Learning Control Knowledge through Cases in Schedule Optimization Problems

Kazuo Miyashita*

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
miya@cs.cmu.edu

Katia Sycara

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
katia@cs.cmu.edu

Abstract

We have developed an integrated framework of iterative revision and knowledge acquisition for schedule optimization, and implemented it in the CABINS system. The ill-structuredness of both the solution space and the desired objectives make scheduling problems difficult to formalize and costly to solve. In CABINS, situation-dependent user's preferences that guide schedule revision are captured in cases together with contextual information. During iterative repair, cases are exploited for multiple purposes, such as (1) repair action selection, (2) repair result evaluation and (3) recovery from revision failures. The contributions of the work lie in experimentally demonstrating in a domain where neither the human expert nor the program possess causal knowledge that search control knowledge can be acquired through past repair cases to improve the efficiency of rather intractable iterative repair process. The experiments in this paper were performed in the context of job-shop scheduling problems.

1 Introduction

Recently there has been increased interest in approaches of planning by incrementally modifying previous plans in order to accommodate parts of specifications in old plans or recover from execution failures. [4, 5, 9, 10] Most current approaches have the following common characteristics: (1) they are motivated solely by considerations of computational efficiency, (2) they are concerned only with preserving correctness, and (3) they assume the existence of a strong domain model that is utilized to guide modification and repair. These characteristics are limiting in interesting real world tasks since the existence of a strong domain model can almost never be assumed and quality of a plan (as opposed to only correctness) is often a crucial consideration.

We present an approach, implemented in CABINS, to demonstrate that reuse of previous relevant experiences is effective to efficiently repair a plan for quality improvement without strong domain model. Through case-based reasoning (CBR), CABINS learns three categories of concepts: (1) what combinations of effects of application of a particular repair action constitutes an acceptable or unacceptable repair outcome, (2) what heuristic repair actions to choose in a particular repair context, and (3) when to give up further repair. Those concepts are exploited to enhance the incomplete domain model in CABINS and improve efficiency of problem solving and quality of resultant plans. This distinguishes CABINS from previous case-based planning systems. For example, CHEF [4] assumes existence of a model-based simulator for evaluating a derived plan and detecting failure and well-studied domain rules for repair tactic selection. Works by [5, 9] are based on the hypothesis that the plan built by their planner is causally and teleologically correct, and use CBR to find the satisfying plan efficiently. Moreover, in contrast to case-based knowledge acquisition systems, such as PROTOS [1], where the program requires causal explanations from an expert teacher to acquire domain knowledge, in our approach neither the user nor the program are assumed to possess causal domain knowledge. The user cannot give a solid explanation as to her/his selection of repair action, because s/he cannot predict the effects of the selected action on the plan caused by tight interactions. In other words, the user doesn't have *control knowledge* to guide a plan repair process. The user's

*The first author was a visting scientist at CMU while this research was done.

1043-0989/94 $03.00 © 1994 IEEE

expertise lies in his/her ability to perform consistent evaluation of the results of problem solving and impart to the program cases of problem solving experiences and histories of evaluation tradeoffs.

CABINS has been evaluated in the domain of iterative optimization of job shop schedules. [7] In contrast to approaches that utilize a single repair heuristic [6] or use a statically predetermined model for selection of repair actions [8] in scheduling, our approach utilizes a repair model that is incrementally learned and encoded in the case base. Learning allows dynamic switching of repair heuristics depending on the repair context. In [10] plausible explanation based learning has been successfully used to learn schedule repair control rules for speed up. But, the only concept their system can learn is selection between two repair heuristics, which is inappropriately weak for general optimization problems, while it might be sufficient for constraint satisfaction problems where they evaluated their system. CABINS can learn to select among any number of repair heuristics and giving-up further repair.

In this paper we experimentally demonstrate that in a domain where neither the human expert nor the program possess domain causal knowledge, such as job shop scheduling, control knowledge can be acquired through past cases to improve the efficiency of rather costly iterative optimization process while preserving the quality of results.

2 Schedule optimization in CABINS

Scheduling assigns a set of tasks over time to a set of resources with finite capacity. One of the most difficult scheduling problem classes is job shop scheduling. Job shop scheduling is a well-known NP-complete problem [3]. In job shop scheduling, each task (heretofore called a job or an order) consists of a set of activities to be scheduled according to a partial activity ordering. The dominant constraints in job shop scheduling are: temporal precedence constraints that specify the relative sequencing of activities within a job and resource capacity constraints that restrict the number of activities that can be assigned to a resource during overlapping time intervals.

