Using Case-Based Reasoning as a Reinforcement Learning Framework for Optimization with Changing Criteria *

Dajun Zeng *Katia Sycara*
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
(412) 268-8815 (412) 268-8825
Fax: (412) 268-5569
zeng+@cs.cmu.edu katia@cs.cmu.edu

Abstract

Practical optimization problems such as job-shop scheduling often involve optimization criteria that change over time. Repair-based frameworks have been identified as flexible computational paradigms for difficult combinatorial optimization problems. Since the control problem of repair-based optimization is severe, Reinforcement Learning (RL) techniques can be potentially helpful. However, some of the fundamental assumptions made by traditional RL algorithms are not valid for repair-based optimization. Case-Based Reasoning (CBR) compensates for some of the limitations of traditional RL approaches. In this paper, we present a Case-Based Reasoning RL approach, implemented in the CABINS *system, for repair-based optimization. We chose job-shop scheduling as the testbed for our approach. Our experimental results show that* CABINS *is able to effectively solve problems with* changing *optimization criteria which are not known to the system and only exist implicitly in a extensional manner in the case base.*

1 Introduction

Consider an AI program (an agent) that must learn to solve real-world problems, assuming that no complete domain knowledge is available. For each problem it's trying to solve, it needs to collect information about the world (either from its sensors or from interaction with its user) and must choose an action to take. After executing the chosen action, the agent receives a signal (a *reinforcement* signal) from the world that indicates how well the agent is performing. The agent evaluates this *reinforcement* signal and decides either to go to another loop of sense-select-evaluate, or to terminate the problem solving process.

This learning scenario is quite different from standard concept learning, in which a teacher presents the learner with a set of input/output pairs. In the reinforcement learning (RL) scenario, the learner is not told anything about which action to take, but instead must discover which action yields the highest reward by trying different actions. Typically, actions may affect not only the immediate reward, but also the next situation, and through that all subsequent rewards [1].

In this paper, we present a learning agent that solves one of the "hardest" [2] combinatorial optimization problems, i.e., job-shop scheduling problems. Our approach, implemented in the CABINS system, is shown experimentally to be able to learn scheduling problem solving knowledge even when the scheduling criteria change over time. This capability is very important for the following 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 must be defined in advance of problem solving [3]. 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. Second, and equally important consideration is the fact that the problem solving environment and optimization criteria could be changing over time. Therefore, approaches that capture optimization criteria statically or require expensive knowledge-base updating are extremely limiting. On the other hand, approaches that utilize machine learn-

*This paper appears in The Proceedings of the 7th International Conference on Tools with Artificial Intelligence (ICTAI'95).

ing techniques to adapt their behavior to the changing objective criteria and problem solving context are much more promising.

Recently, repair-based optimization has been identified as a very flexible framework for solving optimization problems [4]. Reinforcement learning (RL) is particularly relevant and potentially useful within a repair-based framework. However, some basic assumptions that typical reinforcement learning methods [5, 1] make about the problem domain are violated for complex optimization tasks: (1) The reinforcement signal is typically assumed to be a scalar, which doesn't hold for real-world optimization tasks where evaluation criteria are situation-dependent and changing, (2) RL methods assume that there is an explicit criterion to tell the problem solver when the goal has been reached. However, for optimization tasks, except for toy problems, it is not possible to verify the optimality of a certain solution short of using exhaustive search, which is computationally prohibitive. To address these fundamental issues, instead of using classic reinforcement learning techniques, such as Q-learning [6], or connectionist-based approaches [7], we apply Case-Based Reasoning (CBR) [8] as the primary tool to (1) represent the state space implicitly and approximately in a case-base, (2) generate expected rewards associated with sample points in the state space based on previous problem solving experiences and knowledge about optimization criteria, (in some sense, an approximation of Q used in Q-learning is estimated through CBR), (3) choose the appropriate action at each decision-making point to maximize the expected reward, and (4) utilize failure information as a helpful index to explore temporal credit assignment information. Our experimental results show that CBR could be effectively incorporated within a RL context. Due to the approximate nature of CBR, when CBR-based selection and evaluation are applied in decision-making, we lose many nice properties which Temporal Differences-based approach [5] can provide, such as asymptotic convergence. We believe, however, that our CBR-based approach has good potential for handling much bigger search spaces since it doesn't require an explicit representation of the problem space, and attacking task domains with complicated and dynamically changing decision-making criteria and constraints.

The work reported here extends previous work on the CABINS system [9, 10, 11, 12, 13]. It tests the hypothesis that our CBR-based incremental repair methodology shows good potential within a reinforcement learning context to solve problems with optimization criteria that *change over time*. Our investigation was conducted in the domain of job shop schedule optimization and the experimental results, shown in section 4 confirmed this hypothesis.

