ELSEVIER

0954-1810(95)00001-1

*Artificial Intelligence in Engineering* 9 (1995) 277–287
Elsevier Science Limited
Printed in Great Britain
0954-1810/95/$09.50

# Case-based knowledge acquisition for schedule optimization

**Kazuo Miyashita**

*Electrotechnical Laboratory, 1-1-4, Umezono, Tsukuba, Ibaraki 305, Japan*

In recent years, there have been a lot of efforts in solving scheduling problems by using the techniques of artificial intelligence (AI). Through development of a variety of AI-based scheduling systems, it became well known that eliciting effective problem-solving knowledge from human experts is a difficult task, and human schedulers typically lack the knowledge of solving large and complicated scheduling problems in the sophisticated manner. In this paper, our case-based approach, implemented in the system called CABINS, is presented for capturing a human expert's preferential criteria about schedule quality and control knowledge to speed up problem solving. By iterative schedule repair, CABINS improves the quality of sub-optimal schedules, and during the process CABINS utilizes past repair experiences for (1) repair tactic selection and (2) repair result evaluation. It is empirically demonstrated that CABINS can optimize a schedule along objectives captured in its case base and improve the efficiency of optimization process while preserving the quality of a resultant schedule.

*Key words:* job shop scheduling, case-based learning, optimization, preferential knowledge, search control knowledge.

## 1 INTRODUCTION

Scheduling is a task to assign a set of jobs to a set of resources with finite capacity over time. The goal of scheduling is to produce schedules that respect all constraints and optimize a set of objectives. Scheduling problems have been formulated as a type of combinatorial optimization problems and tackled by several mathematical approaches such as dynamic programming, branch and bound methods and integer programming.[1]

In recent years, the advancement of artificial intelligence technologies has initiated the development of knowledge-based scheduling systems. But, development of knowledge-based systems usually requires a knowledge engineer to inquire of domain experts about their knowledge and skills on the domain and encode them in a computationally operational form (e.g. if–then rules). This step is regarded as a major bottleneck in the system development process, since it is an arduous job for knowledge engineers without background knowledge of the domain to ask the right questions of domain experts and understand their answers correctly. Many systems and methodologies are proposed to assist knowledge engineers in knowledge acquisition and actually used for the development of some knowledge-based scheduling systems. In scheduling problems, the expertise of human experts itself often becomes the subject of controversy. Through past experiences of developing knowledge-based scheduling systems, it has been pointed out that a human scheduler does not have sufficient knowledge for making a good schedule efficiently.[2] Therefore, to obtain useful knowledge for efficient problem solving, several machine learning techniques are applied in the domain of planning and scheduling.[3]

In this paper, the authors first describe the difficulties of job shop scheduling problems and explain past research efforts concerning knowledge acquisition/machine learning in scheduling problems. Then, they describe their case-based schedule revision methodology, implemented in the system called CABINS, for knowledge acquisition and optimization in the scheduling domain, and empirically demonstrate its effectiveness.

## 2 JOB SHOP SCHEDULE OPTIMIZATION

The job shop scheduling problem is one of the most difficult NP-hard combinatorial optimization problems.[1] Job shop scheduling deals with allocation of a limited set of resources to a number of activities (operations) associated with a set of jobs so as to respect given temporal relations (e.g. precedence relations among activities), temporal constraints (e.g. job

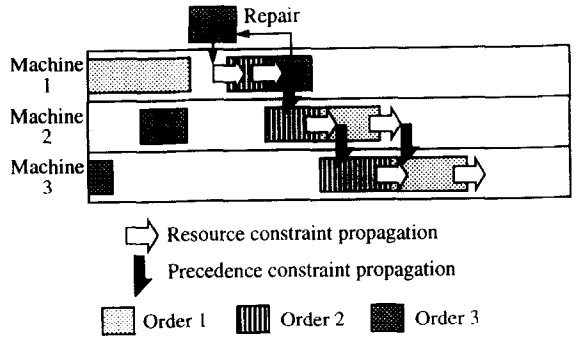


**Fig. 1.** Example of tight constraint interactions.

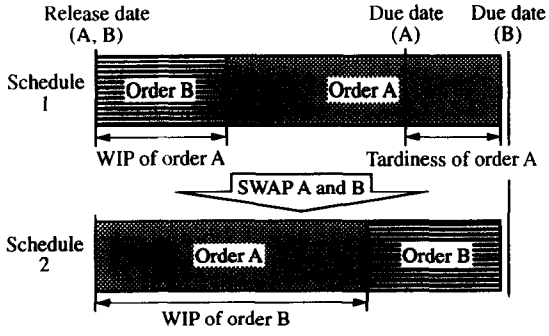


**Fig. 2.** Example of conflicting objectives.

release and due dates) and resource capacity restrictions in order to optimize a set of objectives, such as minimize tardiness, minimize work in process inventory (WIP), maximize resource utilization, etc. Due to the tight interactions among scheduling constraints and the often conflicting nature of optimization criteria, it is impossible to assess with any precision the extent of schedule revision or the impact of a scheduling decision on the global satisfaction of optimization criteria. For example, in Fig. 1, moving forward the last activity 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. In addition, the evaluation itself of what is a 'high quality' schedule is difficult because of the need to balance conflicting objectives and trade-off among them. Such tradeoffs typically reflect user preferences, which are difficult to express as a simplified cost function. For example, WIP and weighted tardiness are not always compatible with each other. As shown in Fig. 2, there are situations where a repair action can reduce weighted tardiness, but WIP increases. Which is a better schedule depends on user preferences in the specific scheduling context where several factors, such as clients, past shipping records and load of factory and warehouse, have meaningful influences.

