CABINS : A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair

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

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

September 15, 1994

*This paper was submitted to *Artificial Intelligence Journal*

Abstract

Practical scheduling problems generally require allocation of resources in the presence of a large, diverse and typically conflicting set of constraints and optimization criteria. The ill-structuredness of both the solution space and the desired objectives make scheduling problems difficult to formalize. This paper describes a case-based learning method for acquiring context-dependent user optimization preferences and tradeoffs and using them to incrementally improve schedule quality in predictive scheduling and reactive schedule management in response to unexpected execution events. The approach, implemented in the CABINS system, uses acquired user preferences to dynamically modify search control to guide schedule improvement. During iterative repair, cases are exploited for: (1) repair action selection, (2) evaluation of intermediate repair results and (3) recovery from revision failures. The method allows the system to dynamically switch between repair heuristic actions, each of which operates with respect to a particular local view of the problem and offers selective repair advantages. Application of a repair action tunes the search procedure to the characteristics of the local repair problem. This is achieved by dynamic modification of the search control bias. There is no a priori characterization of the amount of modification that may be required by repair actions. However, initial experimental results show that the approach is able to (a) capture and effectively utilize user scheduling preferences that were not present in the scheduling model, (b) produce schedules with high quality, without unduly sacrificing efficiency in predictive schedule generation and reactive response to unpredictable execution events along a variety of criteria that have been recognized as important in real operating environments.

1 Introduction

The scheduling task can be described as assigning a limited number of resources to activities over time in a consistent manner, i.e. so as to avoid violation of constraints associated with the problem, such as resource capacity constraints, activity precedence constraints and release dates. The goal of a scheduling system is to produce schedules that respect these problem constraints and optimize a set of objectives, such as minimize tardiness of jobs, minimize work in process inventory (WIP), maximize resource utilization, minimize cycle time etc. The produced schedule should also respect user preferences. Scheduling is difficult to automate for the following reasons:

1. Computational Complexity

   Scheduling is a problem in the "hardest" subset of NP-complete problems [Fre82].

2. Tight Constraint Interactions

   Due to the tight interactions among scheduling constraints and the non-linear nature of scheduling objectives, there is no general way to predict the effect of a local optimization decision on global optimality, even for the simplest objective.

3. Ill-structured Objectives / Preferences

   For practical scheduling problems, it is desirable that multiple optimization objectives (e.g. minimize weighted tardiness, minimize work in process inventory, maximize resource utilization) must be satisfied. Moreover, optimization objectives often interact and conflict with one another. To optimize along one objective alone could jeopardize optimality along other objectives. The relationships between global objectives are extremely difficult to model.

   The definition/evaluation itself of what is a "high quality" schedule is fraught with difficulties because of the need to balance conflicting objectives and tradeoffs among them. Such tradeoffs typically reflect the presence of context-dependent user preferences and domain constraints not captured in the scheduling model. The value of incorporating such user preferences and constraints in operational scheduling

environments is becoming increasingly recognized (e.g. [MBS88]) but good techniques are currently lacking.

4. Dynamic Environment

   Operational environments for scheduling systems (e.g. factories) are dynamic. Unpredictable events, such as machine breakdown or operator absence, often happen during schedule execution. Therefore, a schedule that is only *predictive* (i.e. it is created assuming that the world is static and predictable) will be brittle. It is clear that any effective scheduling system should be *reactive*, i.e. perform schedule revision in response to unforeseen events during schedule execution.

The scheduling problem has been addressed by two general types of methods, *constructive* scheduling and *revision-based* scheduling. In constructive approaches (e.g., [Fox83, Sad91]), a schedule is constructed by incremental construction and merging of partial schedules. In revision-based approaches (e.g., [MJPL90, ZDG90, BC91, LAL92]) a complete but suboptimal initial schedule is incrementally repaired by several techniques, such as a min-conflict heuristic [MJPL90] or simulated annealing. In [OST88], while predictive schedules are generated from scratch, incremental revision has been used to repair a pre-computed schedule in response to unanticipated events during schedule execution. The approach analyzes the implications of specific schedule features and matches them to behavioral characteristics of appropriate reactive actions that are selected according to a static, pre-determined control model. These approaches assume the existence of an explicit optimization function. This assumption is in general limiting since, in practice, optimization criteria reflect context-dependent user preferences and cannot be expressed in terms of a single global objective function.

In this paper, we describe a revision-based approach, implemented in the CABINS system, that provides a unified framework for acquisition of user optimization preferences and tradeoffs, improvement of schedule quality based on these preferences, and reactive schedule management in response to unforeseen events. Unlike other systems that utilize iterative repair to find a feasible solution (e.g. [ZDG90, MJPL90]), where executability of the schedule was not guaranteed at the end of each repair iteration, CABINS produces an executable schedule after each repair that has guaranteed monotonic increase in quality the more time it is allowed for repair, thus exhibiting *anytime*

executable behavior [DB88]. This is a very desirable quality especially in reactive contexts since there could only be a certain limited amount of time for the system to react.

Our approach uses integration of Case-based Reasoning (CBR) [KSS85] and fine granularity constraint-directed scheduling mechanisms based on [SF90]. Integrating CBR with constraint-based scheduling stems from a variety of motivations. Although scheduling is an ill-structured domain, we assume that it exhibits domain regularities that could be captured, albeit only approximately, in a case. In CABINS, a case represents application of a revision action to one activity in the schedule, thus expressing dependencies among features of the schedule, the repair context and a suitable repair action (see section 4.1 for a detailed description of case representation). CBR allows capture and re-use of this dependency knowledge to dynamically adapt the search procedure and differentially bias scheduling decisions in future similar situations. On the other hand, because of the tightly coupled nature of scheduling decisions, a revision in one part of the schedule may cause constraint violations in other parts. Therefore, constraint propagation techniques are necessary to determine the *ripple effects* that spread conflicts to other parts of the schedule as case-based repair actions are applied and specific schedule revisions are made. The evaluation criteria for judging the acceptability of the outcome of a repair action are often multiple, conflicting, context dependent and reflect user judgment of tradeoffs. Therefore, it is difficult to describe the evaluation criteria and the associated tradeoffs in a simple manner. The case base incorporates a distribution of examples that collectively and implicitly capture a user's schedule evaluation preferences and tradeoffs under diverse problem solving circumstances and enable CABINS to induce these tradeoffs from the case base. Hence, user preferences are reflected in the case base in two ways: as *preferences for selecting a repair action* depending on the features of the repair context, and as *evaluation preferences* for the repair outcome that resulted from selection and application of a specific repair action.

A revision-based approach is attractive for solving practical scheduling problems. There are no known efficient search algorithms for schedule optimization except for a very limited set of simple objectives such as make-span (e.g. [ABZ88]) and the amount of computation required for finding a solution is generally unpredictable [Fre82]. Therefore, the construction of a cheap but suboptimal schedule that is then incrementally repaired to meet optimiza-

tion objectives is preferable in practice, because one can interrupt the repair process and use the interim result for execution when no more time is allowed for further repair. For example, dispatch heuristics have very low computational cost, but due to their myopic nature, they must be tailored to particular optimization objectives. Hence, in general they cannot address issues of balancing tradeoffs with respect to a variety of optimization objectives. As a consequence, they result in suboptimal schedules. However, because of their efficiency, they are widely used by practitioners. Therefore, as has already been pointed out by other researchers (e.g., [ZDB+92, MJPL92]), combining a repair methodology, such as a simple gradient search [KS90], neural networks [Joh90], or the one advocated in our work, with a dispatch driven scheduler for creation of the initial schedule is promising for real world scheduling environments. Experimental results reported in section 5.2.1 indicate that CABINS can produce substantial schedule improvements starting with schedules generated by several methods i.e. a number of dispatch heuristic and a constraint based scheduler.

Our approach was evaluated through extensive controlled experimentation on job shop scheduling problems. Experimental results, reported in section 5 show that (1) the approach is potentially effective in capturing user preferences and optimization tradeoffs that are difficult to model, (2) it improves schedule quality irrespective of method of initial schedule generation, (3) it produces high quality schedules at much lower computational cost as compared to simulated annealing, a well-known iterative repair method, and (4) it is suitable as a reactive scheduling method because it maintains high schedule quality and minimizes disruptions in the face of execution time failures.

The rest of the paper is organized as follows: section 2 gives some background in job shop scheduling and presents the constraint-based techniques used in CABINS. Section 3 introduces case-based schedule optimization. Section 4 presents case representation, indexing, retrieval and application to the schedule of a retrieved revision.It also presents an extensive example. Section 5 presents experimental results to validate the approach. Section 6 discusses related work and section 7 conclusions and future work.

2 Job Shop Scheduling

Job shop scheduling deals with allocation of a limited set of resources to a number of activities associated with a set of jobs/orders. The dominant constraints in job shop scheduling are *temporal activity precedence* and *resource capacity* constraints. The activity precedence constraints along with a job's release date and due date restrict the set of acceptable start times for each activity. The capacity constraints restrict the number of activities that can use a resource at any particular point in time and create conflicts among activities that are competing for the use of the same resource at overlapping time intervals. The goal of a scheduling system is to produce schedules that respect temporal relations and resource capacity constraints, and optimize a set of objectives. In our model we allow substitutable resources for each activity of a job, thus being able to deal with *parallel machine job shop scheduling*, a more complicated version of the job shop scheduling problem [MP93]. CABINS's revision based approach has two-phases: (1) create an initial schedule by utilizing any method (e.g. dispatching rules), and (2) improve the (possibly) suboptimal schedule that was generated in the first step so as to incorporate user preferences and tradeoffs.

In the rest of this section, we present the job shop scheduling problem within the framework of constraint satisfaction, and present the search strategy that is used to propagate the effects of repair actions in CABINS.

2.1 Constraints

The job shop scheduling problem requires scheduling a set of jobs $J = \{J_1, \ldots, J_n\}$ on a set of physical resources $RES = \{R_1, \ldots, R_m\}$. Each job J_l consists of a set of operations/activities $A^l = \{A^l_1, \ldots, A^l_{n_l}\}$ to be scheduled according to a process routing that specifies a partial ordering among these activities (e.g., A^l_i BEFORE A^l_j).

Each job J_l has a release date rd_l that signifies the earliest time the job can be started and a job due date dd_l, by which the job should be finished. Each activity A^l_i has a fixed duration du^l_i and a variable start time st^l_i. The domain of possible start times of each activity is initially constrained by the release and due dates of the job to which the activity belongs. In order to be successfully executed, each activity A^l_i requires p^l_i different resources (e.g., a milling machine, a jig and a machinist) R^l_{ij} $(1 \leq j \leq p^l_i)$, for each

of which there may be a pool of physical resources from which to choose, $\Omega_{ij}^l = \{r_{ij1}^l, \ldots, r_{ijq_{ij}^l}^l\}$, with $r_{ijk}^l \in RES (1 \leq k \leq q_{ij}^l)$ (e.g., several possible milling machines).

More formally, the problem can be defined as follows:

VARIABLES : A vector of variables is associated with each activity, $A_i^l (1 \leq l \leq n, 1 \leq i \leq n_l)$, which includes:

1. the *activity start time*, st_i^l, and
2. each *resource requirement*, $R_{ij}^l (1 \leq j \leq p_i^l)$ for which the activity has several alternatives.

CONSTRAINTS : The non-unary constraints of the problem are of two types:

1. *Precedence constraints* defined by the process routings translate into linear inequalities of the type: $st_i^l + du_i^l \leq st_j^l$ (i.e. A_i^l BEFORE A_j^l);
2. *Capacity constraints* that restrict the use of each resource to only one activity at a time translate into disjunctive constraints of the form: $(\forall p \forall q R_{ip}^k \neq R_{jq}^l) \vee st_j^k + du_i^k \leq st_j^l \vee st_j^l + du_j^l \leq st_i^k$. Constraints simply express that, unless they use different resources, two activities A_i^k and A_j^l cannot overlap [1].

Time is assumed discrete, i.e., activity start times and end times can only take integer values. Each resource requirement R_{ij}^l has to be selected from a set of resource alternatives, $\Omega_{ij}^l \subseteq RES$. These constraints include non-relaxable release dates, and initially, non-relaxable due dates between which all activities in a job need to be performed.

2.2 Objectives and Preferences

In practice, scheduling objectives are numerous, complex, often conflicting and the mathematics of the problem can be extremely difficult with even the simplest of objectives [Fre82]. Below, we define the objectives,

[1] These constraints have to be generalized when dealing with resources of capacity larger than one.

that are among the most common in the literature (e.g. [Fre82]), that we used to develop the performance evaluation of CABINS. These objectives are mathematical simplifications of state-dependent objectives that are difficult to model precisely. For example, an optimization criterion such as $WEIGHTED.TARDINESS^2 \times WIP^3$ could be induced by CABINS if user gave consistent evaluation of schedules, but cannot be easily represented in ways that can be explored by traditional schedule approaches.

Waiting time (W_i^l) **:** is the time that elapses between the completion of the preceding activity A_{i-1}^l (or rd_l, if $i = 1$) and the start of processing A_i^l.

Total waiting time (W_l) **:** is the sum of waiting time of all activities that belong to J^l. Clearly $W_l = \sum_{i=1}^{n_l} W_i^l$.

Completion time (C_l) **:** is the time at which processing of J_l finishes. We have the equality: $C_l = rd_l + \sum_{i=1}^{n_l}(W_i^l + du_i^l)$

Lateness (L_l) **:** is simply the difference between the completion time and the due date of J_l : $L_l = C_l - dd_l$.

Tardiness (T_l) **:** is delay in the completion of J_l against its due date dd_l. Note that T_l always takes non-zero value. Thus $T_l = \max(0, L_l)$.

Flowtime (F_l) **:** is the amount of time that J_l spends in the system. $F_l = C_l - rd_l$ or $F_l = \sum_{i=1}^{n_l}(W_i^l + du_i^l)$

Make-span (C_{max}) **:** is the latest completion time of the entire orders. $C_{max} = \max C_i$

Work-in-Process Inventory(WIP) **:** is the summation of total waiting time. $WIP = \sum_{i=1}^{n} W_i$

Weighted Tardiness (T_{wt}) **:** is the weighted average of tardiness. Weight is considered as a penalty cost of being tardy. $T_{wt} = \sum_{i=1}^{n} w_i T_i$

The quality of a schedule is a function of the extent to which it achieves *user's preferences*. We illustrate the necessity of having user's preferences in the scheduling system by using a very simple example. We assume the simplest factory with a single machine and two jobs. Each job consists of a

single activity to be processed on the factory machine. Let us further assume that the two jobs are released to the factory floor at the same time.

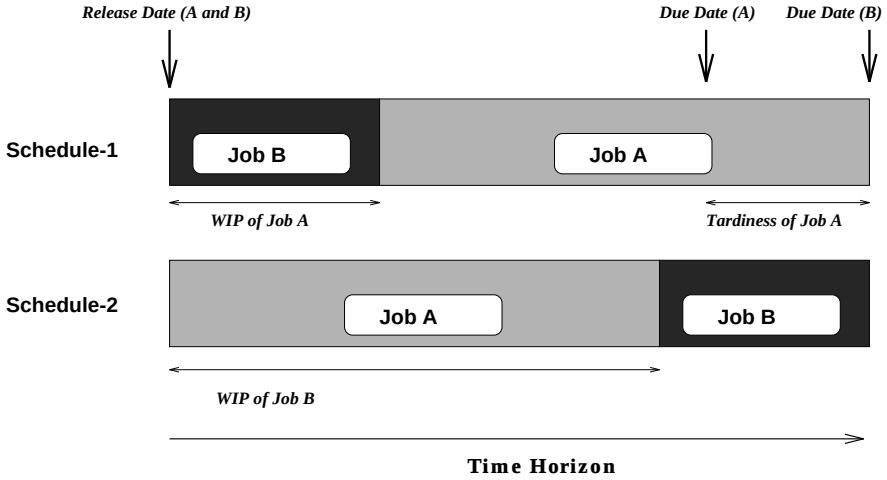

Figure 1: **Example of Conflicting Objectives**

Figure 1 shows two schedule results for this problem. Suppose schedule-1 is generated. In this schedule, job B finishes before its due date but job A is tardy. The WIP of job A is indicated in figure 1 (the WIP of job B is zero). Suppose one wishes to revise the schedule to reduce the tardiness of job A. In this simple schedule, the only possible repair is to switch the positions of job A and job B. The schedule resulting from this switch is schedule-2. In schedule-2, neither job is tardy but the WIP in schedule-2 (the WIP of job B) is larger than in schedule-1. Even in this extremely simple example, it is difficult to decide which schedule is of higher quality without taking into consideration the preferences of the user. Simply adding WIP plus weighted tardiness and minimizing the sum may not be realistic since the relative importance of each of these objectives in the overall sum reflects the tradeoffs the user is willing to make. These tradeoffs may depend upon many factors, such as the importance of the client of each job, past shipping records, load of a factory/warehouse and so on. The combination of those factors produces enormous number of contexts in which user preferences are considered, thus making user's preferences difficult to capture and represent a priori in the problem model. That is the reason that the authors think acquiring preferences adaptively is important.

