Learning to Improve Iterative Repair Scheduling

Monte Zweben
Eugene Davis
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-92-14

January, 1992

Learning to Improve Iterative Repair Scheduling

Monte Zweben
Eugene Davis*
NASA Ames Research Center
M.S. 269-2
Moffett Field, California 94035
zweben@shepard.arc.nasa.gov

Phone: 415/604-4940
Fax: 415/604-3594

January 7, 1992

Abstract

This paper presents a general learning method for dynamically selecting between repair heuristics in an iterative repair scheduling system. The system employs a version of explanation-based learning called *Plausible Explanation-Based Learning (PEBL)* that uses multiple examples to confirm conjectured explanations. The basic approach is to conjecture contradictions between a heursitic and statistics that measure the quality of the heuristic. When these contradictions are confirmed, a different heuristic is selected. To motivate the utility of this approach we present an empirical evaluation of the performance of a scheduling system with respect to two different repair strategies. We show that the scheduler that learns to choose between the heuristics outperforms the same scheduler with any one of two heuristics alone.

Problem Area: Learning and Scheduling
General Approach: Explanation-based
Evaluation Criteria: Empirical Evaluation

*Recom Technologies

1 Introduction

Iterative repair is an general search method that has been applied to complex scheduling and constraint satisfaction problems with good results (e.g., [Zwe90, Min90, Bie91, Lin73, Joh90]). Iterative repair methods construct an initial, possibly-flawed schedule and then iteratively reduce the number of constraint violations until a conflict-free schedule is produced. Thus, the main difference between iterative repair methods and more traditional constraint-based approaches is that iterative repair modifies a complete, imperfect schedule, instead of incrementally constructing a conflict-free schedule.

In reviewing previous research on repair-based scheduling we have found that one of the essential differences among such systems is how informed the repair process is. As illustrated by Figure 1, an "intelligent" repair system (i.e., a point near the right of the x-axis) will generally require fewer, more computationally expensive iterations to find a solution. In contrast, a less informed system may require more iterations to reach a solution, but each iteration can be accomplished with less computational cost. Ideally there exists an "equilibrium" point where neither adding repair knowledge nor removing it yields a system that converges more quickly to conflict-free solutions.

This paper presents a learning method for dynamically selecting between a less-informed repair heuristic and a more informed, but computationally expensive, heuristic. The system begins with the inexpensive repair strategy and then uses *Plausible Explanation-Based Learning (PEBL)* [Esk90] to learn when the less-informed heuristic leads the search awry. When similar situations arise after learning, the system adopts the more informed strategy.

2 Iterative Repair

In this paper, we consider scheduling problems consisting of a set of tasks, a set of resources, and a set of constraints. Each task has a start time, a duration, and a set of resource requirements. The constraints specify restrictions on the possible schedules. For example, a constraint might establish temporal relations between tasks, or determine the maximum capacity of the amount of resource used

