

Informedness vs. Computational Cost of Heuristics in Iterative Repair Scheduling

Monte Zweben * Eugene Davis † Brian Daun * Michael Deale *

NASA Ames Research Center

Artificial Intelligence Research Branch, M.S. 269-2

Moffett Field, CA 94035

U. S. A.

Abstract

This paper explores the tradeoff between the informedness and computational cost of two repair heuristics used in the GERRY scheduling system. GERRY performs rescheduling with *constraint-based iterative repair*, a technique that starts with a complete but possibly flawed schedule and iteratively improves it by using constraint knowledge within repairs. We show empirically how each repair technique performs on a set of random and real-world problems, and conjecture when less "informedness" will help speed convergence to solution.

1 Introduction

Space Shuttle ground processing encompasses the inspection, repair, and refurbishment of shuttle orbiters in preparation for launch. Each orbiter's complete preparation (called a *flow*) at the Kennedy Space Center requires constant attention from flow managers, whose job is to make certain that all tasks are accomplished in a safe and timely manner. Flow Management frequently revises the schedule in order to accommodate unanticipated events, such as the unavailability of specialized personnel, and the need to repair newly discovered problems.

GERRY is a scheduling system that has been used operationally in the shuttle ground processing domain since early 1992. GERRY uses *constraint-based iterative repair* [Zweben *et. al.*, 1990] when considering modifications to schedules. Iterative repair methods such as [Zweben *et. al.*, 1990; Minton *et. al.*, 1990; Biefeld and Cooper, 1991; Lin and Kernighan, 1973; Kurtzman and Aiken, 1989; Johnson *et. al.*, 1990; Morris, 1990] differ from *constructive* scheduling methods in that they begin with a complete, but possibly flawed, set of assignments and then iteratively modify or repair those assignments to improve the overall schedule. Constructive scheduling methods [Fox, 1987; Sadeh and Fox, 1989] incrementally extend valid, partial schedules until a complete schedule is synthesized or until backtracking is required.

*Red Pepper Software Company

†University of Toronto/Recom Technologies

In this paper we explore the tradeoff between the informedness and the computational cost of two repair heuristics in both scheduling and rescheduling problems. We show empirically that increased knowledge in repair selection can greatly improve the convergence speed of a repair-based system. However in some domains (such as that found in shuttle ground processing) a system may easily become overwhelmed by an overly informed heuristic.

2 Problem Class: Fixed Preemptive Scheduling

Scheduling is the process of assigning times and resources to the tasks of a plan, where all 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., $end(T1) \leq 11/23/93\ 12\ 00\ 00$. A resource requirement consists of a type and quantity of a resource; e.g., T1 needs 4 mechanical technicians and 3 cranes. Each resource requirement has a corresponding capacity constraint which states that the resource must not be over-allocated.

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 orbiter sub-systems. Tasks might be constrained by the state conditions (a *state requirement*) and they might cause a change in the state conditions (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 cannot be performed unless the payload bay doors are in designated positions. State effects are changes that tasks impose upon state variables such as the opening of the payload bay doors.

3 Constraint Repair with GERRY

A constraint is considered *violated* if the values assigned to its variables are inconsistent with the restrictions specified by the constraint. Generally we are interested

in *repairing* schedules that contain violated constraints.

GERRY's constraint-based iterative repair mechanism begins with a complete schedule of unacceptable quality and iteratively modifies it until its quality is found to be satisfactory. The quality of a schedule S 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 a non-negative number reflecting the degree to which the constraint is violated. The *weight* function of a constraint returns a non-negative number representing the importance or utility of a constraint.

GERRY performs repairs in each of its iterations by selecting a violated constraint (i.e. one whose product of weight and penalty is greater than zero), identifying candidate modifications to the schedule that might lower the violated constraint's penalty, and finally selecting and performing one of these modifications. This step may be performed several times per iteration.

Repairing any violation typically involves moving a task to a different time. If so, other tasks may be forced to move as well since GERRY always preserves all temporal constraints. We use the Waltz constraint propagation algorithm over time intervals [Waltz, 1975; Davis, 1987] to carry this out (thus enforcing a form of arc-consistency [Mackworth, 1977; Freuder, 1982]). If any temporal constraint violations cannot be resolved by the Waltz algorithm, the initial move that caused the violations is rejected.

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 cached as the best solution so far. If it is not an improvement, it is either accepted anyway with some probability (described below), or it is rejected and the changes are not kept.