2.3 Constraint-Based Search Procedure

The constraint-based search procedure used in CABINS for applying a selected repair action (see section 4.4) is based on [SF90, Sad91]. Search is interleaved with the application of consistency enforcing mechanisms and variable/value ordering heuristics that attempt to avoid dead-end states. A search state is associated with each partial solution. Each search state defines a new constraint satisfaction problem whose variables are the variables that have not yet been instantiated and whose constraints are the initial problem constraints along with constraints reflecting current assignments. A schedule is built by opportunistically selecting an activity to be scheduled and assigning to it a *reservation*, i.e. a resource and a start time. Each time a new activity is scheduled, new constraints are added to the initial scheduling constraints that reflect the new activity reservation. These new constraints are then propagated (consistency checking). If an inconsistency (i.e., constraint violation) is detected during propagation, the system backtracks. Otherwise the scheduler selects a new activity to schedule and a reservation for that activity. The process terminates when all activities have been scheduled successfully.

More specifically, search proceeds according to the following steps:

1. If all operations have been scheduled then stop, else go on to step 2;
2. Apply the *consistency enforcing* procedure;
3. If a dead-end is detected then backtrack (i.e. select an alternative reservation if one is left and go back to step 1, otherwise stop)
4. Select the next operation to be scheduled (*variable ordering* heuristic);
5. Select a promising reservation for that operation (*value ordering* heuristic);
6. Create a *new search state* by adding the new reservation assignment to the current partial schedule and go back to step 1;

The details of each step are as follows:

Consistency Enforcement : The consistency enforcing procedure is a hybrid procedure that differentiates between precedence constraints and

capacity constraints. It guarantees that dead-end states only occur as the result of capacity constraint violations. Essentially, consistency with respect to precedence constraints is enforced by updating in each search state a pair of earliest/latest possible start times for each unscheduled operation.

Consistency enforcement with respect to capacity constraints tends to be significantly more expensive due to the disjunctive nature of these constraints. For capacity constraints, a forward checking type of consistency checking is generally carried out by the system. Whenever a resource is allocated to an operation over some time interval, the forward checking procedure checks the set of remaining possible start times of other operations requiring that resource, and removes those start times that would conflict with the new assignment.

Variable Ordering : Because scheduling is NP-hard, it is important to focus search in ways that avoid dead-end states. This is accomplished by utilizing good *variable* (i.e., activity) and *value* (i.e., reservation) ordering heuristics. A *variable ordering* determines which activity is going to be scheduled next and *value ordering* determines which reservation should be assigned to the selected activity. The *variable ordering* heuristic utilized in the system is called Activity Resource Reliance (ARR) [SF90] and selects the *most critical activity first*, i.e., the activity with the highest probability of being involved in a capacity constraint violation over particular time intervals. For more details on the approach, see [SF90].

Value Ordering : Once the activity to be scheduled next has been selected, the *value ordering* heuristic determines which reservation to assign to the activity. The two value ordering heuristics relevant to this paper are:

Least Constraining Value Ordering (LCV) : This heuristic selects the reservation that is the least likely to prevent other activities from being scheduled. LCV uses an unbiased utility-function (see figure 2) for each activity, i.e. there is no preference for a particular start time out of the activity's available start times.

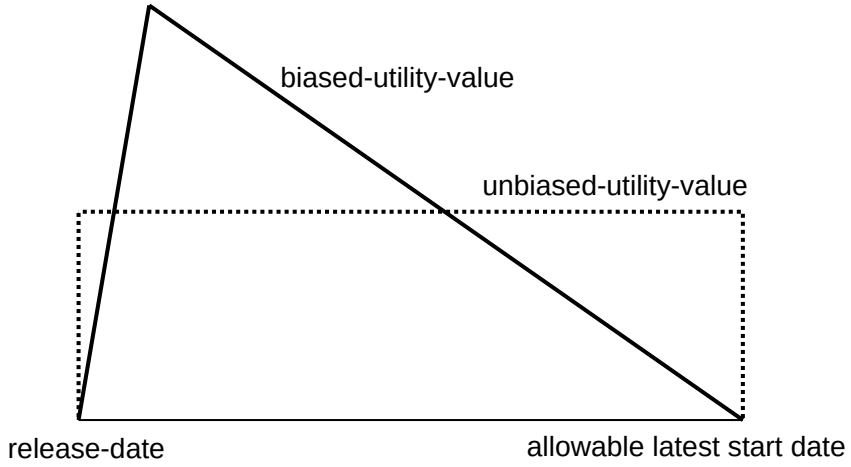

Figure 2: Utility-Functions

Greedy Value Ordering (GV) : This heuristic selects a reservation based on local preferences that are expressed via *static piecewise linear biased utility-function* associated with each activity (see figure 2). This biases value ordering to prefer activity start times with high utility values. For scheduling problems with substitutable resources, static utilities that express differential resource preferences are used in the selection of an activity's reservation.

Experiments in [SF90] [2] on some rather small job shop problems (each with 20 activities) indicate that the ARR variable ordering with LCV value ordering produces suboptimal schedules with minimal backtracking; ARR variable ordering with GV value ordering with statically predetermined utility functions, henceforth referred to as constraint-based scheduling (CBS), was shown to produce high quality schedules as compared to the SMU heuristic ([KY89]).

In CABINS, schedule revision proceeds iteratively, one activity at a time. The set of activities that get involved in constraint violations as a result of repairing one activity is the *conflict set* of the repair. The repair process unschedules the activities in the conflict set and modifies the bias of the utility

[2] The experiments were run on 20 randomly generated scheduling problems.

functions associated with them. This bias reflects the effects of learning context-dependent user preferences and evaluations of repair outcomes that have been stored in the case base. The search procedure with the *modified utility functions*, ARR variable ordering and GV value ordering is used to schedule the conflict set activities that got unscheduled during repair. In other words, each time an activity is repaired, CBS is used to reschedule a subset of the activities (i.e. the members of the conflict set) of the overall schedule with *utility functions that have been adaptively modified* based on information in the case base. Section 4.4 describes the repair process in detail.

3 Case-based Schedule Optimization

In order to optimize schedules to user's satisfaction, we need to know context-dependent user preferences and represent them in the scheduling system to be exploited in the reasoning process. Rule-based approaches, while having the potential to capture context-dependent tradeoffs in rules, require considerable knowledge acquisition effort [Pre90]. Our approach uses case-based reasoning (CBR) which has the potential for dealing with noisy data [RK92, AKA91], acquiring user knowledge in complex domains [Cha93, MBS88], and expending less effort in knowledge acquisition compared with knowledge acquisition for rule-based systems [SM91, LMB91].

Because of the characteristics of the scheduling domain described in the previous section and our interest in capturing context dependent user preferences, CBR seems 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 [Ham89, KH92, Vel92], legal reasoning [Ash87, RA88], argumentation [Syc89], conceptual design [SGK+91], medical diagnosis [Kot88] where the primary concern has been plausibility or correctness of the resulting artifact (plan, argument, design) and computational efficiency of the process rather than artifact quality.

The challenges we faced were to decide 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. This solution is attractive since, if the right information could be transferred from one

scheduling scenario to another, or with little adaptation, the new problem would be solved with relative ease. However, 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 amount of modification needed and the quality of 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 problems that must be avoided [Ham89] is a good guideline and has served many CBR systems well. However, in our 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. Therefore, 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 we could capture a user's evaluation of the results of a single scheduling decision in a case, and, if the result was unacceptable, we could apply another scheduling decision to the same scheduling entity until either all available scheduling decisions had been exhausted or an acceptable result had been obtained. Therefore, it became clear that it was better to have a single activity/operation of a scheduling job as the "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 our purposes. Moreover, contextual information, which can only be provided by having a

complete schedule, is very useful in applying CBR. Therefore, revision-based scheduling was chosen as the underlying scheduling methodology.

Hence in CABINS, a case describes the *application of a schedule revision decision on a single activity of a job.* Operationalization of a schedule revision decision is done by means of a *schedule repair action.* We have identified two classes of schedule repair actions (i.e. strategy and tactic), described in detail in section 4. We use constraint propagation to propagate the effects of a schedule repair action to the rest of the schedule. Each application of a repair 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 ones 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 three categories. The first category consists of the *global features.* Since the results of schedule revision associated with a single activity pertain to the whole schedule, global features that express characteristics of a whole schedule are relevant and operate as contextual information for selection of a particular repair action. The *local features* comprise the second category. Since it is not possible to predict in general the bounds of repair necessitated by application of a repair action (due to constraint ripple effects), and since reasoning about the effects of a repair action on the whole schedule a priori would amount to unlimited lookahead analysis which is in general intractable, we confine the range of lookahead analysis to a limited *repair time horizon* (see section 4.1). Associated with this time horizon, there are local features that allow CABINS to estimate the effects of each repair action.

The schedule resulting from application of a repair action 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 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 third category of case features. Therefore, the case base incorporates a distribution of examples that collectively capture repair performance tradeoffs under diverse scheduling circumstances.

CABINS searches the space of complete schedules. Control for this search is provided by CBR in two ways: First, search control is provided through case-based selection of the next repair action to be applied and second through case-based evaluation of the outcome for the schedule that resulted from application of a selected repair action. The global and local 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. For a more detailed description of case representation and indexing, see section 4.1.

4 CABINS Overview

In CABINS, there are two general types of repairs: *repair strategies* and *repair tactics*. A repair strategy is associated with a particular high level description of classes of schedule defects. Each repair strategy has a variety of repair tactics associated with it. The repair tactics are appropriate for particular specializations of the defect classes. We have identified two general types of repair strategies: *local patching* and *model modification*. Local patching is the selection of repair actions that result in changing the sequence of activities allocated to different resources, or rearranging resource assignments. Local patching is in general less costly and disruptive to factory operations. For example, if the repair goal is to reduce job tardiness, specific local patching strategies include "reduce the slack between activities in the tardy job", and "reduce the idle-time of resources needed by activities in the tardy job". Model modification reformulates the problem by changing model parameters, such as the number of jobs to be scheduled, or global constraints such as changing release or due dates, increasing resource capacity or increasing number of shifts. Model modification strategies facilitate the solution of the problem, since they amount to global constraint relaxations. However, in practice, model modification strategies are costly to implement (e.g., buy new equipment, pay for extra shifts in a factory, subcontract jobs to outside contractors). The default CABINS strategy is local patching, a computationally more challenging task since the system must improve the schedule without relaxing the already imposed constraints (except due date constraints). If local patching is unsuccessful in fulfilling the repair goal, the repair episode is considered a failure. Our experiments were run within these

more stringent assumptions.

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

CABINS can operate in the following modes that exhibit different levels of autonomy:

- **Knowledge acquisition interactive** mode to acquire user preferences and generate the case base.
- **Decision-support interactive** model 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 reused to guide scheduling decisions without any interaction with the user.

In the experiments reported in section 5, CABINS operated autonomously. The repair process in autonomous operating mode has the following basic steps:

1. A job in the initial suboptimal schedule is randomly identifies to be repaired. The random job selection is necessary since CABINS does not have explicit optimization criteria that it could use to select jobs to be repaired in a more informed fashion.
2. The job under current repair consideration is called the *focal_job* and the activity under current repair consideration is called the *focal_activity*. Repair is performed one activity at a time. Activities in a focal_job are repaired in a forward fashion starting with the earliest activity of that job that has "enough" upstream slack. This mechanism focuses attention on activities that have enough slack so they can be moved, thus (a) avoiding unnecessary computations, and (b) limiting the amount of ripple effects (schedule disruption) that could be caused by moving

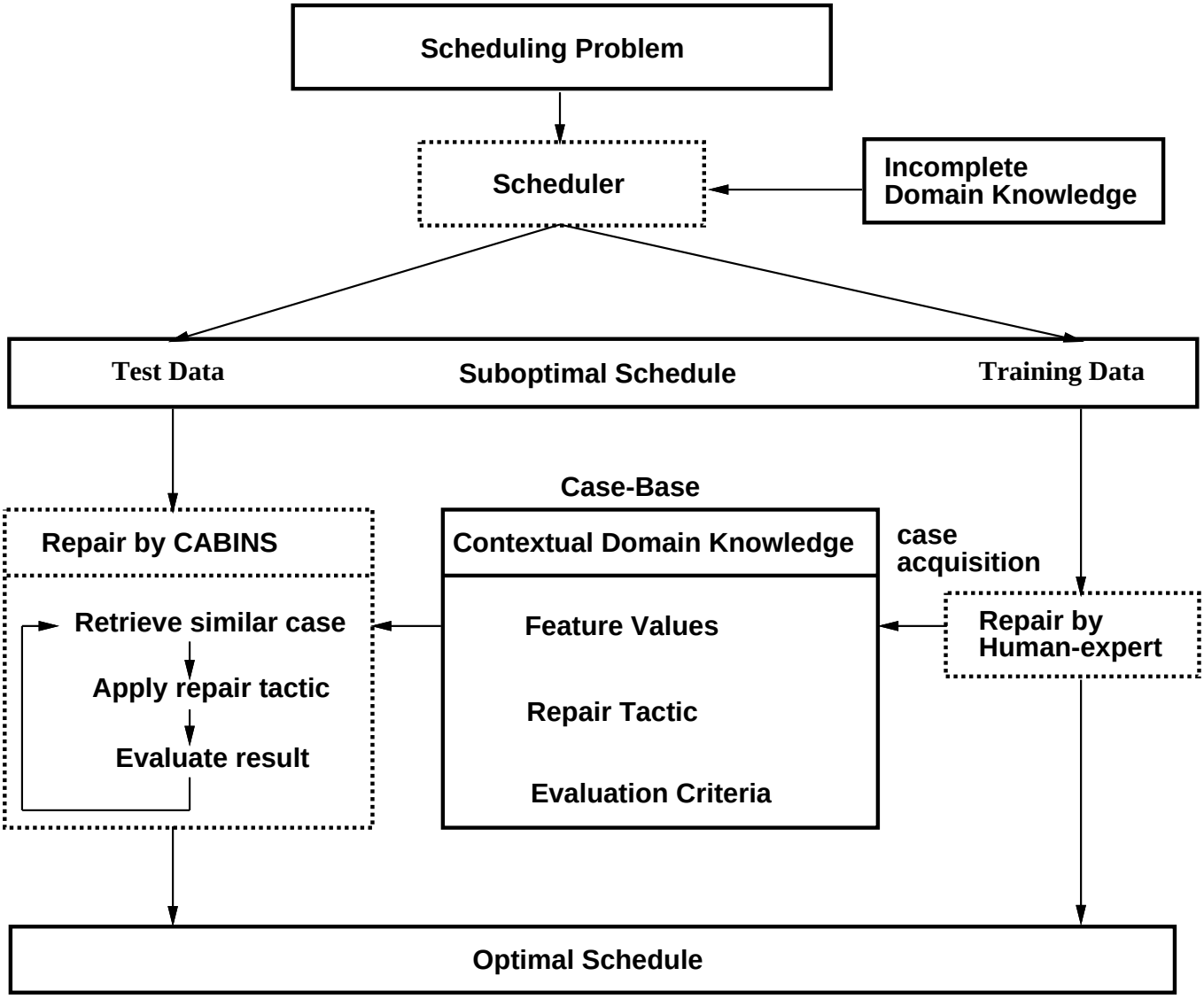

Figure 3: CABINS Architecture

activities that are too tightly scheduled and whose move would cause many constraint violations [3].