## 3 KNOWLEDGE ACQUISITION IN SCHEDULING

Recently a growing number of research activities have been done on acquiring and learning useful knowledge for solving a complex scheduling problem. The goal and methodology of the research can be classified into the following categories:

- Knowledge sharing and reuse. Problem solving for a specific task is extracted and stored in the components in a way that its reusability is independent of the application domains. Development of a knowledge-based system is done by retrieving and assembling appropriate components (e.g. SPARK,[4] CAKE,[5] MULTIS[6]).
- Human–computer cooperative scheduling. When a human expert disapproves a schedule produced by the scheduling system, s/he repairs the schedule and explains the reason for dissatisfaction and validation of a repair. Later, a knowledge engineer analyses user's explanations, and uses them to enhance and improve the content of knowledge bases in the scheduling system (e.g. WATPASS,[7] Scheplan[8]).
- Adaptive dispatching. In FMS environment, a very flexible way of manufacturing control is possible. Induction based learning techniques are applied to create rules for dynamically selecting an appropriate dispatching rule according to the status of manufacturing environment (e.g. PDS,[9] GDCA[10]).
- Control knowledge acquisition. Control knowledge can be exploited to improve efficiency of solving a scheduling problem. This type of knowledge is captured by interviewing with an expert when a detail problem-solving method in the domain is understood or applying the EBL method when sufficient amount of domain theory is available (e.g. SALT,[11] PEBL[12]).
- Problem description acquisition. By using a generic model of scheduling problems as a template for an interview with a human scheduler, a knowledge engineer can easily acquire and formulate knowledge about constraints and physical environment of a scheduling problem without missing the points (e.g. PFF[13]).

In spite of the past research efforts, acquisition of control knowledge in scheduling problems is still difficult, because ill-causality of the scheduling problem hinders understanding of problem solving methods and domain theory of the scheduling problem. And there has been no research concerning the acquisition of user's preferential knowledge on tradeoffs among conflicting objectives in the schedule. We advocated a unified framework of knowledge acquisition and problem solving in the schedule problem and implemented the system called CABINS,[14] which optimizes a schedule by iterative repair using case-based reasoning (CBR)[15] methodology. In the succeeding sections, we explain how CABINS acquires user's preferential and control knowledge from past experience cases, and how

CABINS exploits that knowledge effectively for schedule optimization.

## 4 CBR FOR SCHEDULE OPTIMIZATION

Because of the characteristics of the scheduling domain and the interest in capturing context dependent user preferences and situation sensitive search control knowledge, CBR appears to be a natural method for knowledge acquisition. However, applying CBR to schedule improvement, a numerical optimization problem, is very challenging. In general, CBR has been used for ill-structured symbolic problems, such as planning,[16–18] legal reasoning,[19,20] argumentation,[21] conceptual design,[22] medical diagnosis[23] where the primary concern has been plausibility or correctness of resulting artifacts (plan, argument, design) and computational efficiency of the problem solving process rather than artifact quality.

The challenges in applying CBR to schedule optimization were to determine what constitutes a case in the domain of schedule optimization and what the case indices should be. The intuitive answer would be to consider a whole schedule as a case.[24,25] This solution is attractive since, if the right information could be transferred from one scheduling scenario to another, or with little adaptation, a new problem would be solved with relative ease. In the traditional planning problem, the plan operators capture some form of domain causality in their preconditions and effects. Saving a plan and the derivational trace of how the plan was generated, captures pretty much the planning process and can be easily utilized to solve future similar problems.[16–18] However, it is not true in the scheduling problem. Because of the high degree of nonlinearity of scheduling constraints and objectives, a very small difference between an input problem specification and the problems in the case base can in general result in large variations in the results both in terms of the amount of modifications needed and the quality of a resulting schedule. A second difficulty with respect to having a whole schedule as a case came in the form of what indices to choose. Indexing a case in terms of the goals that must be achieved and the problems that must be avoided[16] is a good guideline and has served many CBR systems well. However, in the scheduling domain, the goals to be achieved (the optimization criteria) cannot be explicitly stated, since they reflect context-dependent user preferences and tradeoffs. Even if the optimization objectives were explicit, because of the nonlinearities of the problem, retrieving a schedule in which the achieved objectives were the same as the desired ones in the current problem would give little or no help in adapting the retrieved schedule to the current problem specifications. Moreover, because of unpredictable ripple effects of constraint propagation and tight constraint interactions, the problems to be avoided are not at all obvious, neither can they be discovered since a causal model for scheduling cannot be assumed.

Since it is impossible to judge *a priori* the effects of a scheduling decision on the optimization objectives, a scheduling decision must be applied to a schedule and its outcome must be evaluated in terms of the resulting effects on scheduling objectives. Thus, having a single scheduling decision as a case seemed to provide advantages in terms of focus and traceability of the problem solving process. Focus and traceability mean that a user's evaluation of the results of a single scheduling decision can be captured in a case, and, if the result was unacceptable, another scheduling decision to the same 'scheduling entity' can be applied until either all available scheduling decisions are exhausted or an acceptable result is obtained. For the above reasons, it became clear that it was better to have a case for a single scheduling entity on which a scheduling decision was applied. Since the result of a scheduling decision needed to be evaluated with regard to the optimization preferences for a schedule as a whole, it is clear that constructive methods which incrementally augment a partial schedule at every scheduling decision point would be unsuitable for the purpose. Moreover, contextual information, which can only be provided by having a complete schedule, is very useful in applying CBR. Therefore, the revision-based method was chosen as the underlying optimization methodology in CABINS.

Hence in CABINS, a case describes the *application of a schedule revision decision on a single scheduling entity.* Operationalization of a schedule revision decision is done by means of a *schedule repair action.* Each application of a schedule repair action results in a new schedule. The search space of CABINS is the space of complete schedules that incorporate acceptable user optimization tradeoffs. Hence the predictive case features that are suitable for case indexing should be those that capture good tradeoffs. Although schedule optimization is ill-structured, we make the hypothesis that there are regularities of the domain that can be captured, albeit in an approximate manner, in these features. In CABINS, indices are divided into two categories. The first category consists of the *descriptive features.* Since the results of schedule revision associated with a single scheduling entity pertain to the whole schedule, it is impossible to make a precise prediction of repair effects in advance of revisions. Descriptive features that express characteristics of a scheduling entity operate as contextual information for selection of a particular repair action and allow CABINS to estimate the effects of each repair action in advance.

