Learning from Failure Experiences in Case-Based Schedule Repair

Katia Sycara

katia@cs.cmu.edu
The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
(412)621-8825

Kazuo Miyashita

miyasita@mcec.ped.mei.co.jp
Matsushita Electric Industrial Co. Ltd.

2-7 Matsuba-cho
Kadoma, Osaka 571, JAPAN
+81-6-905-4809

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. In CABINS, case-based reasoning is used for eliciting situation-dependent user's tradeoffs about repair actions and schedule quality to guide schedule revision for quality improvement. During iterative repair, cases are exploited for multiple purposes, such as (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 taking into consideration failure information improves the efficiency of rather costly 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 that incrementally modify an artifact (e.g., program, plan, design) by reusing previous experiences [1, 2, 3, 4] in order to accommodate changed artifact specifications or recover from execution failures. 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 artifact correctness, and (3) they assume the existence of a strong domain model that is utilized to guide artifact 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 artifact quality (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 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. This distinguishes CABINS from previous case-based systems. For example, CHEF assumes existence of rules for repair tactic selection and a model-based simulator for detecting failures in a derived plan. In contrast to the knowledge acquisition task [5] 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 the artifact. 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. In contrast to approaches that utilize a single repair heuristic [6]

1060-3425/94 $3.00 © 1994 IEEE

Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences, 1994

or use a statically predetermined model for selection of repair actions [7], 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 [8] 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.

CABINS provides a unified framework for predictive schedule improvement, reactive schedule management in response to unforeseen events during schedule execution, and interactive capture of user context-dependent preferences and re-use of the learned knowledge to differentially select repair actions. Experimental results reported in [9] have shown that CABINS substantially increased schedule quality in predictive and reactive situations 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 this paper we experimentally demonstrate that taking into consideration repair failure information in multiple ways improves schedule repair results in terms of quality and efficiency.

1.1 Task Domain

Scheduling assigns a set of tasks 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 [10]. 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. Each job is assigned a release date, the date that it will be ready for starting processing, a due date, the date on which the job should finish, and a set of durations of each job activity. 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

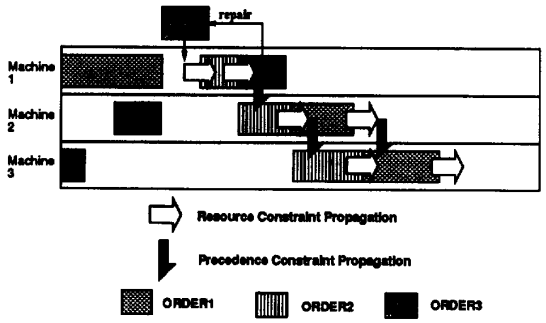

Figure 1: Example of tight interactions

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, minimize cycle time etc.

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. (see figure 1) 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. Tradeoffs are context-dependent and therefore difficult to describe in a simple manner [11]. In CABINS, user feedback 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

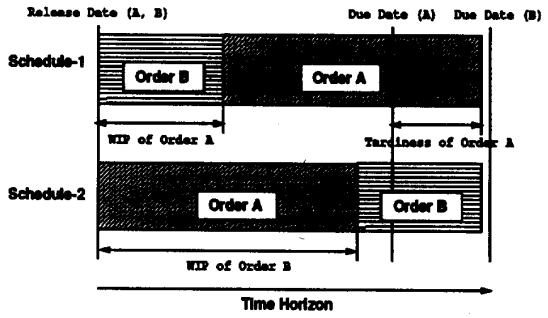

Figure 2: Example of conflicting objectives

selection and application of a specific repair tactic.

2 Overview of CABINS

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-usage) module.

To generate an initial schedule, CABINS uses a state-of-the-art constraint-based scheduler [12]. But the 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. The evaluation categories currently employed are 'acceptable' and 'unacceptable'.
- 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 case acquisition is that, in spite of ill-structuredness of the problem, the following