3. A repair strategy/tactic is selected for the current problem using CBR and is applied. Application of a repair tactic (described in section 4.4) consists of three parts: (a) identifying the activities, resources and time intervals that will be involved in the repair, i.e. the current conflict set, (b) change the utility functions associated with activities in the conflict set, and (c) using the constraint-directed scheduler with utilities assigned in step (b) to make the resource reservations for the activities identified in step (a).

4. After a repair has been executed, CBR is used to predict and evaluate the repair outcome in the context of the current case-base.

5. If repair is deemed a success, find next activity to repair, else (if repair outcome is a failure), CBR is invoked to select the next repair tactic to repair the current focal_activity.

4.1 Case Representation

The repair process should exploit knowledge relating both to the continuing validity of various scheduling decisions, the flexibility of current time and capacity constraints, the trade-offs that are implied by a particular repair, and whether the repair was successful or unsuccessful according to the user's judgment. Figure 4 shows the information content of a case. Appendix A shows an example of a case instance that is in CABINS's case base.

A case describes the application of a particular repair action to an activity. Because of the ill-structuredness of the domain, case features are heuristic approximations that reflect regularities of revision-based scheduling. For example, one of the regularities that would be useful to represent would be repair flexibility, i.e. the notion of how much freedom there is in the current schedule for moving an activity to a new position. Global case features (figure 4) reflect potential repair flexibility for the schedule as a whole. High resource utilization, for example, often indicates a tight schedule

[3] In the current implementation, "enough" upstream slack is heuristically determined as twice the tardiness of the focal_job.

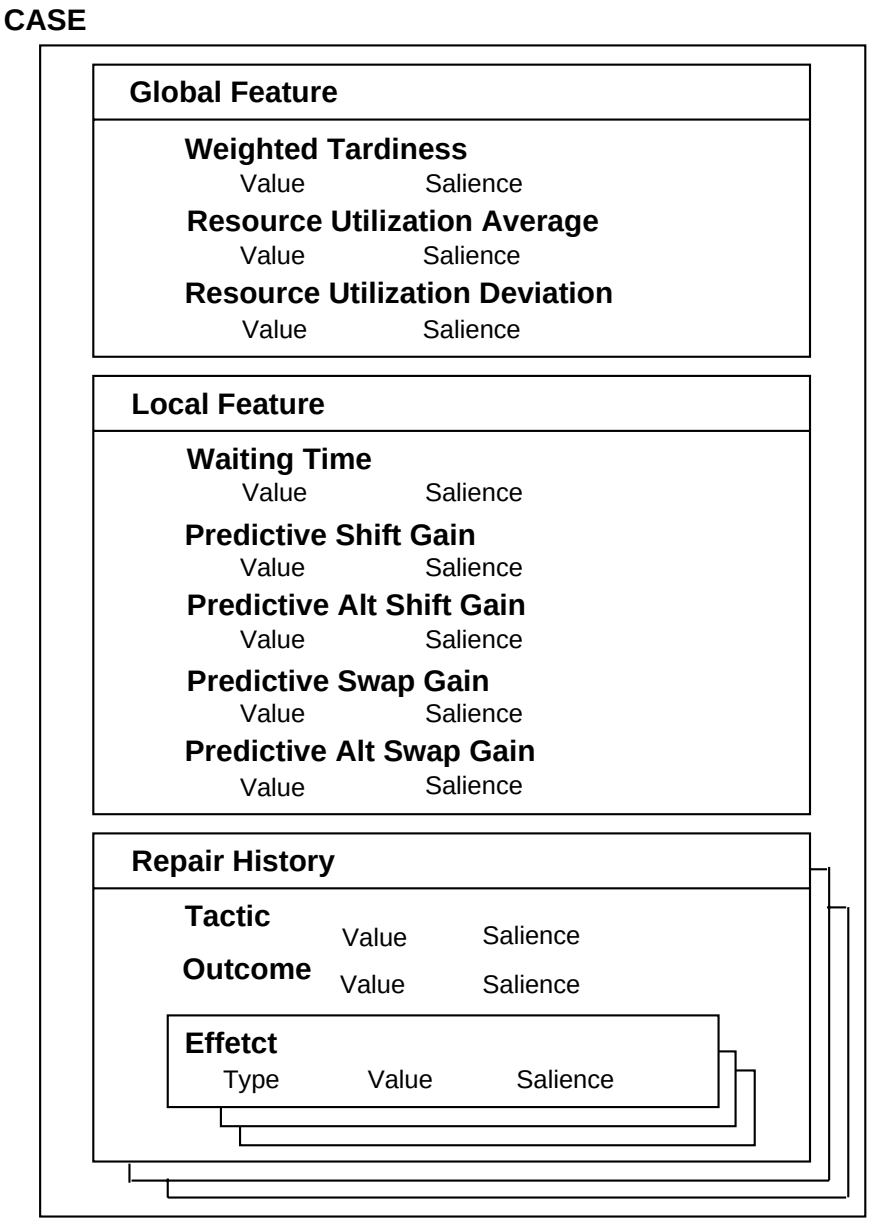

Figure 4: Case Representation

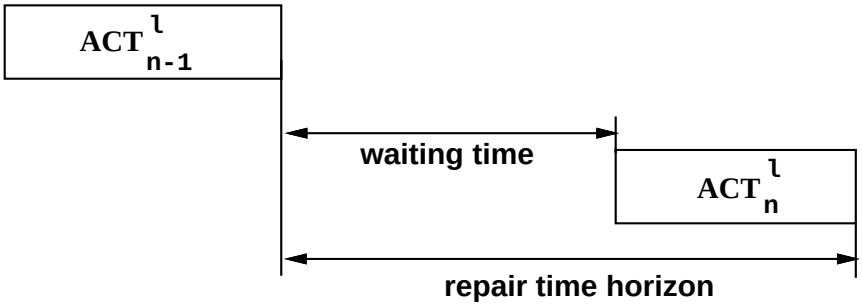

Figure 5: Repair time horizon of focal_activity(ACT_n^l)

without much repair flexibility. High standard deviation of resource utilization indicates the presence of highly contended-for resources which in turn indicates low repair flexibility. Local features reflect flexibility for schedule revision within limited temporal bounds. In particular, the temporal bound that CABINS uses is a time interval called *repair time horizon*. The repair time horizon of a focal_activity is the time interval between the end of the activity preceding the focal_activity in the same focal_job and the end of the focal_activity (see figure 5). The local features that we have identified are in the same spirit as those utilized in [OST88]. 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.

The repair history records the sequence of applications of successive repair actions, the repair effects and the repair outcome. Repair effects describe the impact of the application of a repair action on schedule optimization objectives (e.g., weighted tardiness, WIP). Typically these effects reflect tradeoffs among different objectives. A repair outcome is the evaluation assigned to the set of effects of a repair action and takes values in the set ['acceptable', 'unacceptable']. This judgment is made in the training phase and gets recorded in the case base. An outcome is 'acceptable' if the tradeoffs involved in the set of effects for the current application of a repair action is judged acceptable. If, during case acquisition, the outcome is judged as "unacceptable", the application of the repair tactic is considered a failure and an explanation that expresses tradeoffs with respect to balancing favorable and unfavorable

outcomes on optimization objectives is provided. If during CBR repair the repair outcome is deemed unacceptable, another tactic is selected from success cases to repair the same activity, using as indices global and local case features, the failed tactic, and the indication of the failed outcome. This CBR invocation retrieves similar past failures of the tactic that were successfully repaired and the tactic that was eventually successful in fixing the past failure. The assumption here is that a similar outcome for the same tactic implies similarity of causal structure between the past and current case. Therefore, the eventually successful tactic of a similar failure can potentially be successful in the current problem.

4.2 Case Acquisition

In CABINS, the session starts with an empty case-base. A set of training problems are presented to the user who interacts with CABINS to repair schedules by hand. At first, the user selects the repair tactic that is deemed to be appropriate and uses CABINS's tactic application procedure (see section 4.4) to apply the chosen tactic to the current schedule.

The effects of the repair are calculated. An effect describes the result of the repair with respect to one or more repair objectives. Effects pertain to either the schedule as a whole or to a job. Possible effects pertaining to a schedule as a whole are: weighted tardiness, average resource utilization, deviation of resource utilization, total schedule WIP. Effects that pertain to a job are changes in the tardiness of the job, changes in work-in-process inventory, or changes in resource assignment. So, for example, the tradeoff between utilizing a less preferred machine to reduce a job's tardiness can be reflected in these effects. Due to tight constraint interactions, these effects are ubiquitous in job shop scheduling and make schedule optimization extremely hard. When application of a repair tactic produces a feasible result, the user must decide whether the resulting schedule is acceptable or not based upon those calculated effects. An example of these effects is shown in Appendix A.

An outcome is judged as unacceptable, if the schedule resulting from the application of the revision heuristic does not make any improvement with respect to the user's criteria. This could happen because harmful effects outweighed, in the user's judgment, the effected improvement. For example, if reduction of job tardiness enforces increased utilization of low-quality machine, although the total cost of this repair may be low, it may be unac-

ceptable to a user who worries that the quality of resulting products might be low. Therefore such a repair might be judged as unacceptable. The user's judgment as to balancing favorable and unfavorable effects related to a particular optimization objective constitute the explanations of the repair outcome. The user supplies an explanation in terms of rating the importance of each effect (denoted by "salience" in figure 4). At the end of each repair iteration, the applied repair tactic, the effects of the repair and user judgment / explanation as to the repair outcome are recorded in a case along with the current problem features. If the effects are acceptable to the user, the repair outcome is recorded as "acceptable" and the user tries to repair another activity. If the user does not like the tradeoffs that are incorporated in the repair effects, then the outcome of the current repair tactic ("unacceptable"), the effects calculated by CABINS and the salience assigned by the user are recorded in the repair history of the case. Subsequently, the user tries to utilize another repair tactic to repair the same activity.

The process continues until an acceptable outcome is reached, or failure is declared. Failure is declared when all available tactics have been used to repair an activity, but the user finds each repair outcome unacceptable. The sequence of application of successive repair actions, the effects, user's judgment and explanation in case of failed application are recorded in the repair history of the case. Two remarks are in order here with respect to case acquisition. First, a new case is acquired only when a new activity is under repair. When an activity is repeatedly repaired due to unacceptable repair tactic application results, no new case is acquired, but the repair history of the same case is augmented by each successive repair tactic application, its effects and outcome. In this way, a number of cases are accumulated in the case-base. In section 5, we describe how the cases used in our experiments were acquired. Moreover, in section 5.3 we report current experimental results to investigate the tradeoffs incurred when CABINS operates with different size case bases.

4.3 Case Retrieval

Once CABINS has constructed a case-base from training data, it can perform schedule repair without any interaction with its user. Retrieved cases are for three purposes: selection of a repair tactic to be applied, evaluation of the resulting schedule after application of the selected repair tactic, and, in case of failure, retrieval of a tactic that had fixed a previous similar failure. In

each of these three situations, CABINS utilizes a different set of indices for case retrieval. In order to retrieve cases to select a repair tactic, global and local features of the current case (the current focal_activity) are used. The process of applying a repair tactic is described in section 4.4.

After a repair has been applied and, if the result is a feasible schedule, repair evaluation is performed through CBR. Using the effect features (type, value, and salience) as new indices, CBR is invoked and returns an outcome in the set (acceptable, unacceptable).

If the outcome of current revision is decided as unacceptable, CABINS performs another CBR invocation using as indices the conjunction of the current outcome (unacceptable), the failed heuristic and the case global and local features to find another possibly applicable revision heuristic. Invoking CBR with these indices retrieves cases that have failed in the past in a similar manner as the current revision. This use of CBR in the space of failures is a domain-independent method of failure recovery [Syc88, Sim85], and allows the problem solver to access past solutions to the failure. If the result is acceptable, then CABINS proceeds to repair another activity.

For each of the three case retrieval situations described above, CABINS uses a k-Nearest Neighbor method (k-NN) [Das90] for case retrieval. The space over which the k-Nearest Neighbor calculation is done is the set of features corresponding to each of the three retrieval situations. For example, for case retrieval to select a repair tactic, k-NN is used over the space defined by the values of global and local features. A k-NN calculation finds the k-nearest neighbors, where k is some constant of the current problem from the training data based on pre-determined similarity measures and, in its simplest form, a single nearest neighbor is found and chosen as a classification result.

We selected k-NN instead of 1-NN for the following reasons. [4] In domains, such as scheduling that do not have clear predictive features due to lack of causal structure, there can be many matches other than the nearest match that can potentially contribute to accurate classification. If a large number of near neighbor cases are of the same category (e.g. suggesting swap as the tactic to be applied), a higher confidence can be given to the classification result than if the near neighbors are of many different categories (e.g. some suggesting left_shift and some suggesting swap). For example, in deciding the repair tactic to be applied to the current problem, suppose that we have

[4] In the current implementation of CABINS, k is set to 5.

five nearest neighbors. Three of them are left_shift cases, whose similarity to the current problem is 0.9, 0.2 and 0.1 and the other two are swap cases, whose similarity is 0.8 and 0.75. If we use 1-NN, left_shift is selected as a repair tactic because the nearest retrieved case (with similarity 0.9) uses left_shift as a successful revision tactic. In this method, the occurrence of multiple cases suggesting a different classification result with relatively high similarity could potentially be ignored. We use the sum of the similarity in k-nearest neighbors as a selection criterion, instead of using the frequency of appearance of a class among k-nearest neighbors, in order to avoid the situation where dissimilar cases may have an undue influence on the classification result [5]. In the previous example, swap is selected as a repair tactic by CABINS (since its total similarity is 1.55 vs.1.2 of left_shift).

The similarity between a case and the current problem is computed in CABINS as follows:

$$
\begin{aligned}
Distance_i &= \sqrt{\sum_{j=1}^{N}(Salience_j^i \times \frac{CaseFeature_j^i - ProblemFeature_j}{E\_Dev_j})^2} \\
Similarity_i &= \exp(-Distance_i)
\end{aligned}
$$

where $Salience_j^i$ is the salience of j-th feature of i-th case in the case-base, $CaseFeature_j^i$ is the value of j-th feature of i-th case, $ProblemFeature_j$ is the value of j-th feature in the current problem, E_Dev_j is the standard deviation of j-th feature value of all cases in the case-base, and $Distance_i$ is the dissimilarity between i-th case and the current problem. And $Similarity_i$ is the similarity between i-th case and the current problem.

We utilize the normalized Euclidean distance to measure the dissimilarity between a case and a problem. This prevents certain features from dominating distance calculation merely because they have large numerical values.

4.4 Repair by CABINS

Repair of a schedule is performed by applying the repair tactics selected in each repair iteration by CBR. The repair tactics currently available in CABINS are:

[5] This method has been successfully applied in domains without clear causal structure, such as English word pronunciation and text classification in [SW86, CMSW92].

left_slide : try to move focal_activity on the *same resource* as much to the left on the timeline as possible within the repair time horizon, while preserving the sequence of all the activities.

left_shift : try to move focal_activity on a *same resource* as much to the left on the timeline as possible within the repair time horizon while minimizing the disruptions.

left_shift_into_alt : try to move focal_activity on a *substitutable resource* as much to the left on the timeline as possible within the repair time horizon while minimizing the disruptions.

swap : swap the focal_activity with the activity on its left on the *same resource* within the repair time horizon which causes the least disruptions.

swap_into_alt : swap the focal_activity with activity on its left within the repair time horizon which causes the least disruptions by changing the resource assignment of the focal_activity to a *substitutable resource*.

give_up : give up a further repair of the current focal_activity.

