Rescheduling with Iterative Repair

Monte Zweben
Eugene Davis
Brian Daun
Michael Deale
AI Research Branch, Mail Stop 269-2
NASA Ames Research Center
Moffett Field, CA 94025, USA

NASA Ames Research Center
Artificial Intelligence Research Branch

Technical Report FIA-92-15

April, 1992

Rescheduling with Iterative Repair

Monte Zweben
Eugene Davis*
Brian Daun†
Michael Deale‡
NASA Ames Research Center
M.S. 269-2
Moffett Field, California 94035
zweben@pluto.arc.nasa.gov

Abstract

This paper presents a new approach to rescheduling called *constraint-based iterative repair*. This approach gives our system the ability to satisfy domain constraints, address optimization concerns, minimize perturbation to the original schedule, produce modified schedules quickly, and exhibits "anytime" behavior. The system begins with an initial, flawed schedule and then iteratively repairs constraint violations until a conflict-free schedule is produced. In an empirical demonstration, we vary the importance of minimizing perturbation and report how fast the system is able to resolve conflicts in a given time bound. We also show the anytime characteristics of the system. These experiments were performed within the domain of Space Shuttle ground processing.

Function: Planning and Scheduling
Domain: Space Shuttle Ground Processing
Knowledge: Constraints

*Recom Technologies
†Lockheed Artificial Intelligence Center
‡Lockheed Space Operations Company

Introduction

Space Shuttle ground processing encompasses the inspection, repair, and refurbishment of space shuttles in preparation for launch. During processing the Kennedy Space Center (KSC) flow management team frequently modifies the schedule in order to accommodate unanticipated events, such as lack of personnel availability, unexpected delays, and the need to repair newly discovered problems. If the Space Shuttle ground processing turnaround time could be shortened, even by a small percentage, millions of dollars would be saved. This paper presents GERRY, a general scheduling system being applied to the Space Shuttle ground processing problem.

As originally put forth in [Smi85] and [Zwe90], rescheduling systems should satisfy domain constraints, address optimization concerns, minimize perturbation to the original schedule, produce modified schedules quickly, and exhibit "anytime" behavior[1].

GERRY is a novel approach to rescheduling that addresses these five concerns and gives the user the ability to individually modify each criteria's relative importance. In an empirical demonstration of the system, we vary the importance of minimizing perturbation and report how fast the system is able to converge to a conflict-free schedule (or a near-conflict-free schedule) in a given time bound. We also show the anytime characteristics of the GERRY system.

Problem Class: Fixed Preemptive Scheduling

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

[1] An *Anytime* algorithm [Dea85] can be interrupted at any point during its computation and asked to output a usable solution. Ideally, the algorithm should be restartable where solution quality improves in a well-behaved manner over time.

The Space Shuttle domain also requires the modeling of state variables. State variables are conditions that can change over time; examples include the positions of switches, the configuration of mechanical parts, and the status of systems. Tasks might be constrained by the state conditions (a *state requirement*) and they might cause a change in state condition (a *state effect*). A state requirement asserts that a state variable must have a certain value during a task's scheduled time. For example, in the Space Shuttle domain, certain tasks can not be performed unless the payload bay doors are in designated positions. State effects are changes that tasks impose upon the state variables: when a task has the effect of opening the payload bay doors, the state variable models the doors as "open" until some other task closes them.

Preemption is an additional complicating factor introduced by the Space Shuttle problem. In preemptive scheduling, each task is associated with a calendar of legal work periods that determine when the task must be performed. For example, suppose a task has a duration of 16 hours and a calendar indicating that only the first shift of each non-weekend day is legal. Given that the first shift of the day extends from 8:00 am to 4:00 pm, if the task is started on Monday at 8:00 am, then it will be suspended at the end of the shift (at 4:00 pm). It would restart on Tuesday at 8:00 am and would complete the same day at 4:00 pm. If the task had been started on Friday, however, it would not complete until the following Monday at 4:00 pm.

Preemption effectively splits a task into a set of subtasks. Resource constraints are annotated as to whether they should be enforced for each individual subtask (and not during the suspended periods between subtasks) or during the entire time spanning from the first subtask until the last (including suspended periods). Labor is a resource type that is not typically required during the suspended periods; in contrast, heavy machinery is difficult to relocate and thus may remain allocated during the suspended periods.