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 [Kirkpatrick *et al.*, 1983]. The escape function for accepting inferior solutions is: $Escape(S, S', T) = e^{-|Cost(S) - Cost(S')|/T}$ where T is a "temperature" parameter that is gradually reduced (i.e. *cooled*) 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.

4 Repair Heuristics

The amount of knowledge used to make repair decisions can vary significantly. Intuitively it makes sense to employ knowledge in decision-making when it is available. However using knowledge is not free – computational overhead is incurred to evaluate and use repair knowledge. This is analogous to the *utility problem* of machine learning [Minton, 1988] which states that learning can degrade the performance of a problem solver if the learned knowledge is not useful. Similarly, a knowledge-intensive repair method could degrade the

problem solver if the method is overly expensive and does not provide enough heuristic power to compensate for its expense.

One repair-based heuristic that has achieved notable success in the scheduling domain is MIN-CONFLICTS [Minton *et al.*, 1990]. A common use of this technique is to randomly select a violated constraint, consider all possible moves for the tasks involved in the violation, and perform the move that minimizes the resulting number of constraint violations. In effect, MIN-CONFLICTS performs *lookahead* to determine which move to perform.

GERRY's repair framework also randomly selects a constraint to repair. However, in the selection of candidate repairs, we have found that considering all possible moves in a given repair is computationally infeasible. A single repair might need to consider thousands of candidate moves. In addition, the most promising moves are typically the shortest ones, since these moves are less likely to unnecessarily perturb the schedule.

Instead, GERRY identifies a small set of candidate schedule modifications that are deemed most likely to have a chance to repair the violation while minimizing unnecessary perturbation. The types of modifications depend upon the constraint being repaired:

- A repair to a resource constraint always moves one of the violated tasks to a new location. For each task participating in the violation, two candidate moves are identified. These moves are to the nearest times (one earlier and one later) that would satisfy the violated constraint for this task.
- A repair to a state constraint cannot always be achieved by task-moves, since the required state may not exist anywhere on the schedule. It may be possible then to insert a new *achiever* task whose sole purpose is to set the desired state over the time interval that the constraint specifies. This option is considered along with moving the task to the nearest locations (one earlier and one later) where the state constraint is already properly set (or could be properly set with the insertion of a new achiever task).

Once a set of candidate modifications has been identified, GERRY can perform a wide variety of heuristics to select the actual repair to perform.

4.1 Static Repair Measurement

One simple heuristic consists of simply taking measurements of the existing schedule and making an informed guess as to which of the available repairs appears most promising. Each repair receives a score based upon the measurements taken; these scores serve as a basis for selecting the repair to perform. We have found this inexpensive heuristic, which we refer to as the *static* measurement heuristic, to be quite effective in the domain of shuttle ground processing. The kinds of measurements performed for the scoring of available repairs, again, are a function of the constraint being repaired:

- For resource constraints, each candidate move is scored by several criteria, and each score is normalized by what are considered the best and worst pos-

sible scores for that criteria. A net score is derived from a linear combination of all individual scores. GERRY uses the net scores to weigh each move before randomly selecting one. The following scoring criteria are used:

1. Assign higher scores to moves whose task's requirement more closely matches the amount of resource overallocation.
 2. Assign higher scores to moves whose task has fewer temporal dependents.
 3. Assign higher scores to shorter moves.
- For state constraints, prefer to insert a new task that will satisfy the proper state for the violated constraint if this can be done *without* actually having to move the task with the violation. Otherwise, score each task-move (to where the state is or can be satisfied properly) according to the magnitude of the move itself.

4.2 Lookahead Repair Measurement

An alternative repair selection heuristic is to perform lookahead on each candidate modification, record the resulting cost, then select the modification that reduces the cost by the greatest amount. This is a version of MIN-CONFLICTS on the restricted subset of candidate modifications the system has identified.

Clearly the lookahead heuristic has more knowledge at its disposal when choosing a repair. But is the computational expense of that enhanced knowledge too great? We knew that our static measurement heuristic performed well in our application domain, and were curious how well the lookahead heuristic would perform in its place. We expected that each iteration of the lookahead method would lower the cost more than an iteration of the static method, but also expected the less-informed static repair mechanism to be able to perform more iterations per unit time than the lookahead method.

5 Experiments

We performed experiments in order to explore the comparative utility of the static and lookahead measurement heuristics. For both heuristics, the same method (described above) is employed to select the constraint to repair and the set of candidate modifications to the schedule. The two methods were compared on both scheduling and rescheduling problems and on both artificially generated and actual Space Shuttle data.