In recent work we have expanded the set of tactics to 11 and are currently performing additional experiments with them. The process of applying a repair tactic has the following steps:

1. Determine the *predictive* start time of the focal_activity being repaired. The predictive start time of an activity is a temporary start time that is calculated by each repair tactic as a desirable start time for a focal_activity. The ripple effects of a repair, the conflict set, consists of all the activities that may need to be re-scheduled due to constraint violations arising from moving the focal_activity to the predictive start time. Note that this predictive start time may *not* be exactly the same as the start time that will result from execution of the repair (step 5 below).

    - For left_shift or left_shift_into_alt, the "predictive" start time is the start time that minimizes capacity over-allocation as a result of moving the focal_activity on the same (or substitutable) resource within the focal_activity's repair time horizon.

- For swap or swap_into_alt, the "predictive" start time is the start time that causes the least amount of precedence constraint violations on the same (or substitutable) resource within the focal_activity's repair time horizon.

2. Project the effects of moving the focal_activity to the predictive start time and designated resource. This is done by performing constraint propagation to identify capacity constraint violations.

3. Adjust the reservations of all the activities in the conflict set by simple right-shifting or left-shifting so that all conflicts are resolved.

4. Change the bias of the start time utility function (see Fig. 2) of the activities in the conflict set in favor of start times calculated in step 3. If the tactic being applied involves a substitutable resource, also change the resource utility-function so that the substitutable resource has utility higher than the resource on which the focal_activity is currently scheduled. Changing the utility functions biases selection of start times by the value ordering heuristic (section 2.3) in favor of those with higher utility values, thus reflecting the preferences encoded in the case base.

5. Unschedule the focal_activity and all members of its conflict set and re-schedule them using the opportunistic constraint-directed scheduler with ARR variable ordering, GV value ordering and the utility functions defined in step 4.

6. Restore the start time utility-function of the affected activities to reflect no bias for the next repair iteration.

The above process results in a conflict free revised schedule. The effects of the revision are calculated, and CBR is invoked with the effects as the relevant indices to evaluate the repair outcome. Note that an activity A_i^j can be moved under two different situations. First, A_i^j can be moved when it is the current focal_activity. Second, it can be moved when it is in the conflict set of another focal_activity.

Figure 6 gives a detailed example that graphically shows how the local repair action *left_shift* can be applied. In this simplified example, we have

three jobs and each of them has three activities. Suppose the current focal_activity is A_3^3 and *left_shift* has been chosen as the repair tactic. The first step of revision is to find an appropriate start time for activity A_3^3. Left_shift dictates that activity A_3^3 should be starting as soon as possible within the given repair time horizon. Therefore, the utility function associated with A_3^3 that used to reflect the preference for starting A_3^3 as late as possible (indicated in the figure by "Utility function of A_3^3 before repair") is adjusted accordingly. In the figure, the new utility function is indicated as "Utility function of A_3^3 after adjustment". The next step is to find the conflict set which consists of all affected activities by moving A_3^3 to the left. The members of the conflict set are shown in the figure. The utility function of each activity in the conflict set is also adjusted to reflect these changes. In the figure, we show as an example the adjustment of the utility for activity A_1^2. After these utility functions have been adjusted, the focal_activity and the activities in the conflict set are unscheduled and the constraint-based scheduler is called to re-schedule them. The resulting repaired schedule is shown at the bottom of the figure 6.

4.5 An Example

We briefly illustrate the repair process with a very simple example schedule to be repaired shown in figure 7 . In the gantt chart, each row shows assignments of activities on each resource, along the timeline, and each white box corresponds to an assignment of an activity. The number inside a white box identifies the job which the activity belongs to. For example, the first activity on resource2 is the first activity of Job2, identified in our text as A_1^3. We write R_i to indicate the ith resource, J_j to identify the jth job and A_k^n to identify the kth activity of job n. The example has ten jobs $(J_1, \ldots, J_{10})$ and each job has five activities with the linear precedence constraint. (e.g., A_1^n BEFORE A_2^n, ... , A_4^n BEFORE A_5^n). Resources R_1 and R_2, R_3 and R_5 are substitutable; resource R_4 is a bottleneck. Suppose that the current focal_job is J_8 and the current focal_activity is A_4^8. The indices used to retrieve the similar cases from the case-base are calculated as follows:

1. Global features:

   Weighted Tardiness: In this particular case, the weighted tardiness of the whole schedule is 460.

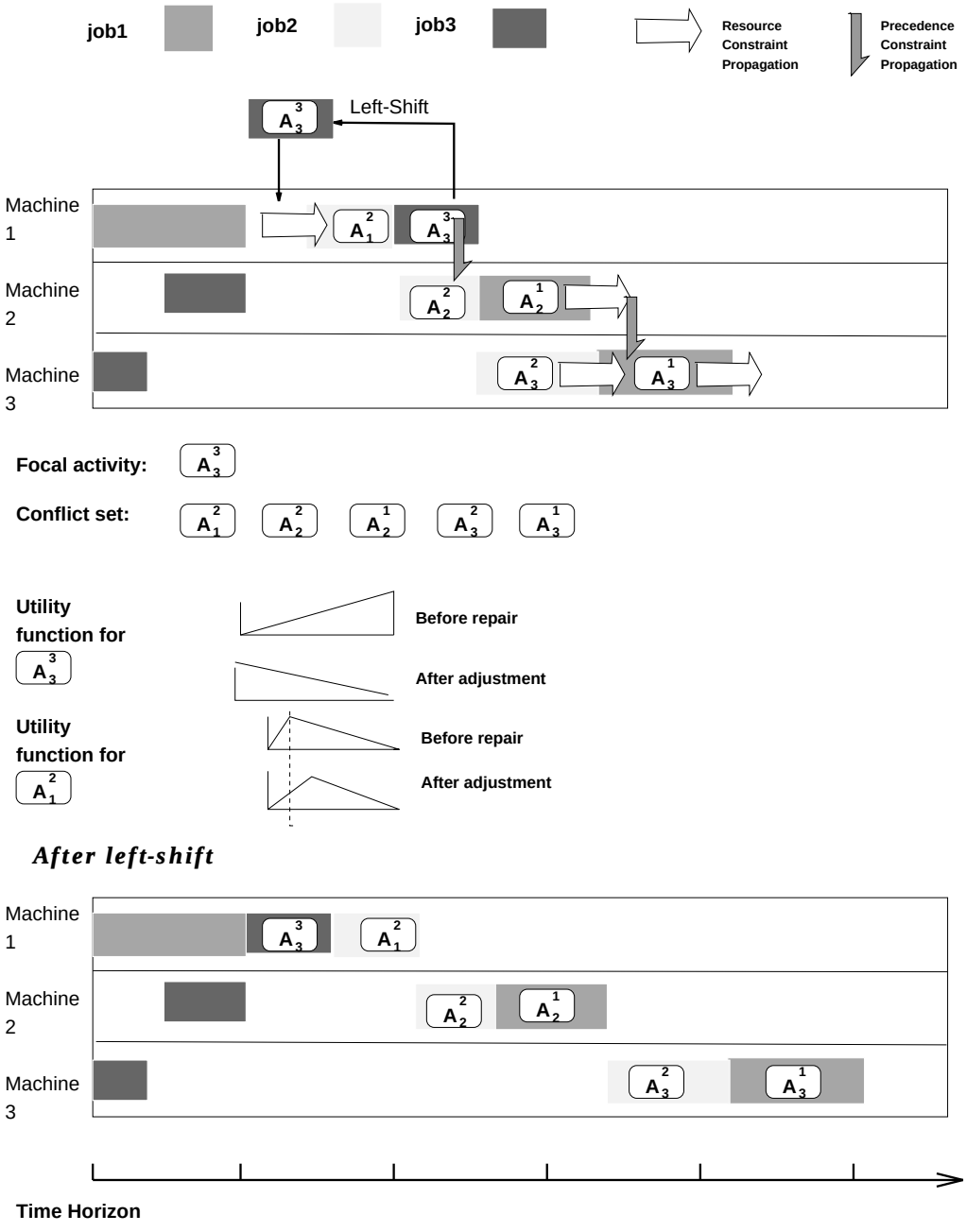

Figure 6: Example of repair tactic application: left_shift

Resource Utilization Average: This feature can be calculated as the ratio of overall utilization of resources to overall availability of resources. The value of this feature is 0.544.

Resource Utilization Deviation: The deviation of resource utilizations across the different resources is equal to 0.032.

2. Local features:

Waiting Time: This feature is defined as the time elapsing between the completion of the preceding activity (A_3^8) and the start of the present focal activity (A_4^8). In our case, it is equal to $1180 - 620 = 560$.

The *predictive_shift_gain* is computed in CABINS as follows:

$$\frac{predictive\_start\_time - current\_start\_time}{waiting\_time} \times repairability$$

where *predictive_start_time* is defined in section 4.4, *current_start_time* and *waiting_time* are the parameters associated with focal activity. We heuristically estimate the *repairability* within the given repair time horizon by a hyperbolic tangent function.

For our example, the value of *predictive_shift_gain* for A_4^8 is 0.705.

Predictive Alt Shift Gain: The calculation of this feature is very similar to that of *predictive_shift_gain*. In this case, since the required resource of activity A_4^8 is a bottleneck resource, R_4, that does not have any substitutable resources, the value of *predictive_alt_shift_gain* is 0.

Predictive Swap Gain: To calculate *predictive_swap_gain*, CABINS uses the same formulas as for *predictive_shift_gain*, but the *predictive_start_time* is calculated differently. (See Section 4.4) For this example, *predictive_swap_gain* is 0.96.

Predictive Alt Swap Gain: The value of this feature is 0 since A_4^8 requires the bottleneck resource R_4 which does not have substitutable resources.

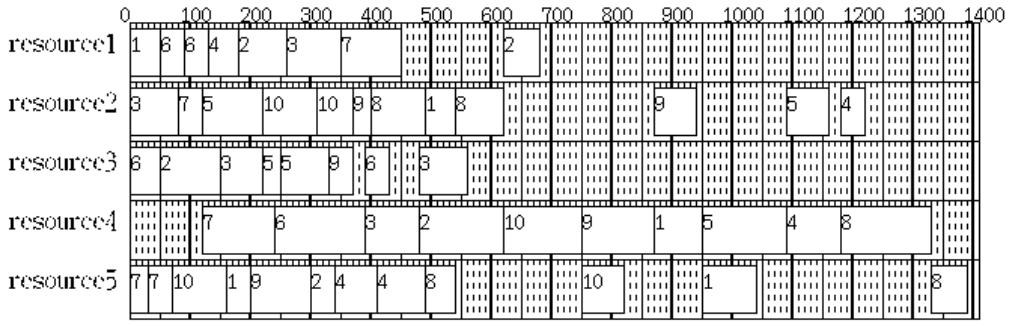

Figure 7: Original Schedule Results

Case based retrieval is performed with the global and local indices. It turns out that case-based retrieval found the case shown in Appendix A as the most similar and thus selected *swap* as the repair tactic for the focal_activity A_4^8.

To apply swap, CABINS calculates the activity with which A_4^8 will be swapped. To do this, CABINS selects the activity which, if swapped with A_4^8, will result in least amount of precedence constraint violations. From the figure 7 we can see that actually there are 5 activities swappable with A_4^8 within the repair horizon. These activities are: A_4^{10}, A_4^9, A_4^1, A_4^5 and A_4^4. At first glance, it may appear that it would be better if A_4^8 was swapped with A_4^{10} because by doing so A_4^8 will be finished as early as possible. However, it is not the best choice since if A_4^8 is swapped with A_4^{10}, it will cause a lot of downstream ripple effects contrary to the primary intention of keeping repair effects as localized as possible. After calculation of the estimated possible effects, CABINS decides to swap A_4^4 with A_4^8. Job J_4 has weight 3 and weighted tardiness $3 \times (1370 - 1320) = 150$. The effect of applying the swap tactic is that A_4^8 and A_4^4 are unscheduled on R_4 and A_4^8 is re-scheduled to start at time 1090 (the start time of activity A_4^4 prior to the swap). Due to the larger duration of activity A_4^8, now there is the ripple effect of a precedence constraint violation between activity A_4^4 and its successor activity A_5^4 on resource R_2. (In general, many activities could be affected and must be rescheduled as described in section 4.4). Constraint propagation discovers this constraint conflict and shifts activity A_5^4 further to the right on resource

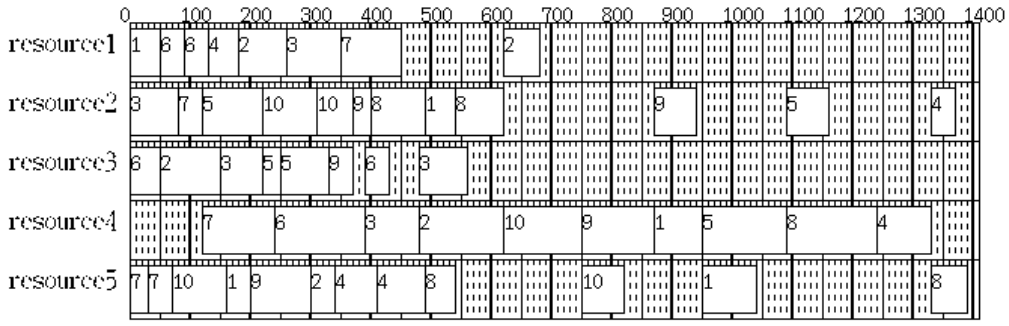

Figure 8: Schedule Results after Repair on A_4^8

R_2 resulting in the repaired schedule shown in figure 8.

Then, the effects of repairing A_4^8 are calculated. CABINS *estimates* the local effects on the focal_job J_8 and calculates global effects on the whole schedule. Machine utilization did not change but J_8 had an estimated decrease in weighted tardiness of 180 time units and an estimated decrease in WIP of 200 units [6]; J_4 had an increase in weighted tardiness of 150 units and an increase in WIP of 750 units. Global weighted tardiness decrease is $180 - 150 = 30$ and global WIP increase is 750. CBR is invoked using the these effects and applied repair tactic as indices to determine whether this repair outcome is acceptable. If there are more success cases than failure cases in the retrieved k-nearest neighbors, it is considered that the effects reflect tradeoffs in the user's preferences (in this example, little weight on WIP) and the outcome is considered acceptable. If, on the other hand, a failure case is retrieved, then the outcome is considered unacceptable, reflecting the user preferences for minimization of weighted tardiness without the expense of increasing WIP.

In this example, CBR invocation with effects as indices retrieves as the closest matching case, the case shown in Appendix B, where the effects match the effects associated with the swap repair tactic. Therefore, the outcome is deemed "acceptable".

[6] These decreases cannot be precisely determined until the last activity of J_8, A_5^8, is repaired

5 Evaluation of the Approach

We conducted a set of experiments to test the following hypotheses:

1. Our approach is potentially effective in capturing user preferences and optimization tradeoffs that are difficult to model.

2. Our approach improves schedule quality irrespective of method of initial schedule generation.

3. Our approach produces high quality schedules at much lower computational cost as compared to simulated annealing, a well-known iterative repair method.

4. Our approach is suitable as a reactive scheduling method because it maintains high schedule quality and minimizes disruptions in the face of execution time failures.

These hypotheses are difficult to test since, due to the subjective and ill-defined nature of user preferences, it is not obvious how to correlate scheduling results with the captured preferences or how to define quality of a schedule whose evaluation is subjective.

To address these issues, we had to devise a method to test the hypotheses in a consistent manner. To do that, it is necessary to know the optimization criterion that would be implicit in the case base, so that the experimental results can be evaluated. In the experiments reported here, we used two different explicit criteria (weighted tardiness; WIP+weighted tardiness) to reflect the user's optimization criteria and built a rule-based reasoner (RBR) that goes through a trial-and-error repair process to optimize a schedule. Since the RBR was constructed not to select the same repair action after application of a selected repair tactic was evaluated as unacceptable, it could go through all the repair actions before giving up further repair. Each of these applications of a repair action would be gathered in the repair history of the case for the particular activity under repair. For each repair, the repair effects were calculated and, on this basis, since the RBR had a predefined evaluation objective, it could evaluate the repair outcome consistently. Thus, we used the RBR with different rules each time to generate different case bases, each for a different explicit optimization objective. Naturally, an objective, though known to the RBR, is not known to CABINS and is only implicitly and

indirectly reflected in an extensional way in each case base. By designing an objective into the RBR so it could be reflected in the corresponding case base, we got an experimental baseline against which to evaluate the schedules generated by CABINS.