The activity precedence constraints along with a job's release date and due date restrict the set of acceptable start times for each activity. The capacity constraints restrict the number of activities that can use a resource at any particular point in time and create conflicts among activities that are competing for the use of the same resource at overlapping time intervals. The goal of a scheduling system is to produce schedules that respect temporal relations and resource capacity constraints, and optimize a set of objectives, such as minimize tardiness, minimize work in process inventory (WIP) (i.e., the time an order spends in a factory waiting to be processed), maximize resource utilization etc.

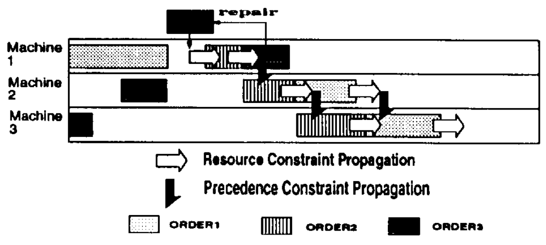
repair
Machine 1
Machine 2
Machine 3
Resource Constraint Propagation
Precedence Constraint Propagation
ORDER1
ORDER2
ORDER3

Figure 1: **Example of tight interactions**

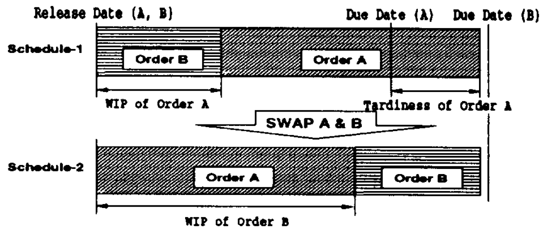

Figure 2: **Example of conflicting objectives**

CABINS *incrementally revises a complete but sub-optimal schedule* to improve its quality. Revision consists in identifying and moving activities in the schedule. Because of the tight constraint interactions, a revision in one part of the schedule may cause constraint violations in other parts. It is generally impossible to predict in advance either the extent of the constraint violations resulting from a repair action, or the nature of the conflicts because of interacting influences of constraint propagations. For example, in figure 1 moving forward the last activoty of ORDER3 creates downstream cascading constraint violations. Therefore, a repair action must be applied and its repair outcome must be evaluated in terms of the resulting effects on scheduling objectives. The evaluation criteria are often context dependent and reflect user judgment of tradeoffs. For example, WIP and weighted tardiness are not always compatible with each other. As shown in figure 2, there are situations where a repair action can reduce weighted tardiness, but WIP increases. Trade-offs are context-dependent and therefore difficult to describe in a simple manner. In CABINS, the user's repair process on a sub-optimal schedule is used to incrementally acquire context dependent schedule evaluations and their justifications. These are recorded in the case base and can be re-used to evaluate future repair outcomes. Hence, user preferences are reflected in the case base in two ways: as *preferences for selecting a repair tactic* depending on the features of the repair context, and as *evaluation preferences* for the repair outcome that resulted from selection and application of a specific repair tactic.

2.1 CABINS overview

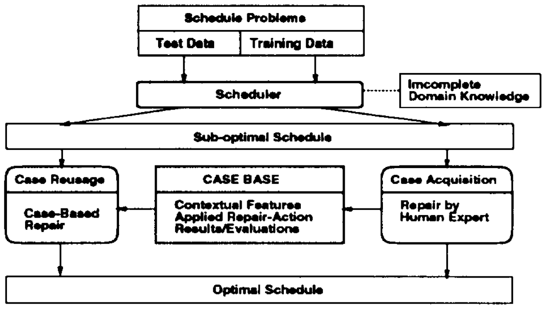

Figure 3: **CABINS Architecture**

Figure 3 depicts the overall architecture of CABINS implementation. CABINS is composed of three modules: (1) an initial schedule builder, (2) an interactive schedule repair (case acquisition) module and (3) an automated schedule repair (case re-use) module.

To generate an initial schedule, CABINS can use one of several scheduling methods including traditional dispatching rules and constraint-based scheduler. But any scheduler can't always produce an optimal solution to the job shop scheduling problem, because the complete knowledge of the scheduling domain model and user's preferences are not available to the scheduler. In order to compensate for the lack of those types of knowledge, CABINS gathers the following information in the form of cases through interaction with a domain expert in its training phase.