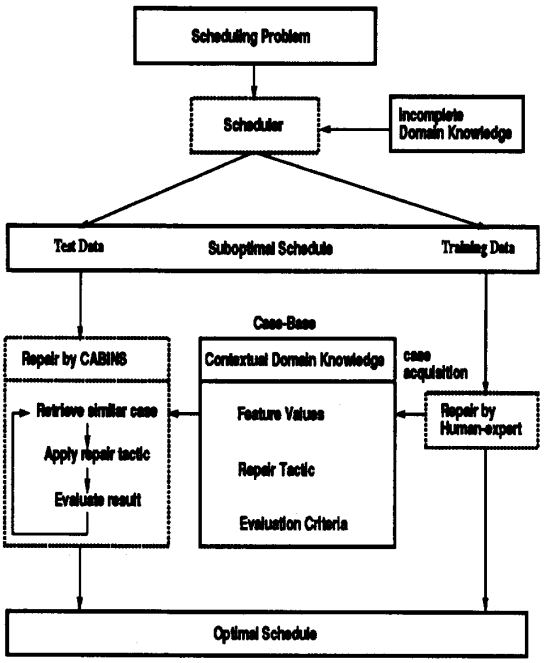

Figure 3: CABINS Architecture

three types of domain knowledge constitute useful case features.

- Repair heuristics : a set of local patching heuristics that can be applied to a sub-optimal 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 are of two types, local and global and will be described in detail in section 2.1.
- 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.

Once a case base is created, CABINS can repair sub-optimal schedules automatically by using case-based reasoning in selection of repair tactics and in evaluation of repaired results.

2.1 Case Representation

Within a job, repair is performed on one activity, the *focal_activity* at a time. In CABINS, a case describes the application of a particular modification to a

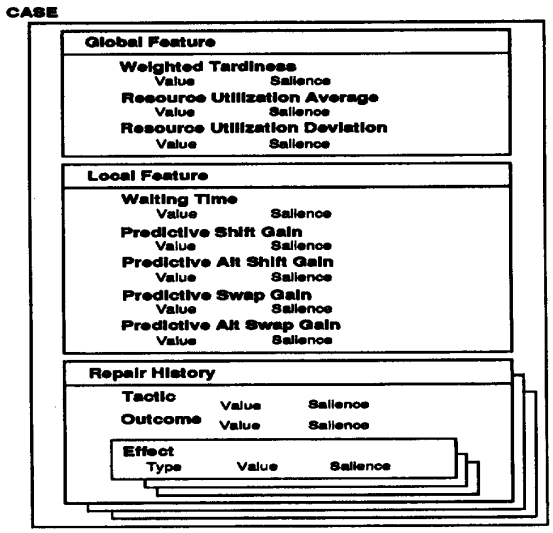

Figure 4: CABINS Case Representation

focal_activity. Each case is indexed in terms of surface features relating to the flexibility of temporal and capacity constraints surrounding the focal_activity, the repair tactic used, repair effects and the repair outcome. 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, 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.

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. These features are in the same spirit as those utilized in [7]. 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 repair history records the sequence of applications of successive repair actions, the repair outcome and the effects. The repair history is used as a record of evidences that show an existence of a certain causal structure in a problem implicitly. 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.

2.2 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 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). Repairing an activity, i.e., unscheduling it from its current position and re-scheduling at another time interval may cause conflicts with other activities. In each tactic application, the conflicting activities are all re-scheduled. For details of the approach, see [9].

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. For example, repair of the current focal_activity may decrease WIP by 200 units and decrease weighted tardiness of the focal_order by 180 units while at the same time increasing weighted tardiness of another order by 130 units and increasing WIP by 300 units. If the user evaluates the repair outcome as 'acceptable', CABINS pro-

ceeds 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 failure is declared. Failure is declared when there are no more tactics to be applied to the current focal_activity [1]. 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.

2.3 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) in case of failure, deciding whether to give up or which repair tactic to use next by using global and local features and a list of repair history as indices. 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. As a case retrieval mechanism, CABINS uses a variation of k-Nearest Neighbor method (k-NN). [13] 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.

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

3 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 [12]. 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.

In the experiments reported here, we used a simple metric, minimizing weighted tardiness [2], 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

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

compare CABINS. Since the RBR is constructed not to select the same tactic again 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 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.

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.

3.1 Evaluation of Three Repair Strategies

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 (in time horizon and allocatable resources), 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 (i.e., 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 effectiveness of case based schedule repair, 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 compared the following three repair strategies:

- *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.

- *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. 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.

- *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) usage of failure cases 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.