We evaluated the approach on a benchmark suite of job shop scheduling problems where parameters, such as number of bottlenecks, range of due dates and activity durations were varied to cover a broad range of parallel machine job shop scheduling problem instances. In particular, the benchmark problems have the following structure: each problem has 10 orders of 5 activities each. Each order has a linear process routing specifying a sequence where each order must visit bottleneck resources after a fixed number of activities, so as to increase resource contention and make the problem tighter. Two parameters were used to cover different scheduling conditions: a range parameter, RG, controlled the distribution of order due dates and release dates, and a bottleneck parameter, BK, controlled the number of bottleneck resources. To ensure that we had not unintentionally hardwired knowledge of the problem into the solution strategies, we used a problem generator function that embodied the overall problem structure described above to generate parallel job shop scheduling instances where the problem parameters were varied in controlled ways. In particular, six groups of 10 problems each were randomly generated by considering three different values of the range parameter (static, moderate, dynamic), and two values of the bottleneck configuration (1 and 2 bottleneck problems). The slack was adjusted as a function of the 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 order were also randomly generated.

Generating problem instances "in the neighborhood" of a problem by controlled variation of problem parameters is a well-accepted method in Operations Research and knowledge-based scheduling communities for evaluating the performance of scheduling methods (e.g., [Sad91, SC93]). The problem instances, although randomly generated, shared features of problem structure (e.g., each problem has 5 machines, of which 1 and 2 machines are bottlenecks, and substitutable machines exist for the non bottleneck machines etc), and CBR can exploit the captured regularities in the structure of the problems for transfer to later problem solving. It is interesting to note that this transfer carries over even if the number of orders is varied (see Table 5).

The benchmark problems are variations of the problems originally reported in [Sad91] and used as a benchmark by a number of researchers (e.g. [Mus93, LS93]). Our problem sets are, however, different in two respects: (a) we allow substitutable resources for non-bottleneck resources, thus solving the parallel machine rather than the simple job shop scheduling problem, and (b) the due dates of orders in our problems are tighter by 20 percent than in the original problems.

A cross-validation method was used to evaluate the capabilities of CABINS. Each problem set in each class was divided in half. The overall training sample, consisting of 30 problems, each of which has 50 activities, was repaired by RBR to gather cases. As has been explained in the section on case acquisition (section 4.2), a case is acquired for each activity that is the current focal_activity (irrespective of the number of tactics available or number of tactics used in the activity's repair). Of course, an activity (and consequently a job) may be repaired more than once during an overall repair cycle, since it is repaired as a focal_activity but also as an activity in the conflict set of another focal_activity, and thus must be moved. Allowing each activity to be a focal_activity once for each problem would give a maximum of 30X50 = 1,500 cases for each training sample (for each different experimental optimization objective). In practice, some of the activities did not become focal_activities to be repaired because they did not have enough upstream slack (see section 4), so that for each training sample, CABINS was trained with approximately 1,100 cases. These cases were then used for case-based repair of the validation problems (the other 30 problems). We repeated the above process by interchanging the training and the test sets. Reported results are for the validation problem sets. Since it is not possible to theoretically predict the bounds of repair or the global optimum, in the experiments, CABINS was allowed to run for three overall repair cycles.

5.1 Preference Acquisition

To test the hypothesis that CABINS could acquire user preferences, we constructed through RBR two case bases, the first to reflect the user's preference for repairs that minimize weighted tardiness and the second to reflect the more complex criterion of minimizing the combination of weighted tardiness and WIP. The cases constituted the only source of knowledge for CABINS. In other words, there was no objective given to CABINS explicitly. The case-

bases were used both as a source of suitable repairs, and also as a source of advice regarding repair evaluation.

Graphs in Fig. 9 show the comparison of the performance by CABINS using "weighted tardiness" case base (labeled in the graphs as CABINS(WT)) and the performance by CABINS using the "weighted tardiness and WIP" case base (labeled in the graphs as CABINS(WT+WIP)). From the results, we observe that CABINS(WT) generated higher quality schedules with respect to minimizing weighted tardiness than CABINS(WT+WIP) in all six problem classes. Conversely, CABINS(WT+WIP) generated higher quality schedules with respect to WIP, and weighted tardiness plus WIP than CABINS(WT) in the all problem classes. In a nutshell, CABINS(WT) tries to optimize a schedule only in terms of weighted tardiness and neglects WIP, but CABINS(WT+WIP) takes into account the tradeoffs between weighted tardiness and WIP in schedule repair. These results indicate that CABINS can acquire different and subjective user preferences on the tradeoffs of diverse objectives in scheduling from the cases. Thus in our approach, unlike traditional heuristic scheduling approaches [Fre82, MP93], it is not necessary to devise a particular heuristic to suit the optimization criterion. Only the case-base must be changed for different optimization objectives. In addition, unlike traditional search-based scheduling approaches such as branch-and-bound, dynamic programming, tabu search, simulated annealing and so on, our approach doesn't require an explicitly represented objective function. CABINS has the potential for inducing more complicated form of user's objectives (e.g. allowing handling of exceptional situations) from the cases. It is true that user's objectives could be elicited by intensely interviewing domain experts and represented in the form of rules as we have done in constructing RBR modules to gather cases in the experiments. But, (1) rule-based knowledge acquisition is extremely laborious [Pre90] and (2) a scheduling problem is so ill-structured that even a domain expert cannot have a sufficient knowledge for making a good schedule efficiently [KLSF91]. Nevertheless, the CBR-based methodology of CABINS can induce efficient control model from the cases obtained through the applications of insufficient rules.

In another set of experiments with objective WIP+WT, we used RBR itself to repair the set of test problems. Table 1 shows that repair by CABINS is about 40% more efficient than repair by RBR and it improves the quality of schedules by about 12% more than repair by RBR. A potential explanation

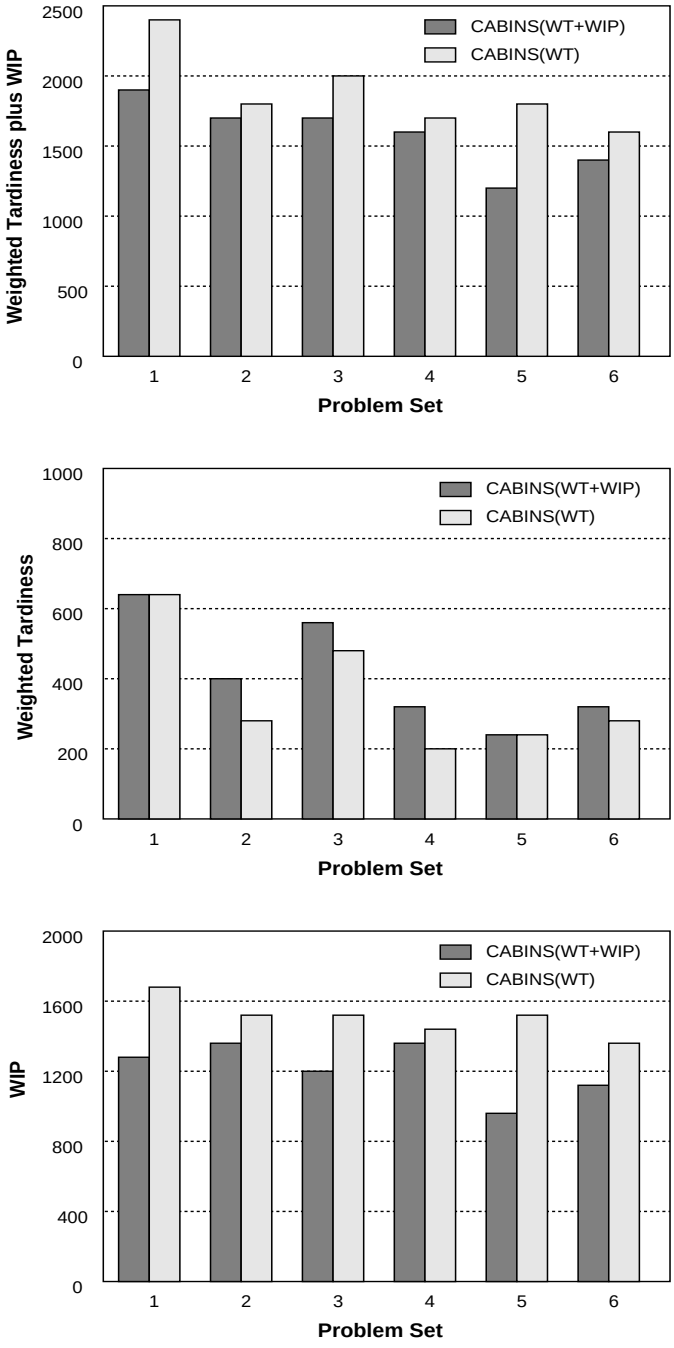

Figure 9: Scheduling Results with Different Case Bases

Table 1: Repair by RBR and CABINS

	WT.	WIP	WT+WIP	CPU Sec.
Repair by RBR	375.2	1446.6	1821.8	498.6
Repair by CABINS	405.3	1195.0	1600.3	296.5

for these results is that, as described in section 4.3, CABINS can effectively utilize failure information stored in the cases (Refer to [MS94] for more details and some experimental results.)

5.2 Predictive and Reactive Scheduling

We evaluated CABINS against other scheduling methods using standard criteria (e.g. [OST88, ZDG90]) for evaluating schedule revision quality. These criteria are also appropriate for planning. These criteria were: (a) Attendance to scheduling objectives: what it the quality of the revision with respect to the desired optimization criteria? (b) Amount of disruption: how many changes to the original schedule are made? (c) Efficiency of revision: how quick is the revision process? In particular, can the revision process be responsive to schedule execution in the sense of allowing execution to proceed as rapidly as possible? Although we subscribe to the view that both schedule generation and schedule repair can be viewed as an iterative repair process, for ease of readability, we have described our experiments in two separate subsections 5.2.1 and 5.2.2.

Schedule quality and efficiency is important in both predictive schedule generation and reactive schedule management. Responsiveness of the schedule revision process is crucial during handling of schedule execution failures (and opportunities) to patch up the schedule quickly and allow execution to proceed. Minimizing schedule disruption is most important during reactive management of a schedule. Once a schedule starts executing, it is important to preserve continuity of domain activity, since there could be substantial cost in having to attend to discontinuities introduced by reactive schedule revision (e.g. set-up costs when resource assignments have been changed). These criteria must be balanced and traded off against each other.

The results show that in predictive schedule generation, the methodology

improves the quality of schedules generated by a variety of scheduling methods and also generates schedules of higher quality along a variety of optimization objectives with lower processing cost as compared to simulated annealing, a well-known iterative optimization method [JAMS89, ZDG90, LAL92]. In recovering from execution time failures, the approach (1) attends to schedule quality both in terms of optimization objectives, and disruption, and (2) is responsive in that it allows continuation of execution without delays in response to execution failures.

5.2.1 Predictive Schedule Repair

In predictive schedule repair, the primary objective in our experiments was to optimize schedule quality at a low computational cost. To investigate our experimental hypotheses, we compared CABINS with Simulated Annealing. Simulated Annealing (SA) is a well known iterative improvement approach to combinatorial optimization problems, which is reported to be able to yield solutions of better quality at the cost of larger computational efforts in a number of combinatorial optimization domains, such as computer-aided design of integrated circuit, image processing and neural network theory ([JAMS91, LAL92]). SA has also been applied to job-shop scheduling domain for the makespan objective and is reported ([LAL92]) to have a potential of finding shorter makespans than the state-of-the-art tailored heuristic, e.g. shifting bottleneck procedure ([ABZ88]).

The details of the our SA implementation are given as follows:

1. Generate an initial schedule
2. Select an activity randomly
3. Unless all the available repair actions have been tried, do the following:
   (a) Select a repair action among the remaining un-tried repair tactics;
   (b) Apply the chosen repair tactic to the activity under repair;
   (c) Evaluate the resulting repaired schedule with respect to the explicit objective (WIP+WT)
   (d) If the resulting schedule is better than the schedule before repair in terms of the objective, then the revision procedure goes on to repair next randomly-chosen activity;

Otherwise the revision procedure goes on to repair next randomly-chosen activity with probability $exp(-\Delta/Temp)$, in which Δ is defined as the difference of schedule evaluations after repair and before repair.

The temperature *Temp* is updated (decreased by a fixed percentage every time) when a fixed number (currently 250) of repair actions have been applied and the revision procedure will be terminated if a pre-set maximum computational effort has been reached. We ran each experiments 5 times and reported the best results among these 5 separate runs (since SA incorporates a probabilistic factor, the results are not necessarily the same across the different experimental runs).

In order to test the generality of the approach, we repeated the same set of experiments 4 times, where each time the initial (seed) schedule was generated using a set of well regarded dispatch heuristics and a constraint-based scheduler (CBS). The dispatch rules selected to generate the initial schedule are widely used in practical job shop scheduling problems, namely the Earliest Due Date (EDD) rule, the Weighted Shortest Processing Time (WSPT) rule and the WSPT with order time urgency factor (R&M) rule. These heuristics have been reported to be particularly good at reducing tardiness under different scheduling conditions [MP93]. We also used the constrained-based scheduler (CBS), which uses ARR variable ordering heuristic and GV value ordering heuristic with pre-determined biased start time utility-functions (see section 2.3).

In our experiments, the user's objective function was assumed to be minimizing weighted linear combination of work-in-process inventory (WIP) plus weighted tardiness. This is a multi-objective function that is difficult to optimize heuristically. WIP and weighted tardiness are not always compatible with each other. There are situations where WIP is reduced, but weighted tardiness increases.

Table 2 presents the average results of all 60 problems in the benchmark. Based on the results, we make a variety of observations. First, CABINS improved the initial schedule across all scheduling methods according to the objectives. It should be noted that these dispatch heuristics have been extensively used in Operations Research experimentation with very good results [Bak74, MRV84]. The initial schedules generated by the dispatch heuristics can be considered as local minima, in the sense that they cannot be

Table 2: Repair by CABINS and SA based on Different Methods of Initial Schedule Generation

	WT.	WIP	WT+WIP	CPU Sec.
Schedule by EDD	956.0	1284.6	2240.6	0.1
Repair by CABINS	349.5	1311.2	1660.7	73.5
Repair by SA	340.5	1333.4	1673.9	388.2
Schedule by WSPT	584.0	1241.0	1825.0	0.1
Repair by CABINS	321.0	1254.9	1575.9	72.1
Repair by SA	328.5	1320.4	1648.9	398.3
Schedule by R&M	556.0	1242.0	1798.0	0.1
Repair by CABINS	305.3	1264.9	1570.2	84.9
Repair by SA	330.1	1290.8	1620.9	450.5
Schedule by CBS	1173.0	1481.0	2654.0	17.4
Repair by CABINS	405.3	1195.0	1600.3	296.5
Repair by SA	395.5	1220.0	1615.5	1380.0

easily improved. For example, these initial schedules are very *tight*, in that there is no on-purpose machine idleness. We conjecture that it would be more difficult to improve an initial schedule with higher quality. For example, it would be more difficult to improve an "R&M"-generated schedule than an "EDD"-generated one. The experimental results support this conjecture (EDD-generated schedule has been improved by 25.9 percent and R&M-generated schedule has been improved by a 12.6 percent). Second, we observe that the better the quality of the initial schedule, the better the quality of the repaired result. Third, CABINS generated schedules of comparable quality but was on the average 4-5 times more efficient than simulated annealing. It seems that the contextual information captured in the CABINS case base and the system's use of failure information in the repair history is effectively used to guide the search and prune unpromising paths thus making CABINS much more efficient than the random search of simulated annealing.