Preemptive scheduling requires additional computational overhead since for each task the preemption times must be computed and appropriate constraint manipulation for each time assignment must be performed.

In summary, the input to a scheduling problem is a set of tasks, each with a work duration, a work calendar, a set of temporal constraints, a set of resource requirements, and a set of state requirements and effects. A solution to the problem is a decomposition of each task into its preempted subtasks, where each subtask is assigned a start and end time and a resource pool for

each resource request. A solution is satisfied if the subtasks of each task are consistent with its preemption work calendar and all temporal, resource, and state constraints are satisfied.

Rescheduling

Rescheduling is necessitated by changes that occur in the environment. Systems can respond in three ways: schedule again from scratch, remove some tasks from the schedule and restart from an intermediate state, or repair the schedule where the changes occurred.

Scheduling from scratch reconsiders the scheduling problem in light of exogenous events. In [Ham86], [Sim88] and [Kam90], the authors argue that it is more efficient to modify flawed plans than to plan from scratch. Moreover, since scheduling from scratch will generate a new schedule without considering any values from the previous solution, a high amount of perturbation is likely to occur.

To schedule from an intermediate state, all tasks affected by the exogenous events are first removed from the schedule; scheduling then is resumed considering the exogenous events. For example, suppose T_1, T_2, T_3, and T_4 are tasks in a schedule that are constrained to be sequential in the order shown. If T_3 is delayed, then only T_3 and T_4 would be removed from the schedule before restarting, because the other tasks are unaffected by the delay. This approach is complex, because a dependency analysis is required to determine whether a schedule modification could affect any particular task. Further, even though a task is unaffected by an exogenous event, it may be possible to provide a better schedule by reconsidering its assignments. For example, placing an unaffected task much later in the schedule might cause little perturbation and allow many tasks (which are affected by the exogenous events) to fit in its place. Unfortunately, this opportunity would be missed if the unaffected tasks are not considered in the rescheduling process.

GERRY adopts the third approach, which is to repair the constraints that are violated in the schedule.

Constraint-Based Iterative Repair

Constraint-based iterative repair begins with a complete schedule of unacceptable quality and iteratively modifies it until its quality is found satisfac-

tory. The quality of a schedule is measured by the *cost* function: $Cost(s) = \sum_{c_i \in Constraints} Penalty_{c_i}(s) * Weight_{c_i}$, which is a weighted sum of constraint violations. The *penalty* function of a constraint returns an integer reflecting its degree of violation. The *weight* function of a constraint returns an integer representing the importance or utility of a constraint.

In GERRY, repairs are associated with constraints. Local repair heuristics that are likely to satisfy the violated constraint can then be encoded without concern for how these repairs would interact with other constraints. Of course local repairs do occasionally yield globally undesirable states, but these states, if accepted (see below), are generally improved upon after multiple iterations.

Repairing any violation generally involves moving a set of tasks to different times: at least one task participating in the constraint violation is moved, along with any other tasks whose temporal constraints would be violated by the move. In other words, all temporal constraints are preserved after the repair. We use the Waltz constraint propagation algorithm over time intervals [Wal75, Dav87] to carry this out (thus enforcing a form of arc-consistency [Mac77, Fre82]). The algorithm recursively enforces temporal constraints until there are no outstanding temporal violations. This scheme can be computationally expensive, since moving tasks involves checking resource constraints, calculating preemption intervals, etc.[2]

At the end of each iteration, the system re-evaluates the cost function to determine whether the new schedule resulting from the repairs is better than the current solution. If the new schedule is an improvement, it becomes the current schedule for the next iteration; if it is also better than any previous solution, it is stored as the best solution so far. If it is not an improvement, with some probability it is either accepted anyway, or it is rejected and the changes are not kept. When the changes are not kept, it is hoped that repairs in the next iteration will select a different set of tasks to move and the cost function will improve.

The system sometimes accepts a new solution that is worse than the current solution in order to escape local minima and cycles. This stochastic technique is referred to as simulated annealing [Kir83]. The escape function for accepting inferior solutions is: $Escape(s, s', T) = e^{-|Cost(s)-Cost(s')|/T}$

[2]Note that all temporal constraints are also preserved (using the same Waltz algorithm) whenever the user manually moves tasks.

where T is a "temperature" parameter that is gradually reduced during the search process. When a random number between 0 and 1 exceeds the value of the escape function, the system accepts the worse solution. Note that escape becomes less probable as the temperature is lowered.

