Improving Schedule Quality through Case-Based Reasoning

Kazuo Miyashita
Matsushita Electric Industrial Co. Ltd.,
2-7 Matsuba-cho, Kadoma, Osaka 571, JAPAN
miyasita@mcec.ped.mei.co.jp

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

Abstract

We describe a framework, implemented in CABINS, for iterative schedule revision based on acquisition and reuse of user optimization preferences to improve schedule quality. Practical scheduling problems generally require allocation of resources in the presence of a large, diverse and typically conflicting set of constraints and optimization criteria. The ill-structuredness of both the solution space and the desired objectives make scheduling problems difficult to formalize. CABINS records situation-dependent tradeoffs about repair actions and schedule quality to guide schedule improvement. During iterative repair, cases are exploited for: (1) repair action selection, (2) evaluation of intermediate repair results and (3) recovery from revision failures. The contributions of the work lie in experimentally demonstrating in a domain where neither the user nor the program possess causal knowledge of the domain that (a) taking into consideration failure information in the form of failed cases or a repair history of a case improves schedule quality, (b) schedule quality improves with increasing case size and (c) preserving the case base rather than inducing rules gives better results.

Introduction

Recently there has been increased interest in approaches that incrementally modify an artifact (e.g., program, plan, design) by reusing previous experiences in order to accommodate changed artifact specifications or recover from failures. Most current approaches have the following common characteristics: (1) they are motivated by considerations of computational efficiency (Kambhampati and Hendler, 1992; Veloso, 1992; Simmons, 1992), (2) they are concerned with preserving artifact correctness not addressing optimization issues (Kambhampati and Hendler, 1992; Veloso, 1992; Simmons, 1992; Hammond, 1989), and (3) they assume the existence of a strong domain model that is utilized to guide artifact modification and repair (Kambhampati and Hendler, 1992; Veloso, 1992; Simmons, 1992; Hammond, 1989). For example, CHEF (Hammond, 1989) uses rules rather than CBR for repair tactic selection, uses a model-based simulator for detecting failures in a generated plan, and addresses plan correctness issues (recovery from a failed plan) but ignores issues of plan optimization. Such characteristics limit current approaches in their ability to handle interesting real world tasks since the existence of a strong domain model can almost never be assumed and improving artifact quality (as opposed to only correctness) in terms of a set of evaluation criteria is often a crucial consideration.

We present an approach, implemented in CABINS, that demonstrates that reuse of previous relevant experiences is effective not only to ensure artifact correctness but also to improve quality. Through case-based reasoning (CBR), CABINS learns two categories of concepts: (1) what heuristic repair actions to choose in a particular repair context, and (2) what combinations of effects of application of a particular repair action constitutes an acceptable or unacceptable repair outcome in terms of optimization criteria. In contrast to the knowledge acquisition task (Bareiss, 1989) where the program interacts with an expert teacher to acquire domain knowledge, in our approach neither the user nor the program possess causal domain knowledge. The user cannot predict the effects of modification actions on artifact correctness or quality. In the domain of scheduling, for example, a modification could result in worsening schedule quality or introducing constraint violations (see next section). The user's 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 improvement of job shop schedules. Experimental results reported in (Miyashita and Sycara, 1993) have shown that CABINS substantially increased schedule quality along a variety of optimization criteria (improvements ranged from 30-70 percent) without undue degradation in efficiency as compared with (1) a

state of the art constraint-based scheduler, and (2) a variety of well regarded dispatch heuristics that are used in production management. In contrast to approaches that utilize a single repair heuristic (Minton *et al.*, 1990) or use a statically predetermined model for selection of repair actions (Ow *et al.*, 1988), our approach utilizes a repair model that is incrementally learned and encoded in the case base. Learning allows dynamic selection and application of repair actions depending on the repair context. In (Zweben *et al.*, 1992) plausible explanation based learning has been successfully used to learn schedule repair control rules for speed up. Our experimental results show that in the context of CABINS, keeping the case base rather than inducing rules gives better results in terms of schedule quality.

In this paper we experimentally demonstrate that (a) taking into consideration failure information improves performance results, (b) result quality improves with increasing case size and (c) preserving the case base rather than inducing rules gives better results.