The schedule resulting from a repair action application must be evaluated in terms of user-defined tradeoffs. The user cannot predict the effects of modification actions on schedule correctness or quality since a modification could result in worsening schedule quality

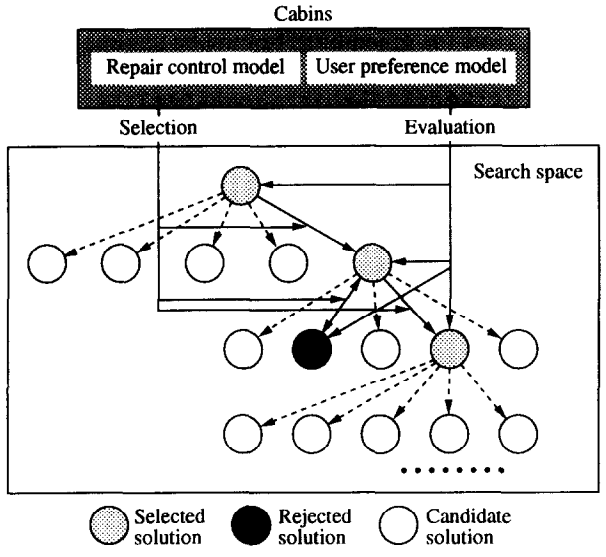


**Fig. 3.** Search space and search control in CABINS.

or introducing constraint violations. Nevertheless, the user can perform consistent evaluation of the results of schedule revisions. This evaluation is recorded in the case as part of the case's repair history. The *repair history* constitutes the second category of case features. Thus, the case base incorporates a distribution of examples that collectively capture repair performance tradeoffs under diverse scheduling circumstances.

CABINS searches for an 'optimal' schedule over the space of complete schedules. Figure 3 shows the schematic diagram of the search space and search control in the CABINS system. CABINS revises the current solution iteratively to improve the solution quality. For each step of the search, CABINS selects a solution among the neighbours of the current solution. The neighbourhood size for the current solution (i.e. the number of potential solutions for each revision) is equal to *Number_of_Repair_Actions* × *Number_of_Repair_Objects*. In a scheduling problem, *Repair_Actions* are several heuristics that modify the assignments of resources to activities in the schedule and *Repair_Objects* are typically the activities in the schedule. The number of revision cycles required to obtain a final solution cannot in general be predicted in advance because of tight constraint interactions in the scheduling problem. Hence the search space for a large scheduling problem can be intractably big. To reduce the required research efforts, CABINS has the following mechanisms of search control using CBR: a *repair control model* provides the search control through case-based selection of the next repair action to be applied, and a *user preference model* provides the search control through case-based evaluation of the result of the application of a selected repair action. The descriptive features are the indices that are used to retrieve a case that suggests the next repair action to be applied. The features associated with the repair history are used to retrieve cases that suggest evaluations of a repair outcome.

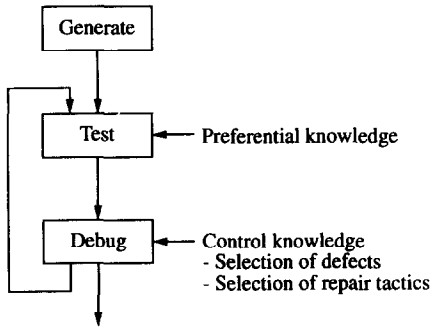


**Fig. 4.** GTD process.

## 5 SCHEDULE OPTIMIZATION BY CABINS

CABINS optimizes a given schedule by iterative repair based upon a GTD (generate-test-debug) process (Fig. 4). CABINS differs from other GTD-based scheduling methodologies in that it doesn't use a fixed objective function for optimization. CABINS optimizes a schedule by iterative repair until a further repair is unnecessary or impossible. Repair methods are selected by the control knowledge and a result of the repair is evaluated by user's preferential knowledge. Both types of knowledge are acquired through interaction with a human scheduler and stored as cases in the case base of CABINS. First, CABINS gathers user's judgements on repairing schedules of the training problems in the case base. Once a sufficient number of cases are accumulated, CABINS can automatically repair a schedule by reusing user's judgements in the past similar cases (see Fig. 5).