To further investigate CABINS's behavior vis a vis initial schedule generation method, we again used training and test sets of 5 resources and 10

	WT	WIP	WT+WIP	CPU Sec.
Initial Schedule	3875.0	1470.5	5345.5	0.1
Repair by CABINS	1740.0	1432.5	3172.5	81.2
Repair by SA	1723.8	1418.6	3142.4	323.3

Table 3: Repair by CABINS on Randomly Generated Initial Schedules

problems. The initial schedule for each problem is randomly generated from scratch. To do this, we took into account the precedence constraints and resource constraints (disregarding due date constraints) so generation of an executable schedule was guaranteed. As expected, the qualities of these initial schedules are very low (compared to the ones generated by the dispatch heuristics and CBS). From the table 3 we can see that CABINS also performs well on these randomly generated initial schedules. The behavior of CABINS with regard to the method of initial schedule generation confirms intuitions in the Operations Research community (e.g. [MP93]) that the higher the quality of the initial solution, the better the repaired solution. This is also consistent with the behavior of other repair-based methods, for example the behavior of simulated annealing in our experiments, and also the min-conflict heuristic's behavior for constraint satisfaction problems [MJPL92].

Other interesting experimental results we got so far are:

- **Evaluation of revision control model learning**

  We conducted another set of experiments to ascertain the effectiveness of case-based learning of the control model for selecting the repair actions. The results without learning were obtained by random, not case-based, selection and application of the same repair tactics for activity repair. The results showed that repair did not improve schedule quality of approximately 90% of the example problems.

- **Evaluation of scalability**

  To test the scalability of our approach we generated an additional set of 60 problems each with 20 jobs, each of which uses 5 resources. Usually, in real operating environments, the factory configuration (e.g. number

Table 4: Repair by CABINS on 5 resources and 20 orders Problems

	WT	WIP	WT+WIP	CPU Sec.
Schedule by EDD	2106.8	5440.3	7547.1	0.3
Repair by CABINS	648.5	5538.4	6186.9	171.0
Schedule by WSPT	718.4	5310.2	6028.7	0.5
Repair by CABINS	561.2	5332.1	5893.3	190.0
Schedule by R&M	709.5	5218.3	5927.8	0.6
Repair by CABINS	548.6	5237.8	5786.4	164.5
Schedule by CBS	2396.5	6260.7	8657.2	203.0
Repair by CABINS	692.2	6246.0	6938.2	880.0

and type of machines) is likely to remain relatively the same for reasonably long periods. The number of orders, however, is very likely to fluctuate due to varied customer demands and other economic factors. Based on these assumptions, in our experimentation, we focused on varying number of jobs rather than number of resources. The 20-job problems were generated from the same problem generator function by varying the same parameters (as for the set of 10-job problems) in controlled ways. The knowledge acquisition method was the same as for the 10-job problems, i.e. RBR was used to acquire a training case base with 30 problems each of which has 20 jobs and 5 resources. We also used cross validation approach. The pattern of results was the same as for the first set of 60 problems. The results are shown in Table 4.

- **Evaluation of knowledge transferability**

  In order to test generalization issues in case-based learning and transferability of acquired knowledge, (1) we collected the cases through solving the 5 resources and 10 jobs benchmark problems (using RBR). and (2) we used the case-base collected in Step 1 to solve the 5 resources and 20 jobs problems. The results are shown in Table 5. We see that although the results we got based on *5 resource and 20 jobs* case-base are better (reported in table 4), CABINS still performs very well on the bigger problems using the original *5 resource and 10 jobs* case-base. We

also see that the pattern of CABINS behavior, i.e. improving schedule quality independent of initial schedule generation still holds.

	WT.	WIP	WT+WIP	CPU Sec.
Schedule by EDD	2106.8	5440.3	7547.1	0.3
Repair by CABINS	824.5	5429.4	6253.9	234.2
Schedule by WSPT	718.4	5310.2	6028.7	0.5
Repair by CABINS	633.7	5342.1	5975.8	222.0
Schedule by R&M	709.5	5218.3	5927.8	0.6
Repair by CABINS	598.2	5229.9	5828.1	194.5
Schedule by CBS	2396.5	6260.7	8657.2	203.0
Repair by CABINS	924.2	6252.1	7176.3	973.8

Table 5: Repair on 5 X 20 Problems using Case-Base collected from 5 X 10 problems

From the knowledge acquisition and practical point of view, the results are quite encouraging. They show that CABINS has potential for application in operational factory environments, since knowledge transferability will alleviate the knowledge acquisition burden without much affecting overall system performance and quality of scheduling results.

5.2.2 Repair in Response to Unpredictable Execution Events

Reactive schedule repair involves (1) recognition of the conflicts that are introduced in the schedule as a result of an unexpected and uncontrollable change in the execution environment, (2) propagation of the conflicts, and (3) selection and application of a repair action. Before we present and discuss the experimental hypotheses, evaluation criteria and results, we present the reactive repair steps taken by CABINS.

The first step in reactive repair is the recognition of conflicts introduced in the schedule as a result of unexpected events in the execution environment. In general, there are two types of conflicts that can be recognized:

Temporal conflicts: These are conflicts reflecting inconsistencies between the scheduled and actual start and end times of activities

Resource conflicts: These are conflicts reflecting inconsistencies in the resource capacity currently available and the capacity required for processing activities

In the second step, the effects of the introduced conflicts are propagated downstream (forward in time) from the point in time where the unexpected event happened (right-shifting). This involves undoing the reservations that become inconsistent as a result of the unexpected event and propagating their effects to determine the consequences (ripple effects) of the unexpected event for the rest of the schedule. The result of this step is a feasible schedule but typically of much worse quality than the predictive schedule before the occurrence of the deleterious unexpected event.

In the third step, CABINS is used to repair the suboptimal schedule that resulted in the second step. The mechanisms that CABINS uses in reactive repair are exactly the same used for predictive optimization (except, of course, that no attempt is made to repair activities that have being already executed before the unexpected event happened). If the unexpected event is loss of capacity (e.g. a machine breakdown), the activity that was being processed on the resource at the time of breakdown must also be re-scheduled.

We illustrate the repair process by an example. Figure 10 shows a predictive schedule for one of the problems that were used for experimentation with predictive schedule optimization (see section 5.2.1). In particular, it is one of the two-bottleneck problems with static start time for all jobs. In this schedule, the weighted tardiness is 240 units.

After computing the predictive schedule, a machine breakdown is created in the middle of the schedule. The broken machine, M, is the busiest non-bottleneck machine. The breakdown was timed to occur at the first 20% of total execution time so as to increase its deleterious effects on the rest of the schedule. The estimated duration of the breakdown was 10 times the average duration of the activities in the problem. M is assumed available for processing at the end time of the breakdown.

The effects of the breakdown are propagated downstream (forward in time). In particular, the activities that were scheduled on the broken machine M and whose scheduled reservations overlapped with the time interval

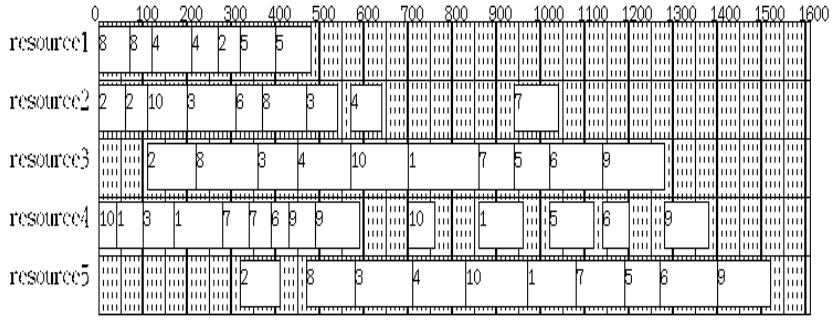

Figure 10: Schedule Result before Machine Breakdown

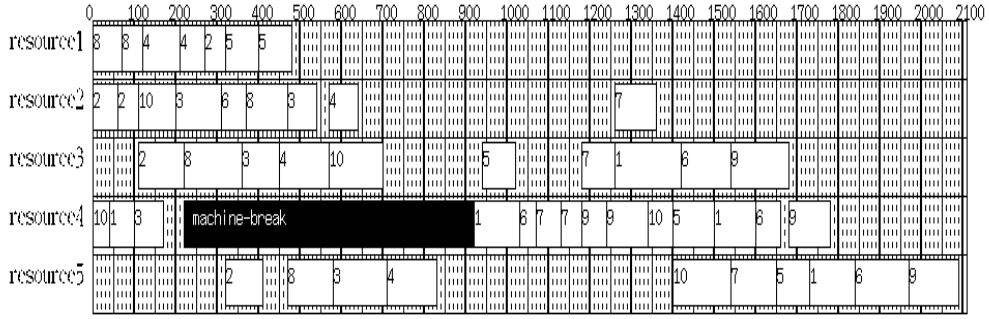

Figure 11: Schedule Result after Machine Breakdown

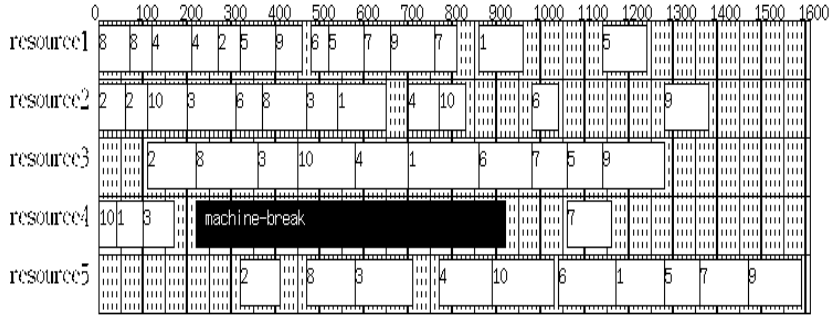

Figure 12: Schedule Result after Reactive Repair

of the breakdown, are unscheduled and re-scheduled in the same sequence on M after the end time of the breakdown (this has been called right-shifting in [OST88]). Right-shifting of these activities on M typically results in constraint conflicts of related activities that are fixed by the constraint propagation mechanisms in CABINS so that a feasible but worse schedule results. Figure 11 shows the schedule resulting from the machine breakdown and its propagated effects. The weighted tardiness of this schedule is 4500, a more than ten-fold worsening of quality. Delaying schedule execution till M is fixed, which is equivalent to right-shifting, is clearly not an option in practice. It is of the utmost importance that the schedule be repaired to enable execution continuity.

CABINS is applied to repair the schedule of figure 11. Because of the big delays that arise as a consequence of capacity loss, we assume optimization of weighted tardiness as the repair objective. Figure 12 shows the schedule resulting after repair by CABINS. Weighted tardiness has been decreased ten-fold (from 4500 to 450).

In general, there are three responses (repair strategies) that a planning/scheduling system can have to the occurrence of unexpected events during execution. First, do not attempt any repair. This strategy results in not taking advantage of any opportunities (e.g., activities finishing earlier than their scheduled end time, additional resources becoming available) or incurring execution delays entailed by deleterious events (e.g., partial or total loss of resource capacity, activities finishing later than their scheduled times). This strategy is obviously suboptimal. A second repair strategy could be to throw away the rest of the plan/schedule and re-plan/re-schedule from the point of the occurrence of the unexpected event. It has been speculated in the literature (e.g., [OST88, ZDG90]) that such strategy may efficiently produce high quality schedules but may increase schedule disruption (though no measure of disruption was given in previous work). A third repair strategy could be incremental revision of the existing schedule. It has been argued in the literature that an incremental repair process that achieves efficient generation of high quality schedules and also allows continuation of execution while minimizing schedule disruption would be the most desirable. To date no experimental evidence has been provided (a) in favor of incremental schedule repair as opposed to re-scheduling, or (b) exhibiting an incremental repair approach that performs well on all the above repair objective simultaneously.

We demonstrate CABINS's reactive capability with respect to execution

time failures, since they are the ones that typically happen and against which a scheduling system must guard. In the set of experiments we performed, CABINS was used to repair a predictive schedule in response to unexpected capacity loss. CBS was used for re-scheduling. Our experimental results demonstrate that the incremental repair methodology of CABINS is superior to re-scheduling in performing reactive schedule repairs in response to execution failures along all the desirable evaluation criteria.

We measured disruption with respect to three criteria:

1. Difference of start times between the repaired schedule and the original predictive schedule (before occurrence of the unexpected capacity loss)

2. Difference in resource assignment of activities in the repaired versus the original schedule

3. Difference in sequencing of activities on a resource in the repaired versus the original schedule

These changes between the repaired and the original schedule qualify as measures of schedule disruption since they could cause changes (with attendant costs) in resource set-up activities, process routing, and expected job finish times. For example, in a manufacturing environment, changes in start times may cause changes in plans for product warehousing, material preparation and product shipment plans; change of resource assignments may change product routings in the factory floor resulting in the need to change programs of material handling equipment (such as automated guided vehicles); changes in activity sequencing on a machine may cause changes in machine set-ups and worker assignments. Such changes cause serious difficulties in the smooth continuation of schedule execution on a factory floor. Obviously, the degree of severity of such changes depends on the nature of the manufacturing process and the factory floor layout. Therefore, a unified measure of disruption of a schedule is hard to formulate.

We compare the performance of reactive repair against CBS re-scheduling in the two different machine breakdown scenarios, each of which has ten sets of problems. In the first experiment, a machine breakdown, whose duration is 10 times the average activity duration, is simulated on the two-bottleneck resource problems, and in the second experiment, a similar machine breakdown is simulated on the set of one-bottleneck problems.

	CABINS reactive-repair	**CBS(GV) re-schedule**
Start Time Disruption	6380	8980
Routing Disruption	9	11
Sequencing Disruption	21	27
Repaired Wei. Tar. (%)	98.8	91.6
CPU time (second)	172.9	6.7

Table 6: Reactive repair vs Re-scheduling

Table 6 shows the average results across all experiments.

The results show that in terms of disruption and quality CABINS outperformed re-scheduling by CBS. However, CABINS's efficiency is much worse than CBS. In some problems, such as problem 1 for example, CABINS spends as much as 40 times more time as CBS.

However, upon further examination, this result is misleading. The reason is the rapid and monotonic repair behavior of CABINS. As shown in figure 13, for example, CABINS achieved better result quality than CBS at the time point when CBS finished re-scheduling. From figure 13 we see that after 9.3 seconds, CABINS has achieved a weighted tardiness of 1430 units compared to 1560 units achieved by re-scheduling in the same time period (9.3 secs). Since, in contrast to the re-scheduling method which does not provide incremental schedule feasibility, CABINS' incremental reactive repair results in a feasible (executable) schedule after every repair iteration, if the repair process is stopped after 9.3 secs, the schedule produced by CABINS can be executed and is of higher quality than the one produced by CBS re-scheduling during the same time period. This behavior was consistency exhibited in all experiments.