2 Repair-based Optimization and Reinforcement Learning

A general optimization task can be described as follows:

$$\max f(x_1, x_2, \ldots, x_n)$$

subject to:	$C_j(x_1, x2, \ldots, x_n) \geq 0, j = 1, 2, \ldots, m$
where	$f(.)$: objective function
	$x_i, i = 1, 2, \ldots, n$: decision variables
	$C_j(.)$: constraints over the decision variables.

Two categories of problem solving strategies are commonly used to calculate the optimal solution $(\bar{x_1}, \bar{x_2}, \ldots, \bar{x_n})$ which maximizes $f(.)$. One of them is the constructive approach, which tries to find the optimal solution from scratch. At each problem solving step, only partial solutions are generated and/or assembled. The problem solving stops once a complete solution is attained, which is presumably optimal or satisficing. The other approach, called repair-based or revision-based, doesn't solve the optimization problem directly from scratch, but instead first finds an easy-to-compute, complete, and most likely suboptimal solution that is to be incrementally repaired to meet optimization objectives. The advantage of the repair-based approach for optimization problems for which there is no known efficient constructive algorithm has been recently realized by both Operations Research and AI communities [4, 14].

Within a repair-based optimization framework, the search space consists of all the possible solutions. The components of a repair-based approach are transform operators used for generating a new complete solution given an old one, and control knowledge for choosing the right transform operator so that a sequence of state transitions will lead to a global optimum. Typically, a certain transform operator focuses on one particular aspect of the problem and tries to improve it. Therefore transform operators are inherently of local nature.

Figure 1 shows a typical problem solving session using the repair-based perspective. Different search paradigms have been proposed to efficiently explore the search space, such as hill-climbing and some variation of hill-climbing including Simulated Annealing and Tabu search aimed at avoiding local minima. Hill-climbing-like searches might be very useful in situations where (1) no other available domain knowledge

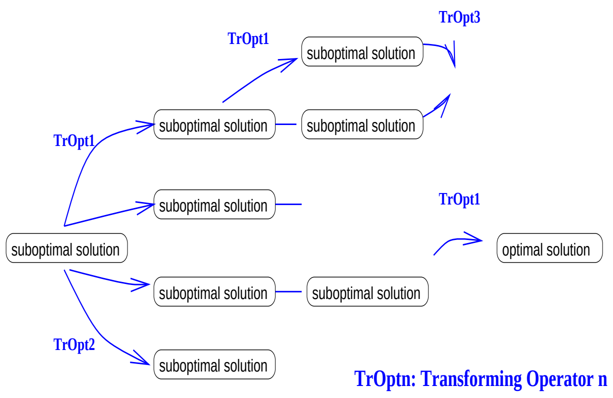

Figure 1: A repair-based Problem Solving Session

can be exploited except for the knowledge of objective criteria and transform operators, or (2) examples for learning must be generated through exploration of the search space. However, hill-climbing approaches are in general computation-intensive and not practical in real-world applications. Furthermore, in real world application, both the objective function and the constraints might change over time. A more flexible and powerful framework for solving complex optimization problems is needed.

It is straightforward to see the correspondence between the repair-based problem solving paradigm and the decision making model commonly used in the RL literature. Let S be the set of states and A be the finite set of actions available to the learning agent for optimization tasks. Each possible feasible solution (which is not necessarily optimal) corresponds to a state. The action set is comprised of all possible transform operators. At each time step t, the agent observes the system's current state $s_t \in S$ (a suboptimal solution), chooses an action $a_t \in A$ (a transform operator) and executes this action. As a result, the agent receives a payoff $R(s_t, a_t)$ [1] and the system makes a transition to a new state s_{t+1}. Unlike in traditional RL problems, the state transition is typically deterministic for repair-based optimization. Noticing the similarity between this re-formulated repair-based optimization problem and the traditional reinforcement learning decision-making model, one might conclude that reinforcement learning methods can be easily adapted to repair-based optimization problem solving. Unfortunately, this is not easy for the following reasons: (1) S for optimization tasks is potentially very large, and, more often than not, infinite, which makes it impossible to explicitly represent the state space, (2) a stopping criterion is not known for optimization problems, and (3) temporal credit assignment problems for optimization tasks tend to be very difficult.

[1] For optimization tasks, $R(s_t, a_t)$ is equal to 0 unless $f(s_t)$ reaches a maximum or is deemed to be satisfactory. A positive reward is assigned when a maximum or a satisfying solution is reached.