### 5.1 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 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.

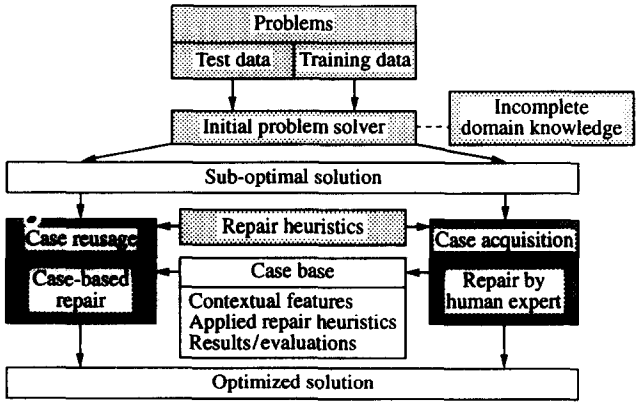


**Fig. 5.** CABINS architecture.

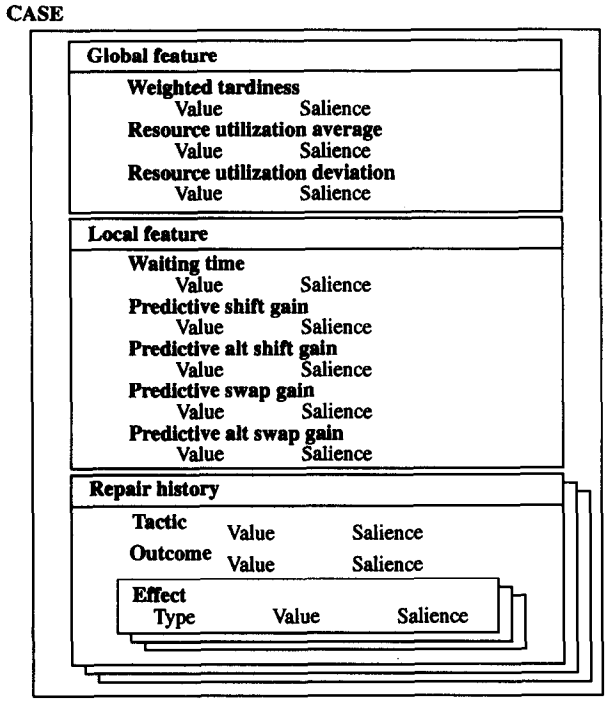


**Fig. 6.** Case representation.

Figure 6 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.

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 the existence of a certain causal structure in a problem implicitly. 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 a repair tactic application 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 value describes the impact of the application of a repair action on the scheduling objective designated in the effect type. The effect salience is assigned when the outcome is 'unacceptable', and it indicates the significance of the effect to the repair outcome.

### 5.2 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 user's optimization criterion is to minimize job tardiness, the most tardy job 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, (b) execute the repair and re-schedule the activities identified in (a).

After tactic selection and application, the repair effects are calculated and shown to be 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 of favourable and unfavourable effects. Then, 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. 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.

As cases are acquired in the course of actual problem solving by a user, CABINS can elicit user's context-dependent knowledge without requiring excessive burdens from the user.

### 5.3 Case application

Once enough cases have been gathered, CABINS repairs sub-optimal schedules without user's interaction. Following the procedure shown in Fig. 7, CABINS repairs the schedules by (1) invoking CBR with global and local features as indices to decide the most appropriate repair tactic to be used for each focal_activity, (2) applying the selected repair tactic, (3) invoking CBR using the repair

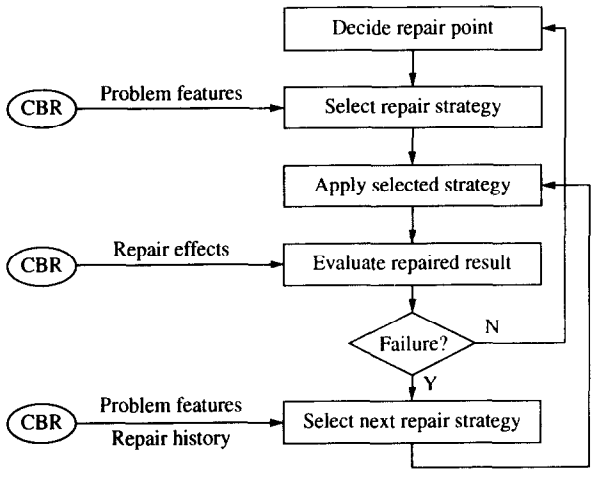


**Fig. 7.** Case application.

effect features (type, value and salience) as indices to evaluate the repair result, (4) in case of failure, invoking CBR with global and local problem features and history of failed repair tactics as indices to decide which repair tactic to use next. Experiments in using different indexing schema in case of failure are described in section 8.

In CABINS concept are defined extensionally by a collection of cases. As a case retrieval mechanism, CABINS uses a variation of $k$-nearest neighbour method where not the frequency but the sum of similarity of $k$-nearest neighbours is used as a selection criterion. The similarity between the $i$th case and the current problem is calculated as follows:

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

where $SL_j^i$ is the salience of the $j$th feature of the $i$th case in the case base, and its value has been heuristically defined by the user; $CF_j^i$ is the value of the $j$th feature of the $i$th case, $PF_j$ is the value of the $j$th feature in the current problem, $E\_D_j$ is a standard deviation of the $j$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.

## 6 AN EXAMPLE

We briefly illustrate the repair process with the very simple example schedule to be repaired shown in Fig. 8. The example has ten jobs ($J_1, \ldots, J_{10}$) and each job has five activities with linear precedence constraints (e.g. $O_1^n$ BEFORE $O_2^n, \ldots, O_4^n$ BEFORE $O_5^n$). Resources $R_1$ and $R_2$, $R_3$ and $R_5$ are substitutable; resource $R_4$ is a bottleneck. Suppose that the job under repair is $J_8$. This job has a weight of 2, a due date of 1250 and the scheduled end-time of its last activity is 1390. Hence it has a weighted tardiness of $2 \times (1390 - 1250) = 280$. Suppose the current focal_activity is $O_4^8$. CBR is invoked with global features (weighted tardiness = 280, resource utilization average = 0·544, resource utilization deviation = 0·032) plus the set of local features as indices and selects swap as a repair tactic. One can see from the figure that this is a good choice since the focal_activity is scheduled on machine $R_4$, which doesn't have any substitutable machine and any idle time in the repair time horizon (time between the end of $O_3^8$ and the end of $O_4^8$).