- A suggestion of which repair heuristic to apply : a user's decision on what repair heuristic to be applied to a given schedule for quality improvement.
- An evaluation of a repair result : a user's overall evaluation of a modification result.
- An explanation of an evaluation : when a user evaluates the modification result as unacceptable, s/he indicates the set of undesirable effects that have been produced. The explanation given to CABINS consists of the numerical rating of each identified effect.

Our basic assumption on knowledge acquisition through CBR is that, in spite of ill-structuredness of the problem, the following three types of domain knowledge are available and constitute useful case features.

- Repair tactics : a set of local patching heuristics that can be applied to a schedule.
- Descriptive features : attributes of a schedule that describe a particular scheduling situation and might be useful in estimating the effects of applying repair heuristics to the schedule. These features will be described in detail in section 2.2.
- Evaluation criteria : quantification of different aspects of the effects of applying repair heuristics to the schedule. The degree of importance on these criteria is in general user- and state-dependent.

In the following sections, we explain the implementation of CABINS as an application system of the case-based reasoning methodology.

2.2 Case representation

In each repair iteration, CABINS focuses on one activity at a time, the *focal_activity*, and tries to repair it. A case in CABINS describes the application of a particular modification to a focal_activity. Figure 4 shows the information content of a case. The global features reflect an abstract characterization of potential repair flexibility for the whole schedule. High 'Resource Utilization Average', for example, often indicates a tight schedule without much repair flexibility.

Associated with the focal_activity are local features that we have identified and which potentially are predictive of the effectiveness of applying a particular repair tactic. For example, 'Predictive Shift Gain' predicts how much overall gain will be achieved by moving the current focal_activity earlier in its time horizon. In particular, it predicts the likely reduction of the focal_activity's waiting time when moved to the left within the repair time horizon. Because of the ill-structuredness of job shop scheduling, local and global features are heuristic approximations that reflect problem space characteristics.

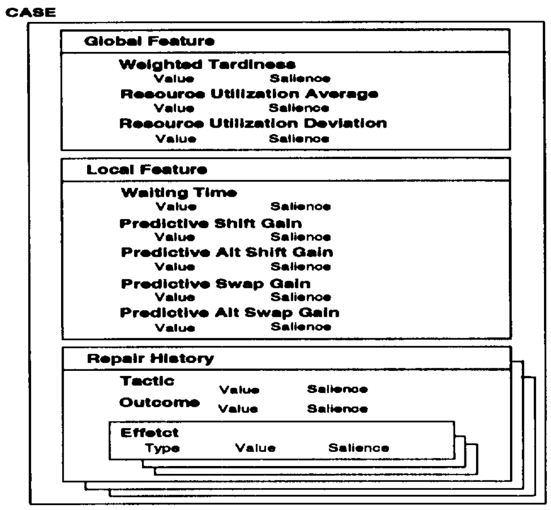

Figure 4: **CABINS Case Representation**

The repair history records the sequence of applications of successive repair actions, the repair outcome and the effects. Repair effect values describe the impact of the application of a repair action on scheduling objectives (e.g., weighted tardiness, WIP). A repair outcome is the evaluation assigned to the set of effects of a repair action and takes values in the set ['acceptable', 'infeasible', 'unacceptable']. Typically the outcome reflects tradeoffs among different objectives. The outcome of application of a repair tactic is 'infeasible', if the application of repair heuristic results in an infeasible schedule, i.e. a schedule that violates domain constraints. If the application of a repair tactic results in a feasible schedule, the result is judged as either acceptable or unacceptable with respect to the repair objectives by a domain expert. An outcome is 'acceptable' if the user accepts the tradeoffs involved in the set of effects for the current application of a repair action. Otherwise, it is 'unacceptable'. The effect salience is assigned when the outcome is 'unacceptable', and it indicates the significance of the effect to the repair outcome. This value is decided by a domain expert subjectively and interactively. The user's judgment as to balancing favorable and unfavorable effects related to a particular objective constitute the explanations of the repair outcome.

2.3 Case acquisition

To gather enough cases, sample scheduling problems are solved by a scheduler. CABINS identifies jobs that must be repaired in the initial sub-optimal schedule. Those jobs are sorted according to the significance of defect, and repaired manually by a user according to this sorting. For example, if the user's optimization criterion is to minimize order tardiness, the most tardy order is repaired first. The user selects a repair tactic to be applied. Tactic application consists of two parts: (a) identify the activities, resources and time intervals that will be involved in the repair, and (b) execute the repair by applying constraint-based scheduling to reschedule the activities identified in (a).

