 604-6527
(415) 604-6997 FAX

NASA Ames Research Center
Artificial Intelligence Research Branch

Technical Report FIA-91-16

September, 1991

Iterative Repair for Scheduling and Rescheduling

Monte Zweben
Eugene Davis*
Michael Deale†
NASA Ames Research Center
M.S. 244-17
Moffett Field, California 94035
zweben@kronos.arc.nasa.gov

September 30, 1991

Abstract

This paper describes an iterative repair search method called constraint-based simulated annealing. Simulated annealing is a hill climbing search technique capable of escaping local minima. We demonstrate the utility of our constraint-based framework by comparing search performance with and without the constraint framework on a suite of randomly generated problems. We also show results of applying the technique to the NASA Space Shuttle ground processing problem. These experiments demonstrate that the search method scales to complex, real-world problems and reflects interesting anytime behavior.

1 Introduction

Iterative repair scheduling techniques typically hill-climb through a space of complete schedules repeatedly making patches or fixes to the schedule's weak points [Zwe90, Min90, Bie91]. The informedness of individual repairs can range from weak, random repairs to very knowledge intensive repairs. This paper presents a search framework for incorporating repair knowledge and investigates the continuum of repair informedness by empirically contrasting different repair strategies on randomly generated rescheduling problems.

The fundamental questions addressed by our experiments are:

*Recom Software
†Lockheed Space Operations Company

- Is it faster to perform many weak, but cheap to compute repairs as opposed to fewer, more computationally intensive, smart repairs.
- Can a knowledge intensive repair strategy scale to a real-world problem?

Our results bear evidence that knowledgeable repairs do converge overwhelmingly faster than weaker repairs. Additionally, experiments with knowledgeable repairs on the Space Shuttle ground processing domain indicate that the informed strategy can scale to complex, real-world problems.

The specific iterative repair method we use is constraint-based simulated annealing. Simulated annealing [Kir83] enables a hill-climbing search technique to escape local minima. The technique has been applied to design problems, scheduling problems, and traditional combinatoric problems [Kir83, Joh90a, Joh90b, Ott89]. Annealing begins with a rough (but complete) solution to a problem and then iteratively modifies the solution until it is of acceptable quality. It escapes local minima by conservatively considering poor solutions that eventually lead to more desirable schedules.

The main contribution of our work is the constraint-based representation that enables one to exploit knowledge within the simulated annealing framework. This modular and extensible representation scheme results in a more informed search when compared to the stochastic methods generally utilized by simulated annealing systems. The organization of this paper is as follows. First we formulate rescheduling as constraint satisfaction and then present the constraint-based simulated annealing algorithm. We then report our empirical results and finally attempt to portray our work in the context of previous work.

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

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

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 2 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.

As originally described in [Ow,88], 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.

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 consistent with its preemption work calendar,
2. the aggregate duration of subtasks sums to their parent 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

3.1 Constraints

Constraints are defined by the functions depicted in Figure 3.1.4. 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 3.1.4). 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 [Wal75, Dav87]. In constraint satisfaction terminology, this algorithm enforces arc-consistency [Mac77, Fre82]. 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. Section 5.3.2 discusses the effects of this stochastic behavior.

The complexity of this repair is dependent on the data structure representing resource availability. The aggregate usage of a resource pool is represented as a history [Wil86] or time line implemented as lists of tuples. The first element of the tuple represents a time interval, the second represents a value, and the third is the set of tasks using the resource. For example,

```
( ([0 100] 16 nil)
  ([100 200] 10 (T1 T2))
  ([200 :pos-infinity] 16 nil))
```

indicates that tasks T1 and T2 use six units of the resource from time 100 to 200 and no other utilizations exist. Using this data structure, the worst-case complexity of the resource capacity repair is $O(T^2)$ where T is the number of tasks. This is the case where all tasks use the same resource and every task must loop down the entire history to score the severity of bottleneck criterion.

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

- 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.

temporal constraints. In the future, we plan on extending this repair with options resembling the modal truth criterion of non-linear planners [Cha87]. 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. Since state variables are also represented as histories, searching for the next time with the correct state is of complexity $O(T)$, where T is the set of tasks that change the state variable.

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. 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

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

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. 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 = S';
             Else { With probability Escape do
                          Old = NewC; S = S';
                  };
          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.