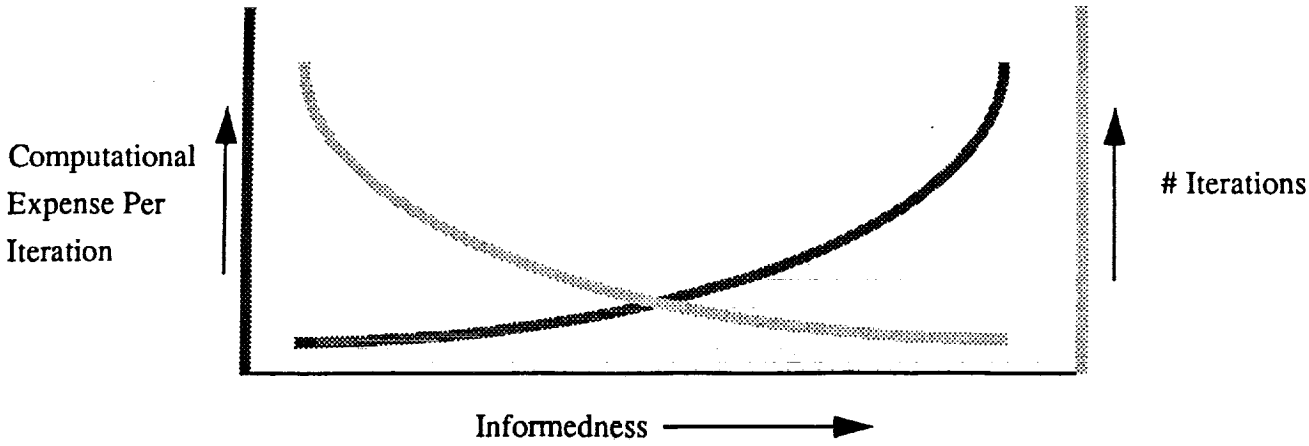

Figure 1: Informedness vs. Computational Cost of Repairs

at any point. The goal is to find a schedule that minimizes the number of constraint violations or *conflicts* while optimizing some objective function.

As mentioned above, iterative repair methods begin with a complete, possibly flawed schedule, then reduce the number of constraint violations until a conflict-free schedule is produced. In our formulation of the scheduling problem, we represent attributes such as the start time of a task, the resource assigned to each task, etc., as variables. Thus, each repair constitutes re-assigning one or more variables with different values. To simplify our discussion below, we will often refer to repairing a conflict by "moving a task," which is a prototypical repair for scheduling problems.

3 Problem Specification

Our experiments were conducted using the *Space Shuttle ground processing* problem as a test domain. To prepare each space shuttle for launch, a multitude of inspection, repair, and refurbishment tasks must be completed by a designated deadline. Each task demands some amount of resources such as technicians, engineers and quality inspectors. Tasks are also temporally related to each other. Our scheduling system is used to track existing Space Shuttle missions in parallel with the more manual techniques currently in place. We expect our system to evolve into a primary scheduling and rescheduling role at the Kennedy Space Center (KSC) in support of the flow management teams that coordinate ground processing.

Each task in the Space Shuttle ground processing scheduling problem is associated with a variable that represents its start time. The output of the scheduling process is an assignment of start times such that all constraints are satisfied.

There are several different types of constraints. Temporal constraints enforce predecessor/successor relationships between tasks as well as more complex temporal relations [All84, Vil86] and due dates. Since each task also requires varying amounts of resources, capacity constraints are used to ensure that no resource is overallocated. For example, an activity for testing an engine of the Space Shuttle could overlap with an inspection of the forward reaction control thrusters. Here both tasks compete for technicians of the same skill level. If both require 10 of the 15 technicians available during that time, then the capacity constraints on the activities would be violated.

For each shuttle mission, a set of generic tasks (called the generic flow) must be performed. The generic flow includes tasks for inspecting, testing, repairing, and installing in all systems of the Shuttle. In addition to the generic tasks, other groups of tasks (called options) are added. An option is a set of temporally related tasks, each having individual resource requirements. Options may be added for a variety of reasons; for example, certain maintenance operations are only required after every fifth flight.

A mission is composed of the tasks and constraints from both the generic flow and the necessary options. Options are attached as a postrequisite to one generic task and as a prerequisite to another and are thus anchored into the generic flow. There are no other temporal constraints on tasks in the options.

Our learning method attempts to transfer learned information from a training set of missions to a different set of missions that share similar options.

4 Constraint-Based Iterative Repair

Our learning method was applied to the constraint-based iterative repair system described in [Zwe90]. This system begins with an assignment of all variables and improves the assignments until the cost function reaches zero. For the experiments reported here, the cost function is simply the number of conflicts in the schedule.

The initial schedule is determined by the Critical Path Method (CPM) [Hil80]. This algorithm schedules all tasks as early as possible considering only temporal constraints. It is an O(n) algorithm that results in a schedule that contains only resource violations.

