Using Case-Based Reasoning to Acquire User Scheduling Preferences that Change over Time *

Katia Sycara, Dajun Zeng

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

Kazuo Miyashita

Production Engineering Devision
Matsushita Electric Industrial Co.
Kadoma, Osaka 571, Japan
miyasita@mcec.ped.mei.co.jp

Abstract

Production/Manufacturing scheduling typically involves the acquisition of user optimization preferences. The ill-structuredness of both the problem space and the desired objectives make practical scheduling problems difficult to formalize and costly to solve, especially when problem configurations and user optimization preferences change over time. This paper advocates an incremental revision framework for improving schedule quality and incorporating user dynamically changing preferences through Case-Based Reasoning. Our implemented system, called CABINS, *records situation-dependent tradeoffs and consequences that result from schedule revision to guide schedule improvement. The preliminary experimental results show that* CABINS *is able to effectively capture both user* static *and* dynamic *preferences which are not known to the system and only exist implicitly in a extensional manner in the case base.*

1 Introduction

Scheduling deals with allocation of a limited set of resources to a number of activities [1]. One of the most difficult scheduling problem classes is job shop scheduling. In job shop scheduling, each task (interchangeably called an order or a job) consists of a set of activities to be scheduled according to a preset partial ordering which reflects *precedence constraints. Capacity constraints* restrict the number of activities that can be assigned to a resource during overlapping time intervals. The activity precedence constraints along with a task's release date and due date determine the set of acceptable start times for each activity. The goal of a scheduling system is to produce schedules which satisfy all temporal relations and resource capacity constrains. The scheduling system will also try to optimize the resulting schedule based on a set of objectives, such as minimize weighted tardiness, minimize inventory cost of Work-In-Process, maximize resource utilization, etc.

*This papers appears in *The Proceedings of the Eleventh IEEE Conference on Artificial Intelligence Applications (CAIA '95).*

The scheduling problem is difficult to solve for a number of reasons. First, it is an NP-complete problem [7]. Second, scheduling objectives are typically not well-defined. For example, the user might want to minimize both weighted tardiness and work-in-process to meet due dates and to diminish the inventory cost. However, what type of combination of objectives will perfectly reflect the user's preferences? Even if the user knows that some combination of weighted tardiness and W.I.P. must be optimized, does the objective in the form of $Weighted\ Tardiness + W.I.P.$ make more sense than $Weighted\ Tardiness \times W.I.P.$ or the opposite? Even if the user knows that the additive form is preferred, still the relationship existing between these two objectives needs to be specified, for example, is $0.5 \times Weighted\ Tardiness + 0.5 \times W.I.P$ more appropriate or $0.2 \times Weighted\ Tardiness + 0.8 \times W.I.P.$? Third, the user's scheduling preference might change over time. Fourth, due to the tight interactions among scheduling constraints and the often conflicting nature of optimization criteria, it is impossible to evaluate precisely the impact of a single scheduling decision or local revision action on the global satisfaction of optimization criteria. In effect, real-world scheduling practice often involves both quantitative and qualitative constraints as well as situation-dependent preferences and subjective user-dependent judgement. Finding ways to incorporate these considerations when building scheduling systems, namely, how to trade-off between conflicting objectives or preferences and how to bias optimization procedure adaptively will be a real challenge to both researchers and practitioners in the scheduling domain.

Our thesis is that learning through case-based reasoning can capture user optimization preferences that change over time in ill-structured and complex tasks, such as job shop scheduling, and utilize the changing user preferences as control knowledge to guide solution optimization. We believe that our approach will be effective in real-world optimization tasks for a variety of reasons. First, traditional search methods, both Operations Research-based and AI-based, that are used in combinatorial optimization, need explicit representation of the optimization objectives that are defined in advance of problem solving [20]. In many practical problems, such as scheduling and design, optimization criteria often involve context- and user-dependent tradeoffs which are impossible to represent as an explicit and static optimization function. For example, a user's preferences might depend on partial results obtained during problem solving. For such situations, it is very difficult for the user to completely define his/her preferences in advance of problem solving. The number and dimensions of the alternatives involved in decision making might be very large preventing the user from giving general applicable preference evaluation. Second, expert system approaches, while having the potential to capture context-dependent tradeoffs in rules, require considerable knowledge acquisition effort and traditionally have not dealt with optimization concerns [18]. Third, and equally important consideration is the fact that the problem solving environment and the preferences of the user could be changing over time. Therefore, approaches that capture preferences statically or require expensive knowledge-base updating are extremely limiting. On the other hand, approaches that acquire knowledge through continuous interaction with a user and utilize machine learning techniques to adapt their behavior to the changing user preferences and problem solving context are much more promising.