3.2.1 Noise: Escaping Local Minima

In the algorithm presented above, we accept "worse" solutions with the Escape probability function specified in Figure 3.2.1. This permits the algorithm to follow paths that are undesirable with respect to the cost function. These paths are later repaired and usually result in much improved solutions. The Escape temperature[3] parameter (T) controls the likelihood that poor solutions will be accepted; higher temperatures are more aggressive. For example, consider the case where the temperature is high (T=100) and the new solution is similar to the current solution (i.e., their costs differ by 5). In this case, the probability that the new solution will be accepted is .95123. However, if we lower the temperature to 5, the probability is only .36787. In addition to being sensitive to the temperature parameter, the escape function is also sensitive to the degradation in solution quality. Even if the temperature were aggressively set to 100, when the cost of new solution differs by 100, the probability of acceptance would only be .36787.

The algorithm begins with a high temperature and is reduced according to a "cooling" schedule. As a result, the algorithm initially jumps around the search space but then makes more careful repairs. Currently, we begin with a temperature of 100 and reduce it after

[3]The name of this parameter is reminiscent of the algorithm's physical chemistry origin.

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

  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.

several iterations to 75. When the cost is low, we then reduce the temperature to 25. After many iterations, the temperature is further reduced to 10.

As stated previously, we introduce noise in the search process to escape local minima. One explanation for this is that the cost function does not accurately reflect how "close" a candidate solution is to the actual solution; it is only a measure of the flaws in a candidate solution. For example, a logical assumption is that if only one availability constraint is violated, then the algorithm is quite close to a solution, however this is misleading. It may require over 20 repairs to achieve the overall goal of zero cost, because 20 tasks must be reassigned. In fact, all paths that lead to a solution may need to transition through a state of higher cost.

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 [Dea88] to be classified as an anytime algorithm. Their criteria classifies anytime algorithms as those that:

1. can be interrupted and restarted
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 Experiments

Our experiments show that constraint-based simulated annealing is practically useful on large scale, complicated problems, and that it converges to acceptable schedules faster than the weaker repair techniques. We show that constraint penalties focus the search on the weak areas of the schedule, and that the heuristic knowledge embodied in constraint repair functions accelerate convergence.

5.1 NASA Application - Space Shuttle Processing

To prepare the Space Shuttle for launch, a plethora of inspection, repair, and installation activities take place around the clock, for about 60 days, by a large team technicians, engineers, and supervisors at the Kennedy Space Center (KSC). In cooperation with the Lockheed Space Operations Company, we are investigating the use of our rescheduling system to help coordinate this process. We have modeled the Space Shuttle processing environment with about 500 activities that are split into a approximately 4000 subtasks. There are 900 temporal constraints, 3600 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 aganist cumulative time. These graphs indicate that the algorithm scales to very large problems and maintains its anytime characteristics.

5.2 Artificial Problem Generation

In order to contrast constraint-based annealing with weaker repair methods, we have randomly generated a set of 25 rescheduling problems. Randomly generated problems ensure that our techniques transfer to problems that differ from the application discussed above. For each, problem we generate the following information:

1. the total number of tasks
2. the work-duration for each task
3. the number of resource classes
4. the number of instances per resource class
5. the capacity for each resource instance
6. the number of resource requests for each task
7. the required quantity for each resource request

We generate a set of tasks and resources with respect to this data and also probabilistically generate a set of temporal constraints between activities.

To facilitate experimentation, we made many simplifying assumptions when generating problems. All activities use a contiguous calendar (no holidays, no work shifts). The only temporal constraints generated assert that the end of a task must be less than or equal to another task's start. There are no state requirements or effects in the generated problems and each problem has a fixed start time and a milestone due date at its end.

Given a scheduling problem, we search for a solution using a systematic backtracking search method [Zwe89, Esk90]. Given this schedule, we generate a rescheduling problem by moving an arbitrary set of tasks by a random amount and direction.

5.3 Rescheduling Strategies

We hypothesized that constraint-based simulated annealing would converge faster to acceptable solutions when compared to weaker repair strategies because of two major reasons:

1. Constraints focus the search on the weaker areas of the schedule because they provide a type of local blame assignment. In contrast, traditional annealing applications and many other hill-climbing systems rely strictly on the global cost function to judge solutions.
2. Constraint repairs strongly bias the search towards promising solutions. Weaker repair methods rely on probability functions that do not offer much heuristic power.