The swap repair tactic roughly calculates the effects of swapping the current focal_activity with each activity within the current focal_activity's time horizon and selects the activity that gives the biggest net gain (note that swapping an activity that is scheduled earlier with one that is scheduled later will now delay the earlier activity). In the example, suppose that activity $O_4^4$ is selected as the activity to be swapped with the current focal_activity $O_4^8$. The effect of applying the swap tactic is that $O_4^8$ and $O_4^4$ are unscheduled on $R_4$ and $O_4^8$ is rescheduled to start at time 1090 (the start time of activity $O_4^4$ prior to the swap) and $O_4^4$ is moved to start at time 1180 (the start time of activity $O_4^8$ prior to the swap). Because the new assignments of two activities overlap each other, constraint propagation is invoked and the assignment of $O_4^4$ is further delayed. Due to the delay of activity $O_4^4$, now there is the ripple effect of a precedence constraint violation between activity $O_4^4$ and its successor activity $O_5^4$ on resource $R_2$ (in general, many activities could be affected and must be rescheduled). Constraint propagation discovers this constraint conflict and shifts activity $O_5^4$ further to the right on resource $R_2$. Since

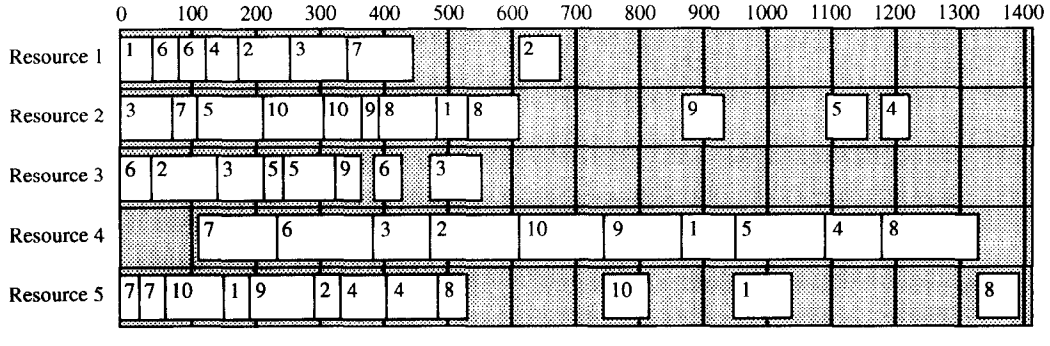


**Fig. 8.** Original schedule results.

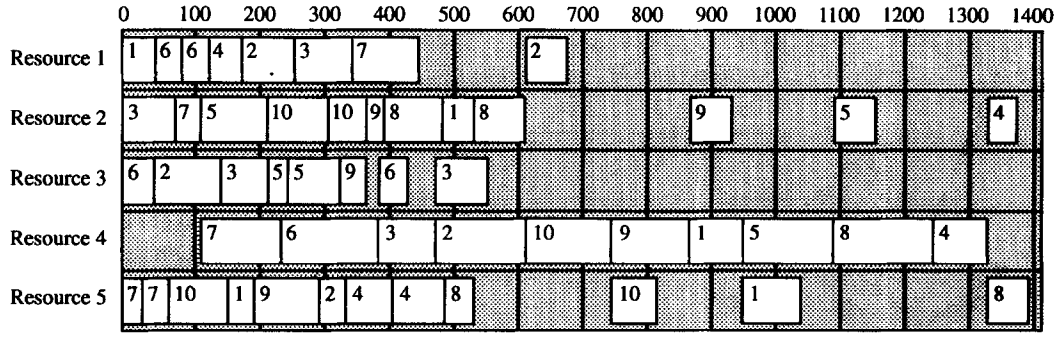


**Fig. 9.** Schedule results after repair on $O_4^8$.

job $J_4$ has weight 3, its weighted tardiness is now $3 \times (1370 - 1320) = 150$. The repaired schedule result is shown in Fig. 9.

CABINS calculates both local effects (i.e. effects on the repair target, $J_8$) and global effects (i.e. effects on the whole schedule) for result evaluation. In this example, 'local_weighted_tardiness' is estimated as +180 time units and 'local_inprocess_inventory' is estimated as +120 units, both being improved by the change of $O_4^8$. And 'global_weighted_tardiness' is +30 units (i.e. 180–150) and 'global_inprocess_inventory' is −750 units (as the waiting time in $J_4$ increases by 950 units). CBR is invoked using these effect values as indices to determine whether this repair outcome is acceptable or not. If there are more significant 'SUCCEEDED' cases than 'FAILED' cases in the retrieved $k$-nearest neighbours, the repair is considered reflecting the tradeoffs of user's preference (in this example, little weight on 'global_inprocess_inventory) and the outcome is considered as acceptable. Otherwise, the outcome is considered as unacceptable, thus showing that loss in 'global_inprocess_inventory' is more critical than possible gain in weighted tardiness according to the user's preferences.

## 7 ACQUIRING PREFERENTIAL KNOWLEDGE

We hypothesize that CABINS can acquire user's preferential knowledge by accumulating cases of user's judgement on schedule repair results. To validate our hypothesis, we did experiments to see whether CABINS can optimize schedule quality along various optimization criteria which CABINS does not know explicitly. We compared CABINS with a set of well-regarded dispatch heuristics, widely used in manufacturing job shop scheduling, and with a constraint-based scheduler. The dispatch rules selected for the comparison are the earliest due date (EDD) rule, the weighted shortest processing time (WSPT) rule and the WSPT with job time urgency factor (R&M) rule. The constraint-based scheduler (CBS) uses backtrack search with sophisticated variable and value ordering heuristics.[26] These schedulers are known to achieve near optimal performance with respect to job tardiness and work-in-process-inventory (WIP) under various scheduling conditions.

To cover different scheduling conditions, six groups of 10 problems each were randomly generated using three different values in due date and release date parameter distribution (static, moderate, dynamic), and two values of bottleneck configuration (one bottleneck, two bottlenecks). Each problem has five resources and 10 jobs of five operations each. Each job has a linear 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 tighter. The slack was adjusted as a function of the range and bottleneck parameters to keep demand for bottleneck resources close to 100% over the major part of each problem. Durations for activities in each job were also randomly generated.

To assess CABINS performance accurately, we applied a two cross-validation method. Each problem set in each class was divided in half. In one a half initial schedule was generated using a constraint-based scheduler and then repaired heuristically to gather cases. These cases were used to repair the other half of the problem set. We repeated the above process by interchanging the sample- and the test-set. Our results are the average of the two sets of results. In the experiments, CABINS used two types of case base, one of which was trained under the optimization criteria of minimizing weighted tardiness and the other was trained to minimize combination of weighted tardiness and WIP.