Task Domain

Scheduling assigns a set of jobs to a set of resources with finite capacity over time. One of the most difficult scheduling problem classes is job shop scheduling. Job shop scheduling is a well-known NP-complete problem (French, 1982). In job shop scheduling, each job consists of a *set of activities* to be scheduled according to a *partial activity ordering*. Each job is assigned a release date, the date that it will be ready for starting processing, and a due date, a date on which the job should finish. Each activity within a job is assigned a set of *substitutable* resources on which the activity can be performed, and an activity duration. For example a drilling activity could be performed utilizing either a drilling machine or a milling machine. 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 minimization of job tardiness (i.e., how late a job will finish), minimization of weighted tardiness (the sum of tardiness of all jobs, each weighted by its importance), minimization of work in process inventory (WIP) (i.e., the time a job spends in a factory waiting to be processed), maximization of resource utilization, etc.

CABINS *incrementally revises a complete 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 (a) predict in advance either the extent of the constraint violations resulting from a repair action, or the nature of the conflicts. or (b) judge a priori the effects of a repair action on the optimization objectives. 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 preferences* with respect to tradeoffs. For example, WIP and weighted tardiness are not always compatible with each other. There are situations where WIP is reduced, but weighted tardiness increases. Tradeoffs are context dependent and therefore difficult to fully describe a priori even for a human expert. In CABINS, evaluation feedback is used to incrementally acquire context dependent schedule evaluation tradeoffs and their justifications. These are recorded in the case base and can be re-used to evaluate future schedule revision outcomes. Hence, preferences are reflected in the case base in two ways: as *preferences for selecting a repair action*, and as *evaluation preferences* for the repair outcome that resulted from selection and application of a specific repair action.

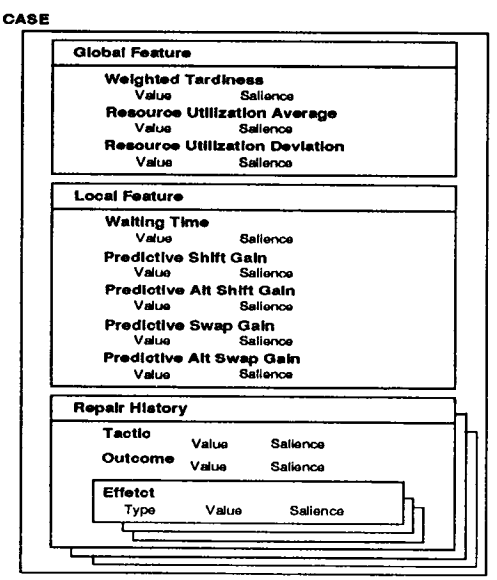

Figure 1: CABINS Case Representation

Overview of CABINS

Case representation

Within a job, repair is performed one activity at a time. At each iteration, the current job whose activity is being repaired is called the *focal_job* and the current activity being repaired is called the *focal_activity*. A case describes the application of a particular modification to an activity. Case indices are of three types (figure 1). First, there are features that reflect potential repair flexibility for the schedule as a whole, (global features). High resource utilization, for example, often indicates a tight schedule without much repair flexibility. High standard deviation of resource utilization indicates the presence of highly contended-for resources which in turn indicates low repair flexibility. Second, there are features that reflect flexibility for schedule revision within limited temporal bounds (local features). In particular, the temporal bound that CABINS uses is a time interval called *repair time horizon*. The repair time horizon of a focal_activity is the time interval between the end of the activity preceding the focal_activity in the same focal_job and the end of the focal_activity (see figure 2).

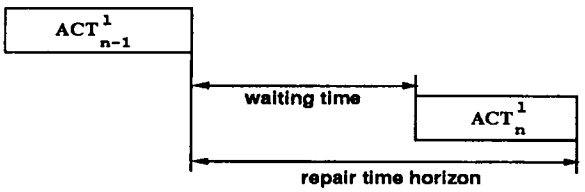

Figure 2: Repair time horizon of focal_activity(ACT_n^1)

Associated with the repair time horizon are local features that we have identified and which potentially are predictive of the effectiveness of applying a particular repair tactic. These features are in the same spirit as those utilized in (Ow *et al.*, 1988). 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.