To confirm our hypotheses, we compared three different repair strategies. The first is Constraint-Based Simulated Annealing (CSA) as described above. The other techniques are more stochastic in nature and less knowledge-intensive. The second strategy - probabilistic constraint repair (PCR), uses constraints to localize repairs but performs them in a strict probabilistic manner. The final strategy - random (R), is completely stochastic, in that it uses constraints only in the calculation of the cost function.

The PCR strategy modifies schedules by repairing violated constraints. For each constraint drawn from a subset of violated constraints, it probabilistically decides to substitute

an alternative resource or to move the task involved in the violation. When moving a task, it probabilistically generates the location of the move. Whenever PCR moves a task, it exploits the MOVE operator to preserve temporal constraints.

The R strategy probabilistically decides to move a task or to swap a resource pool without reference to the violated constraints. The selection of which task to move (and the location of the move) or the selection of the resource to substitute is also probabilistically driven. The R strategy also exploits the MOVE operator for shifting tasks.

We tested each of the above strategies on 25 problems that were generated in the manner described in the previous section. Since repair functions are probabilistic, we calculate average results over repeated trials.[4] The cost threshold was zero and the time bound was 15 minutes. In the next section we analyze the results of these experiments.

5.3.1 Empirical Results

Figure 6. presents the average best cost of each strategy over time. For a more meaningful presentation, we have normalized cost so that each curve indicates the time it takes to converge towards zero from a cost of one for every generated problem. Clearly, the constraint-based simulated annealing techniques converge toward acceptable solutions faster than the other techniques. Some of the generated test problems were overconstrained and did not have a zero cost solution. For these problems, all three techniques used the maximum amount of time. However, the CSA method had superior schedules upon termination. PCR and R could not reach zero violations within the time bound in many cases where CSA could. As expected, the use of constraints in the PCR method helped to focus the search but did not perform as well as the more informed CSA method.

One significant disadvantage of constraint-based simulated annealing is that it is incomplete. This implies that when the algorithm reaches its termination bound, there is no way to ascertain whether a solution actually exists or whether the search would find the solution if there were more time. Further, if there is no solution, the system will always continue repairing until it reaches its iteration bound (or is interrupted). Again, we believe that the average case behavior of the technique is worthy of sacrificing this completeness.

5.3.2 Effects of Randomness

CSA uses heuristic repair knowledge to bias probabilistic choices. We have observed that this stochastic behavior can result in mixed performance. Figure 7. demonstrates this behavior as an envelope of convergence. Here we show the average convergence time between the best and the worst trials. This indicates that the technique is sensitive to the probabilistic

[4]The constraint-based simulated annealing experiments were repeated 20 times and the other strategies were repeated 10 times. The convergence characteristics of PCR and R did not vary much between repetitions and therefore were repeated fewer times. We observed that R and PCR were more sensitive to the temperature parameter than CSA and therefore, with the hope of giving these techniques a boost, we optimized the cooling strategy for these strategies to the best one we could develop empirically. The cooling strategy for CSA was not changed.

decisions for a particular run, but reflects attractive average case behavior. In fact, when the system is having great difficulty converging, it is feasible that simply restarting the problem would resolve its difficulty. We plan to experiment with this "forget and retry" strategy in the future.

5.3.3 Discussion

The convergence properties of constraint-based simulated annealing depend upon the probability that a conflict will occur after modifying a schedule. This probability has four main components: 1) the probability that an activity will be concurrent with other activities; 2) the probability that concurrent activities will compete for resources; 3) the tightness of resource availability; and 4) the probability that the parallel activities will require contradictory states. Unfortunately, our empirical studies have not yet provided ample data to draw conclusions about the exact effect of these characteristics upon rescheduling. We expect that future experimentation will reveal these relationships and that they will help assess the difficulty of a problem before rescheduling. If one were capable of roughly estimating the difficulty of the problem, the cumulative time bound could be selected more wisely. We are exploring metrics that measure the connectedness of the temporal constraint graph, the amount of overlap between the types of resources that tasks request, the fraction of total resource capacity that tasks request on average, and finally the number of activities vieing for state conditions. We believe that a combination of these measures will be a useful estimator of search difficulty.