In the system, repairs are associated with constraints. This allows knowledge engineers to encode local repair heuristics that are likely to satisfy the constraint, without requiring them to anticipate how repairs interact with other constraints. Of course local repairs do occasionally yield globally undesirable states.

In our experiments, the system repairs up to ten constraint violations per iteration. Repairing a violation 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.

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 not an improvement, with some probability (see below) it is either accepted anyway, or it is rejected and the changes are not kept.

The system will sometimes accept 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 probability distribution 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 during the search process.

To summarize, the repair algorithm 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.

4.1 The Dimensions of Repair Knowledge

In [Zwe91], the authors outline five different dimensions of repair-based techniques that affect the tradeoff between informedness and computational complexity. This paper is concerned with two of these dimensions: the depth of repairs and the evaluation criteria used to select repairs.

The *depth* of a repair strategy is the degree of lookahead employed when choosing a repair. A strategy with depth 0 only considers the current state (current schedule) in evaluating what repair to make. A strategy with depth 1 evaluates a potential repair by considering the state that would result after the repair. For example, the min-conflicts method [Min90] is a depth 1 method since it selects the repair that most reduces the total number of conflicts. Similarly, a strategy with depth 2 would evaluate a repair by considering the repairs that would be required *after* this current repair was executed. Some systems employ variable-depth lookahead: moving a task may require additional tasks to move if the system automatically preserves temporal constraints.

The evaluation criterion measures the quality of a repair. After a set of candidate repairs has

been generated, each is evaluated and one selected. Examples of evaluation criteria are the number of conflicts in a resulting schedule or the amount of perturbation from a previous state. Ideally, a good evaluation criterion should accurately measure both the likelihood of reaching a conflict-free state from the resulting schedule and the quality of the resulting schedule with respect to the objective function.

4.2 Two Heuristic Repair Methods

In this section we present two repair strategies for resource capacity constraints that differ in depth of lookahead and evaluation criteria. One repair strategy uses no lookahead (depth 0) and evaluates the quality of the repair with a simple evaluation function over task attributes. The other strategy employs a form of min-conflicts that uses a variable depth lookahead.

4.2.1 A Depth Zero Repair Strategy

To repair a violated resource capacity constraint, an offending task is selected and reassigned to a new time. The heuristic used to select a candidate 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.

For each of the tasks contributing to the violation, the system considers only the *next earlier* and *next later* available times for a move, rather than exploring many or all possible times. Each candidate move is scored using a linear combination of the *fitness* and *temporal dependents* 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 two computationally inexpensive heuristic criteria: degree of fitness and number of temporal dependents.

4.2.2 A Variable Depth Repair Strategy

As before, for each of the tasks contributing to a violation, the system considers moving the task to the next earlier and next later available time. However, this method selects the repair that minimizes the constraint violations in the resulting state. This method uses a variable depth lookahead because it counts conflicts *after* the Waltz algorithm moves an arbitrary number of tasks to preserve temporal constraints. The main advantage of this strategy is that it can identify interactions (i.e., new constraint violations) that would occur after a candidate repair. However, this technique is more computationally expensive than the depth 0 strategy because each candidate move must be performed, the resulting state evaluated, and the original state restored in order to evaluate the next move.

5 Learning to Select Repair Strategies

Here we describe a general method for learning to select between repair strategies. We define *goodness* as a quality metric for a repair which, for these experiments, is simply the number of conflicts resulting after a repair. For example, a repair which results in fewer conflicts would have a high goodness rating. The first step of the training process is to gather goodness statistics for each task moved within a repair. During this first phase, the system uses the depth 0 heuristic over a training set of missions, resulting in a table of goodness statistics. Figure 2 depicts a typical statistics table for a training suite.