5.1 Problem Data Sets

Scheduling problems were constructed using a problem generator. These problems were built with varying characteristics (e.g. number of tasks, task durations, temporal dependencies, amount and frequency of resource requests and state requirements and effects). The problem generator initially assigns all tasks in the schedule to their earliest locations that preserve temporal constraints: this changes the initial scheduling problem into a rescheduling problem, where the resource and state constraints must now be repaired.

All of the rescheduling experiments used Space Shuttle data. One data set, corresponding to the STS-43 mission of the orbiter Atlantis, contained no state constraints. A rescheduling experiment was generated by taking a conflict-free schedule and moving 10 pending tasks by a random amount. After this "perturbation", the rescheduler was invoked to resolve any conflicts created by the moves.

The penalty of a resource capacity constraint was 1 if the resource was overallocated. When a required state variable was not set correctly, the penalty of the associated state constraint returned 1. Weights for all constraints were set to 1.

All experiments were run on a Sun SPARCstation 2 with 32MB of memory. Each experiment ran until there were no outstanding violations or a 30-minute CPU time bound was reached. Since the repair functions are probabilistic, we calculated average results over at least 10 repeated trials for each experiment.

5.2 Empirical Results

Table 1 presents results for the entire experimental suite. For each problem the average time to solution, percent timeouts, and average cost at timeout are reported for the static (S) and lookahead (L) repair heuristics. The "Due Date" column represents the position of the completion date for the problem as a percentage beyond the shortest possible schedule with respect to temporal constraints only. In other words, if $Due\ Date = 30$, and ES_{SOP} is the earliest start for the *StartOfProject* task and EF_{SOP} is the earliest finish for the *EndOfProject* task, then all tasks had to be scheduled in a time frame $= 1.3 \times (EF_{EOP} - ES_{SOP})$. Problems with smaller "Due Date" values were more tightly constrained and therefore more difficult to schedule. The data shows that for most of the experiments run, especially the larger problems, static value selection outperforms lookahead.

Figure 1 shows average best cost against time for the 50-task problem with increasingly smaller "Due Date" values. These graphs show that as the problem becomes more constrained, the lookahead technique fares increasingly better on average. In the highly constrained graph, we can see a "crossover" behavior, where the static repair technique quickly brings the cost to a low value, but has a hard time going much further. In contrast, the lookahead technique requires more time to get to the same low cost, but can find better schedules more efficiently beyond that point.

This seems to indicate that the static repair measurement is not as useful in resolving the more highly constrained violations. The heuristic is blind to secondary effects it may cause (e.g. new violations on other resources), and is thus more prone to cycle between moves that it scores highly. Lookahead is not blind to secondary effects, since it will base its repair selection on each repair's resulting cost for the entire schedule.

Figure 2 shows representative behavior from our rescheduling tests on STS-43 and STS-50. In the STS-43 tests, the static heuristic was superior, while in the STS-50 tests, the static heuristic's advantage is not as clear. We believe that the competitiveness of lookahead

Problem Name	Due Date	Initial Cost	Average Time to Solution (sec)		Percent Timeouts		Average Cost at Timeout	
			S	L	S	L	S	L
20 Tasks	30	42	-	-	100	100	5	5
20 Tasks	50	42	15	43	55	60	2	3
20 Tasks	70	42	14	29	5	20	1	2
50 Tasks	50	297	-	-	100	100	55	37
50 Tasks	100	297	328	229	0	35	-	1
50 Tasks	150	297	98	195	0	26	-	1
100 Tasks	30	322	-	1072	100	85	38	16
100 Tasks	50	322	663	470	15	0	7	-
100 Tasks	100	322	65	130	0	0	-	-
500 Tasks	150	3465	-	-	100	100	1007	2053
500 Tasks	200	3465	-	-	100	100	902	2103
500 Tasks	250	3465	-	-	100	100	892	1975
STS-43		128	467	816	10	10	4	4
STS-43		51	13	93	0	0	-	-
STS-43		95	25	157	0	0	-	-
STS-43		158	573	423	0	20	-	5
STS-43		63	26	148	0	0	-	-
STS-43		104	16	107	0	0	-	-
STS-43		100	62	188	0	0	-	-
STS-43		159	32	271	0	0	-	-
STS-43		109	254	439	0	0	-	-
STS-43		135	306	508	0	0	-	-
STS-50		32	111	179	10	30	1	1
STS-50		90	532	721	20	0	7	-
STS-50		58	90	328	0	0	-	-
STS-50		62	-	-	100	100	4	5
STS-50		25	24	67	0	0	-	-
STS-50		11	22	17	0	0	-	-
STS-50		64	37	20	0	0	-	-
STS-50		19	14	27	0	0	-	-
STS-50		110	17	62	0	0	-	-
STS-50		36	-	-	100	100	1	1