We have implemented our Case-Based iterative revision approach in the CABINS system to demonstrate the capability of acquiring user *context-dependent and dynamically changing* optimization preferences in job shop scheduling domains. The work reported here extends previous work on the CABINS system [26, 16, 17]. It tests the hypothesis that our CBR-based incremental revision methodology shows good potential for capturing user preferences in ill-structured domains that *change over time* and reuse them as control knowledge for solution improvement. The preliminary experimental results, shown in section 5 confirmed our hypothesis.

2 Built-in Learning Capability: Case-Based Reasoning

Capturing user's situation-dependent preferences has become a very active area in Knowledge-Based Systems and Decision Support Systems (DSS) research. This problem arises in a variety of application domains, such as office electronic secretary [6], financial portfolio selection [10], digital circuit design [12] and production scheduling domain [19]. Automating the acquisition of user preferences is important in practice for the following reasons. First, for many ill-structured domains such as production scheduling, the overall objectives of the system typically are not explicitly available. It may take a tremendous amount of knowledge acquisition effort to elicit them in a practical environment. As a matter of fact, the success of a knowledge-based system/DSS depends to a large extent on its capability of acquiring user preferences. Second, it is not always possible to extract user's objectives or preferences when the knowledge-based system is being built. Sometimes, although it is theoretically feasible to get all the information concerning user's objectives, the high cost typically associated with knowledge extraction and acquisition might prevent doing so practically. Therefore, the capability of capturing and learning the user's preferences through interaction with users during the system's routine operation is highly desirable. Last but not least, the world is changing. So, it is necessary to develop knowledge acquisition techniques that reflect changing preferences and evaluation criteria. A knowledge-based system incapable of handling changing world will be very fragile and easy to fail.

Research in machine learning provides the opportunity and possibility for knowledge-based systems to acquire user-dependent and situation-dependent knowledge without suffering too much from knowledge acquisition bottleneck. Several learning paradigms have been proposed to deal with capturing preferences. ID3 and Neural-Net-based learning algorithms have been successfully applied to capture user's preferences in mundane meeting scheduling domain [6]. Some general extensions of explanation-based learning have also reported [12, 14] to be effective for digit circuit design task.

Instead of using ID3-like inductive learning algorithms or neural-net based approaches, we chose Case Based Reasoning (CBR) paradigm to capture user optimization preferences that change over time in job shop scheduling domain. CBR is the problem solving paradigm where previous experience are used to guide problem solving [8, 24, 9]. In CBR *learning is an integral part of case based problem solving*. Case-based reasoning has been successful in dealing with exceptional data [22], acquiring user knowledge in complex domains [4, 13], and expending less effort in knowledge acquisition compared with knowledge acquisition for rule-based systems [23, 11]. CBR presents the following advantages as an appropriate framework for knowledge acquisition. First, rule induction require great computational effort to update the rule base and generate the new rules as well as to maintain consistency between acquired rules. The *computational overhead* problem is alleviated in CBR because of the inherent *incremental* property of Case-Based Reasoning. In a CBR system, when a new problem is encountered and solved, the case base is incrementally updated without the need for any other reasoning or consistency maintenance calculations. Case matching and retrieval can be done efficiently if the memory is organized effectively. Second, the rules learned through induction might not be very understandable to the user since the underlying learning algorithm is based on the *statistical* characteristics of the previously collected cases and relevant features, which might not be meaningful to the user. The *understandability* problem is much worse for neural-net based systems [21]. On the other hand, cases are understandable since they embody concrete experiences of user interactions. Third, after rule induction, contextual information associated with the underlying example instances is thrown away. Contextual information is potentially useful to the user to understand the system's reasoning. CBR does keep all the contextual information.