System responsiveness in reactive contexts is of great concern. To see whether the results of 13 are robust across different breakdown scenarios, we repeated the experiments with four different variations of duration of machine breakdown. In each experiment, the breakdown duration was 4, 6, 8, and 10 times the average activity duration. Figure 14 shows those results. The graph shows that reactive repair is very efficient at first and then saturates until no further improvement is possible. This characteristic of CABINS' repair process is very suitable for reactive repair since it allows continuation

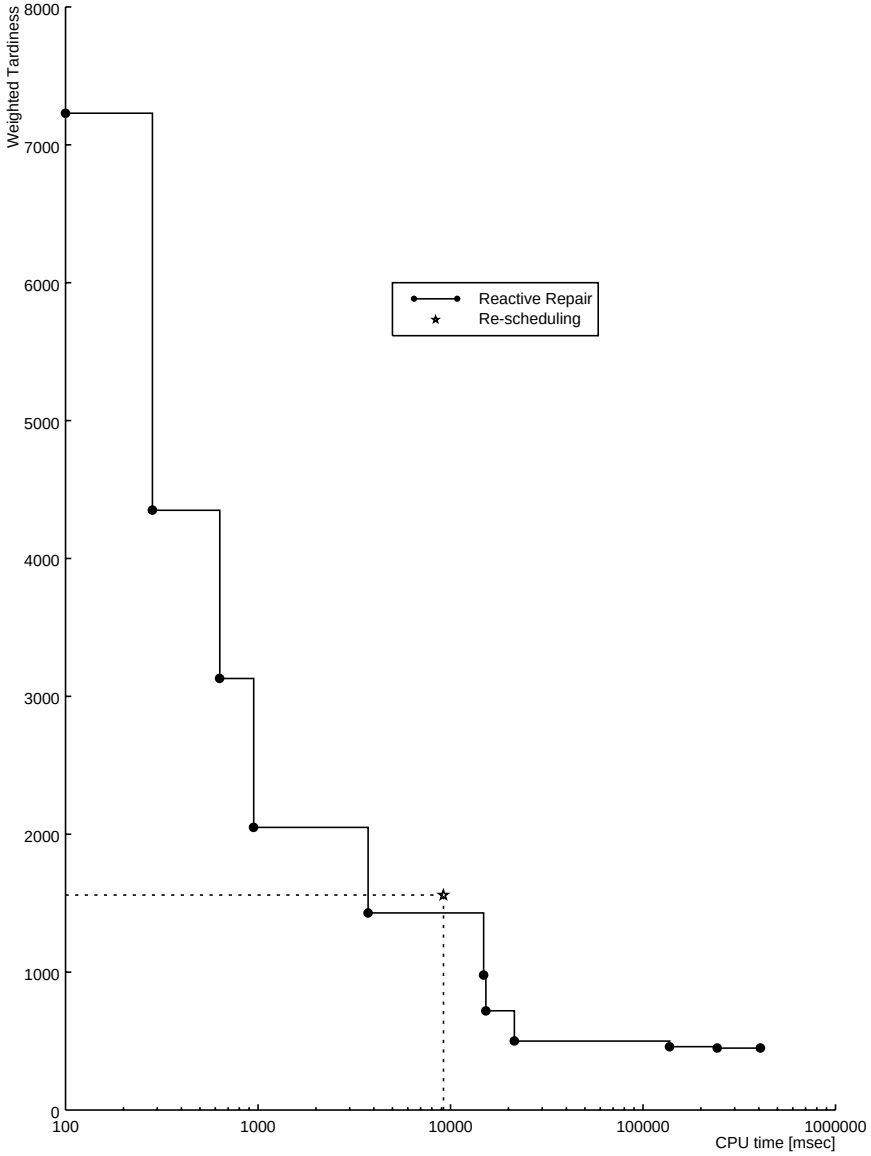

Figure 13: Repair Responsiveness of CABINS in Problem 1

of execution with minimal delay (most of the schedule quality loss is repaired very rapidly).

5.3 How Many Cases Are "Enough"?

The graphs in figure 15 compare the performance of CABINS with different sized case-bases. The results were obtained based on CABINS with WT+WIP type of case-bases. A case of approximately 4,500 cases was generated by RBR. This was done by allowing 3 overall repair cycles for a training set of 30 problems each of which has 50 activities. To get the case bases of different sizes, an appropriate number of cases for each situation was randomly selected and deleted from the approximately 4,500 size case base. This method of generating a new case base by random deletion of cases from a bigger case base is similar to the *ablation* study performed in [Bar89]. The initial schedule generation method was CBS. 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 performance 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 (figure 15) 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 cases) capture well characteristics of the problem space, and additional 1000 new cases may give much redundant information. When the size of 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

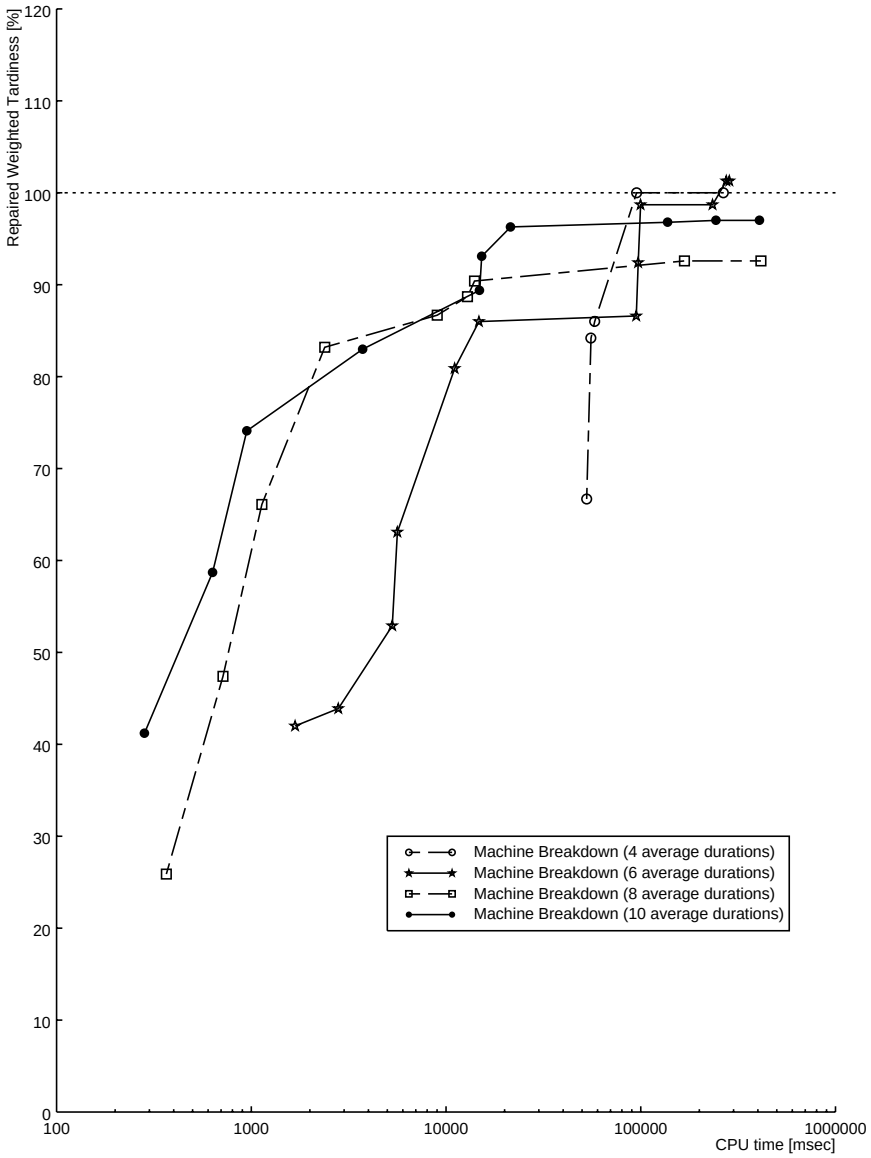

Figure 14: Repair Ratio

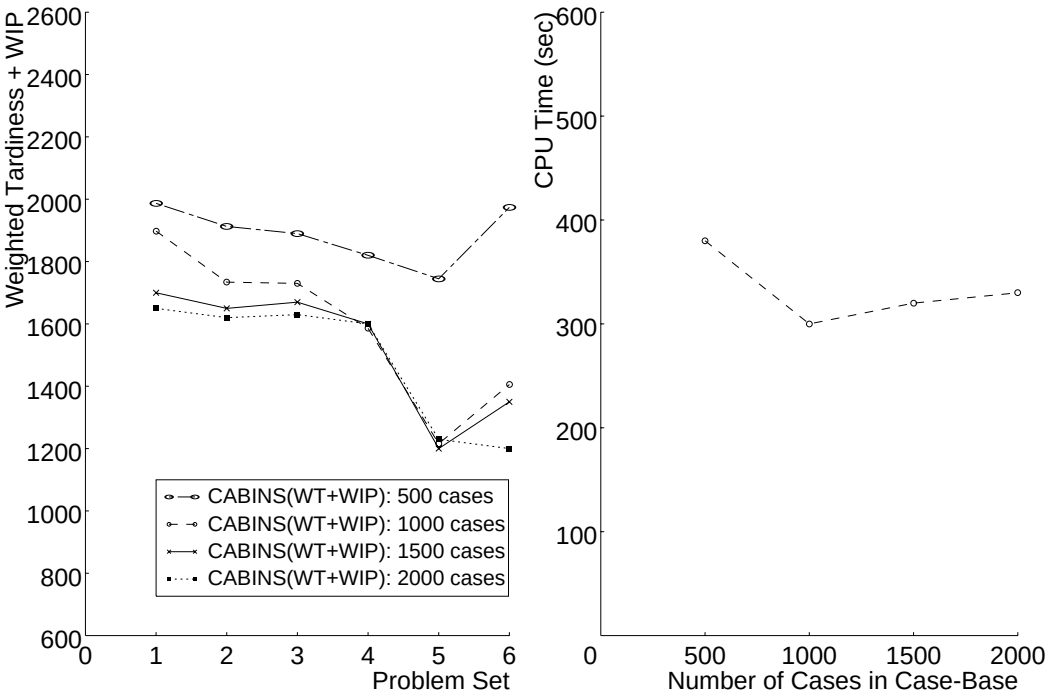

Figure 15: **Effect of case-base sizes in quality and efficiency**

the case-base with 1000 cases might be the optimal choice. 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, in terms of schedule quality improvement, case bases with more than 1000 cases do not seem to provide payoff proportional to the case base size increase.

5.4 Discussion

The experimental results show that the CBR-based repair method not only has the potential to capture different user optimization preferences but also performs well in terms of producing schedules of high quality as compared with other constructive scheduling methods. As compared with simulated annealing, another repair-based method, CBR-based repair produces schedules of comparable quality with substantial computational savings. In addition, CBR-based repair exhibits desirable anytime characteristics and outperforms re-scheduling by a constructive constraint-based method in terms of minimizing disruption and maintaining high schedule quality.

In this section, we will attempt to answer the question "what makes the approach powerful"? We believe the power of the approach stems from the following four reasons. First, as has been pointed out by others (e.g. [MJPL92]), revision-based approaches by making available a complete assignment (a complete schedule for our domain) provide more information that can guide search as compared with constructive methods where only a partial assignment is available. Our CBR-based revision method captures such relevant information in global case features and exploits it as contextual information during case retrieval. Second, although job shop schedule optimization belongs to the category of "hard" NP-complete problems, the case features were able to capture some important domain regularities, such as repair flexibility. This was complemented by keeping information about failed applications of revisions in the repair case history and also keeping failed cases in the case memory. These failures were exploited by CBR to prune unpromising paths in the search space in future similar situations. Third, experimental results and discussion presented in section 5.3 support the hypothesis that the cases CABINS acquired and used in the reported experiments seem to cover the solution space in a fairly evenly distributed fashion, thus allowing CBR-based repair to take advantage of this coverage.

Since, however, we cannot conjecture whether good quality solutions are evenly distributed in the search space as a whole, backtrack search, for example, could be potentially disadvantaged if good solutions are "bunched up" in particular parts of the search space [MJPL92, Lan92], whereas dispatch heuristics are too myopic to take advantage of promising search paths.

Finally, we believe that some of the regularities in the structure of the experimental problems were captured in cases during the training phase and this information was transferable to solve the test problems. Moreover, this information seems to transfer also across problem size as the the results in Table 5 indicate. The table shows that the cases acquired during training with a set of 10-job problems were effective in solving test problems with 20 jobs. The question then arises to what extent the information captured in cases from one set of problems can transfer to job shop optimization problems with different problem structure. This question, albeit of great theoretical and practical importance, is very difficult to answer in a theoretical way. In contrast to other NP-complete problems (e.g. graph-coloring, satisfiability, traveling salesman) for which insightful analysis has been performed (e.g. [MR92, CKT91]) as to their structure and properties that characterize "easy" or "hard" problem instances, similar characterization of job shop schedule optimization problems is currently an open research problem (e.g.[CKT91, Bak74]). Due to the tight constraint inter-dependencies in job shop optimization, it is not known what constitutes "problem structure", i.e. what features of a problem make it difficult or easy to solve, or make one problem substantially similar or different from another. It is for this reason that, except for some simple optimization objectives, such as minimize flowtime for *one-machine* problems where it has been proven that the WSPT heuristic finds the optimal solution, it is currently impossible to theoretically prove schedule optimality for a particular technique. It is only after some proposed problem has defied solution by extensive experimentation by many researchers that it is understood *ipso facto* to be difficult [ABZ88, Bak74]. Most importantly, even if there were good approaches to characterize problem structure in job shop optimization, with explicit optimization criteria, this would not help with our analysis since CABINS does not have an explicit objective function, but instead aims at capturing implicitly context-dependent user preferences.

6 Related Work

Our work shares the same motivations and goals with the work in [MBS88] where the motivations for interactive user manipulation of schedules is presented. In that work, the system monitors the user's manipulation of a schedule, requesting the reasons for each revision that is made. This information is then used to augment/refine the system's knowledge. The approach seems promising but has not been experimentally tested.

Our approach is rooted on concepts and mechanisms of a long line of research in constraint-directed scheduling [Fox83, SOL+86, Sad91]. In that work, schedules are generated by incrementally constructing and merging partial schedules. That work has extensively investigated various properties and aspects of this scheduling methodology and has proposed sophisticated procedures and techniques for constraint-directed scheduling. Although this research tradition has come to view scheduling as an opportunistic repair process, it has operated under static design assumptions (e.g. deterministic application of variable and value ordering heuristics in [Sad91], or statically determined control level model for application of repair actions [OST88]). Our approach advances the state of the art by learning to dynamically adapt the focusing mechanism of the search procedure and by adapting the repair model according to current problem solving circumstances and user preferences and tradeoffs.

Our approach, generates schedules by repair based scheduling in the space of complete schedules. In this respect it is similar to [ZDG90, ZDDD93, MJPL90, BC91]. In [ZDG90, ZDDD93] simulated annealing has been used to perform iterative repair. Knowledge in the form of constraint types and evaluation criteria has been added to the basic simulated annealing framework and has been shown to improve convergence speed [ZDDD93]. [ZDDD92] has studied the tradeoff of minimizing perturbations vs. speed of convergence to a conflict free schedule and vs. schedule quality measured in terms of number of violated resource constraints. In [MJPL90, MJPL92] the min-conflict heuristic, a repair heuristic that chooses the repair that minimizes the number of conflicts that result from a one-step lookahead has been investigated and its performance analyzed. Though the heuristic has been shown to be powerful for solving the N-queens problem, it has been shown inadequate for some types of job shop scheduling constraint satisfaction problems [Mus93] when the initial assignment is random. This is because min-conflicts relies

on a good initial assignment [MJPL92]. The CBR-based repair of CABINS, on the other hand has been shown experimentally to improve schedule quality irrespective of initial schedule generation method, although the percent improvement and the quality of the final repaired schedule varies. In [BC91] schedule modifications are procedurally encoded. Small snapshots of the scheduling process, called chronologies, are used to focus the search by using information gained incrementally during the scheduling process to locate, classify and resolve bottlenecks. In [ZDB+92] plausible explanation based learning (PEBL) has been applied to learn search control rules to increase search efficiency in scheduling tasks for NASA Space Shuttle payload and ground processing. PEBL enables a system to generalize a given target concept (e.g. chronic resource contention) over a distribution of examples. The cost function is to minimize the number of remaining conflicts in the schedule. Unlike all the above systems, CABINS doesn't have any explicit objectives to optimize, but applies case-based learning techniques to acquire user optimization preferences from the records of user's repair decisions and optimizes schedules based on the acquired objectives.

The repair-based scheduling methods considered here are related to the repair-based methods that have been previously used in case-based planning systems (e.g. [Vel92, KH92, Ham89]). Previous case-based systems for incremental solution revision have been motivated primarily by concerns of computational efficiency, preserving plan correctness rather than improving plan quality, and have assumed the existence of a strong domain model to get information as to plan correctness. For example, CHEF [Ham89] assumes the existence of a model-based simulator to evaluate a derived plan and detect a plan failure and uses well-studied domain rules for selecting repairs. Research by [KH92, Vel92] are based on the hypothesis that the plan built by their planner is causally and teleologically correct, and use CBR to find the satisfying plan efficiently.