Table 1: Experimental Results: For Static (S) and Lookahead (L) heuristics, CPU time for problems that converge, percentage of problems that do not converge, and average cost at timeout for non-converging problems.

on STS-50 is because there are fewer candidate repairs to explore on average when repairing a state constraint as opposed to a resource constraint. In addition, a repair to a state constraint can be accomplished without moving any tasks, so lookahead has less work to do in measuring each candidate modification. Consequently, it is computationally less expensive to perform this limited form of lookahead.

6 Conclusions and Future Work

Our experiments suggest that the selection of an efficient repair mechanism in a system such as GERRY depends at least in part on two problem characteristics – problem size and constraint tightness. Static evaluation – a less informed technique – serves better on larger and less constrained problems; lookahead techniques are especially effective on more difficult and smaller problems. Clearly lookahead suffers in problems with many tasks, where the overhead of preserving temporal relations is substantial. It is apparent that the size of the shuttle

ground processing problem is a more dominating feature than the tightness of its resource and state constraints.

The tabular data shows that neither heuristic is immune from cycling. We see that occasionally no solution is found on problems where the average time to solution is well below the 30-minute timeout. Perhaps using a heuristic that selects a modification randomly in these cases would help escape the cycling behavior.

We observed (although the data is not presented here) that when both static and lookahead methods reached a cost of zero, the lookahead technique did so in a fewer number of iterations. Thus the lookahead mechanism more often chose the best repair candidate, even though it may have spent too much time in doing so.

From the graphs shown, we see that early in the repair process it is better to perform *some* kind of repair quickly, instead of wasting time evaluating which repair will reduce the cost by the greatest amount. But later in the repair process as a system is attempting to resolve a final few “tough” violations, the lookahead heuristic