In GERRY the types of constraints that can contribute to the cost function include the resource, state, and perturbation constraints.

Resource Constraints

The penalty of a resource capacity constraint is 1 if the resource is overallocated. If K simultaneous tasks overallocate the resource, then all K tasks are considered violated. One of these tasks will be selected in an attempt to repair as many of the K violations as possible. The heuristic used to select this task considers the following information:

Fitness: Move the task whose resource requirement most closely matches the amount of overallocation. A task using a significantly smaller amount is not likely to have a large enough impact on the current violation being repaired. A task using a far greater amount is more likely to be in violation wherever it is moved.

Temporal Dependents: Move the task with the fewest number of temporal dependents. A task with many dependents, if moved, is likely to cause temporal constraint violations and result in many task moves.

Distance of Move: Move the task that does not need to be shifted significantly from its current time. A task that is moved a greater distance is more likely to cause other tasks to move as well, increasing perturbation and potentially causing more constraint violations.

For each of the tasks contributing to the violation, the system considers moving the task to its *next earlier* and *next later* times such that the resource is available, rather than exploring many or all possible times. This reduces the computational complexity of the repair and, like the "distance to move" criterion above, tends to minimize perturbation.

Each candidate move is scored using a linear combination of the *fitness*, *temporal dependents*, and *distance to move* heuristic values. The repair then chooses the move stochastically with respect to the scores calculated. After

the repair is performed, the Waltz algorithm moves other tasks in order to preserve temporal constraints.

In summary, this repair strategy only considers two possible moves for a task participating in a violation: one earlier and one later. The evaluation criterion used to select a repair is based upon three computationally inexpensive heuristic criteria: degree of fitness, number of temporal dependents, and distance to move.

State Constraints

The penalty of a state constraint is 1 if the required state is not set. To repair a state constraint, the task with the violated state requirement is reassigned to a different time when the state variable takes on the desired value. Similar to the resource capacity constraints, the system considers only the next earlier and next later acceptable times and selects between these randomly. We are currently investigating improvements to this repair and expect to extract more useful heuristics from our experts. One effort underway is the development of a repair that can introduce new tasks into the schedule, thus yielding a behavior generally associated with AI planning systems.

Perturbation Constraint

The penalty function of the perturbation constraint returns the number of tasks that differ from their original temporal assignments. Since the weighted penalty of this constraint contributes to the cost of a solution, schedules with significant perturbation tend to be rejected at the close of an iteration. We are in the process of experimenting with repairs for this constraint that augment the information provided by its penalty and weight. Below we show how varying the weight of this constraint can affect convergence speed and solution quality.

To summarize, constraint-based iterative repair begins with a complete but flawed schedule and isolates the violated constraints. Tasks are moved according to the repairs embodied in the violated constraints. A new schedule is accepted if the new cost is lower than the previous cost, or if a random

number exceeds the value of the escape function; otherwise it is rejected and new repairs are attempted on the previous schedule. The process repeats until the cost of the solution is acceptable to the user, or until the user terminates the repair cycle. The system may also terminate itself if a prespecified number of iterations have been attempted or if a prespecified CPU time bound has been reached.

Experiments

The problem domain for the experiments consisted of the tasks, resources, temporal constraints, and resource constraints from the STS-43 Space Shuttle ground processing flow. A rescheduling problem was generated by taking the original conflict-free schedule and randomly moving ten tasks. Five such problems were generated for the results reported below. The first and last tasks of the original schedule were anchored in time so repairs could not extend the duration of the entire flow.

In the experiments, we maintained the resource constraint weight at ten, and varied the perturbation constraint weight from zero (perturbation was of no concern) to 50 (perturbation was extremely important). The system terminated its search when all resource constraints were satisfied or when its run time exceeded ten minutes. Upon termination, the system returned the best solution found. Each rescheduling run was performed with the same settings 20 times in order to minimize stochastic variance.

Figure 1 presents the results of our experiments on the five problems from three different perspectives. The first graph plots the number of perturbations for the returned solution against the weight of the perturbation constraint. As expected, with a higher perturbation weight, the best solution has fewer perturbations.

The second plot shows the quality of a returned solution (measured as the number of violated resource constraints), as a function of the perturbation weight. As the graph shows, GERRY has more difficulty satisfying resource constraints as perturbation becomes more important.