CABINS as a knowledge acquisition system is also related to previous case-based knowledge acquisition systems (e.g. Protos [Bar89]). These approaches usually require causal explanations from an expert teacher to acquire domain knowledge. In CBR-based schedule repair embodied in CABINS, neither the user nor the program are assumed to possess causal domain knowledge. The user cannot give a solid explanation as to her/his selection of repair action, because s/he cannot predict the effects of the selected action on the plan caused by tight interactions. The user's expertise lies in the

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.

7 Conclusions and Future Work

In this paper, we advocate a framework for knowledge acquisition and iterative repair 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, and (b) on capturing and re-using user scheduling preferences and judgments. The capability of acquiring user optimization preferences is important in domains without strong domain models because usually explicitly expressed objectives are unavailable. Even if they were available, new optimization heuristics would need to be developed, evaluated and implemented complicating the design and maintenance of the system. CABINS provides a framework for alleviating these problems. Our experimental results show the potential of the approach to capture and effectively utilize user scheduling preferences that were not present in the scheduling model. The results indicate that different scheduling objectives implicitly reflected in the case base differentially bias the schedule repair procedure. Further experimental results show that for well defined objectives reflected in the case base, CABINS produces schedules with higher quality as compared with other repair-based scheduling methods, such as simulated annealing. In addition, CABINS is robust in the sense that it always improved the quality of a schedule regardless of which method was used for generating the seed schedule. It seems that the effort expended to capture a large number of cases can be amortized by future repeated use of the case base to get high quality schedules efficiently. More importantly, CABINS can acquire the cases through user interaction during the process of solution improvement without imposing undue overhead on the user. We believe that CABINS has the potential for accommodating acquisition of user preferences that change over time. Future work will investigate this issue and issues of automating hierarchical

abstraction of the repair process, and dealing with more complex objectives and larger problems.

8 Acknowledgments

This research was partially supported by the Defense Advance Research Projects Agency under contract #F30602-91-C-0016. This work was performed when the first author was a visiting scientist at the Robotics Institute of Carnegie Mellon University under the support of Matsushita Electric Industrial Co. We would like to thank Mr. Dajun Zeng for help with the experimentation and insightful discussions.

APPENDIX A: Case Instance

```
T_case {
  name = "exp_0_0_8:order5-1:1:activity5_2";
  slots = (
    Slot {
      feature = weighted_tardiness;
      value = 470.000000;
      salience = 1.000000;
    };
    Slot {
      feature = resource_utilization_average;
      value = 0.789000;
      salience = 1.000000;
    };
    Slot {
      feature = resource_utilization_deviation;
      value = 0.0403749;
```

```
    salience = 1.000000;
  };
);
SI_slots = (
  Slot {
    feature = waiting_time;
    value = 580.000000;
    salience = 0.333;
  };
  Slot {
    feature = predictive_shift_gain;
    value = 0.806000;
    salience = 0.667;
  };
  Slot {
    feature = predictive_alt_shift_wip_gain;
    value = 0.106000;
    salience = 0.333;
  };
  Slot {
    feature = predictive_swap_gain;
    value = 0.903000;
    salience = 0.667;
  };
  Slot {
    feature = predictive_alt_swap_gain;
    value = 0;
    salience = 0.333;
  };
);

solutions = (
  Solution {
    tactics_type = SWAP;
    effects = (
      Effect {
        effect_type = WEIGHTED_TARDINESS;
```

```
  salience = 0.667;
  domain = "whole_schedule";
  previous_value = 470.000000;
  current_value  = 380.000000;
  gain = 90.000000;
};

Effect {
  effect_type = RESOURCE_UTILIZATION_AVERAGE;
  salience = 0.667;
  domain = "whole_schedule";
  previous_value = 0.789000;
  current_value  = 0.789000;
  gain = 0;
};

Effect {
  effect_type =  RESOURCE_UTILIZATION_DEVIATION;
  salience = 0.667;
  domain = "whole_schedule";
  previous_value = 0.0403749;
  current_value  = 0.0403749;
  gain = 0;
};

Effect {
  effect_type  = INPROCESS_INVENTORY;
  salience = 0.667;
  domain = "whole_schedule";
  previous_value = 2240.000000
  current_value  = 2250.000000;
  gain = -10.000000;
};

Effect {
  effect_type  = INPROCESS_INVENTORY;
  salience = 0.333;
```

```
          domain = "job7";
          previous_value = 290.000000;
          current_value  = 310.000000;
          gain = -20.000000;
        };
        ..........
      );
      result = ACCEPTABLE;
    };
  );
}
```

APPENDIX B: Case Used for Evaluation

```
T_case {
  name = "exp_2_0_4:order7-1:1:activity7_2";
  slots = (
    Slot {
      feature = weighted_tardiness;
      value = 670.000000;
      salience = 1.000000;
    };
    Slot {
      feature = resource_utilization_average;
      value = 0.682000;
      salience = 1.000000;
    };
    Slot {
      feature = resource_utilization_deviation;
      value = 0.074000;
```

```
    salience = 1.000000;
  };
);
SI_slots = (
  Slot {
    feature = waiting_time;
    value = 280.000000;
    salience = 0.333;
  };
  Slot {
    feature = predictive_shift_gain;
    value = 0.406000;
    salience = 0.667;
  };
  Slot {
    feature = predictive_alt_shift_wip_gain;
    value = 0.306000;
    salience = 0.333;
  };
  Slot {
    feature = predictive_swap_gain;
    value = 0.103000;
    salience = 0.667;
  };
  Slot {
    feature = predictive_alt_swap_gain;
    value = 0.304000;
    salience = 0.333;
  };
);

solutions = (
  Solution {
    tactics_type = LEFT_SHIFT;
    effects = (
      Effect {
        effect_type = WEIGHTED_TARDINESS;
```

```
      salience = 0.667;
      domain = "whole_schedule";
      previous_value = 670.000000;
      current_value  = 930.000000;
      gain = -260.000000;
    };

    Effect {
      effect_type = RESOURCE_UTILIZATION_AVERAGE;
      salience = 0.667;
      domain = "whole_schedule";
      previous_value = 0.682000;
      current_value  = 0.682000;
      gain = 0;
    };

    Effect {
      effect_type =  RESOURCE_UTILIZATION_DEVIATION;
      salience = 0.667;
      domain = "whole_schedule";
      previous_value = 0.074000;
      current_value  = 0.074000;
      gain = 0;
    };

    Effect {
      effect_type  = INPROCESS_INVENTORY;
      salience = 0.667;
      domain = "whole_schedule";
      previous_value = 1940.000000
      current_value  = 1990.000000;
      gain = - 50.000000;
    };
    ..........
  );
  result = UNACCEPTABLE;
};
```

```
Solution {
  tactics_type = SWAP;
  effects = (
    Effect {
      effect_type = WEIGHTED_TARDINESS;
      salience = 0.667;
      domain = "whole_schedule";
      previous_value = 670.000000;
      current_value  = 600.000000;
      gain = 70.000000;
    };

    Effect {
      effect_type = RESOURCE_UTILIZATION_AVERAGE;
      salience = 0.667;
      domain = "whole_schedule";
      previous_value = 0.682000;
      current_value  = 0.682000;
      gain = 0;
    };

    Effect {
      effect_type =  RESOURCE_UTILIZATION_DEVIATION;
      salience = 0.667;
      domain = "whole_schedule";
      previous_value = 0.074000;
      current_value  = 0.074000;
      gain = 0;
    };

    Effect {
      effect_type  = INPROCESS_INVENTORY;
      salience = 0.667;
      domain = "whole_schedule";
      previous_value = 2480.000000;
      current_value  = 3180.000000;
```

```
        gain = -700.000000;
      };

      Effect {
        effect_type  = WEIGHTED_TARDINESS;
        salience = 0.333;
        domain = "job7";
        previous_value = 790;
        current_value  = 210;
        gain = 580;
      };

      Effect {
        effect_type  = INPROCESS_INVENTORY;
        salience = 0.333;
        domain = "job9";
        previous_value = 290;
        current_value  = 410;
        gain = -120;
      };
      ....
    );
    result = ACCEPTABLE;
  };
);
}
```

References

[ABZ88] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

[AKA91] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[Ash87] K.D. Ashley. *Modeling Legal Argument: Reasoning with Cases and Hypotheticals.* PhD thesis, University of Massachusetts, Amherst, 1987.

[Bak74] K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974. Course Textbook.

[Bar89] Ray Bareiss. *Exemplar-Based Knowledge Acquisition : A Unified Approach to Concept Representation, Classification, and Learning.* Academic Press, New York, NY, 1989.

[BC91] E. Biefeld and L. Cooper. Bottleneck identification using process chronologies. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 218–224, Sydney, Australia, 1991.

[Cha93] Alok R. Chaturvedi. Acquiring implicit knowledge in a complex domain. *Expert Systems With Applications*, 6(1):23–35, 1993.

[CKT91] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, 1991.

[CMSW92] Robert H. Creecy, Brij M. Masand, Stephen J. Smith, and David L. Waltz. A traiding MIPS and memory for knowledge engineering. *Communications of ACM*, 35(8):48–64, 1992.

[Das90] Belur V. Dasarathy, editor. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamos, CA, 1990.

[DB88] T. Dean and M. Boddy. An analysis of time dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, Saint Paul, Minnesota, 1988. AAAI.

[Fox83] Mark Fox. *Constraint-Directed Search: A Case Study in Jop Shop Scheduling.* PhD thesis, Department of Computer Science, Carnegie Mellon University, 1983.

[Fre82] Simon French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood, London, 1982.

[Ham89] Kristian J. Hammond. *Case-Based Planning : Viewing Planning as a Memory Task*. Academic Press, New York, NY, 1989.

[JAMS89] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part I (graph partioning). *Operations Research*, 37(6):865–892, 1989.

[JAMS91] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part Ii (graph coloring and number partitioning). *Operations Research*, 39(3):378–406, 1991.

[Joh90] Mark D. Johnston. SPIKE: AI sheduling for NASA's Hubble Space Telescope. In *Proceedings of the Sixth Conference on Artificial Intelligence for Applications*, pages 184–190, Santa Barbara, CA, 1990. IEEE CS.

[KH92] Subbarao Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3):193–258, 1992.

[KLSF91] K. Kempf, C. LePape, S. F. Smith, and B. R. Fox. Issues in the design of AI-based schedulers: workshop report. *AI Magazine*, 11(5):37–46, 1991.

[Kot88] P. Koton. Reasoning about evidence in causal explanations. In *Proceedings of the 1988 Case-Based Reasoning Workshop*, pages 260–270, Clearwater, Fla., 1988.

[KS90] John J. Kanet and V. Sridharan. The electronic leitstand: a new tool for shop scheduling. *Manufacturing Review*, 3(3):161–169, 1990.

[KSS85] J. Kolodner, R. Simpson, and K. Sycara. A process of case-based reasoning in problem solving. In *Proceedings of the Ninth*

International Joint Conference on Aritificial Intelligence, pages 284–290, Los Angeles, CA, 1985. IJCAI.

[KY89] Naiping Keng and David Y. Y. Yun. A planning/scheduling methodology for the constrained resource problem. In *Proceedings of the Eleventh International Joint Conference on Aritificial Intelligence*, pages 998–1003, Detroit, MI, 1989. IJCAI.

[LAL92] Peter J. M. Van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

[Lan92] P. Langley. Systematic and non-systematic search strategies. In *Proceedings of AAAI-92*, San Jose, CA, 1992. AAAI.

[LMB91] 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.

[LS93] J. Liu and K. Sycara. Distributed constraint satisfaction through constraint partition and coordinated reaction. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, Hidden Valley, PA., 1993.

[MBS88] 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.

[MJPL90] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings, Eighth National Conference on Artificial Intelligence*, pages 17–24, Boston, MA., 1990. AAAI.

[MJPL92] S. Minton, M Johnston, A. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

[MP93] Thomas E. Morton and David W. Pentico. *Heuristic Scheduling Systems: With Application to Production Systems and Product Management.* John Wiley and Sons Inc., New York, N.Y., 1993.

[MR92] Ron Musick and Stuart Russell. How long will it take? In *Proceedings of AAAI-92*, pages 466–471, San Jose, CA, 1992. AAAI.

[MRV84] T. E. Morton, R. M. Rachamadugu, and A. Vepsaleinen. Accurate myopic heurisrics for tardiness scheduling. Technical Report 36083-84, Graduate School of Industrial Administration, Carnegie Mellon University, 1984.

[MS94] Kazuo Miyashita and Katia Sycara. Learning control knowledge through cases in schedule optimization problems. In *Proceedings of the Tenth IEEE Conference on Aritificial Intelligence for Application*, pages 33–39, San Antonio, TX, 1994. IEEE.

[Mus93] N. Muscettola. Scheduling by iterative partition of bottleneck conflicts. In *Proceedings of the Ninth Conference on AI Applications*, pages 49–55, Orlando, Fla., March 1993. IEEE.

[OST88] P. S. Ow, S. F. Smith, and A. Thiriez. Reactive plan revision. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 77–82, St-Paul, Minnesota, 1988. AAAI.

[Pre90] D. S. Prerau. *Developing and Managing Expert Systems: Proven Techniques for Business and Industry.* Addison-Wesley, Reading, MA, 1990.

[RA88] E.L. Rissland and K.D. Ashley. Credit assignment and the problem of competing factors in case-based reasoning. In *Proceedings of the Case-Based Reasoning Workshop*, pages 327–344, Clearwater, Fla., 1988.

[RK92] David Ruby and Dennis Kibler. Learning episodes for optimization. In *Machine Learning : proceedings of the Ninth International Workshop (ML92)*, pages 379–384, 1992.

[Sad91] Norman Sadeh. *Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[SC93] Smith S.F. and Cheng C.C. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of The Eleventh National Conference of Artificial Intelligence*, Washington, D.C., 1993. AAAI.

[SF90] Norman Sadeh and Mark S. Fox. Variable and value ordering heuristics for activity-based job-shop scheduling. In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, pages 134–144, Hilton Head Island, SC, 1990.

[SGK+91] K. Sycara, R. Guttal, J. Koning, S. Narasimhan, and D. Navinchandra. Cadet: A case-based synthesis tool for engineering design. *International Journal of Expert Systems*, 4(2), 1991.

[Sim85] R.L. Simpson. *A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation.* PhD thesis, School of Information and Computer Science Georgia Institute of Technology, Atlanta, GA, 1985.

[SM91] E. Simoudis and J.S. Miller. The application of CBR to help desk applications. In *Proceedings: Case-Based Reasoning Workshop*, pages 25–36, 1991.

[SOL+86] S. F. Smith, P. S. Ow, C. LePape, B. McLaren, and N. Muscettola. Integrating multiple scheduling perspectives to generate detailed production plans. In *Proceedings SME Conference on AI in Manufacturing*, Long Beach, CA, September 1986.

[SW86] Craig Stanfill and David Waltz. Toward memory-based reasoning. *Communications of ACM*, 29(12):1213–1228, 1986.

[Syc88] K. Sycara. Patching up old plans. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Montreal, Canada, 1988.

[Syc89] K. Sycara. Argumentation: Planning other agents' plans. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, Mich, 1989.

[Vel92] Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1992.

[ZDB+92] M. Zweben, E. Davis, D. Brian, E. Drascher, M. Deale, and M. Eskey. Learning to improve constraint-based scheduling. *Artificial Intelligence*, 58(1-3):271–296, 1992.

[ZDDD92] M. Zweben, E. Davis, B. Daun, and M. Deale. Rescheduling with iterative repair. In *Proceedings of AAAI-92 workshop on Production Planning, Scheduling and control*, San Jose, CA., 1992. AAAI.

[ZDDD93] M. Zweben, E. Davis, B. Daun, and M. Deale. Iterative repair for scheduling and rescheduling. *IEEE Transactions on System, Man and Cybernetics*, 23(6):1588–1596, 1993.

[ZDG90] M. Zweben, M. Deale, and M. Gargan. Anytime rescheduling. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 251–259, San Diego, CA., 1990. DARPA.