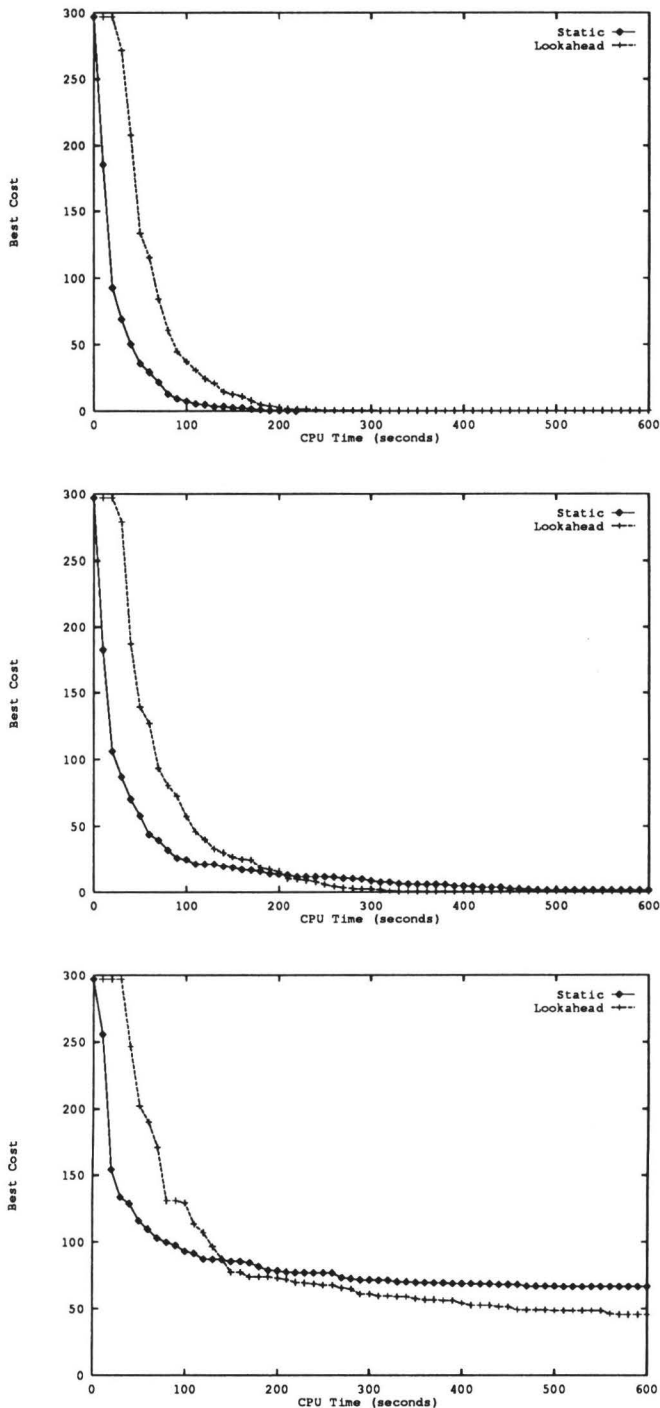


Figure 1: Results of scheduling the 50-task problem with different Due Date values. (upper) 150% (underconstrained); (middle) 100% (moderately constrained); (c) 50% (highly constrained).

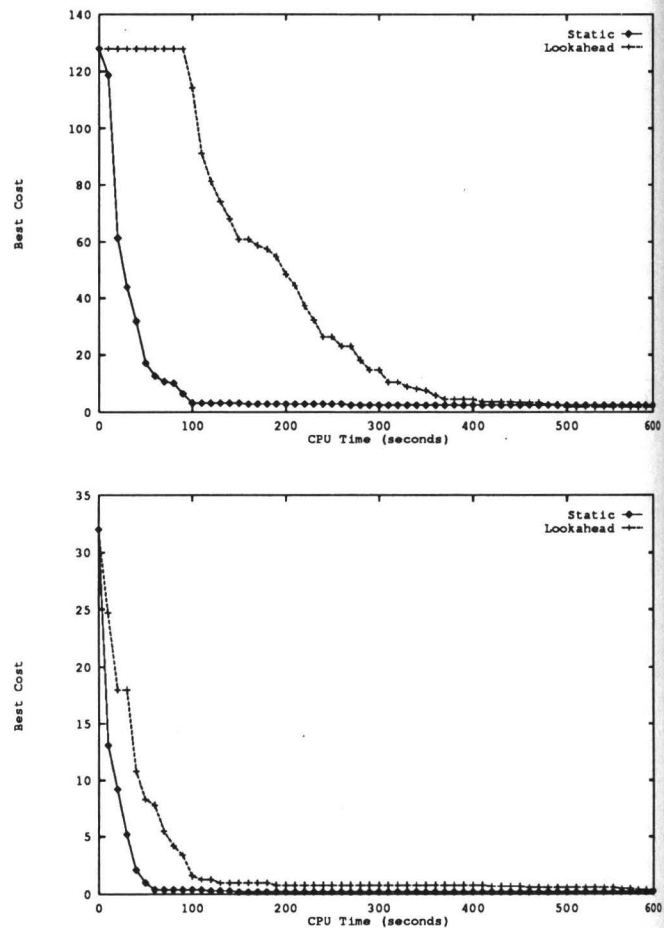


Figure 2: Representative rescheduling runs from the Space Shuttle Ground Processing domain. (upper) STS-43; (lower) STS-50.

becomes more necessary to make productive moves in a timely manner.

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 switch dynamically between heuristics [Zweben *et. al.*, 1992].

7 Related Work

Our work was heavily influenced by previous constraint-based scheduling [Fox, 1987; Fox and Smith, 1984; Sadeh and Fox, 1989] and rescheduling efforts [Ow *et. al.*, 1988].

ISIS [Fox, 1987] 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 restarting the beam search from

an intermediate state.

OPIS [Fox and Smith, 1984; Ow *et. al.*, 1988], 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 its 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 [Johnson *et. al.*, 1990a; Johnson *et. al.*, 1990b]. In contrast with our constraint-based repair, their repairs were generally uninformed.

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 [Sussman, 1973] and, more recently, the repair strategies used in the GORDIUS [Simmons, 1988] generate-test-debug system, in the PRIAR plan modification system [Kambhampati, 1990], and the CHEF case-based planner [Hammond, 1986].

Miller *et al.* [Miller *et. al.*, 1988], Currie and Tate [Currie and Tate, 1991], and Drummond and Bresina [Drummond and Bresina, 1990] describe other efforts that deal with resource and deadline constraints.

8 Acknowledgments

We wish to thank Mark Drummond, Peter Friedland, Steve Minton, Holly Pease and the anonymous reviewers for their comments.