The third category of case indices is a set of features that reflect the sequence of revisions to an activity (repair history). The repair history records the sequence of applications of successive repair actions, the repair effects and the repair outcome. Repair effects describe the impact of the application of a repair action on schedule optimization objectives (e.g., weighted tardiness, WIP). Typically these effects reflect tradeoffs among different objectives. A repair outcome is the evaluation assigned to the set of effects of a repair action and takes values in the set ['acceptable', 'unacceptable']. This judgement is made in the training phase and gets recorded in the case base. An outcome is 'acceptable' if the tradeoffs involved in the set of effects for the current application of a repair action is judged acceptable. If, during case acquisition, the outcome is judged as "unacceptable", the application of the repair tactic is considered a failure and an explanation that expresses tradeoffs with respect to balancing favorable and unfavorable outcomes on optimization objectives is provided. If during CBR repair the repair outcome is deemed unacceptable, another tactic is selected from success cases to repair the same 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 successfully repaired and the tactic that was eventually successful in fixing the past failure. 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.

Case acquisition

To gather enough cases, sample scheduling problems are solved by a constraint-based scheduler (Sadeh and Fox, 1990). CABINS identifies jobs in a schedule that must be repaired. Those jobs are sorted according to the significance of defect, and repaired according to this sorting. For example, if the optimization criterion is to minimize job tardiness, the most tardy job is repaired first. A repair tactic is selected 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). Repairing an activity, i.e., unscheduling it from its current position and rescheduling at another time interval may cause conflicts with other activities. In each tactic application, the focal_activity and the conflicting activities are all rescheduled. For details of the approach, see (Miyashita and Sycara, 1993).

The tactics currently available in CABINS are:

left_shift : try to move the focal_activity on the *same* resource as much to the left on the timeline as possible within the repair time horizon, so as to minimize the amount of resource capacity contention created by the move.

left_shift_into_alt : try to move the focal_activity on a *substitutable* resource as much to the left on the timeline as possible within the repair time horizon, so as to minimize the amount of resource capacity contention created by the move.

swap : swap the focal_activity with the activity on the *same* resource within the repair time horizon which

causes the least amount of precedence constraint violations.

swap_into_alt : swap the focal_activity with the activity on a *substitutable* resource within the repair time horizon which causes the least amount of precedence constraint violations.

After tactic selection and application, the repair effects are calculated and evaluated. For example, repair of the current focal_activity may decrease WIP by 200 units and decrease weighted tardiness of the focal_job by 180 units while at the same time increasing weighted tardiness of another job by 130 units and increasing WIP by 300 units. If the repair outcome is evaluated as 'acceptable', CABINS proceeds to repair another activity and the process is repeated. If the evaluation of the repair outcome is "unacceptable", an explanation is supplied, the repair is undone and another repair tactic is selected for the same focal_activity. The process continues until an acceptable outcome for the current focal_activity is reached, or failure is declared. Failure is declared when there are no more tactics to be applied to the current focal_activity. The sequence of applications of successive repair actions, the effects, the repair outcome, and the 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.

In the experiments reported here, we used a simple metric, minimizing weighted tardiness, [1] as an objective function to evaluate the performance of CABINS. Although there is no straightforward way to modify a schedule to optimize a realistic multi-criteria objective function, by using a single-criterion objective we built a rule-based reasoner (RBR) that goes through a trial-and-error repair process to optimize a schedule and forms an experimental baseline against which to compare CABINS. Since the RBR is constructed not to select the same tactic after tactic failure, it could go through all the tactics 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 a clearly-defined objective function (which is available only in a user's mind) was used for evaluation, RBR can work as an emulator of a human scheduler, whose expertise lies in the ability of consistent evaluation. Therefore, we used RBR not only to make a comparison baseline for the CABINS experiment results but also to generate the case base for CABINS. So far, CABINS has been trained with 1,000 cases.

Once a case base is created, CABINS can repair a suboptimal schedule through CBR. CABINS repairs a schedule by (1) recognizing schedule suboptimalities, (2) focusing on an 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 activity, (4) invoking CBR using the repair effect features (type, value and salience) as indices to evaluate the repair result, and (5) in case of failure, deciding whether to give up or which repair tactic to use next by using global and local features and the repair history as indices. In the experimental study section, we report results about the effectiveness of indexing schemes that in situations of failure utilize different types of failure information.