CABINS is different from other case-based learning systems. Current case-based knowledge acquisition systems, (e.g., [2]) require causal explanations from an expert teacher to acquire domain knowledge. In our approach neither the user nor the program are assumed to possess causal domain knowledge. As a matter of fact, in ill-structured domains such as job shop scheduling, it is impossible to have causal knowledge. 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 acquires, stores and reuses two categories of concepts that reflect user preferences: (1) what heuristic local optimization action to choose in a particular context, and (2) what combinations of tradeoffs resulting from the application of a particular local optimization action constitutes an outcome that is acceptable or unacceptable to a user. These two concept types are recorded in the case base and used by CABINS to guide iterative optimization search and infer optimization tradeoffs to evaluate the current solution. The user's optimization preferences are not explicitly represented as case features or in terms of a cost function but are implicitly and extensionally represented in the case base.

3 User Optimization Preferences

User preferences influence decision-making in many ways and can be reflected in knowledge-based systems in several levels of abstraction [3]. In particular, we identify three types of preferences: *objectives*, *constraints* and *procedurals*. Objectives are related to the positive and negative consequences of the alternatives that a decision maker wishes to maximize or minimize. In decisions with multiple objectives, usually there are trade-offs between the objectives. The structure of the trade-offs could be very complicated in the sense that an analytically simple combination of the objectives might not be expressive enough to reflect the intentions and evaluation criteria of the user. Constraints consist of the conditions under which the objectives will be optimized. Essentially they define the limits of space of acceptable outcomes. Many of the constraints in practice turn out to be relaxable, thus increasing the number of potential solutions, but also making the problem solving more difficult, since the problem solver must decide (a) which constraints will be relaxed and (b) in which order. Finally, procedurals specify the preference over different sequencings of lower-level decisions in order to optimize the overall set of outcomes of decision making.

Although user preferences may change over time, it is often the case in many application domains that these changes won't occur very rapidly and drastically. More specificly, we assume that (1) the preferences change smoothly, (e.g., we are not expecting that the user will rapidly shift from maximizing a certain objective to minimizing it), and (2) the problem solving context will not change drastically over the problem solving horizon.

The main reason why we made these assumptions is that we expect that the information contained in current knowledge-base based on *previous* problem solving experiences should be *useful* and *relevant* for the current and future problem solving, although the knowledge based on previous experiences might be a little bit out-of-date. If the problem solving context and user preferences change drastically between problem solving sessions, the knowledge base must be rebuilt anew each time.

Changing by a considerable degree means that some of the previously acquired data might be out of date or irrelevant. This can be handled computationally in two ways. First, only keep the most recent cases as the basis of future reasoning, or second assign higher weight to temporally closer cases than to the more remote ones, when case similarity is calculated. However, since CBR is not an exact problem solving paradigm (it performs partial matching), and since it is difficult to determine such weights, we believe that reasoning based on a *rolling time window of data*, i.e., keeping an approximately constant number of the most recent cases is an effective way of reflecting user changing preferences [6, 15]. In order to make the idea of time window operational from the implementation point of view, the time window size must be determined. There exist some obvious trade-offs in terms of the window sizing. If the window is too large, the system will be *sluggish* in responding to user's preference changes. If the time window is too small, the cases might not adequately cover the problem space to reliably provide advice. There are no general ways to address these system design tradeoffs. We believe that they can be answered experimentally for each domain and task. We can, however, make some general observations. In domains with simple causal relationships, the size of the time window can be kept relatively small. On the contrary, a bigger time window might be indispensable for a more ill-structured domain.

4 Overview of CABINS

CABINS uses a *revision-based approach* for schedule optimization, i.e., a complete but suboptimal schedule is generated by dispatch heuristics or a constraint-based scheduler and then incrementally revised using revision actions, called repair tactics. In each revision iteration, CABINS tries to repair a particular activity, called a *focal_activity*. Repair means moving the activity to a different place in the schedule. In general, this will result in constraint violations that in turn must be resolved. Due to the tight constraints of job shop scheduling, these constraint violations can ripple through the whole schedule. To find an activity to repair, CABINS identifies jobs that must be repaired in the initial sub-optimal schedule, which are called *focal_job*'s. The activities of a focal_job are repaired in sequence starting with the earliest in the current schedule.