After tactic selection and application, the repair effects are calculated and shown to the user who is asked to evaluate the outcome of the repair. If the user evaluates the repair outcome as 'acceptable', CABINS proceeds to repair another focal_activity and the process is repeated. If the user evaluates the repair outcome as 'unacceptable', s/he is asked to supply an explanation in terms of rating the salience/importance of each of the effects. The repair is undone and the user is asked to select another repair tactic for the same focal_activity. The process continues until an acceptable outcome for the current focal_activity is reached, or the repair is given up. Repair is given up when there are no more tactics to be applied to the current focal_activity. When repair of the current focal_activity is given up, CABINS carries on repair of another activity. The sequence of applications of successive repair actions, the effects, the repair outcome, and the user's explanation for failed application of a repair tactic are recorded in the repair history of the case. In this way, a number of cases are accumulated in the case base.

2.4 Case re-use

Once enough cases have been gathered, CABINS repairs sub-optimal schedules without user interaction. CABINS repairs the schedules by (1) recognizing schedule sub-optimalities, (2) focusing on a focal_activity to be repaired in each repair cycle, (3) invoking CBR with global and local features as indices to decide the most appropriate repair tactic to be used for each focal_activity, (4) invoking CBR using the repair effect features (type, value and salience) as indices to evaluate the repair result, and (5) when the repair result seems unacceptable, deciding whether to give up or which repair tactic to use next. Experiments of using different indexing schema in case of failure are described in the following section.

In CABINS concepts are defined extensionally by a collection of cases. The similarity between i-th case and the current problem is calculated as follows :

$$exp(-\sqrt{\sum_{j=1}^{N}(SL_j^i \times \frac{CF_j^i - PF_j}{E\_D_j})^2})$$

where SL_j^i is the salience of j-th feature of i-th case in the case-base, and its value has been heuristically defined by the user. CF_j^i is the value of j-th feature of i-th case, PF_j is the value of j-th feature in the current problem, E_D_j is the standard deviation of j-th feature value of all cases in the case-base. Feature values are normalized by division by a standard deviation of the feature value so that features of equal salience have equal weight in the similarity function.

3 Learning control knowledge

In job shop schedule repair, we don't know how many different features are necessary to precisely predict the most successful tactic to be applied. But, since scheduling constraints are tightly interconnected the necessary number of features for fully representing a problem must be very large. Therefore, the number of features in the current case representation could be insufficient. But we should keep the number of features moderate, because if a case has a large number of features, the number of cases required for training increases drastically (dimensionality problem). Therefore, with moderate number of features and training cases, we can't avoid making some wrong predictions by using inductive learning. To compensate for the lack of a large number of case descriptive features, we can use failure experiences to derive useful information (i.e. retrieve more similar cases from the case base) to improve predictive accuracy.

Our hypothesis is that CBR enables CABINS to (1) learn a control model of repair action selection from cases that are created from superficial rules and (2) improve its competence both in repair quality and efficiency by utilizing failure information recorded in the cases. To analyze the correctness of our hypothesis, we divided cases into three categories: *immediate success cases* where the first application of a repair tactic was evaluated as success, *eventual success cases* where a focal_activity was repaired after several failed tactic applications, and *failure* cases where a focal_activity couldn't be repaired. We experimentally implemented the following three repair strategies: *One-shot repair*, *Exhaustive repair* and *Limited exhaustive repair*.

In One-shot repair, CABINS selects a repair tactic by retrieving the most similar immediate success cases, applies it to a focal_activity and proceeds to repair the next focal_activity regardless of repair outcome.

In Exhaustive repair, CABINS selects a repair tactic and applies it to repair a focal_activity. If the repair outcome is deemed either unacceptable or infeasible, another tactic is selected from eventual success cases to repair the same focal_activity, using as indices global and local case features, the failed tactic, and the indication of the failed outcome. This CBR invocation retrieves similar past failures of the tactic that were eventually successfully repaired and the tactic that was eventually successful in fixing the past failure. For example, when a repair result is judged as unacceptable by a user after application of the left_shift tactic, another case that has the most similar global and local features is retrieved from eventual success cases where left_shift has failed. The tactic that finally succeeded in repairing the selected case is retrieved and used to repair the current activity. The intuition here is that a similar outcome for the same tactic implies similarity of causal structure between the past and current case. Therefore, the eventually successful tactic of a similar failure can potentially be successful in the current problem.