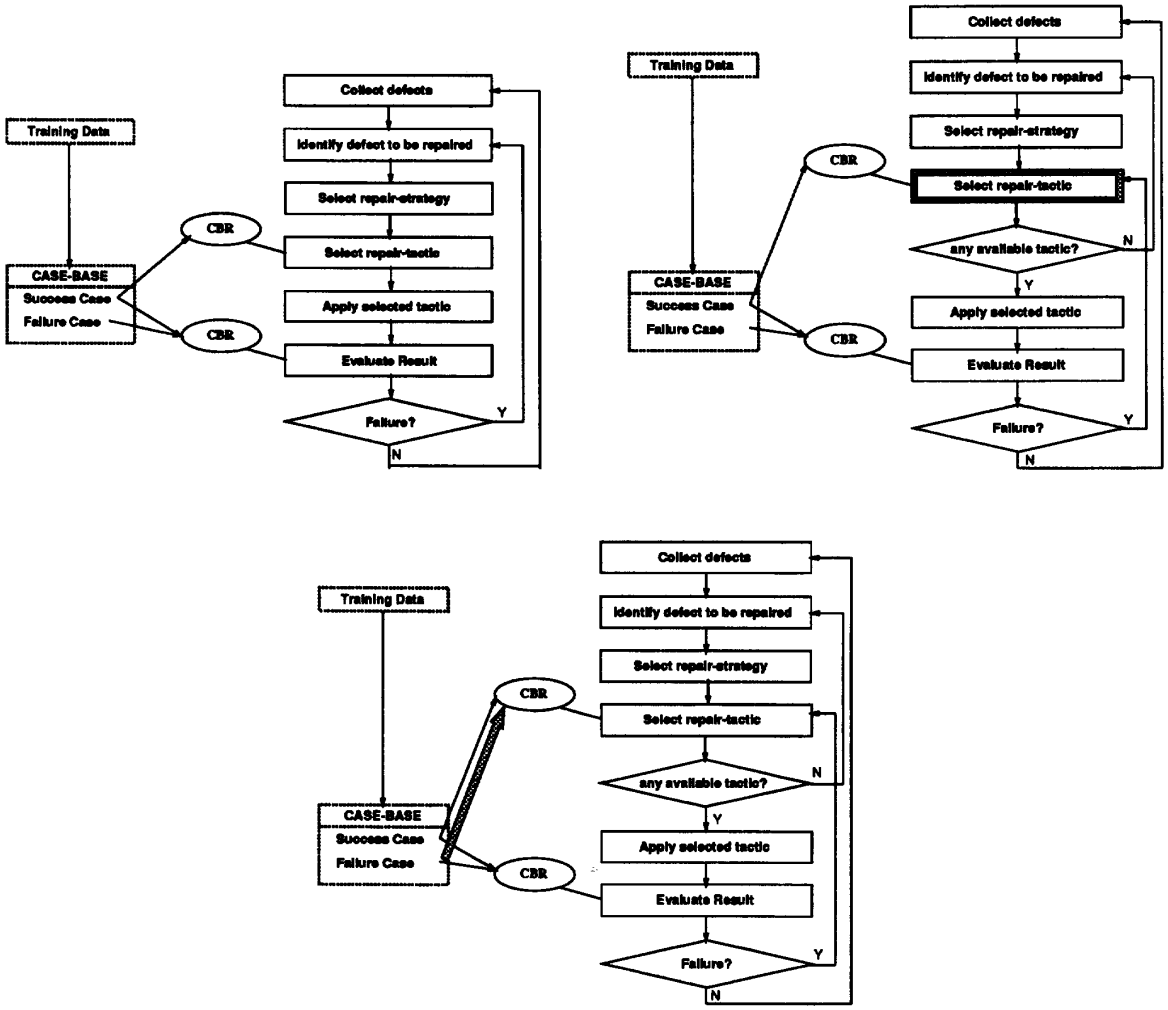

Figure 5: Three repair strategies compared

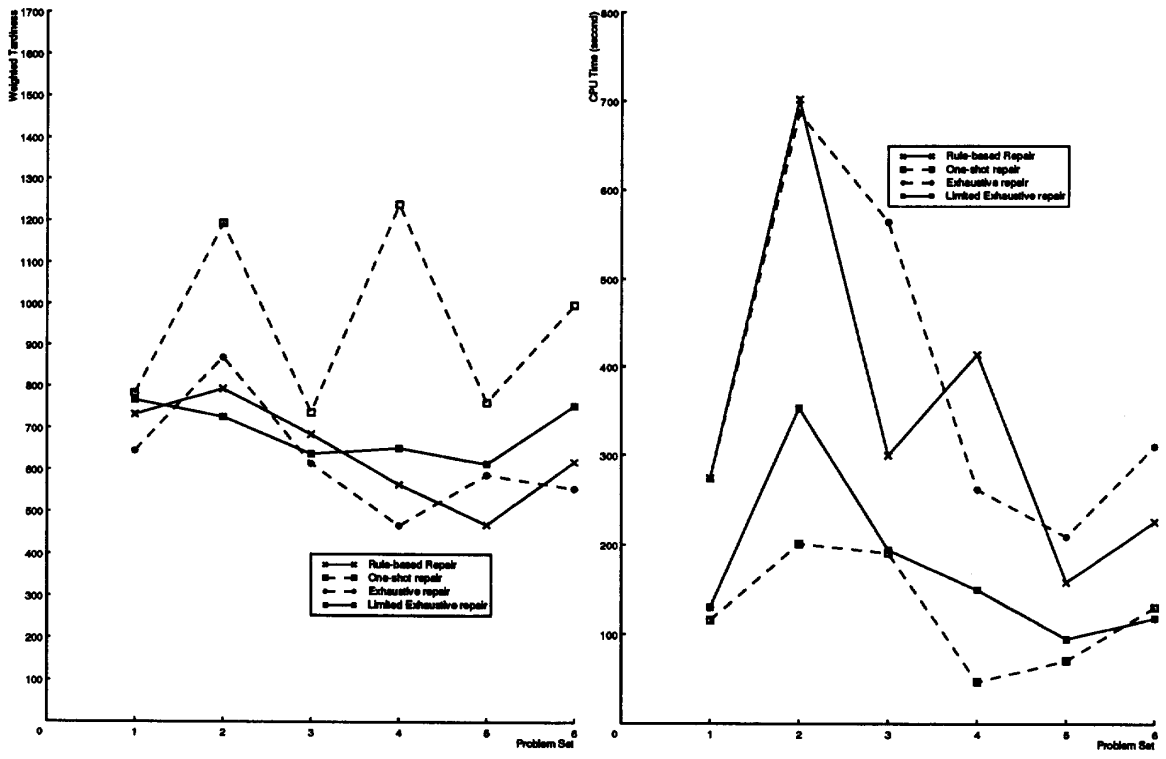

Figure 6: Effect of repair strategies in quality and efficiency

The flow chart of each repair strategy is shown in figure 5.

The graphs in figure 6 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 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 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. One potential reason for these effects is that, as we described in section 1.1, the effects of schedule repair is 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 a framework for acquisition and reuse of past problem solving experiences for plan revision in domains, such as job shop scheduling, that do not have a strong domain model. We examined various ways of exploiting failure information in such a domain. Our experiment results show that our methodology can improve its own performance by: (1) using failure experience as 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 failure recovery that allows the problem solver to access past solutions to the failure without strong domain knowledge.

References

[1] S. Kambhampati and J. A. Hendler, "A validation-structure-based theory of plan modification and reuse," *Artificial Intelligence*, vol. 55, pp. 193–258, 1992.

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

[3] R. G. Simmons, "The roles of associational and causal reasoning in problem solving," *Artificial Intelligence*, vol. 53, pp. 159–207, 1992.

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

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

[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*, (Boston, MA.), pp. 17–24, AAAI, 1990.

[7] P. S. Ow, S. F. Smith, and A. Thiriez, "Reactive plan revision," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, (St-Paul, Minnesota), pp. 77–82, AAAI, 1988.

[8] M. Zweben, E. Davis, B. Daun, E. Drascher, M. Deale, and M. Eskey, "Learning to improve constraint-based scheduling," *Artificial Intelligence*, vol. 58, no. 1-3, pp. 271–296, 1992.

[9] K. Miyashita and K. Sycara, "A framework for case-based revision for schedule generation and reactive schedule management," *Journal of Japanese Society for Artificial Intelligence*, Submitted.

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

[11] K. Miyashita and K. Sycara, "Cabins : Case-based interactive scheduler," in *Working Notes of AAAI Spring Symposium on Practical Approaches to Scheduling and Planning*, (Stanford, CA), AAAI, 1992.

[12] N. Sadeh, *Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[13] B. V. Dasarathy, ed., *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* Los Alamitos, CA: IEEE Computer Society Press, 1990.