CABINS starts with an empty case base. For each activity repair attempt, CABINS interacts with the user to gather the following relevant information to be included in a case: (a) a suggestion of which local repair action to apply, (b) an evaluation of the repair result and (c) an explanation of an evaluation, if the user evaluates the repaired result as *unacceptable*. *A case in* CABINS *describes the application of a particular repair action to a focal_activity.* If a user evaluates the result of a repair attempt as unacceptable, he/she is asked to select another repair tactic and repair of the same focal_activity is attempted anew. When all available repair tactics have been unsuccessful in repairing a given focal_activity, the user tries to repair another activity. In this way, a number of cases are collected in the case base. CABINS can be used in the following modes:

- **Knowledge acquisition interactive** mode to acquire user preferences and generate the case base through interaction with the user.
- **Decision-support interactive** mode where the previously acquired case base that incorporates user preferences suggests revision actions and evaluation outcomes to the user who can accept a suggestion or override it with a new suggestion.
- **Automatic** mode where previously acquired user preferences are re-used to guide scheduling decisions without any interaction with the user. In automatic mode, the schedule is automatically generated and incrementally revised.

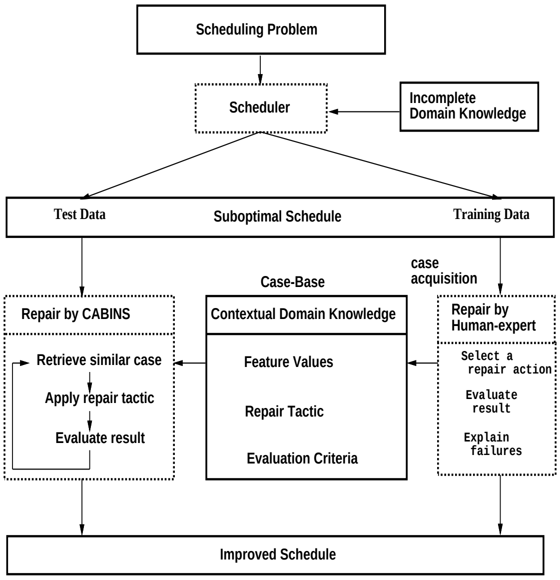

Figure 1: CABINS Architecture

The overall architecture of CABINS is depicted in Figure 1. CABINS consists of three modules: (1) an initial schedule builder, (2) an interactive schedule repair (case acquisition) module and (3) an automated schedule repair (case re-use) module.

Case Representation The content of a case is shown in Figure 2. The global features give an abstract characterization of potential repair flexibility or the lack thereof for the whole schedule. 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 details on case features, refer to [17].

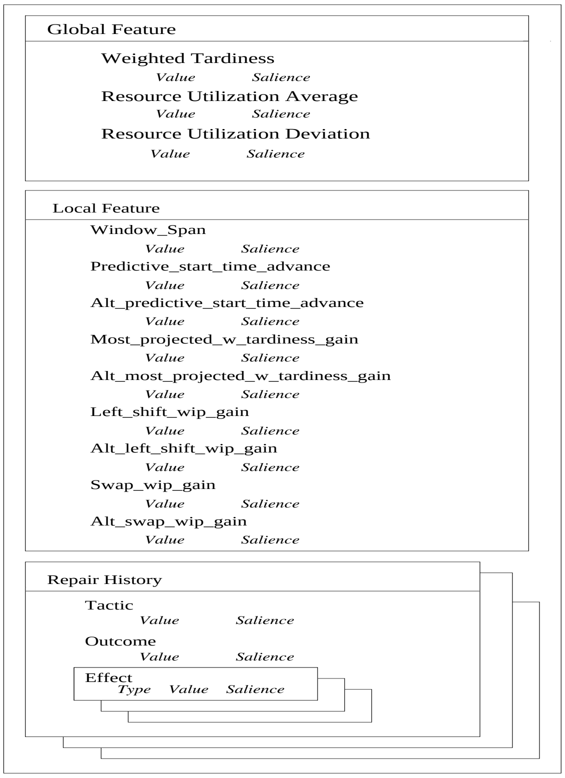

Figure 2: CABINS Case Representation

In order to bound the ripple effects of repair, a repair tactic is used only within a bounded time horizon, 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. CABINS currently has 11 repair tactics. Examples of repair tactics are: (1)*left_shift:* try to move the focal_activity on the same resource as much to the left on the time-line as possible within the repair time horizon, so as to minimize the amount of capacity over-allocation created by the move. (2)*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.