6 Related Work

Our work was heavily influenced by the criteria put forth in [Ow,88] and by the functionality of the ISIS,OPIS, and Cortes schedulers [Fox83, Fox84, Sad89]. These systems all exploit constraint representations but do so within a systematic search framework. In addition to beam search, OPIS also employs rescheduling repair strategies that resemble dispatch heuristics. Like ISIS, OPIS, and Cortes, we use constraint-based representations to focus search but we use our constraints within the basic simulated annealing framework described in [Joh90a, Joh90b].

Repair-based frameworks date back to the "fixes" used in the Hacker planning system [Sus73] and are still used in current systems such as GEMPLAN [Lan88]. GEMPLAN exploits domain locality to improve a systematic search strategy through the repair space (as opposed to our iterative improvement strategy).

Our experiments corroborate with a parallel study of a repair-based technique called MIN-CONFLICTS [Min90]. For any violated constraint, the MIN-CONFLICTS heuristic chooses the repair that minimizes the number of remaining conflicts. However, it is important to note that MIN-CONFLICTS may be too expensive to use in a preemptive scheduling domain where the granularity of time is small. This is because of the overhead of re-splitting a task whenever a new start time is considered multiplied by the large number of possible times.

For example, consider a violated resource capacity constraint that has only one pool to assign (i.e., substitution is impossible). For each task contributing to the violation the system must:

1. Search for the next available time given a resource availability history.[5]
2. Move the task to that time.
3. Re-split the task, inheriting resource and state constraints to the split tasks when specified.
4. Repeat for all tasks moved by the Waltz algorithm's enforcement of temporal constraints.
5. Test the number of conflicts.

Rough experiments have indicated that pure MIN-CONFLICTS is much slower than our more local heuristics when tried on the Space Shuttle problems.

Our work is also related to FORBIN [Mil88] in that deadline and resource requirements are addressed, but our representations and search techniques differ greatly. FORBIN represents time-changing information as propositions maintained by the TMM - Time Map Manager [Dea85] and use traditional graph search algorithms to maintain consistency among these propositions. Tate et. al. also address planning and scheduling with deadline and resource requirements in O-Plan [Bel85]. They use a blackboard framework to systematically search through a space of repairs for outstanding plan flaws.

We also have similar goals as Drummond and Bresina [Dru90]. They are developing an *anytime* agent architecture based upon beam search that explicitly represents uncertainty and disjunctive schedules. Their algorithm is anytime with respect to the certainty of goal achievement. With more time, the system *robustifies* its behavior by developing contingency plans for the likely deviations.

Finally, our technique is also closely related to the Jet Propulsion Laboratory's OMP scheduling system [Bie89, Bie91]. OMP uses procedurally encoded patches in an iterative improvement framework. It stores small snapshots of the scheduling process (called *chronologies*) which allow it to escape cycles and local minima. OMP addresses the flexible preemptive scheduling problem.

7 Conclusions and Future Work

Our experiments suggest that our constraint framework and the knowledge encoded in this framework is an extremely effective search aid. The framework is modular and extensible in that one can declare new constraints as long as their weight, penalty, and repair functions are provided. We look forward to a more extensive analysis of the sensitivity of our search method to the parameters of the algorithm and to the characteristics of the given problems.

[5] A pure interpretation of MIN-CONFLICTS would consider every time point.

We are engaging in a rigorous application effort hopefully resulting in daily use of our system in support of Space Shuttle processing. We also expect to extend our previous research in machine learning and scheduling [Esk90] to augment iterative improvement search methods. We are considering techniques for learning the general conditions under which "worse" solutions should be pursued instead of solely relying upon a probabilistic approach. We are also investigating the automatic tuning of the parameters of our algorithm as well as learning temporal abstractions. Finally, we would like to empirically compare and contrast CSA to other rescheduling systems in a larger set of experiments.

Acknowledgements

Thanks to Brian Daun, Todd Stock, and Ellen Drascher for all their contributions. We thank Cindy Mollakarimi, Danielle Schnitzius, and Mark Yvanovich for all their help at KSC. Special thanks to Eric Clanton, Flow Manager of the orbiter Endeavour, and Wayne Bingham, Vehicle Operations Chief of the orbiter Columbia, for their patience and help in our Space Shuttle application effort.

References

[Bel85] Bell, C.,Currie, K., and Tate, A. Time Window and Resource Usage in O-Plan. Technical report, AIAI, Edinburgh University, 1985.