### 7.1 Experimental results

The results in Fig. 10 show a comparison of the resultant schedule quality by the above scheduling methods and CABINS. The optimization criteria used in the experiments were weighted tardiness (left graph) and the combined objective of minimizing weighted tardiness and WIP (right graph). In the graphs, CABINS(WT) and CABINS(WT + WIP) represent CABINS with a case base trained to optimize weighted tardiness and CABINS with a case base trained to optimize the combination of weighted tardiness respectively. From the graphs, it is shown that CABINS with correctly

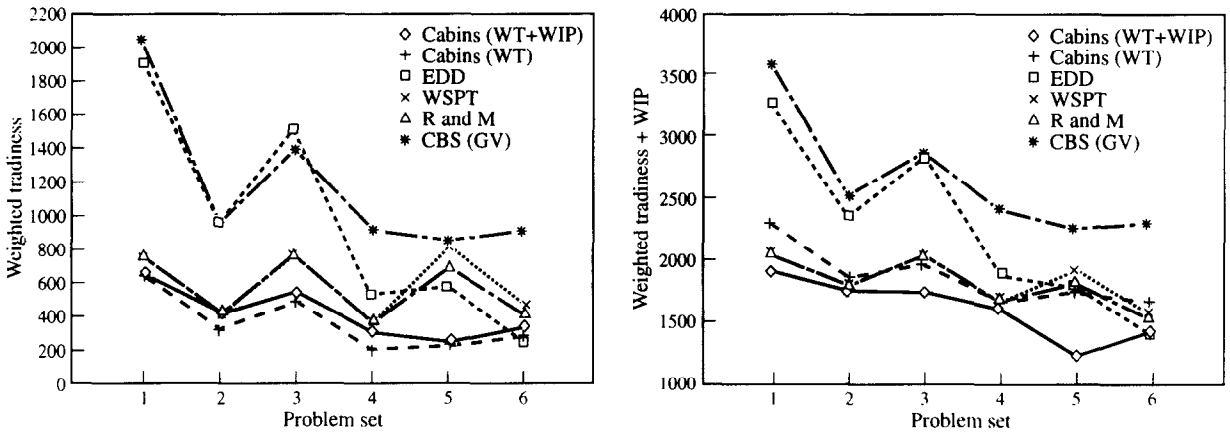


**Fig. 10.** Comparison of schedule quality.

trained case base outperformed all the other methods (including CABINS with wrong case base) across both one and two bottleneck problems in all experiments. From these experiments, we see that CABINS can acquire user's scheduling objectives and optimize schedules along acquired objectives.

## 8 ACQUIRING CONTROL KNOWLEDGE

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, (2) improve its competence both in repair quality and efficiency by utilizing failure information recorded in the cases. To analyse the correctness of our hypothesis, we classified cases into two types, one of which is a *success case* where a repair was done successfully and the other of which is a *failure case* which failed to repair a schedule, and experimentally implemented the following three repair strategies (see Fig. 11):

- *One-shot repair.* CABINS selects a repair tactic by retrieving the most similar case from 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 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 successfully repaired and the tactic that was successful in repairing a schedule. For example, when a repair result is judged as unacceptable by a user after application of left_shift tactic because the user doesn't like side effects caused by the tactic, another case that has the most similar global and local features is retrieved from the success cases which have the failure record of left_shift tactic because of user's unacceptance in repair history of the case. And a tactic that finally succeeded in repair of the selected case is used for another repair trial of the current problem. The intuition here is that a similar outcome of the same tactic implies a similarity of causal structure between the past and current cases. Therefore, the repair tactic which succeeded in the cases that have similar failed tactic applications 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 and success cases are checked 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,

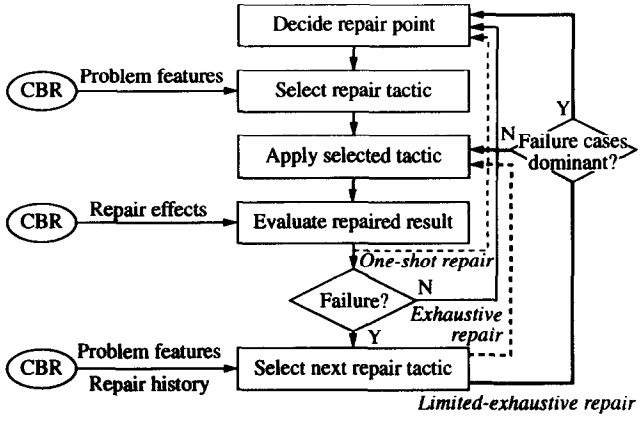


**Fig. 11.** Three repair methods in CABINS.

we bias (negatively) usage of failure cases by placing a threshold on the similarity value and currently its value is heuristically fixed as 0·75. 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 favour of success cases which are moderately similar to the current problem.

To verify our hypothesis, we did experiments with six types of scheduling problems, the same as the experiments in the previous section. In the experiments, reported here, we used a metric, minimizing weighted tardiness, as an objective function to evaluate the performance of CABINS. Of course, CABINS does not know this metric but has to guess it from the contents of the case base. And 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 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 an exact objective function (in real world job shop scheduling problems, which is very complex and is available only in a user's mine) for evaluation, RBR can work as an emulator of a human scheduler, who cannot repair a schedule in the most efficient way but can make consistent evaluation on 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. Thus, CABINS has been trained with about 100 cases by RBR.

### 8.1 Experimental results

The graphs of Fig. 12 show comparative results with respect to schedule quality improvement (weighted tardiness) and repair efficiency (in CPU seconds) 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 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.

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 the effects of schedule repair are pretty unpredictable because of ill-causality of scheduling problems 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.