The repair history represents the sequence of applications of successive repair actions, the repair outcome and the repair effects. Repair effect values describe the impact of the application of a repair action on scheduling objectives (e.g., weighted tardiness, WIP). A repair outcome is the evaluation assigned by a user to the set of effects of a repair action. Typically user optimization criteria/preferences are reflected in the evaluation of the repair outcome. 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 (See Figure 2) is assigned when the outcome is 'unacceptable', and indicates the significance of the effect to the repair outcome.

Case Retrieval and 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, e.g., finding out all the tardy jobs, (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, (5) when the repair result is evaluated by CBR as *unacceptable*, deciding whether to give up or which repair tactic to try next.

As a case retrieval mechanism, CABINS uses a variation of k-Nearest Neighbor method (k-NN) [5]. For the detailed formula for similarity calculation, see [17].

5 Experimental Evaluation of Capturing Changing Preferences

Extensive experiments have been conducted with CABINS [16, 26, 25]. It has been experimentally shown that CABINS (1) is capable of capturing diverse *static* user optimization preferences and re-using them to guide solution quality improvement, (2) is robust in the sense that it improves solution quality independent of the method of initial solution generation, and (3) produces high quality solutions. In this paper, we report preliminary results from a set of experiments which are aimed at testing the following hypothesis: our CBR-based incremental modification and re-use methodology can be effective in capturing user's preferences *that change over time* which are reflected in the selection of schedule repair actions and evaluation of repair consequences. This hypothesis is very difficult to verify due to the subjective, ill-defined and dynamic nature of the user preferences. In order to evaluate the experimental results consistently, we built a rule-based reasoner (RBR) with known optimization function that goes through a trial-and-error repair process to optimize a schedule. For each repair, the repair effects were calculated and the corresponding repair outcomes were evaluated based on the optimization criteria known to the RBR. RBR was used to generate a case base for CABINS. Since RBR knows the exact objective function for evaluation, it can work as an emulator of a human scheduler, which cannot repair a schedule very efficiently, but can make consistent evaluations of repair results. Note that the objectives and user preferences, though known to RBR, are not known to CABINS and are only implicitly and extensionally reflected in the case-base. By incorporating explicit objectives into the RBR so they could be reflected in the case base we get an experimental baseline against which to evaluate the schedules generated by CABINS.

We evaluated our approach on a suite of job shop scheduling problems where parameters, such as number of bottleneck resources, range of due dates and activity durations were varied to cover a range of job shop scheduling problem instances with the following structure:

1. a range parameter controlled the distribution of job due dates and release dates, which might take one of the following values: *static* — both the due dates and release dates remained constant across all the problems within a class, *moderate* — the due dates and release dates varied for problems in the same class, but the variances were relatively small, *dynamic* — the variabilities among the dates were relatively big.

2. a bottleneck parameter controlled the number (1 or 2) of bottleneck machines.

Therefore, six groups of problems were generated with random assignment of *resource* and *execution duration* for each activity. For each group, 55 scheduling problem instances were generated randomly, resulting in a total of $55 \times 6 = 330$ problem instances. Each problem had 10 jobs and 5 machines. There were 5 activities for each job. Each job had a linear routing. Each activity could be executed on two substitutable machines. Bottleneck machines, however, had no substitutes.

A cross-validation method was used to evaluate the performance of CABINS. Each problem set in each group was divided in half. The training samples were repaired by RBR to gather cases. These cases were then used for case-based repair of the validation problems. We repeated the above process by interchanging the training and validation experimental sets. Reported results are for the validation problem sets.

5.1 Experimental Design

For each group of problem instances, the following steps were followed. First, 5 problems were randomly chosen out of the 55. These five problems were repaired by RBR using *Weighted Tardiness* as the optimization criterion. [1] The main reason for the creation of this initial case-base is to keep the size of the time window of cases approximately fixed. If we didn't construct the initial case-base, then the number of the cases used by CABINS to repair the first subset of validation problems would be roughly only half the number of the cases used by CABINS for other subsets of validation problem instances.

Second, the remaining 50 problems were divided into two subsets: one subset was the training sample which would be repaired by RBR to gather cases, the other subset served as the validation problem set to be repaired by CABINS. In order to simulate the dynamic preference changes, we randomly divided further the problem instances in both subsets into 5 categories, each of which contained 5 individual problem instances and was assigned to a different objective function respectively. Table 1 succinctly shows the experimental design.