Finally, the third plot shows the convergence time (in cpu seconds) as a function of the perturbation weight. Average time to solution generally increased as the perturbation weight increased.

It is interesting to note that for smaller weights on the perturbation constraint (< 20), the increase in resource violations is small while the drop in

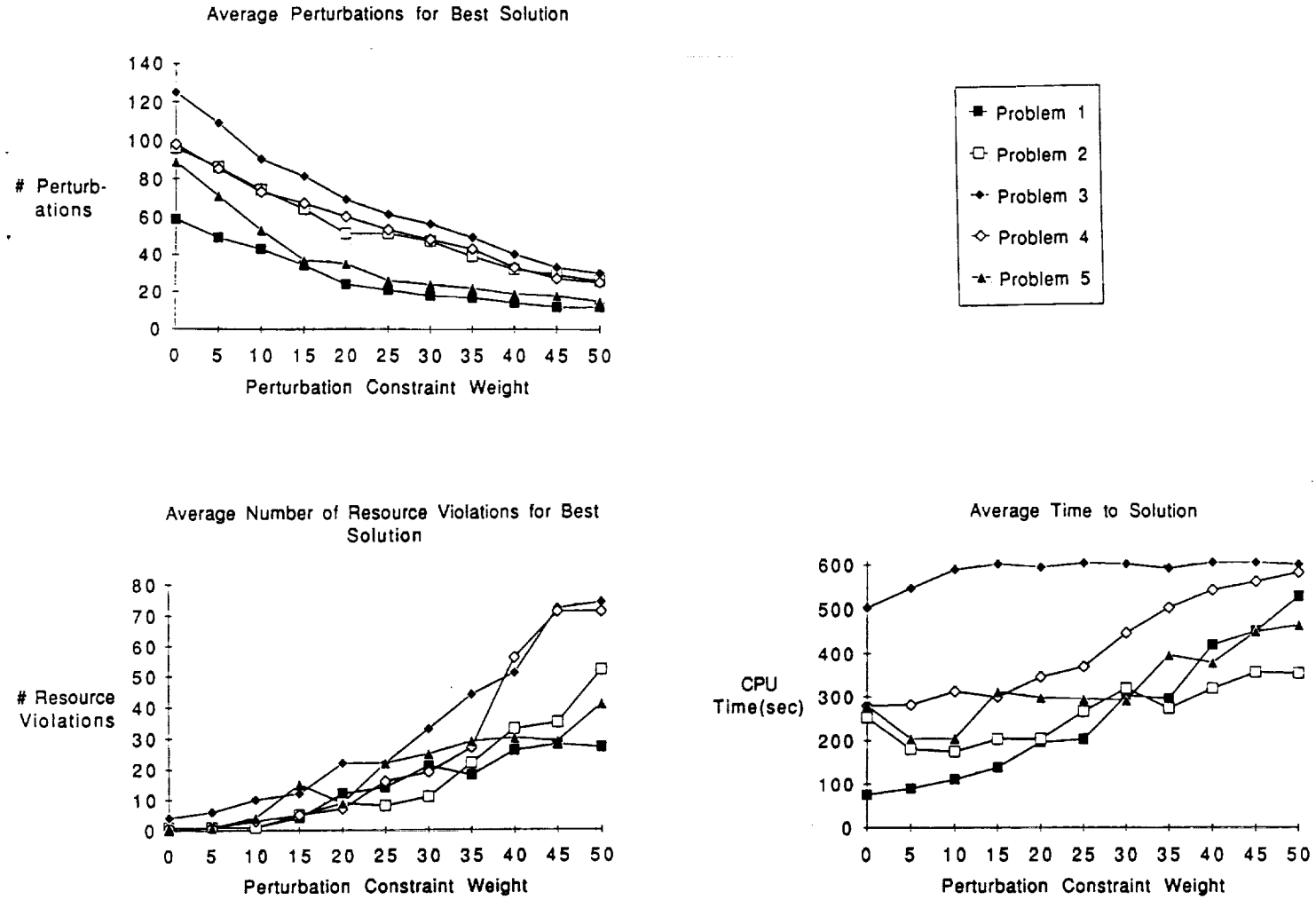


Figure 1: Experimental Results: number of perturbations versus perturbation constraint weight; number of resource violations versus perturbation constraint weight; average run time versus perturbation constraint weight

number of perturbations is fairly large. As the perturbation weight increases beyond 20, resource violations rise quickly, and the drop in perturbations slows.