## 9 SCALING-UP A CASE BASE

The graphs in Fig. 13 show a comparison of CABINS' performance with different sized case bases. The results were obtained based on CABINS with WT + WIP type of case bases. From the viewpoint of knowledge acquisition, an interesting question is when knowledge acquisition can be terminated because sufficient knowledge has been acquired to enable high quality perfor-

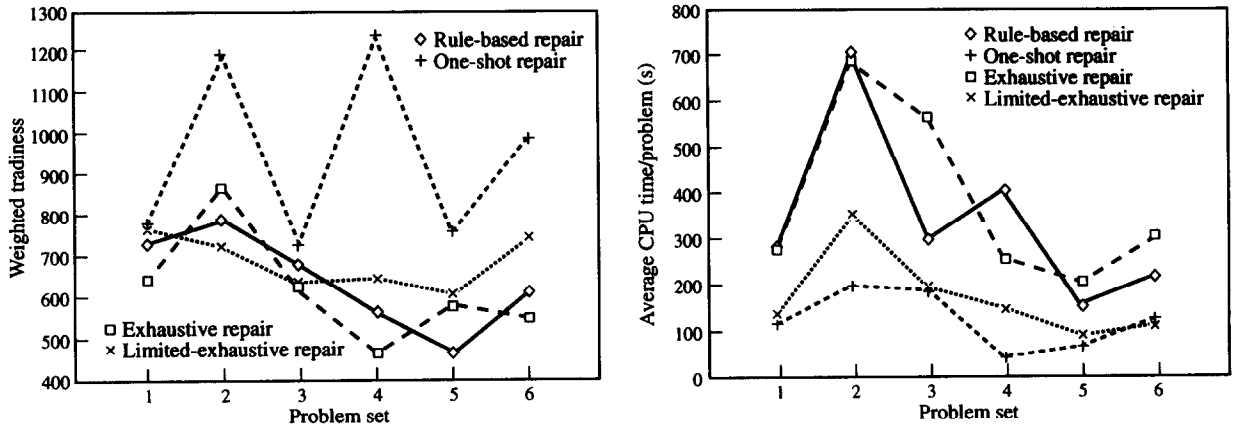


**Fig. 12.** Comparison of schedule quality and efficiency.

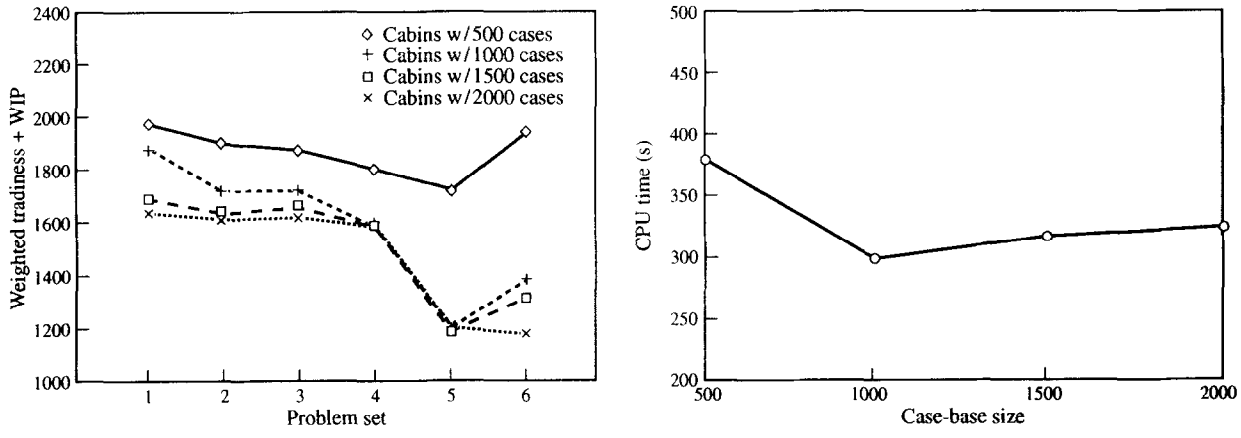


**Fig. 13.** Effect of case base sizes on quality and efficiency.

mance of a knowledge based system. For case-based knowledge acquisition, this question becomes how many cases would be enough for knowledge capture and reuse and for guaranteeing overall satisfactory performance. Unfortunately, it is very difficult to answer this question in general due to the ill-structuredness of the scheduling problem and the approximate nature of CBR (since no causal model is available). We believe, however, that there exists some appropriate size of the case base which will give us relatively satisfactory results in terms of schedule quality without excessive overhead for case acquisition or case retrieval from the case base.

Our experimental results (Fig. 13) support this hypothesis as follows: (1) The larger the number of cases, the better the schedule quality. However, the marginal payoff from the increase in case base size decreases. This can be explained partially by the fact that some number of cases (say, 1000) captures well the characteristics of the problem space, and an additional 1000 new cases may give much redundant information. When the size of a case base is relatively small, every time new cases are acquired, we may get information about a different part of the problem space which results in higher quality improvement. (2) In terms of efficiency of the system, we observe from the graphs that the case base with 1000 cases might be the optimal choice. Intuitive explanation of the results is that CABINS with less cases cannot search the repair solution space efficiently because of its poor control knowledge and CABINS with more cases spoils the search efficiency because of the increased case retrieval time (i.e. CABINS suffers from a utility problem[27]).