After ample statistics are gathered, the system enters a second phase which employs Plausible Explanation-Based Learning (PEBL). PEBL was originally developed to learn variable ordering in a depth-first backtracking scheduling system [Esk90] and is a modest extension over previous EBL methods [Mit86, DeJ86, Min88]. With PEBL, explanations are not converted into search control

Goodness Statistics				
Tasks	*Options*			
	G	O_1	O_2	O_3
G_1	1	1	1	3
G_2	2	1	2	1
⋮	⋮	⋮	⋮	⋮
O_{11}	2	1	1	1
O_{12}	2	1	2	1
⋮	⋮	⋮	⋮	⋮
O_{21}	3	6	7	10
O_{22}	2	5	9	12
⋮	⋮	⋮	⋮	⋮
O_{31}	4	9	12	15
O_{32}	3	8	13	16
⋮	⋮	⋮	⋮	⋮

PEBL-Conjectures				
Tasks	*Options*			
	G	O_1	O_2	O_3
G_1	0	0	0	0
G_2	0	0	0	0
⋮	⋮	⋮	⋮	⋮
O_{11}	0	0	0	0
O_{12}	0	0	0	0
⋮	⋮	⋮	⋮	⋮
O_{21}	.4	.5	.8	.9
O_{22}	.3	.6	.9	.9
⋮	⋮	⋮	⋮	⋮
O_{31}	.5	.6	.9	.7
O_{32}	.5	.6	.7	.7
⋮	⋮	⋮	⋮	⋮

Figure 2: *Goodness Statistics* (left) : each cell O_{ij}, O_k is the expected violations that will result from moving task j of option i when option O_k is part of the mission. *PEBL-Conjectures* (right) : each cell O_{ij}, O_k is the probability that moving task j of option i will contradict the goodness statistics when option O_k is part of the mission.

advice until confirmed with multiple examples. Evidence measuring the accuracy of the explanations is gathered by retrying explanations in similar circumstances.

The PEBL domain theory is used to conjecture that our variable depth strategy should be used when the goodness statistics contradict the advice offered by the depth 0 strategy. That is, the tasks suggested by the less-informed heuristic would not be preferred if the decision were based upon the goodness statistics alone. More specifically, if the highest candidate tasks proposed by the depth 0 strategy are not suggested by the statistical data as one of the top candidates to move, then it is likely that the depth 0 criteria has over-abstracted the problem details, and may create unnecessary constraint violations. PEBL then conjectures that since this local strategy has been myopic, a contradiction exists that can only be resolved with more global information. When training is terminated, the system is capable of selecting repair strategies according to this evidence.

To foster transfer between problems, the goodness statistics and PEBL-conjectures are organized into tables that relate tasks to the generic and options as shown in Figure 2. New problems can exploit learned knowledge when they contain options that appeared in the training set.

After learning, when selecting a task to move for a repair, the system uses the PEBL-conjectures to decide whether it should believe its less-informed heuristic or use the lookahead strategy. To make this decision the system averages the relevant PEBL-conjectures for the task selected by the depth 0 strategy. If this result exceeds a threshold, then lookahead is performed. In our experiments, the threshold is set at 0.5.

For example, suppose some mission is composed of options O_1 and O_3, and task O_{31} is selected by the depth 0 repair. Since the PEBL-conjectures of $(O_{31}, G) = .5$, $(O_{31}, O_1) = .6$, $(O_{31}, O_3) = .7$ average to .6, and since this exceeds the .5 threshold, our system will employ the lookahead strategy.

To summarize, learning proceeds with a statistics gathering phase followed by a PEBL stage. The statistics quantify the goodness of the depth 0 strategy. Then the PEBL phase is used to conjecture contradictions between the depth 0 strategy and the statistics. This technique can be used with any two repair heuristics and a variety of goodness metrics.

6 Evaluating the Method

To evaluate the success of our approach, we generated a set of scheduling problems by specifying range limits for various problem parameters as input to the system. An individual scheduling problem consists of a generic flow, a set of option flows, and a set of resource pools. A "suite" of scheduling problems can be considered a single scheduling domain. That is, for each suite, the generic flow, the set of resource pools and the set of options are fixed. A generic flow is comprised of 100 to 150 tasks. For each suite, there is a set of 8 possible options, each consisting of 10 to 50 tasks. There are 6 resource pools for each suite, with capacities ranging from 7 to 10 units.