[1] Of course, CABINS does not know this metric but had to induce it from the contents of the case base.

Case retrieval

In CABINS concepts are defined extensionally by a collection of cases. As a case retrieval mechanism, CABINS uses a variation of k-Nearest Neighbor method (k-NN). (Dasarathy, 1990) where not the frequency but the sum of similarity of k-nearest neighbors is used as a selection criterion. 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. Salience and values of features are numeric and have 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 a 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.

Experimental Studies

To evaluate CABINS, we performed a set of controlled 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 60 job shop scheduling problems at random from problem generator functions where the above problem parameters were varied in controlled ways. Each problem has 5 resources and 10 jobs of 5 activities each. Each job has a process routing specifying a sequence where each job must visit bottleneck resources after a fixed number of activities, so as to increase resource contention and make the problem more difficult. We also varied job due dates and release dates, as well as the number of bottleneck resources (1 and 2). Six groups of 10 problems each were randomly generated by considering three different values of the due date range parameter (static, moderate, dynamic), and two values of the bottleneck configuration (1 and 2 bottleneck problems). The slack was adjusted as a function of the due

date range and bottleneck parameters to keep demand for bottleneck resources close to 100 percent over the major part of each problem. Durations for activities in each job were also randomly generated. These problems are variations of the problems originally reported in (Sadeh, 1991). Our problem sets are different in two respects: (a) we allow substitutable resources for non-bottleneck resources, and (b) the due dates of jobs in our problems are more constrained by 20 percent.

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 the RBR emulator 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.

Evaluation of three repair strategies

Our hypothesis is that CBR enables CABINS to improve its competence both in repair quality and efficiency compared with RBR by utilizing different types of failure information recorded in the cases.

We experimentally compared three repair strategies:

(1) *one-shot repair*, where CABINS selects a repair tactic, applies it to a focal_activity and proceeds to repair the next focal_activity regardless of repair outcome.

(2)*exhaustive repair*, where CABINS selects a repair tactic and applies it to repair an activity. If the repair outcome is deemed unacceptable, another tactic is selected from success cases to repair the same 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 successfully repaired and the tactic that was eventually successful in fixing the past failure. The intuition here is that similar outcome for the same tactic imply 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.

(3) *limited exhaustive repair*, where CABINS gives up further repair when it determines that it would be a waste of time. To decide whether to give up further repair, previous repair failed cases are utilized in conjunction with repair successes to determine case similarity. If the most similar case is a failure, CABINS gives up repair of the current activity and switches its attention to another 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) usage of failures by placing a threshold 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 cases, this biases tactic selection in favor of success cases which are moderately similar to the current problem.

The graphs in figure 3 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. 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 believe that this result stems from the following two reasons: (1) greediness - exhaustive repair applies the tactic from the most similar cases no matter how small their similarity is, and (2) stubbornness - exhaustive repair continues to repair an activity without giving up when the problem seems difficult. The quality of repair 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 both rule-based repair and 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. With respect to repair quality, we can observe the following: (1) one shot repair does not have enough information to induce an adequate repair model, and (2) prediction accuracy can be improved by using information about failed application of a repair tactic as an additional index feature.

Comparison with different sized case-bases

The graphs in figure 4 show the comparison of CABINS' performance with different sized case-bases. In the experiments, we randomly chose half of the cases in the original case-base (used in the comparative repair strategy experiments) and created a new case-base. Then, we solved the same sixty problems by limited exhaustive repair with each of the case-bases. The graphs depict that CABINS with full case-base outperforms CABINS with half case-base both in quality and efficiency. This means that, as the case base of CABINS is enriched, its competence increases.