Actually, both in terms of CPU time and quality improvement, the case base with 1000 cases obviously outperforms the case base with 500 cases. Moreover, case bases with more cases than 1000 do not seem to provide a payoff proportional to the case base size increase. In Veloso's research,[28] the issue of the tradeoff between optimal case retrieval time interval and search efficiency is discussed in planning domain. However, the assumptions on which their theoretical analysis was based seem not to hold in an ill-structured domain, such as scheduling. How to theoretically predict the optimal size of case base is still an open research problem and we are currently investigating it.[29]

## 10 CONCLUDING REMARKS

In this paper, we advocated a unified framework for knowledge acquisition and iterative revision for schedule optimization. The approach utilizes CBR-based mechanisms for recording user preferences, repair tactics and explanations, and constraint-based scheduling for application of the selected repair tactics. The approach is predicated on (a) the existence of a set of schedule repair tactics, each of which operates with respect to a particular local view of the problem and offers selective advantages for improving schedule quality, (b) on capturing user scheduling preferences and judgements, (c) on retrieving and re-using these preferences to dynamically change scheduling utilities during revision. Our experimental results show that the approach (1) was able to capture and effectively utilize user scheduling preferences that were not present in the scheduling mode, (2) outperformed other scheduling methods along multiple evaluation criteria. We also examined various ways of exploiting past repair experiences for control knowledge acquisition. Our experiment results show that our methodology can improve its own performance by: (1) using failure experience as a contextural index of the problem, (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 that allows the problem solver to improve its efficiency while preserving quality of results in the domain without strong domain knowledge.

## ACKNOWLEDGEMENT

The author is indebted to Professor Katia Sycara at the Robotics Institute of Carnegie Mellon University for fruitful discussions and valuable advice.

## REFERENCES

1. French, S., *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood, London, 1982.
2. Kempf, K., LePape, C., Smith, S. F. & Fox, B. R., Issues in the design of AI-based schedulers: Workshop report. *AI Magazine*, 1991, **11**, 37–46.
3. Minton, S., *Machine Learning Methods for Planning and Scheduling*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
4. Klinker, G., Bhola, C., Dallemagne, G., Marques, D. & McDermott, J., Usable and reusable programming constructs. *Knowledge Acquisition*, 1991, **3**, 117–35.
5. Hori, M., Nakamura, Y. & Hama, T., Methodology for configuring scheduling engines with task-specific components. *Proceedings of the Second Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1992, Kobe, Japan, pp. 215–29.
6. Mizoguchi, R., Tijerino, Y. & Ikeda, M., Task ontology and its use in a task analysis — Two-level mediating representation in MULTIS —. *Proceedings of the Second Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, Kobe, Japan, 1992, pp. 185–98.
7. Mckay, K., Buzacott, J. & Safayeni, F., The scheduler's knowledge of uncertainty: The missing link. *Proceedings of IFIP Working Conference on Knowledge Based Production Management Systems*, 1988, Galway, Ireland.
8. Numao, M. & Morishita, S., A scheduling environment for steel-making process. *Proceedings of the Fifth Conference on AI Applications*, 1989.
9. Shaw, M. J., A pattern-directed approach to flexible manufacturing: A framework for intelligent scheduling, learning and control. *International Journal of Flexible Manufacturing Systems*, 1989, **2**, 121–44.
10. Chaturvedi, A. R., FMS scheduling and control: Learning to achieve multiple goals. *Expert Systems With Applications*, 1993, **6**, 267–86.
11. Stout, J., Caplain, G., Marcus, S. & McDermott, J., Toward automating recognition of differing problem-solving demands, *International Journal of Man-Machine Studies*, 1988, **29**, 599–611.
12. Zweben, M., Davis, E., Brian, D., Drascher, E., Deale, M. & Eskey, M., Learning to improve constraint-based scheduling. *Artificial Intelligence*, 1992, **58**, 271–96.
13. Liu, B., Problem acquisition in scheduling domain. *Expert Systems with Applications*, 1993, **6**, 257–65.
14. Miyashita, K. & Sycara, K., Adaptive case-based control of schedule revision. In *Intelligent Scheduling*, ed. M. Zweben & M. Fox. Morgan Kaufmann, San Mateo, CA, 1994.
15. Kolodner, J., Simpson, R. & Sycara, K., A process of case-based reasoning in problem solving. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985, Los Angeles, CA, pp. 284–90.
16. Hammond K. J., *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, New York, 1989.
17. Kambhampati, S. & Hendler, J. A., A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 1922, **55**, 193–258.
18. Veloso, M. M., Learning by analogical reasoning in general problem solving, PhD thesis, Carnegie Mellon University, 1992.
19. Ashley, K. D., Modeling legal argument: reasoning with cases and hypotheticals. PhD thesis, University of Massachusetts, Amherst, MA, 1987.
20. Rissland, E. L. & Ashley, K. D., Credit assignment and the problem of competing factors in case-based reasoning. *Proceedings of the Case-Based Reasoning Workshop*, Clearwater, FL, 1988, pp. 327–44.
21. Sycara, K., Argumentation: Planning other agents' plans. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989.
22. Sycara, K., Guttal, R., Koning, J., Narasimhan, S. & Navinchandra, D., CADET: A case-based synthesis tool for engineering design. *International Journal of Expert Systems*, 1991, **4**.
23. Koton, P., Reasoning about evidence in causal explanations. *Proceedings of the Case-Based Reasoning Workshop*, Clearwater, FL, 1988, pp. 260–70.
24. Koton, P., SMARTplan: A case-based resource allocation and scheduling system. *Proceedings of the Case-Based Reasoning Workshop*, Pensacola, FL, 1989, pp. 285–94.
25. Mark, W. S., Case-based reasoning for autoclave management. *Proceedings of the Case-Based Reasoning Workshop*, Pensacola, FL, 1989, pp. 176–80.
26. Sadeh, N., Look-ahead techniques for micro-opportunistic job shop scheduling. PhD thesis, Carnegie Mellon University, 1991.
27. Minton, S., *Learning Effective Search Control Knowledge: An Explanation-based Approach*. Kluwer Academic Publishers, Boston, MA, 1988.
28. Veloso, M.M. & Carbonell, J. G. Toward scaling up machine learning: A case study with derivational analogy in PRODIGY. In *Machine Learning Methods for Planning*, ed. S. Minton, Morgan Kaufmann, San Mateo, CA, 1993.
29. Miyashita, K., Case-based knowledge acquisition for solving ill-structured optimization problems. PhD thesis, Osaka University, 1994.