Since there is no repair for the perturbation constraint, the system will terminate when there are no remaining resource capacity constraint violations. Given available time, a perturbation repair would be potentially useful in repairing a perturbation and attempt to find a lower cost solution.

Anytime Characteristics

As soon as flow managers arrive at work they investigate logs, visually inspect the shuttle, and attend brief status meetings to monitor the work performed during the previous evening. A few hours later, they present a 3-day detailed schedule to all relevant KSC personnel. If the flow managers use a tool to assist them in resolving scheduling conflicts and optimizing the flow, the tool must deliver usable schedules before they attend this meeting. The ability to be interrupted at any time and return the best solution is essential in this environment. Near-term violations that could not be resolved by the system in the permitted time may be resolvable by the flow managers because they can exploit conditions that are not modeled within the system. For example, since task locations are not modeled, tasks that are located near each other can share resources even though the system registers overallocations. Therefore we are optimistic that the system will reflect effective *anytime* behavior for the flow managers.

In summary, our algorithm is interruptible, restartable, and outputs a solution when terminated. As demonstrated in Figure 2, the solution quality increases as a step-function of time. These runs are representative of the system's general performance.

Related Work

Our work was heavily influenced by previous constraint-based scheduling [Fox87, Fox84, Sad89] and rescheduling efforts [Ow,88].

ISIS [Fox87] and GERRY both have metrics of constraint violation (the *penalty* function in GERRY) and constraint importance (the *weight* function in GERRY). In contrast with our repair-based method, ISIS uses an incremental, beam search through a space of partial schedules and reschedules by

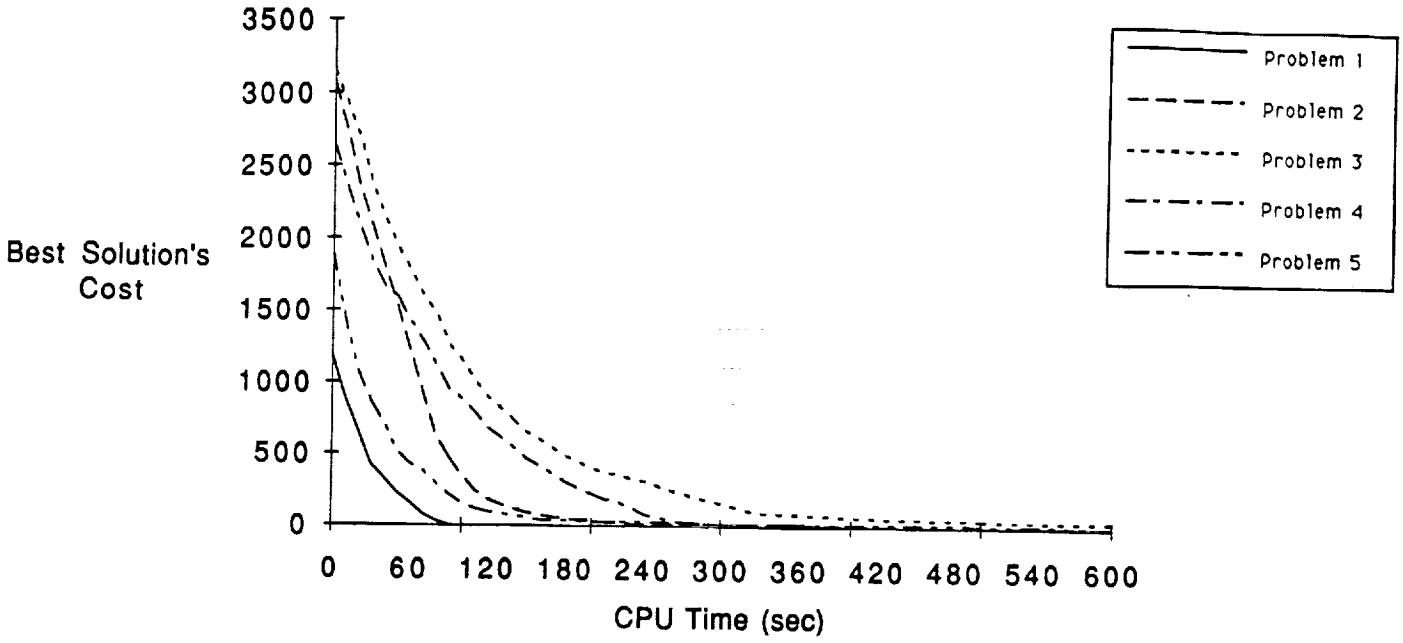


Figure 2: Anytime Characteristics: Best Cost versus Run Time

restarting the beam search from an intermediate state.