Comparison of CBR and rule induction

We tested the hypothesis that keeping the cases rather than inducing rules for repair tactic selection would result in better quality repairs. The graphs in figure 5 show the comparison of CABINS' performance with case-based reasoning and CABINS' performance with rules induced from the case base by C4, a decision tree induction algorithm (descendant of ID3) (Quinlan, 1993). The results show that CABINS'

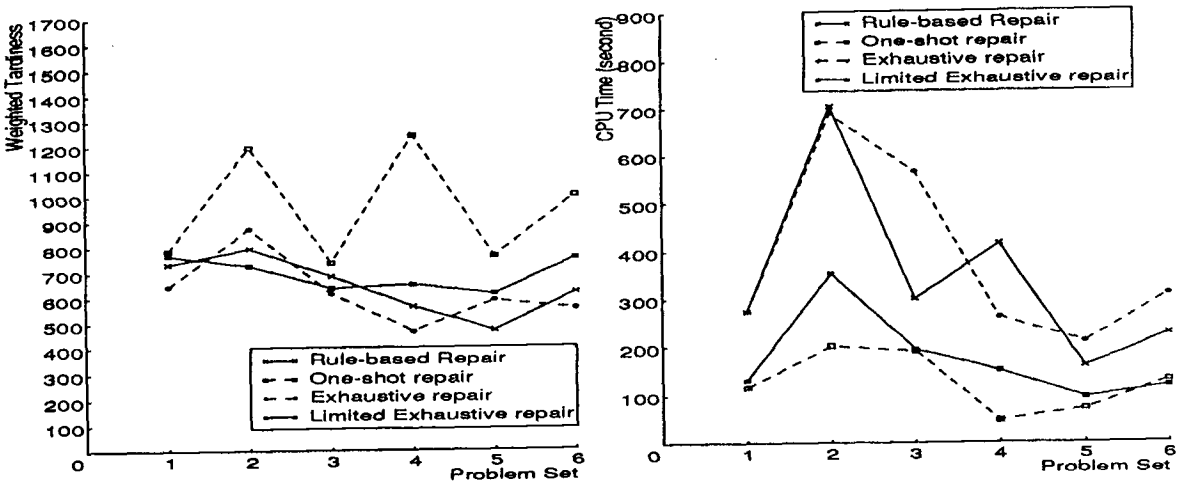

Figure 3: Effect of repair strategies in quality and efficiency

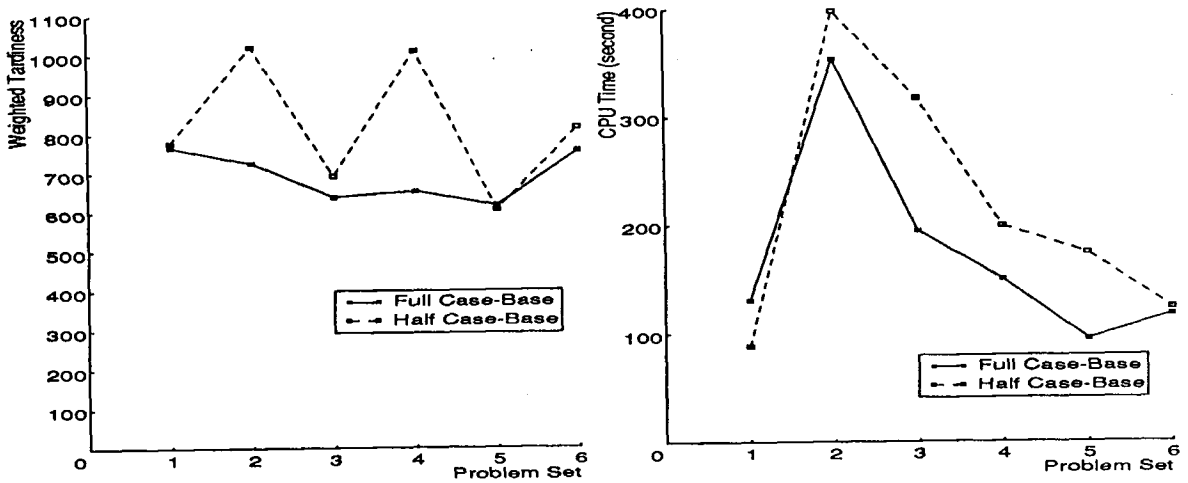

Figure 4: Effects of case-base size in quality and efficiency

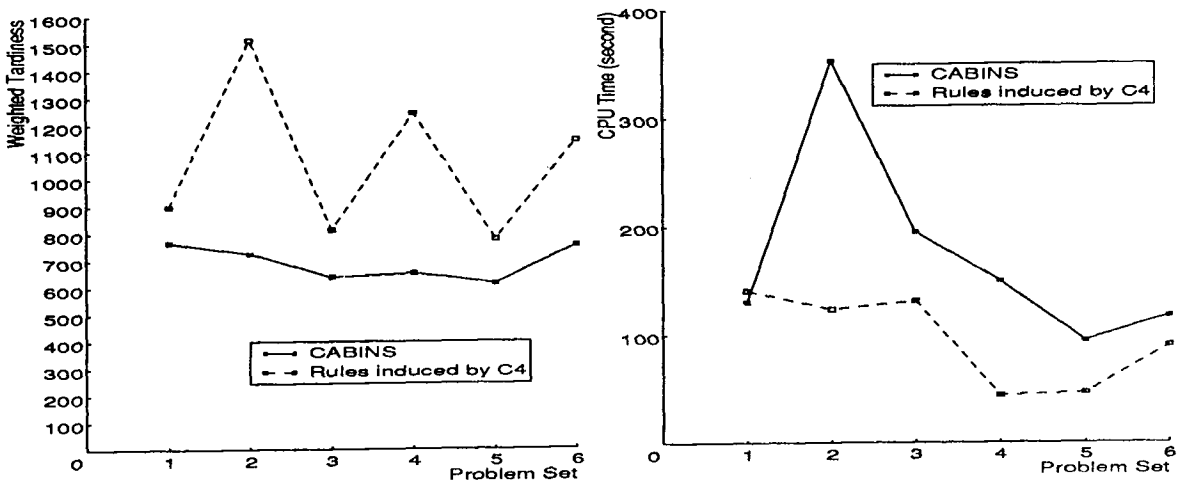

Figure 5: Comparison of CABINS and C4 in quality and efficiency

performance with C4-induced rules is better with respect to efficiency but much poorer in terms of quality than CABINS' performance with case-base. This drawback of C4 stems from the fact that geometrically, C4 (and most of other decision-tree induction programs) can't produce nonrectangular decision regions in the decision space. In the rules used for repairing schedules and creating cases, there are many conditions specifying the relationships of attributes, such as `If attribute-A is greater than attribute-B, then C`. To approximate the decision behavior of the nonrectangular regions produced by those rules, C4 has to fit many small rectangle sections in the form of a staircase function, which requires more training data.(Weiss and Kulikowski, 1990; Quinlan, 1993)

Conclusions

We described a framework for acquisition and reuse of past problem solving experiences for plan revision in domains, such as job shop scheduling, without a strong domain model. Our experimental results show that our methodology can outperform rule based methods, and improve its own performance by: (1) using failures and their repairs as additional indices, and (2) trading off the use of success and failure cases depending on the context in which a repair tactic is applied. In addition, our experimental results showed that increasing case base size improves both quality and efficiency. Finally, CBR techniques used in CABINS, though lower in efficiency, result in superior solution quality compared with rule induction.

Acknowledgments

This work was done when the first author was a visiting scientist at the Robotics Institute of Carnegie Mellon University under the support of Matsushita Electric Industrial Co.,. C4 program used in this paper was implemented by Dr. Wray Buntine at NASA Ames Research Center as a module of his IND software package.

References

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

Dasarathy, Belur V., editor 1990. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, CA.

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

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

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

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

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

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

Quinlan, J. Ross 1993. *C4.5: programs for machine learning.* Morgan Kaufmann Publisher, Inc., San Mateo, CA.

Sadeh, Norman and Fox, Mark S. 1990. Variable and value ordering heuristics for activity-based job-shop. In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, Hilton Head Island, SC. 134–144.

Sadeh, Norman 1991. *Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling.* Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University.

Simmons, Reid G. 1992. The roles of associational and causal reasoning in problem solving. *Artificial Intelligence* 53:159–207.

Veloso, Manuela M. 1992. *Learning by Analogical Reasoning in General Problem Solving.* Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University.

Weiss, Sholom M. and Kulikowski, Casimir A. 1990. *Computer Systems That Learn : Classification and Prediction Methods from Statistics, Neural Nets, Machine Learnig and Expert Systems.* Morgan Kaufmann Publisher, Inc., San Mateo, CA.

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