In Limited exhaustive repair, CABINS gives up further repair when it determines that it would be a waste of time. To decide whether to give up further repair, failure cases are utilized in conjunction with immediate success cases and eventual success cases to determine case similarity. If the most similar case is a failure, CABINS gives up repair of the current focal_activity, and switches its attention to another focal_activity. Since, in difficult problems, such as schedule repair, failures usually outnumber successes, if both case types are weighted equally, overly pessimistic results could be produced (i.e., CBR suggests giving up too often.) To avoid this, we bias (negatively) the use of failure cases by placing a threshold [1] on the similarity value. Failure experiences whose similarity to the current problem is below this threshold are ignored in similarity calculations. Since the similarity metric selects the tactic which maximizes the sum of the most similar k (in current implementation k = 5) cases, this biases tactic selection in favor of success cases which are moderately similar to the current problem.

3.1 Experiments

To verify our hypothesis and compare the above three repair strategies, we performed 6 sets of con-

[1]Currently its value is heuristically fixed as 0.75.

trolled experiments where job shop schedule parameters, such as number of bottlenecks, range of due date, and activity durations were varied to cover a broad range of job shop scheduling problems. To ensure that we had not unintentionally hardwired knowledge of the problem into the solution strategies, we generated 10 job shop scheduling problems at random for each problem set.Each problem has 5 resources and 10 jobs of 5 activities each.

In the experiments reported here, we used minimizing weighted tardiness [2], as an objective function to evaluate the performance of CABINS. We built a rule-based reasoner (RBR) that goes through a trial-and-error repair process to optimize a schedule in terms of weighted tardiness based on the tactic selection rules acquired from a human scheduler. Since the RBR was constructed not to select the same tactic again after tactic failure, it could go through all the tactics [3] before giving up repairing an activity. For each repair, the repair effects are calculated and the repair outcome is correctly determined by comparing the change in the objective function. Since RBR knows the exact objective function for evaluation, it can work as an emulator of a human scheduler, who cannot repair a schedule in the most efficient way but can make consistent evaluations of repair results. Therefore, we used RBR not only for generating the case base for CABINS but also for making a comparison baseline for the CABINS experiment results to see whether CABINS can learn effective control rules from the cases made by an inefficient teacher. So far, CABINS has been trained with 1,000 cases by RBR.

To make an accurate determination of CABINS' capabilities, we applied a two-fold cross-validation method. Each problem set in each class was divided in half. One half was repaired by RBR to gather cases. These cases were used to iteratively repair the other half of the problem set. We repeated the above process interchanging the sample set and the test set. Our results are the average of the two sets of results using case-based repair.

3.2 Evaluation of three repair strategies

The graphs in figure 5 show comparative results with respect to schedule quality improvement (weighted tardiness) and repair efficiency (in CPU secs) among limited exhaustive repair, exhaustive repair, one-shot repair and rule-based repair, which is an emulation of repair by a human scheduler. The results show that one-shot repair is the worst in quality (even compared to rule-based repair) but best in efficiency. Exhaustive repair outperformed one-shot repair and rule-based repair in quality. But, the efficiency of exhaustive repair was worse than that of one-shot repair or rule-based repair. We attribute the increase in CPU time for exhaustive repair to two reasons: (1) greediness - exhaustive repair applies the tactic from the most similar success cases no matter how small their similarity is, and (2) stubbornness - exhaustive repair continues to repair the current focal_activity without giving up when the problem seems difficult.

The quality of repairs by limited exhaustive repair is only slightly worse than that by exhaustive repair, but is still comparable with that of rule-based repair. The efficiency of limited exhaustive repair is much higher than either rule-based repair or exhaustive repair. Although the efficiency of limited exhaustive repair is comparable with that of one-shot repair, the quality of repairs by limited exhaustive repair is much better than that of one-shot repair. One potential reason for these results is that, as described in section 2, the effects of schedule repair are pretty unpredictable and there is a good chance that another application of repair tactic may make the problem, which once seemed difficult, easier by changing the existing schedule fundamentally so that we can go back to the problem afterwards and repair it without wasting much effort. With respect to repair quality, we can observe the following: (1) immediate success cases alone do not have enough information to induce a complicated causal model of schedule repair process, and (2) prediction accuracy of repair tactic selection can be improved by using information about failed application of a repair tactic as additional index feature.