OPIS [Fox84, Ow,88], which is the successor of ISIS, opportunistically selects a rescheduling method. It chooses between the ISIS beam search, a resource-based *dispatch* method, or a repair-based approach. The *dispatch* method concentrates on a bottleneck resource and assigns tasks to it according to the dispatch rule. The *repair* method shifts tasks until they are conflict-free. These "greedy" assignments could yield globally poor schedules if used incorrectly. Consequently, OPIS only uses the dispatch rule when there is strong evidence of a bottleneck and only uses the repair method if the duration of the conflict is short. In contrast, GERRY uses the simulated annealing search to perform multiple iterations of repairs, possibly retracting "greedy" repairs when they yield prohibitive costs.

Our use of simulated annealing was influenced by the experiments performed in [Joh90a, Joh90b]. In contrast with our constraint-based repair, their repairs were generally uninformed. In [Zwe92b] we show that constraint repair knowledge improves convergence speed.

The repair-based scheduling methods considered here are related to the repair-based methods that have been previously used in AI planning systems such as the "fixes" used in Hacker [Sus73] and, more recently, the repair strategies used in the GORDIUS[Sim88] generate-test-debug system, in the PRIAR plan modification system [Kam90], and the CHEF cased-based plan-

ner [Ham86].

In [Min90], it is shown that the *min-conflicts* heuristic is an extremely powerful repair-based method. For any violated constraint, the *min-conflicts* heuristic chooses the repair that minimizes the number of remaining conflicts resulting from a one-step lookahead. However, in certain circumstances this lookahead could be computationally prohibitive. In [Zwe91], the authors investigate the tradeoff between the informedness of a repair and its computationally complexity. There it is shown that the resource repair described above outperformed a lookahead heuristic on the STS-43 Space Shuttle problem. However, on smaller problems the lookahead heuristic was superior.

Our technique is also closely related to the Jet Propulsion Laboratory's OMP scheduling system [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.

[Mil88], [Bel85], and [Dru90] describe other efforts that deal with resource and deadline constraints. [Dru90] and [Bo91] address related *anytime* issues.

Conclusions and Future Work

Our experiments suggest that our constraint framework and the knowledge encoded in this framework is an effective search tool that allows one to adjust the importance of schedule perturbation and other objective criteria. The framework is modular and extensible in that one can declare new constraints as long as their weight, penalty, and repair functions are provided. It also reflects favorable *anytime* behavior.

In future experiments, we hope to better characterize the components of repair informedness and computational complexity. We are currently evaluating candidate metrics of problem difficulty that could be used to guide the selection of repair heuristics. Additionally, we are developing machine learning techniques that allow systems to learn when to dynamically switch between heuristics [Zwe92a].

With respect to the Space Shuttle application, the system is expected to be in daily use sometime this year. Our most significant barrier is gathering accurate models of tasks in an electronic form. We also plan to develop constraints that minimize weekend labor.

References

[Bo91] Boddy, M. Anytime Problem-Solving Using Dynamic Programming. In *Proceedings of AAAI-91*, 1991.

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

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

[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.

[Dru90] Drummond, M. and Bresina J. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction . In *Proceedings of AAAI-90*, 1990.

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

[Fox87] Fox, M. *Constraint-Directed Search: A Case Study of Job Shop Scheduling.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

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

[Ham86] Hammond, K. J. CHEF: A Model of Case-Based Planning. In *Proceedings of AAAI-86*, 1986.

[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.

[Kam90] Kambhampati, S. A Theory of Plan Modification. In *Proceedings of AAAI-90*, 1990.

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

[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.

[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.

[Sim88] Simmons, R.G. Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems. Technical report, MIT Artificial Intelligence Laboratory, 1988.

[Smi85] Smith, S. and Ow, P. The Use of Multiple Problem Decompositions in Time Constrained Planning Tasks. In *IJCAI-85 Proceedings*, 1985.

[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.