[1] Using Weighted Tardiness as the evaluation criterion was an arbitrary decision. Any objective satisfying the assumption of smooth preference changes would be acceptable.

OBJ_1	$Weighted\ Tardiness$
OBJ_2	$0.8 \times Weighted\ Tardiness + 0.2 \times W.I.P.$
OBJ_3	$0.5 \times Weighted\ Tardiness + 0.5 \times W.I.P.$
OBJ_4	$0.2 \times Weighted\ Tardiness + 0.8 \times W.I.P.$
OBJ_5	$W.I.P.$

Table 1: Notations for Different Objectives

Assigning OBJ_i to the subsets of scheduling problem instances in a certain order can be viewed as a reasonable simulation of temporal transition of user preferences for obtaining results that satisfy a particular objective. The specific assignment of the objective functions $(OBJ_j, j = 1, \ldots, 5)$ to the subsets of problem instances is shown as follows: Let $ProblemSet_j^i$ denote a subset of problem instances, where i designates either repair by RBR to gather cases (when $i = RBR$) or repair by CABINS (when $i = CAB$). The subscript j takes the value $[1, 2, 3, 4, 5]$ to refer to one of the five subsets of the problems (each of them contains 5 problem instances), respectively. The objective function for evaluating the solution quality for the problems in $ProblemSet_j^i$ is OBJ_j, where $j = 1 \ldots 5$. Although we, the experiment designers, knew the objective function for every problem set and RBR also knew it (since the simulated human scheduler needs to know the objective function to perform consistent evaluation), CABINS didn't know the objective explicitly but only implicitly through its case base. To simplify the notation, we use $ProbleSet_0^{RBR}$ to denote the 5 problem instances we initially chose to be repaired by RBR to collect the initial case-base. The overall experimentation process was as follows:

1. Solve the problems in $ProblemSet_0^{RBR}$ using RBR to collect the cases. These cases will serve as the *set-up* problem solving experience. The objective used by RBR is OBJ_1. We denote the cases gathered in this step by $Cases_0$.

2. Solve the problems in $ProblemSet_1^{RBR}$ using RBR to accumulate cases based on the criterion OBJ_1. $Cases_1$ is used to stand for the cases RBR collected in this step.

3. Solve the problems in $ProblemSet_1^{CAB}$ through CABINS. The case-base used in this step consists of the cases included in $Cases_0$ and $Cases_1$.

4. Collect the cases using RBR through solving the problems in $ProblemSet_2^{RBR}$ based on the objective function OBJ_2. The cases are denoted by $Cases_2$.

5. Solve the problems in $ProblemSet_2^{CAB}$ using CABINS. The cases from $Cases_1$ and $Cases_2$ are utilized.

6. Solve $ProblemSet_j^i, j = 3, 4, 5$ in the same manner.

In general, the experiments followed the pattern: (1) accumulate the cases through RBR based on the problem solving experience on $ProblemSet_i^{RBR}$. The cases gathered are denoted by $Cases_i$, and (2) solve $ProblemSet_i^{CAB}$ using CABINS based on the cases from $Cases_i$ and $Cases_{i-1}$.

The experimental results presented in Table 2 show the overall average of CABINS performance across all 6 groups of problems. [2]

Obj.	Weight on Wei.Tar.	Weight on W.I.P.	Wei.Tar. improvement	W.I.P improvement
OBJ_1	1.0	0.0	20%	-10%
OBJ_2	0.8	0.2	18%	2%
OBJ_3	0.5	0.5	15%	7%
OBJ_4	0.2	0.8	10%	8%
OBJ_5	0.0	1.0	8%	10%

Table 2: Experimental Results: Quality Improvement when Preferences Change

From the results, we observe that CABINS is capable of automatically and dynamically adjusting its control knowledge to be biased according to the user preferences reflected implicitly in the case-base. When the user gave more importance to minimizing $Weighted\ Tardiness$, CABINS faithfully echoed that change in terms of focusing more efforts on $Weighted\ Tardiness$ rather than on $W.I.P.$ The same thing happened when the user was more interested in reducing $W.I.P.$.

We are currently conducting another set of experiments, where the sequence of steps in our initial experiments are shuffled (e.g., solving $ProblemSet_i^{CAB}$ based on the cases from $Cases_i$ and $Cases_{i+2}$), to investigate quantitatively the importance of the smoothness assumption of preference changing.