We generated three major testing problem sets: (D)ifficult, (F)ew options, and (L)oose. In problem sets (D) and (L), each option has a 50 % chance of being included in a particular problem, so that each problem on average consists of 4 of the 8 options. In problem set (F) there was only a 20 % chance that each option would be included in a problem. Thus problem set (F) was easier than (D) because fewer options cause less interaction. All missions have an overall due date that all tasks must precede. Both the (D) and (F) problem sets have earlier due dates for each mission while the (L)oose problem set has later due dates. In short, problem set (D) includes many options in a tight time bound; problem set (F) is easier than (D) because there are fewer options; problem set (L) is easier than (D) because its time bound is less restrictive.

For the training phase, we randomly selected 10 (out of the 256 possible) scheduling problems to learn the characteristics of the likely task/option interactions. After training, we compared the respective speed of convergence to an acceptable solution (i.e., no constraint violations or reaching a prespecified time bound) for the variable depth lookahead approach, the depth 0 approach, and the approach suggested by PEBL on new problems from the same problem suite (see Figure 3). Testing consisted of more than 1600 runs for the three problem sets.

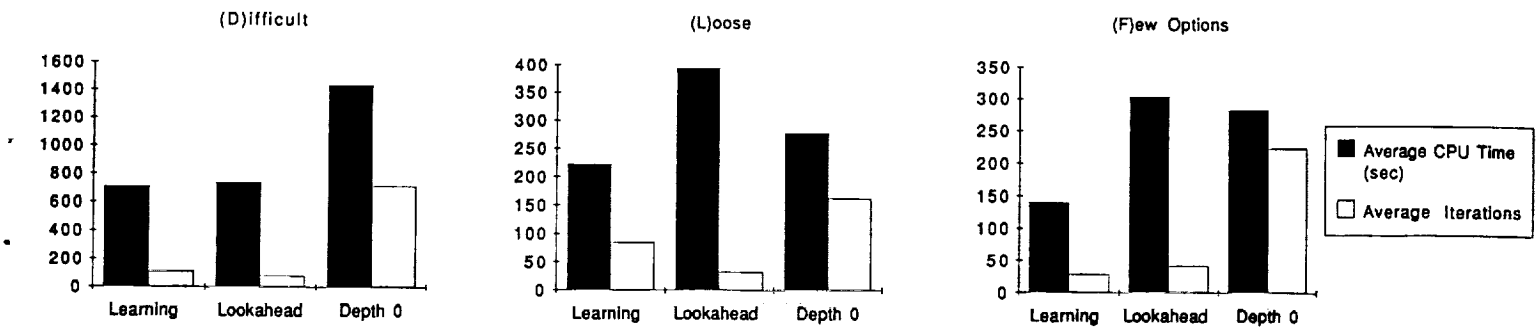

Figure 3: Average CPU Time and Iterations

6.1 Discussion of the Results

The (D)ifficult problem set consisted of numerous tasks with multiple resource requests and tight milestone constraints. In this problem set, the system tended to favor lookahead and performed only 4 % better than the lookahead strategy. However, the system did appear to learn about interactions because it performed 51 % better than the depth 0 strategy.

The (F)ew Options set had fewer interacting tasks in each problem and were therefore considerably easier than the (D) set. Here learning performed 54 % better than lookahead and again 51 % better than the depth 0 strategy.

In the (L)oose problem set, the depth 0 strategy performed much better than the lookahead strategy. Here learning performed 44 % better than lookahead and 20 % better than the depth 0 strategy.

The computational expense of lookahead was clearly worthwhile for problem set (D), but in (F) and (L) using lookahead was overkill. The learning method outperformed the other strategies because it only invests in lookahead when its experience suggests that the depth 0 strategy would make the wrong decision.

7 Related Work

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 the Hacker planning system [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 planner [Ham86].