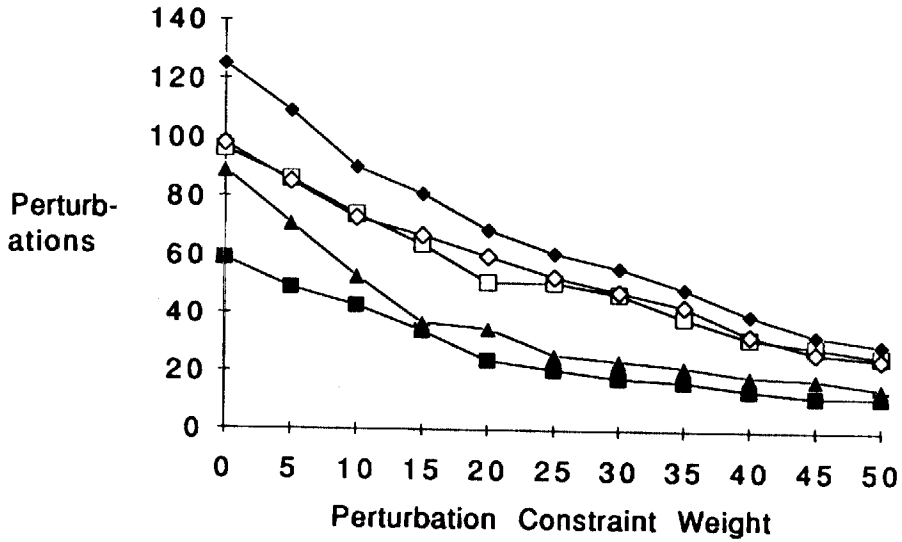
[Zwe90] Zweben, M., Deale, M., Gargan, R. Anytime Rescheduling. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning and Scheduling*, 1990.

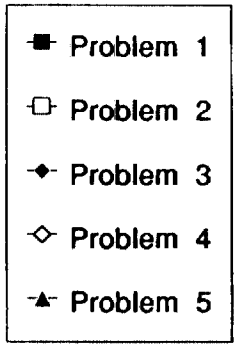
[Zwe91] Zweben, M.,Minton, S. Repair-Based Scheduling: Informedness versus Computational Cost. Submitted to *The First International Conference on AI Planning Systems*, 1991.

[Zwe92a] Zweben, M., Davis, E., Daun, B., Drascher, E., Deale, M., Eskey, M. Learning To Improve Constraint-Based Scheduling. *Artificial Intelligence*, To Appear, 1992.

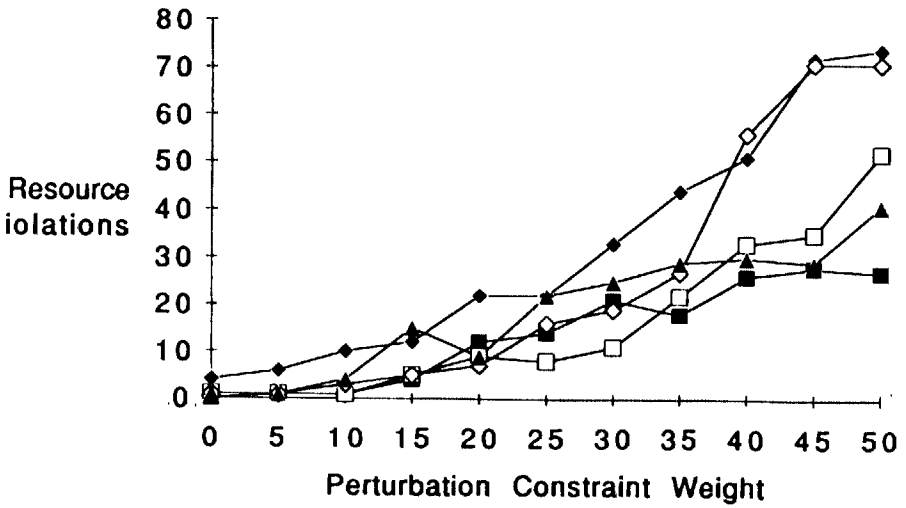
[Zwe92b] Zweben, M., Davis, E., Deale, M. Iterative Repair for Scheduling and Rescheduling. *IEEE Systems, Man, and Cybernetics*, To Appear, 1992.

Average Perturbations for Best Solution





Average Number of Resource Violations for Best Solution



Average Time to Solution