2.1 Job-Shop Scheduling

Scheduling deals with allocation of a limited set of resources to a number of activities. One of the most difficult scheduling problem classes is job shop scheduling [2]. 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 given partial ordering which reflects *precedence constraints*. Another type of constraints, *capacity constraints*, restricts the number of activities that can be assigned to a resource during overlapping time intervals. The goal of a scheduling system is to optimize the resulting schedule based on a set of objectives, such as minimize weighted tardiness, minimize inventory cost of Work-In-Process (WIP), etc. The scheduling problem is difficult to solve for a number of reasons. First, it is an NP-complete problem [2, 14]. Second, scheduling objectives are typically not well-defined and may be changing over time. 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? 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?

In this paper, we focus on solving schedule optimization problems where optimization criteria change over time [2] For experimental comparison of our approach with other scheduling methods, interested readers are referred to [12, 13]. The basic assumption we made about changing optimization criteria is that the changes occur smoothly, (e.g., we are not expecting that the user will rapidly shift from maximizing a certain objective to minimizing it), and the problem solving context will not change drastically over the problem solving horizon. 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 smoothly

[2] The capabilities of CABINS are more extensive than what we described here (e.g., CABINS has the capability of acquiring user preferences). However, in this paper, we restrict our attention on the role of CBR as a complementary framework for reinforcement learning.

changing preferences [15, 16]. Fore more detailed discussion, see [17].

3 Overview of CABINS

CABINS uses a *repair-based approach* for schedule optimization, i.e., a complete but suboptimal schedule is generated by OR-based 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, called *focal_jobs*, that must be repaired in the initial sub-optimal schedule. The activities of a focal_job are repaired in sequence starting with the earliest in the current schedule.

Case Representation *A case in* CABINS *describes the application of a particular repair action to a focal_activity.* The contents of a case include: (1) global features (e.g., Weighted Tardiness, Resource Utilization Average) which give an abstract characterization of potential repair flexibility or the lack thereof for the whole schedule, (2) local features associated with the focal_activity which potentially are predictive of the effectiveness of applying a particular repair tactic, and (3) repair history. For details, refer to [13].

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 actions. 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 effects. Repair effect values describe the impact of the application of a repair action on scheduling objectives (e.g., Weighted Tardiness, Work-In-Process Inventory (W.I.P.)).

Case Retrieval and Re-use CABINS repairs the schedules by: (1) recognizing schedule suboptimalities, 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 action to be used for each focal_activity. As a case retrieval mechanism, CABINS uses a variation of k-Nearest Neighbor method (k-NN) [18]. For the detailed formula for similarity calculation, see [13].

Since the number of possible schedules for job-shop scheduling is potentially infinite, explicit representation of the state space is impossible. In CABINS, the case-base reflects samples of state transition sequences that have been tried out. In addition, each state is represented in terms of abstract features (e.g., global, local features and repair history). This abstracted "representation" of a state, potentially corresponds to multiple schedule instances (multiple states in the repair-based solution space). During problem solving, CABINS uses partial matching to match a given schedule instance (a state) to an appropriate "abstracted" state in order to extract the appropriate control action and evaluates the applicability of this action in the current problem solving situation.

4 Experimental Evaluation of Capturing Changing Preferences

Extensive experiments have been conducted with CABINS [19, 11, 12]. It has been experimentally shown that CABINS (1) is capable of acquiring diverse *static* 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 aimed at demonstrating that CBR-based reinforcement learning can be effective in solving optimization problems with changing optimization criteria. In order to evaluate the experimental results consistently, we built a rule-based reasoner (RBR) with known optimization function that goes through a hill-climbing-based trial-and-error repair process to optimize a schedule. For each repair, RBR calculates repair effects and evaluates the corresponding repair outcomes. RBR generates a case base for CABINS. Note that the optimization criteria, 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 got an experimental baseline against which to evaluate the schedules generated by CABINS [13].

We evaluated our approach on a suite of job shop scheduling problems where parameters, such as number of bottleneck resources (1 bottleneck and 2 bottlenecks), range of the variations of due dates and activity durations (static, moderate, dynamic) were varied to cover a range of job shop scheduling problem instances.

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 has 10 jobs and 5 machines. There are 5 activities for each job. Each job has a linear routing. Each activity can be executed on two substitutable machines. Bottleneck machines, however, have no substitutes. Although the size of these problems may seem small to researchers outside the scheduling community, job-shop scheduling problems of this size have been recognized as very hard problems by AI and OR researchers [20, 14] and there are no known optimal solutions yet for these problems due to the large number of constraints.

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.

4.1 Experimental Design

For each group of problem instances, the following steps were followed. First, 5 problems were randomly chosen out of the 55. These 5 problems were repaired by RBR using $Weighted\ Tardiness$ as the optimization criterion. [3] 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 cases used by CABINS to repair the first subset of validation problems would be roughly only half the number of 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.