References

- [Biefeld and Cooper, 1991] Biefeld, E. and Cooper, L. Bottleneck Identification Using Process Chronologies. In *Proceedings of IJCAI-91*, Sydney, Australia, 1991.
- [Currie and Tate, 1991] Currie, K., and Tate, A. O-Plan: The Open Planning Architecture. *Artificial Intelligence*, 52(1), 1991.
- [Davis, 1987] Davis, E. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32(3), 1987.
- [Drummond and Bresina, 1990] Drummond, M. and Bresina, J. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. In *Proceedings of AAAI-90*, 1990.
- [Fox and Smith, 1984] Fox, M. and Smith, S. A Knowledge Based System for Factory Scheduling. *Expert System*, 1(1), 1984.
- [Fox, 1987] Fox, M. *Constraint-Directed Search: A Case Study of Job Shop Scheduling*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.
- [Freuder, 1982] Freuder, E. C. A Sufficient Condition for Backtrack-Free Search. *J. ACM*, 29(1), 1982.
- [Hammond, 1986] Hammond, K. J. CHEF: A Model of Case-Based Planning. In *Proceedings of AAAI-86*, 1986.
- [Johnson *et. al.*, 1990] Johnson, D.S. and Aragon, C.R. and McGeoch, L.A. and Schevon, C. Optimization by simulated annealing: An experimental evaluation, Part II. *Journal of Operations Research*, 1990.
- [Johnson *et. al.*, 1990a] Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C. Optimization By Simulated Annealing: An Experimental Evaluation, Part I (Graph Partitioning). *Operations Research*, 1990.
- [Johnson *et. al.*, 1990b] Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C. Optimization By Simulated Annealing: An Experimental Evaluation, Part II (Graph Coloring and Number Partitioning). *Operations Research*, 1990.
- [Kambhampati, 1990] Kambhampati, S. A Theory of Plan Modification. In *Proceedings of AAAI-90*, 1990.
- [Kirkpatrick *et. al.*, 1983] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. Optimization by Simulated Annealing. *Science*, 220(4598), 1983.
- [Kurtzman and Aiken, 1989] Kurtzman, C.R. and Aiken, D.L. The Mfive space station crew activity scheduler and stowage logistics clerk. In *Proceedings the AIAA Computers in Aerospace VII Conference*, Monterey, CA, 1989.
- [Lin and Kernighan, 1973] Lin, S, Kernighan, B. An Effective Heuristic for the Travelling Salesman Problem. *Operations Research*, 21, 1973.
- [Mackworth, 1977] Mackworth, A.K. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1), 1977.
- [Miller *et. al.*, 1988] Miller, D., Firby, R. J., Dean, T. Deadlines, Travel Time, and Robot Problem Solving. In *Proceedings of AAAI-88*, St. Paul, Minnesota, 1988.
- [Minton *et. al.*, 1990] 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.
- [Minton, 1988] Minton, S. *Learning Effective Search Control Knowledge: An Explanation-based Approach*. PhD thesis, Carnegie Mellon University, 1988.
- [Morris, 1990] P. Morris. Solutions without exhaustive search: An iterative descent method for binary constraint satisfaction problems. In *Proceedings the AAAI-90 Workshop on Constraint-Directed Reasoning*, Boston, MA, 1990.
- [Ow *et. al.*, 1988] Ow, P., Smith S., Thiriez, A. Reactive Plan Revision. In *Proceedings AAAI-88*, 1988.
- [Sadeh and Fox, 1989] Sadeh, N. and Fox, M. S. Preference Propagation in Temporal/Capacity Constraint Graphs. Technical report, The Robotics Institute, Carnegie Mellon University, 1989.

- [Simmons, 1988] Simmons, R.G. Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems. Technical report, MIT Artificial Intelligence Laboratory, 1988.
- [Sussman, 1973] Sussman, G.J. *A Computational Model of Skill Acquisition*. PhD thesis, AI Laboratory, MIT, 1973.
- [Waltz, 1975] Waltz, D. Understanding Line Drawings of Scenes with Shadows. In P. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.
- [Zweben *et. al.*, 1992] Zweben, M., Davis, E., Daun, B., Drascher, E., Deale, M., Eskey, M. Learning To Improve Constraint-Based Scheduling. *Artificial Intelligence*, 58(1-3), 1992.
- [Zweben *et. al.*, 1990] Zweben, M., Deale, M., Gargan, R. Anytime Rescheduling. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning and Scheduling*, 1990.