6 Conclusions

In this paper, we advocate a framework based on Case Based Reasoning for knowledge acquisition that can be reused as control knowledge to guide iterative revision for job-shop schedule optimization. The capability of acquiring user optimization preferences is important in ill-structured domains because typically a complete set of explicitly expressed objectives and constraints are not available. We believe that the general framework advocated here could be applied to additional ill-structured domains besides scheduling. CABINS provides a framework for alleviating the knowledge acquisition bottleneck. Our experimental results show the potential of the approach to capture and re-use user optimization preferences that were not explicitly known to the system. The results indicate that CABINS can reflect the trend of

[2] CABINS is implemented in C and all the experiments are conducted on a DEC5000 UNIX workstation.

the smooth transition of user's changing optimization preferences in the schedules it produces. Moreover, since CABINS can acquire preferences during routine system operation, we believe that it is useful in real operating environments.

References

[1] K.R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.

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

[3] J.M. Bradshaw and J.H. Boose. Decision analysis techniques for knowledge acquisition: Combining information and preferences using *aquinea* and *axotl*. *International Journal of Man-Machine Studies*, 32:121–186, 1990.

[4] A.R. Chaturvedi. Acquiring Implicit Knowledge in a Complex Domain. *Expert Systems with Applications*, 1992.

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

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

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

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

[9] J. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann Publishers, Inc., 1994.

[10] J.K. Lee, H.S. Kim, and S.C. Chu. Intelligent stock portfolio management system. *Expert Systems*, 6(2):74–87, April 1989.

[11] L.M. Lewis, D.V. Minior, and S.J. Brown. A Case-Based Reasoning Solution to the Problem of Redundant Engineering in Large Scale Manufacturing. *International Journal of Expert Systems*, 4(2):189–201, 1991.

[12] S. Mahadevan, T.M. Mitchell, J. Mostow, L. Steinberg, and P.V. Tadepalli. An apprentice-based approach to knowledge acquisition. *Artificial Intelligence*, 64:1–52, 1993.

[13] K. Mckay, J. Buzacott, and F. Safayeni. The scheduler's knowledge of uncertainty: The missing link. In *Proceedings of IFIP Working Conference on Knowledge Based Production Management Systems*, Galway, Ireland, 1988.

[14] T. M. Mitchell, S. Mahadevan, and L. Steinberg. Leap:a learning apprentice for vlsi design. In *Proceeding of the Ninth International Joint Conference on Aritificial Intelligence*, Los Angeles, CA, 1985. IJCAI.

[15] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7), July 1994.

[16] Kazuo Miyashita and Katia Sycara. Learning control knowledge through cases in schedule optimiztion problems. In *CAIA*. IEEE, 1994.

[17] Kazuo Miyashita and Katia Sycara. Cabins: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. *Artificial Intelligence*, To appear, 1995.

[18] D. S. Prerau. *Developing and Managing Expert Systems: Proven Techniques for Business and Industry.* Addison-Wesley, Reading, MA, 1990.

[19] M. Prietula, P.S. Ow, B. Huguenard, and S. Vicinanza. A critiquing model of flexible constraint evaluation for a scheduler's workbench. In *The First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, June 1988.

[20] C.R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems.* Halsted Press, New York, 1993.

[21] E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, Inc., 1991.

[22] David Ruby and Dennis Kibler. Learning Episodes for Optimization. In *Machine Learning : proceedings of the Ninth International Workshop (ML92)*, pages 379–384, 1992.

[23] E. Simoudis and J.S. Miller. The Application of CBR to Help Desk Applications. In *Proceedings: Case-Based Reasoning Workshop*, pages 25–36, 1991.

[24] K. Sycara. *Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods.* PhD thesis, School of Information and Computer Science Georgia Institute of Technology, Atlanta, GA, 1987.

[25] Katia Sycara and Kazuo Miyashita. Case-based acquisition of user preferences for solution improvement in ill-structured domains. In *Proceedings of AAAI-94*, Seattle, Washington, August 1994. AAAI.

[26] Dajun Zeng. Combined machine learning techniques in predictive and reactive scheduling. Graduate School of Industrial Administration, summer paper, Carnegie Mellon University, 1993.