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 changing optimization criteria. 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. 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 cases. These cases will serve as the *set-up* problem solving experiences. 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. $Cases_1$ denotes 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 $Cases_2$.

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

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. [4]

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 optimization criteria reflected implicitly in the case-base. When the more important criterion was 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 criterion changed to give more weight to reducing W.I.P.

To test the scalability of our approach we conducted similar experiments on problems with 5 resource and 20 jobs (See Table 3). The pattern of results was the same as in Table 2.

5 Conclusions

In this paper, we advocated a Reinforcement Learning framework to learn control knowledge for iterative repair-based optimization of job-shop scheduling problems with changing optimization criteria. We presented the fundamental difficulties that traditional RL algorithms will run into in guiding repair-based optimization and proposed CBR techniques to address these problems. Our experimental results showed the potential of the approach to find sequences of appropriate control actions that effectively guide schedule optimization depending on the particular optimization criterion. We believe that the general framework advocated here could also be applied to other ill-structured domains. Current work focuses on investigating the theoretical modeling and algorithmic analysis of capturing changing optimization criteria, and analyzing quantitatively the importance of the smoothness assumption of preference changing.

[4] CABINS is implemented in C and all the experiments are conducted on a DEC5000 UNIX workstation. Since it is not possible to determine in general the optimality of a certain solution, CABINS terminates problem solving when a pre-set number of repairs have been tried.

Obj.	Weight on Wei.Tar.	Weight on W.I.P.	Wei.Tar. improvement	W.I.P improvement
OBJ_1	1.0	0.0	26%	5%
OBJ_2	0.8	0.2	23%	7%
OBJ_3	0.5	0.5	22%	9%
OBJ_4	0.2	0.8	21%	9%
OBJ_5	0.0	1.0	16%	12%

Table 3: Experimental Results for Problems with 5 resources and 20 jobs

References

[1] S. P. Singh, "Tansfer of learning by composing solutions of elemental sequential tasks," *Machine Learning*, vol. 8, pp. 323–339, 1992.

[2] S. French, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. London: Ellis Horwood, 1982.

[3] C. Reeves, ed., *Modern Heuristic Techniques for Combinatorial Problems.* New York: Halsted Press, 1993.

[4] B. D. M. Zweben, E. Davis and M. Deale, "Rescheduling with iterative repair," in *Proceedings of AAAI-92 workshop on Production Planning, Scheduling and control*, (San Jose, CA.), AAAI, 1992.

[5] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[6] C. J. C. H. Watkins, *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, England, 1989.

[7] L.-J. Lin, “Self-improving reactive agents based on reinforcement learning, planning and teaching,” *Machine Learning*, vol. 8, pp. 293–321, 1992.

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

[9] K. Sycara and K. Miyashita, “Learning control knowledge through case-based acquisition of user optimization preferences in ill-structured domain,” in *Machine Learning and Knowledge Acquisition: Integrated Approaches* (G. Tecuci and Y. Kodratoff, eds.), San Mateo, CA: Morgan Kaufmann, 1994.

[10] K. Sycara and K. Miyashita, “Adaptive schedule repair,” in *Knowledge Based Reactive Scheduling* (E. Szelke and R. Kerr, eds.), pp. 107–124, Amsterdam, Holland: North Holland, 1994.

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

[12] K. Sycara and K. Miyashita, “Case-based acquisition of user preferences for solution improvement in ill-structured domains,” in *Proceedings of AAAI-94*, (Seattle, Washington), AAAI, August 1994.

[13] K. Miyashita and K. Sycara, “Cabins: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair,” *Artificial Intelligence*, vol. 76, no. 1–2, 1995.

[14] T. E. Morton and D. W. Pentico, *Heuristic Scheduling Systems: With Application to Production Systems and Product Management*. New York, N.Y.: John Wiley and Sons Inc., 1993.

[15] L. Dent, J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski, “A personal learning apprentice,” in *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI, 1992.

[16] T. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski, “Experience with a learning personal assistant,” *Communications of the ACM*, vol. 37, July 1994.

[17] K. Sycara, D. Zeng, and K. Miyashita, “Using case-based reasoning to acquire user scheduling preferences that change over time,” in *The Proceedings of the Eleventh IEEE Conference on Artificial Intelligence Applications (CAIA ’95)*, (Los Angeles), IEEE, Feb. 1995.

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

[19] K. Miyashita and K. Sycara, “Improving schedule quality through case-based reasoning,” in *Proceedings of AAAI-93 Workshop on Case-Based Reasoning*, (Washington, DC), pp. 101–110, AAAI, 1993.

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