Another system that uses explanations and multiple examples is presented in [Ben91]. In their system, explanations are created which describe how a plan can be "tuned". If multiple conflicting explanations are created, either a compromise between conflicting explanations is attempted, or a new plan is constructed. Additionally, Gratch and DeJong [Gra91] have successfully applied a very similar technique to the two domains in [Min88] and [Etz90]. They use multiple examples to gain confidence in the utility of a learned rule set.

Our idea of taking statistics on the frequency of interactions between option tasks was inspired by the calibration techniques of [Han91], where a heuristic error model is built from problem solving instances. In their work on decision-theoretic control of scheduling systems, they maintain a multi-dimensional histogram, where each dimension is a heuristic value or an outcome attribute. They use the data to determine what heuristics are likely to lead to good solutions.

Finally, we extend previous work in scheduling using multiple perspectives [Smi85] by applying machine learning techniques to minimize the changes in the search strategy to the most critical points in the search space based on experience.

8 Conclusions and Future Work

The overall conclusion of our work is that PEBL is an effective method for learning search control for iterative repair search. However, the empirical results indicate clear avenues of improvement. In general, the approach requires some hand-tuning and human analysis with respect to the amount of

learning and the thresholds specified.

For the iterative repair search strategy, a cost analysis should be incorporated into the domain theory to provide a better indication of when the use of lookahead is beneficial. The domain theory currently reasons about the predicted accuracy of the repair heuristic. Information about the computational expense of a more informed repair heuristic should be incorporated into the domain theory to determine the optimal balance between the two metrics. In future work, we plan to augment PEBL with utility information and perform more comprehensive testing on problems of variable complexity.

Another problem indicated by our empirical analysis is that the iterative repair method is prone to cycle as it approaches a final solution. Often, the system rapidly converges to few constraint violations, but is unable to jump out of the local minimum. In these cases the system is terminated when a time threshold is reached, but the final solutions are non-optimal. One idea is to use PEBL to isolate these local minima situations. If this proves to be a successful approach, it has potential to generalize to related problems and domains.

References

[All84] Allen, J. F. Toward a General Theory of Action and Time. *Artificial Intelligence*, 23, 1984.

[Ben91] Bennett, S. and DeJong, G. Comparing Stochastic Planning to the Acquisition of Increasingly Permissive Plans for Complex, Uncertain Domains. In *Proceedings of the Eighth International Workshop on Machine Learning*, 1991.

[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.

[DeJ86] DeJong, G.F., Mooney, R. Explanation-Based Generalization: An Alternative View. *Machine Learning*, 1, 1986.

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

[Etz90] Etzioni, O. Why Prodigy/EBL Works. In *Proceedings of AAAI-90*, Boston, MA, 1990.

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

[Gra91] Gratch, J. and DeJong, G. A Hybrid Approach to Guaranteed Effective Control Strategies. In *Proceedings of the Eighth International Workshop on Machine Learning*, 1991.

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

[Han91] Hansson, O. and Mayer, A. Decision-Theoretic Control of Artificial Intelligence Scheduling Systems. Technical report, Heuristicrats Research Inc., Berkeley, CA., 1991.

[Hil80] Hillier, F.S., Lieberman, G.J. *Introduction to Operations Research.* Holden-Day, Inc., San Francisco, CA, 1980.

[Joh90] 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.

[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.

[Lin73] Lin, S, Kernighan, B. An Effective Heuristic for the Travelling Salesman Problem. *Operations Research*, 21, 1973.

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

[Min88] Minton, S. *Learning Effective Search Control Knowledge: An Explanation-based Approach.* PhD thesis, Carnegie Mellon University, 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.

[Mit86] Mitchell, T.M., Keller, R.M., Kedar-Cabelli, S.T. Explanation-Based Learning: A Unifying View. *Machine Learning*, 1, 1986.

[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.

[Vil86] Vilain, M., Kautz, H. Constraint Propagation Algorithms for Temporal Reasoning. In *AAAI-86 Proceedings*, 1986.

[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. In *Submitted to The First International Confernce on AI Planning Systems*, 1991.