[Bie89] Biefeld, E. and Cooper, L. Scheduling with Chronology-Directed Search. In *Proceedings of the AIAA Computers in Aerospace VII*, Monterey, CA, 1989.

[Bie91] Biefeld, E. and Cooper, L. Bottleneck Identification Using Process Chronologies. In *Proceedings of IJCAI-91*, Sydney, Austrailia, 1991.

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

[Dav87] Davis, E. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32(3), 1987.

[Dea85] Dean, T. *Temporal Imagery: An Approach to Reasoning about Time for Planning and Problem Solving.* PhD thesis, Yale University, January 1985.

[Dea88] Dean, T., and Boddy, M. An Analysis of Time-Dependent Planning. In *Proceedings of AAAI-88*, 1988.

[Dru90] Drummond, M. and Bresina J. An Anytime Temporal Projection Algorithm for Maximizing Expected Situation Coverage . In *Proceedings of AAAI-90*, 1990.

[Esk90] Eskey, M. and Zweben, M. Learning Search Control for a Constraint-Based Scheduling System. In *Proceedings of AAAI-90*, Boston, MA, 1990.

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

[Fox84] Fox, M. and Smith, S. A Knowledge Based System for Factory Scheduling. *Expert System*, 1(1), 1984.

[Fre82] Freuder, E. C. A Sufficient Condition for Backtrack-Free Search. *J. ACM*, 29(1), 1982.

[Joh90a] Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C. Optimization By Simulated Annealing:An Experimental Evaluation, Part I (Graph Partioning). *Operations Research*, 1990.

[Joh90b] Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C. Optimization By Simulated Annealing:An Experimental Evaluation, Part II (Graph Coloring and Number Partioning). *Operations Research*, 1990.

[Kir83] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. Optimization by Simulated Annealing. *Science*, 220(4598), 1983.

[Lan88] Lansky, A. Localized Event-based Reasoning for Multiagent Domains. *Computational Intelligence*, 4(4), 1988.

[Mac77] Mackworth, A.K. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1), 1977.

[Mil88] Miller, D., Firby, R. J., Dean, T. Deadlines, Travel Time, and Robot Problem Solving. In *Proceedings of AAAI-88*, St. Paul, Minnesota, 1988.

[Min90] Minton, S., Phillips, A., Johnston, M., Laird., P. Solving Large Scale CSP and Scheduling Problems with a Heuristic Repair Method. In *Proceedings of AAAI-90*, 1990.

[Ott89] Otten, R.H J.M., van Ginneken, L.P.P.P . *The Annealing Algorithm.* Kluwer Academic, Boston, MA, 1989.

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

[Sad89] Sadeh, N. and Fox, M. S. Preference Propagation in Temporal/Capacity Constraint Graphs. Technical report, The Robotics Institute, Carnegie Mellon University, 1989.

[Sus73] Sussman, G.J. *A Computational Model of Skill Acquisition* . PhD thesis, AI Laboratory, MIT, 1973.

[Wal75] Waltz, D. Understanding Line Drawings of Scenes with Shadows. In P. Winston, editor, *The Psychology of Computer Vision.* McGraw-Hill, 1975.

[Wil86] Williams, B.C. Doing Time: Putting Qualitative Reasoning on Firmer Ground. In *Proceedings of AAAI-86*, 1986.

[Zwe89] Zweben, M. and Eskey, M. Constraint Satisfaction with Delayed Evaluation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989.

[Zwe90] Zweben, M., Deale, M., Gargan, M. Anytime Rescheduling. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning and Scheduling*, 1990.