4 Conclusions

We described and experimentally validated a framework for acquisition and reuse of past problem solving experiences for control of plan revision in domains, such as job shop scheduling, that do not have a strong domain model. We examined various ways of exploiting past repair experiences in such a domain. Our experimental results show that our methodology can improve its own performance by: (1) using a failure experience as a contextual index of the problem, and (2) trading off the use of success and failure cases depending on the context in which a repair tactic is applied. This use of CBR in the space of failures is a domain independent method of acquiring control knowledge

[2]Of course, CABINS does not know this metric but has to guess it from the contents of the case base.

[3]The tactics currently available in CABINS are: left_shift, left_shift_into_alt, swap, swap_into_alt.

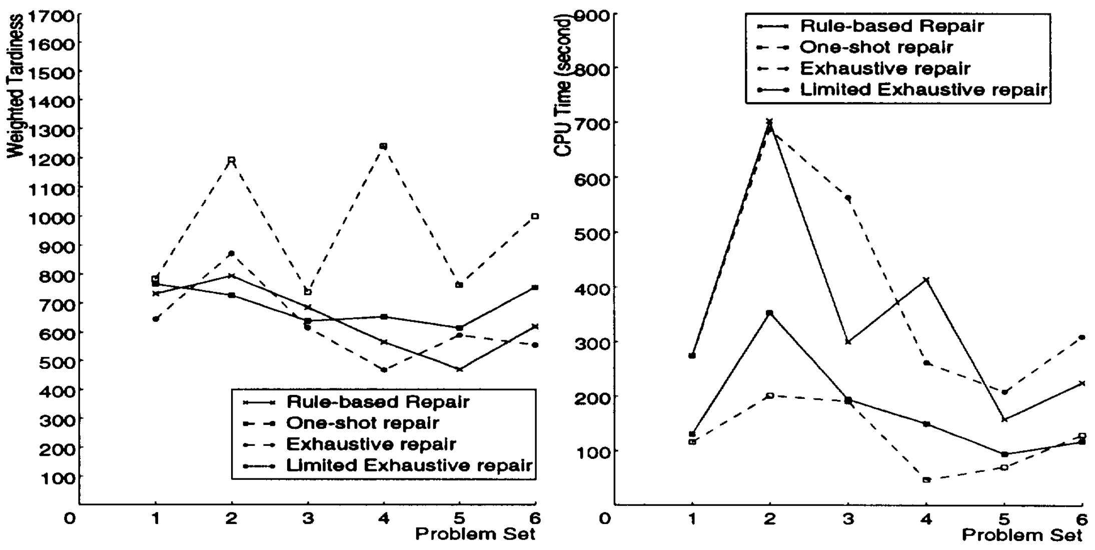

Figure 5: **Effect of repair strategies in quality and efficiency**

that allows the problem solver to improve its efficiency while preserving solution quality in domains without strong domain knowledge. We believe the CABINS approach can be effectively applied to a wide range of problems, such as layout design and personal calendar management[2].

References

[1] Ray Bareiss. *Exemplar-based knowledge acquisition : a unified approach to concept regression, classification, and learning.* Academic Press, New York, NY, 1989.

[2] Lisa Dent, Jesus Boticario, John McDermott, Tom Mitchell, and David Zabowski. A personal learning apprentice. In *Proceedings of the Tenth National Conference on Artificial Intelligence.* AAAI, 1992.

[3] Simon French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.* Ellis Horwood, New York, NY, 1982.

[4] Kristian J. Hammond. *Case-Based Planning : Viewing Planning as a Memory Task.* Academic Press, New York, NY, 1989.

[5] Subbarao Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.

[6] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings, Eighth National Conference on Artificial Intelligence*, pages 17–24, Boston, MA., 1990. AAAI.

[7] Kazuo Miyashita and Katia Sycara. Adaptive case-based control of schedule revision. In M. Fox and M. Zweben, editors, *Knowledge-Based Scheduling.* Morgan Kaufmann, San Mateo, CA, 1993.

[8] P. S. Ow, S. F. Smith, and A. Thiriez. Reactive plan revision. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 77–82, St-Paul, Minnesota, 1988. AAAI.

[9] Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1992.

[10] M. Zweben, E. Davis, D. Brian, E. Drascher, M. Deale, and M. Eskey. Learning to improve constraint-based scheduling. *Artificial Intelligence*, 58(1-3):271–296, 1992.