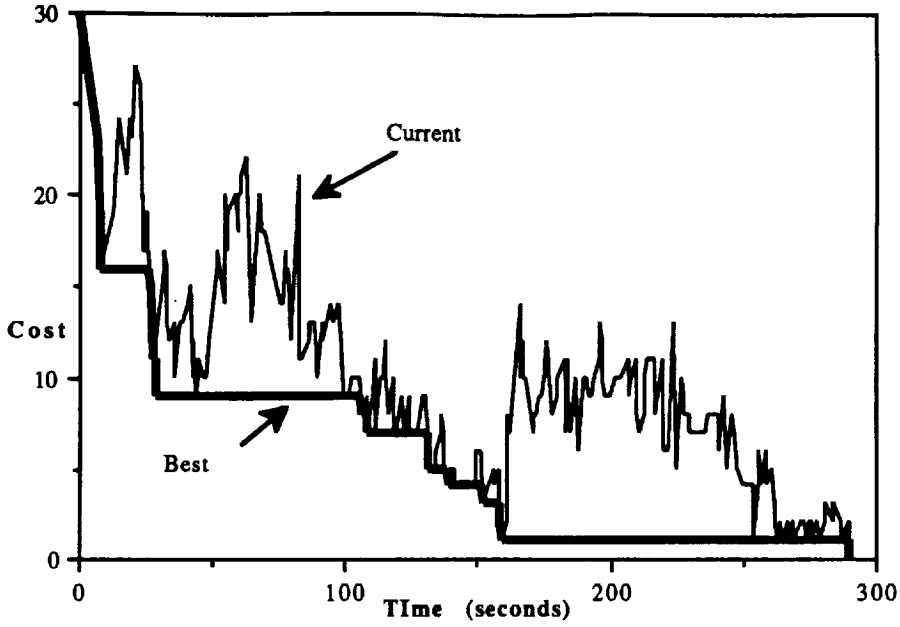

Figure 4: Best Cost and Current Cost

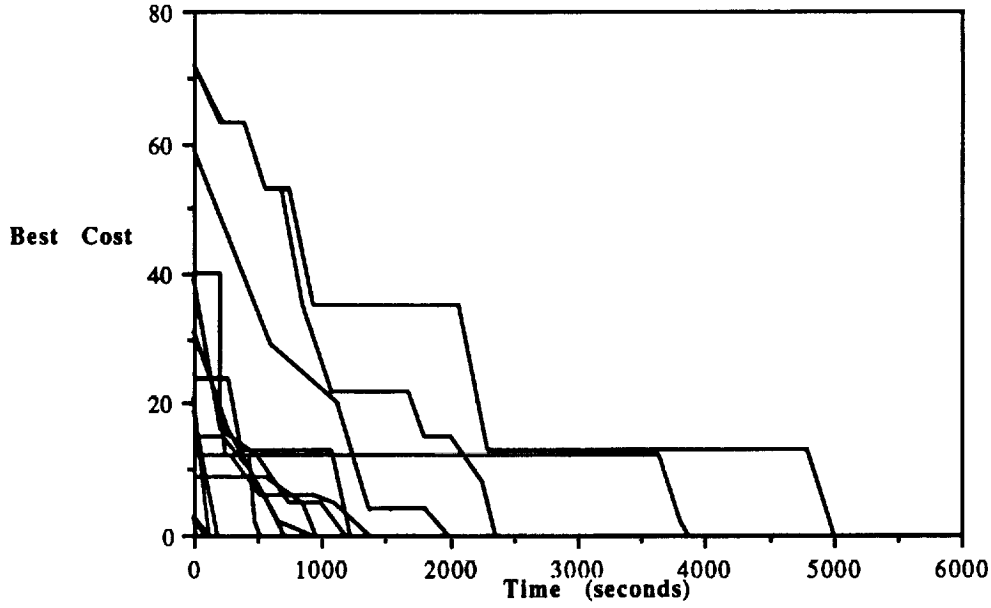

Figure 5: Space Shuttle Rescheduling Problems

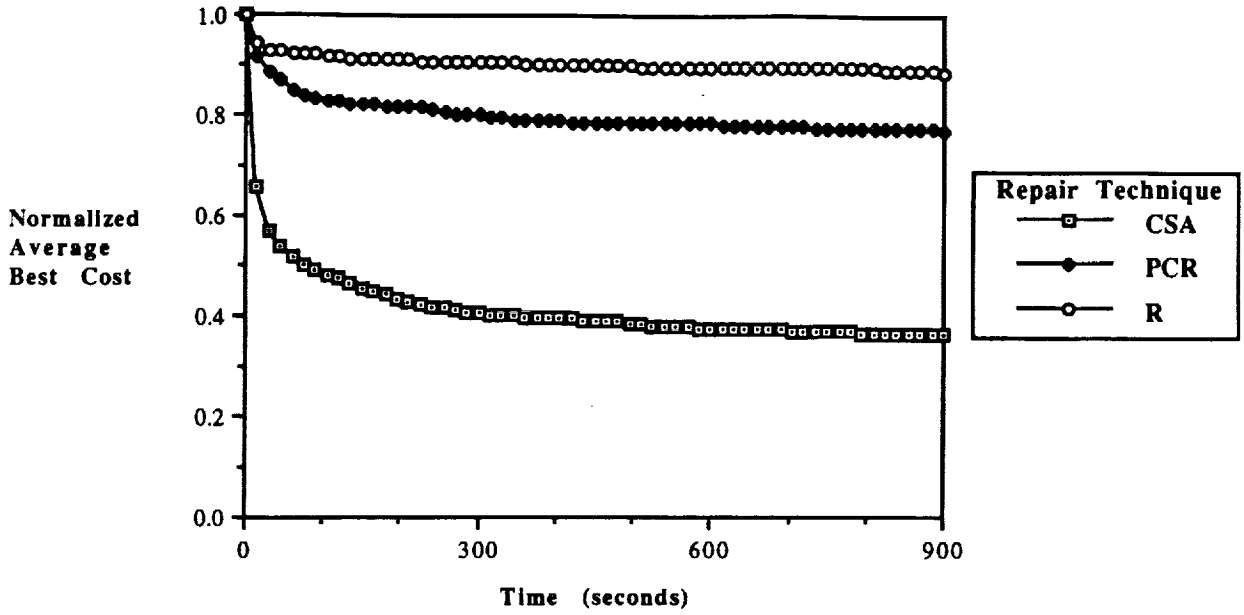

Figure 6: CSA vs. PCR vs. R

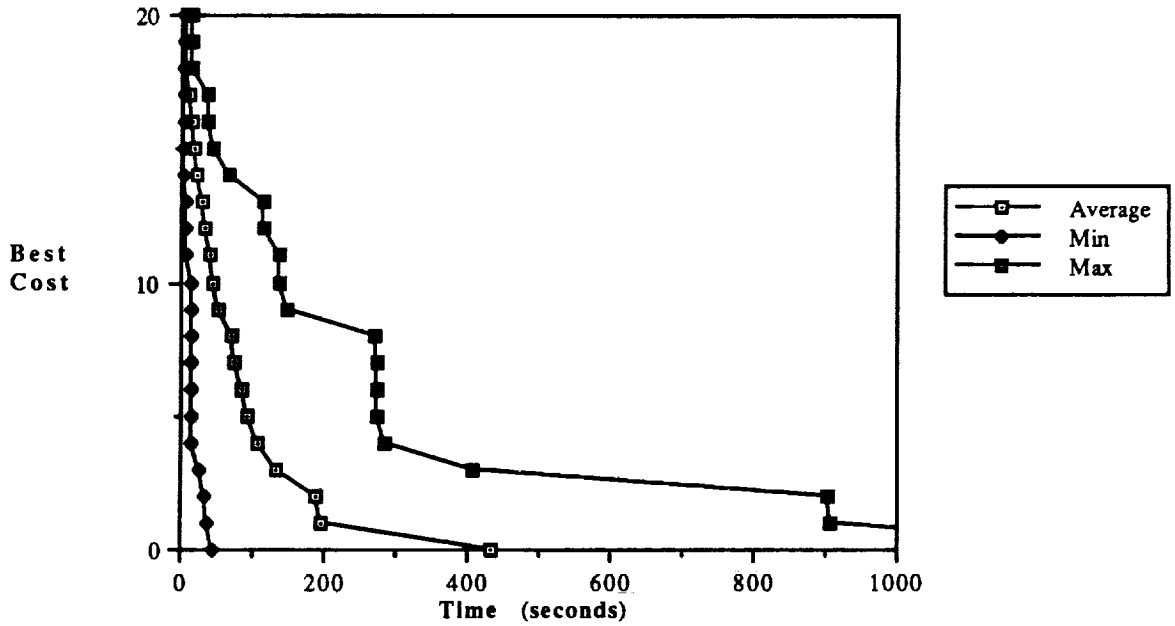

Figure 7: The Effects of Randomness

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Dates attached	

4. TITLE AND SUBTITLE

Titles/Authors - Attached

5. FUNDING NUMBERS

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Code FIA - Artificial Intelligence Research Branch
Information Sciences Division

8. PERFORMING ORGANIZATION REPORT NUMBER

Attached

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Nasa/Ames Research Center

Moffett Field, CA. 94035-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Available for Public Distribution

Peter Friedland 5/14/92 BRANCH CHIEF

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstracts ATTACHED

14. SUBJECT TERMS			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT