A Case-Based Approach to Improve Quality and Efficiency in Ill-Structured Optimization: An Application to Job Shop Scheduling

（悪構造な最適化における品質と効率の改善に対する事例に基づくアプローチ：ジョブショップスケジューリングへの適用）

Kazuo Miyashita

November, 1994

Submitted in partial fullfilment of the requirements
for the degree of Doctor of Philosophy in Engineering

Department of Electronic Engineering
Faculty of Engineering
Osaka University

Abstract

"Good judgment comes from experience, experience comes from bad judgment."

The author has developed a generic framework of iterative revision integrated with knowledge acquisition and learning for optimization in ill-structured domains, and implemented it in the CABINS system. The ill-structuredness of both the problem space and the desired objectives make many optimization problems difficult to formalize and costly to solve. In such domains, neither the system nor the human expert possess causal domain knowledge that can be used to guide solution optimization. Current optimization technology requires explicit formulation of a single global optimization criterion to control heuristic search for the optimal solution. Often, however, a user's optimization criteria are subjective, situation dependent, and cannot be expressed in terms of a single global optimization function. And a domain expert is not supposed to possess heuristics that enable the efficient search of high quality solutions. In CABINS, the control knowledge for meeting context-dependent user's preferences and achieving situation-sensitive efficient search that guides solution revision is captured in cases along with contextual information. During iterative revision of a solution, cases are exploited for multiple purposes, such as revision action selection, revision result evaluation and 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. The approach was tested in the domain of job shop scheduling. Extensive experimentation on a benchmark suite of job shop scheduling problems has shown that CABINS (1) is capable of acquiring user optimization preferences and tradeoffs, (2) can learn to improve the efficiency of rather intractable iterative repair process, and (3) can improve its own competence through knowledge accumulation.

Acknowledgments

I want to express special gratitude to Prof. Riichiro Mizoguchi at the Institute of Scientific and Industrial Research of Osaka University for his generosity of providing me with the ideal environment necessary for completing this research. I was inspired a lot by his energetic and serious attitude towards his world-leading research activities, which encouraged me in pursuing my studies and writing papers. And I am also grateful to my thesis committee members at the Faculty of Engineering of Osaka University, Prof. Kodama of the Department of Electronic Engineering and Profs. Komoda and Nishio of the Department of Information Systems Engineering, for reviewing this thesis and giving valuable comments, which helped me extensively to improve quality of the thesis.

This work was initiated when I was a visiting scientist at the Robotics Institute of Carnegie Mellon University under the support of Matsushita Electric Industrial Co. I could have not finished this research without help from Katia Sycara, my adviser at CMU. Not only her advice was powerful driving force towards the better research direction, but also her patience with my slow advancement and her constant interest in my work encouraged me to keep on working hard. And I would like to thank the members of the CORTES project at Carnegie Mellon University, especially Norman Sadeh for his help with constraint-based scheduling techniques and Dajun Zeng for the experimentation and insightful discussions. I am also grateful for the friendship given by my officemates at CMU, Ali Safavi and Jyshane Liu. Having all of them as my dear friends is the most precious thing I could acquire during my two years stay at CMU. In addition, I am indebted to Oren Etzioni at the University of Washington for distributing his hypothesis testing software and making possible the experiment analysis in this thesis.

Last but most important, I would record here my warmest acknowledgments to Yuka, my wife, for her understanding and support towards my research. Especially I feel indebted for the holidays and weekends she had to spend by herself while I was working on the research. As a token of my respect and thankfulness (Sorry Yuka, but as a Japanese man I naturally hesitate to use the word "love"), I would like to dedicate this thesis to her.

Publications

1. **Journals**

   (a) "A Framework for Case-Based Revision for Schedule Generation and Reactive Schedule Management"
   Journal of Japanese Society for Artificial Intelligence, Vol.**9**, No.3, pp.426-435, 1994, by Miyashita,K. and Sycara,K.

   (b) "Exploiting Failure Information for Case-Based Schedule Repair"
   Journal of Japanese Society for Artificial Intelligence, Vol.**9**, No.4, pp.559-568, 1994, by Miyashita,K. and Sycara,K.

   (c) "Case-Based Scheduling Knowledge Acquisition" (in Japanese)
   Transactions of The Institute of Systems, Control and Information Engineers, Vol.**7**, No.9, pp.353-360, 1994, by Miyashita,K.

   (d) "Case-Based Knowledge Acquisition for Schedule Optimization"
   Artificial Intelligence in Engineering, accepted, by Miyashita,K.

   (e) "Capturing Scheduling Knowledge from Repair Experiences"
   International Journal of Human-Computer Studies, accepted, by Miyashita,K. and Sycara,K. and Mizoguchi,R.

   (f) "CABINS : A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair"
   Artificial Intelligence, accepted, by Miyashita,K. and Sycara,K.

   (g) "Modeling Ill-Structured Optimization Tasks through Cases"
   Decision Support Systems: An International Journal, under review, by Miyashita,K. and Sycara,K. and Mizoguchi,R.

2. **Books**

   (a) "Adaptive Schedule Repair"
   in: Szelke,E. and Kerr,R. (eds.), *Knowledge Based Reactive Scheduling*, North Holland, pp.107-124, 1994, by Sycara,K. and Miyashita,K.

(b) "Adaptive Case-Based Control of Schedule Revision"
in: Zweben,M. and Fox,M. (eds.), *Intelligent Scheduling*, Morgan Kaufmann, 1994, pp.291-308, by Miyashita,K. and Sycara,K.

(c) "Learning control knowledge through case-based acquisition of user optimization preferences in ill structured domains"
in: Tecuci,G. and Kodratoff,Y. (eds.), *Machine Learning and Knowledge Acquisition: Integrated Approaches*, Academic Press, To be published, by Sycara,K. and Miyashita,K.

3. **Surveys**

(a) "Knowledge Acquisition Methods for Building Knowledge-Based Scheduler" (in Japanese)
Journal of the Society of Inspection and Control Engineers, Vol.**33**, No.7, pp.554-560, 1994, Miyashita,K.

4. **International Conferences / Workshops**

(a) "Case-Based Schedule Repair : An Initial Report"
Proc. of IJCAI Workshop on Artificial Intelligence Approaches to Production Planning : Tools for Master Scheduling and Sequencing, pp.18-25, 1991, by Miyashita,K. and Sycara,K.

(b) "CABINS : Case-Based Interactive Scheduler"
Proc. of AAAI Spring Symposium on Practical Approaches to Scheduling and Planning, pp.47-51, 1992, by Miyashita,K. and Sycara,K.

(c) "Incremental Schedule Modification"
Proc. of the AAAI Spring Symposium on Computational Considerations in Supporting Incremental Modification and Reuse, pp.42-45, 1992, by Sycara,K. and Miyashita,K.

(d) "Case-Based Incremental Schedule Revision"
Proc. of AAAI-92/SIGMAN Workshop on Knowledge-Based Production Planning, Scheduling and Control, pp.78-91, 1992, by Miyashita,K. and Sycara,K.

(e) "Improving Schedule Quality through Case-Based Reasoning"
Proc. of AAAI-93 Workshop on Case-Based Reasoning, pp.101-110, 1993, by Miyashita,K. and Sycara,K.

(f) "Predictive and Reactive Scheduling through Iterative Revision"
Proc. of IJCAI-93/SIGMAN Workshop on Knowledge-Based Production Planning, Scheduling and Control, pp.226-237, 1993, by Miyashita,K. and Sycara,K.

(g) "Learning from Failure Experiences in Case-Based Schedule Repair"
Proc. of the Twenty-seventh Hawaii International Conference of System Sciences, 1994, by Sycara,K. and Miyashita,K.

(h) "Learning Control Knowledge through Cases in Schedule Optimization Problems "
Proc. of the Tenth IEEE Conference on Artificial Intelligence for Applications, pp.33-39, 1994, by Miyashita,K. and Sycara,K.

(i) "Case-Based Knowledge Acquisition for Schedule Optimization"
Proc. of '94 Japan/Korea Joint Conference on Expert Systems, pp.23-28, 1994, by Miyashita,K.

(j) "Case-Based Acquisition of User Preferences for Solution Improvement in Ill-Structured Domains"
Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 1994, by Sycara,K. and Miyashita,K.

(k) "Case-Based Knowledge Modeling for Ill-Structured Optimization Problems"
Proc. of the Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW94, 1994, by Miyashita,K. and Sycara,K. and Mizoguchi,R.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In practical problem solving situations, it is often required to search for an optimal or, at least, *satisficing* [Simon, 1981] solution for a given problem with limited computational resources (e.g. CPU time and memory). However, for many real-world problems, such as design, planning and game playing, it is extremely difficult to develop such a computer program that meets the above demand because they belong to the class of *ill structured problems* [Simon, 1973]. Many real world problems are ill structured for two main reasons. First, general problem solving methods, both Operations Research based and Artificial Intelligence based, need explicit representation of objectives in terms of an objective function or an evaluation function to search for an optimal solution [Newell & Simon, 1972; Reeves, 1993]. In many practical problems, optimization criteria often involve context- and user-dependent tradeoffs which are impossible to realistically consolidate in the form of a simple function. For example, in a chess playing program, the concept of "best move" is well defined *in principle*; but in practice, this concept has to be replaced by maximizing some approximate objective function. When the chess playing program has found the move that maximizes this function, it can still be far from finding the move that will "win" the game of chess. Second, since a significant portion of real-world problems are computationally intractable, solving these problems requires formidable amount of computation. Even when a problem can be divided into several tractable subproblems, efforts for solving the entire problem may become large because of deleterious interactions among the various subproblems.

To conquer the above difficulties, knowledge engineering methodology has been proposed to develop a system, which is called an expert system, that finds appropriate quality of solutions efficiently by explicitly exploiting heuristic knowledge elicited from human experts in the problem domain. Until now, a large number of expert systems have been successfully developed and used in realistic operational environments, but expert system approaches, while having a potential to capture context-dependent tradeoffs in rules, require considerable amount of efforts for knowledge acquisition from human experts in the domain [Prerau, 1990]. Moreover, in many complicated optimization problems, even the human experts themselves lack heuristic knowledge potentially useful for solving the problems efficiently with satisfactory quality.

This thesis presents an approach, implemented in the CABINS system, to demonstrate the capability of acquiring control knowledge both for meeting context-dependent user's optimization preferences and achieving situation-sensitive efficient search, and reusing them to guide solution optimization efficiently in ill-structured domains. The approach uses case-based reasoning (CBR) [Kolodner, Simpson, & Sycara, 1985; Kolodner, 1993] which has a potential for dealing with exceptional and/or noisy data [Golding & Rosenbloom, 1991; Ruby & Kibler, 1992; Aha, Kibler, & Albert, 1991], acquiring user knowledge in complex domains [Chaturvedi, 1993], and expending less effort in knowledge acquisition compared with knowledge acquisition for rule-based systems [Simoudis & Miller, 1991]. An optimization method utilized in CABINS is iterative revision of a complete but suboptimal solution. CABINS iteratively searches for a better solution in the neighborhood of a current solution; a revision action is selected and applied to the current solution, and a result is evaluated according to user preferences.

A task domain chosen for illustrating the method and evaluating its performance is a job shop scheduling problem. In the job shop scheduling problem, a solution is required to satisfy every constraint imposed by a user and optimize schedule quality based on user's preferences. At the same time, the solution must be found within reasonable amount of computation. In iteratively revising a schedule, CABINS uses case-based learning to incrementally acquire both the *repair control model* that directs the revision and the *user preference model* that evaluates a result of the revision. CABINS records situation-dependent tradeoffs about repair action selection and schedule quality evaluation 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. And the method enables the system to adjust its repair result evaluation criteria, which change dynamically based on the contexts of the current problem and the repair result.

The rest of this chapter introduces the task domain of this thesis in more details and discusses its difficulties, then briefly describes the overall architecture of the CABINS system.

1.1 Task Domain

The approach was evaluated through extensive controlled experimentation on job shop scheduling problems. Job shop scheduling deals with allocation of a limited set of resources to a number of activities associated with a set of orders. The dominant constraints in job shop scheduling are *temporal activity precedence* and *resource capacity* constraints. The activity precedence constraints along with an order'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, such as minimize tardiness, minimize work in process inventory (WIP), maximize resource utilization, minimize cycle time etc. Job shop scheduling is difficult to automate because of the following reasons:

1. Computational complexity

   The job shop scheduling is a problem in the "hardest" subset of NP-complete problems [French, 1982]. Because of tight interactions among scheduling constraints and non-linear nature of scheduling objectives, there is no general way to predict effects of a local optimization decision on global optimality, even for the simplest objective.

   In some problems, heuristics known and used by human experts in the domain may help reduce complexity of the problems drastically. But, in scheduling problems, expertise of human experts itself often become a subject of controversy. Through past experiences of developing knowledge-based scheduling systems, it has been pointed out that a human scheduler does not have sufficient knowledge for making a good schedule efficiently [Kempf *et al.*, 1991].

2. Complex objectives and preferences

   For practical job shop 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. [Mckay, Buzacott, & Safayeni, 1988]).

The scheduling problem has been addressed by two general types of methods, *constructive* scheduling and *revision-based* scheduling. The constructive scheduling approaches (e.g.

[Fox, 1983; Sadeh, 1991]) that utilize incremental construction of partial schedules to produce a complete schedule, which might be efficient when specific problem structure can be effectively exploited for application of strong heuristics, require extensive search efforts and are inefficient to solve large and complicated problems because of the first reason described above. Revision-based approaches to schedule generation, which have been adopted in the previous research activities (e.g. [Minton *et al.*, 1990; Zweben, Deale, & Gargan, 1990; Biefeld & Cooper, 1991; Laarhoven, Aarts, & Lenstra, 1992]), have a practical advantage, because they can be terminated at any time and produce a schedule. In these systems, an initial schedule is repaired by several techniques, such as a min-conflict heuristic and simulated annealing, assuming an existence of the explicitly defined objective to be optimized. But the second reason above indicates that this assumption is not valid in realistic applications. With the same reason, most of the constructive methods that utilize the explicit objective function are not appropriate either.

The rest of this section presents the job shop scheduling problem within the framework of constraint satisfaction problem (CSP), and describes objectives and preferences in the job shop scheduling problem. Then the revision-based scheduling strategy used in CABINS is discussed at the end of this section.

1.1.1 Constraints

The job shop scheduling problem requires scheduling a set of orders $O = \{O_1, \dots, O_n\}$ on a set of physical resources $RES = \{R_1, \dots, R_m\}$. Each order O_l consists of a set of activities $A^l = \{A^l_1, \dots, A^l_{n_l}\}$ 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 order O_l has a release date rd_l that signifies the earliest time the order can be started and an order due date dd_l, by which the order 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 order to which the activity belongs. If necessary, the model allows for additional unary constraints that further restrict the set of admissible start times of each activity, thereby defining one or several time windows within which an activity has to be carried out (e.g. a specific shift in factory scheduling). 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} $(1 \le j \le p^l_i)$, for each of which there may be a pool of physical resources from which to choose, $\Omega^l_{ij} = \{r^l_{ij1}, \dots, r^l_{ijq^l_{ij}}\}$, with $r^l_{ijk} \in RES(1 \le k \le q^l_{ij})$ (e.g. several possible milling machines).

More formally, the problem can be defined as follows:

<u>VARIABLES</u> : A vector of variables is associated with each activity, $A^l_i(1 \le l \le n, 1 \le i \le n_l)$, which includes:

1. the *activity start time*, st^l_i, and

2. each *resource requirement*, $R^l_{ij}(1 \le j \le p^l_i)$ for which the activity has several alternatives.

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

1. *Precedence constraints* defined by process routings translate into linear inequalities of the type: $st^l_i + du^l_i \le st^l_j$ (i.e. A^l_i BEFORE A^l_j)
2. *Capacity constraints* that restrict the use of each resource to only one activity at a time translate into disjunctive constraints of the form: $(\forall p \forall q R^k_{ip} \neq R^l_{jq}) \vee st^k_j + du^k_i \le st^l_j \vee st^l_j + du^l_j \le st^k_i$. These constraints simply express that, unless they use different resources, two activities A^k_i and A^l_j cannot overlap.

Time is assumed discrete, i.e. activity start times and end times can only take integer values. Each resource requirement R^l_{ij} has to be selected from a set of resource alternatives, $\Omega^l_{ij} \subseteq RES$. Additionally, there are unary constraints restricting the set of possible values of individual variables. These constraints include non-relaxable release dates and non-relaxable due dates between which all activities in an order need to be performed. The model actually allows any type of unary constraint that further restricts the set of possible start times of an activity.

1.1.2 Objectives and Preferences

It is not easy to state *scheduling objectives* in practice. They are numerous, complex, and often conflicting and mathematics of the problem can be extremely difficult with even the simplest of objectives [French, 1982]. The followings show definition of the objectives, which are among the most common in the literature (e.g. [French, 1982]), that are used to develop the performance evaluation of CABINS.

Waiting time (W^l_i) : is the time that elapses between the completion of the preceding activity A^l_{i-1} (or rd_l, if $i = 1$) and the start of processing A^l_i.

Total waiting time (W_l) : is the sum of waiting time of all activities that belong to O_l. Clearly $W_l = \sum^{n_l}_{i=1} W^l_i$.

Completion time (C_l) : is the time at which processing of O_l finishes. $C_l = rd_l + \sum^{n_l}_{i=1}(W^l_i + du^l_i)$

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

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

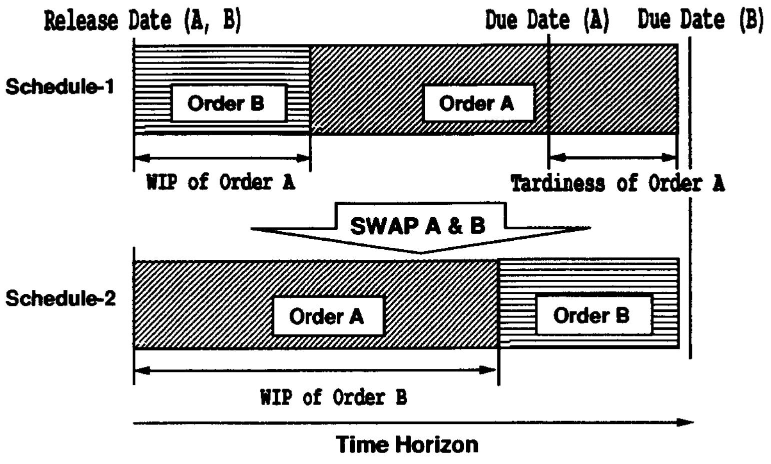

Fig. 1.1: An example of conflicting objectives in a simple scheduling problem

Flowtime (F_l) : is the amount of time that O_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 (*WIP*) : is the summation of total waiting time. $WIP = \sum_{i=1}^{n} W_i$

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

The quality of a schedule is a function of the extent to which it achieves *user's preferences.* A simple example can illustrate the necessity of having user's preferences in the scheduling system. Let us assume the simplest factory with a single machine and two orders; each order consists of a single activity to be processed on the factory machine, and the two orders are released to the factory floor at the same time.

Fig. 1.1 shows two schedule results for such a problem. Suppose schedule-1 is generated. In this schedule, order B finishes before its due date but order A is tardy. The WIP of order A is indicated in Fig. 1.1 (the WIP of order B is zero). Suppose one wishes to revise the schedule to reduce the tardiness of order A. In this simple schedule, the only possible repair is to switch the positions of order A and order B. The schedule resulting from this switch is schedule-2. In schedule-2, neither order is tardy but the WIP in schedule-2 (the WIP of order B) is larger than the WIP in schedule-1 (the WIP of order A). Even in this extremely simple example, it is impossible to determine which schedule is of higher quality without taking into consideration the preferences of the user.

The objectives defined in this section are mathematical simplifications of state-dependent user's preferences on a schedule that are difficult to model precisely. In the above example, user's preference (i.e. tradeoff between WIP and tardiness) may depend upon many factors in the problem, such as a client of each order, past shipping records, load of a factory / warehouse and so on. The combination of these factors produces enormous number of contexts in which user preferences are considered, thus making user's preference difficult to be captured and represented a priori in the problem representation. CABINS can acquire correct optimization criteria to the extent that a user can give consistent evaluation of schedules.

1.1.3 Revision-based Optimization

In CABINS, a schedule is optimized using a revision based methodology. Revision based schedule optimization in CABINS is motivated by several considerations.

- There are no known efficient algorithms of schedule optimization except for a very limited set of simple objectives such as make-span (e.g. [Adams, Balas, & Zawack, 1988]) and amount of computation required for finding a solution is generally unpredictable [French, 1982]. Thus, the construction of a cheap but suboptimal schedule that is then incrementally repaired to meet optimization objectives is preferable in practice, because one can interrupt repair process and use an interim result for execution when no more time is allowed for further repair [Zweben, Deale, & Gargan, 1990]. For example, dispatch heuristics have very low computational cost, but owing to their myopic nature 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 practicians. This indicates that combining a repair methodology, such as a simple gradient search [Kanet & Sridharan, 1990], neural networks [Johnston, 1990], or the one advocated in this thesis, with a dispatch driven scheduler for creation of an initial schedule is promising for real-world scheduling environments.

- An incentive for utilizing incremental schedule repair occurs in response to the need to incorporate context-dependent user preferences and additional constraints that have not been represented in the scheduling model. Even for simple scheduling problems, it is difficult to predictively evaluate the conflicting tradeoffs present in scheduling objectives, and formulate the model to represent user preferences in general and all-inclusive ways, as presented in the example in Section 1.1.2.

  Users' preferences on the schedule are context dependent (e.g. may depend on the state of the scheduling environment at a particular time). Also, interactions among preferences and effective tradeoffs very often depend on the particular schedule produced. This indicates that generally a user of the scheduling system cannot fully specify his/her preferences a priori before getting scheduling results from the system.

By looking over the obtained schedule results, the user often thinks of additional preferences and modifies the schedule. Consider, for example a situation where a human scheduler does not like to use machine-A that is substitutable for machine-B but is of lower quality than machine-B for processing order-X. The reason why high quality results are desired is that order-X belongs to a very important client. Suppose, however, that the produced schedule indicates that order-X is tardy by an amount above an acceptable tardiness threshold on account of high demands on machine-B (by orders more important than order-X). Then the human scheduler may decide to use the less preferable machine, machine-A, for order-X. If the tardiness was below the threshold, he/she may prefer to allow a tardy order. It is very difficult to elicit this type of preference and preference thresholds from the human scheduler in the absence of a particular scheduling context.

Moreover, it is impossible for any given knowledge-based scheduling model to include all the constraints that may be relevant. Current advanced scheduling systems can exploit very complicated models to represent a factory, orders and user's preferences. But no matter how richly the model is constructed, there are always additional factors that may influence the schedule but had not been represented in the model. For example, for a certain foundry it may be good to decrease usage of a sand casting machine during the summer, because the combination of heat and humidity of the weather may make it slower than usual. But how should the model of the scheduling system represent the season, weather or humidity? And isn't it necessary for the model to represent time of the day, strength of wind or health of a machine operator and so on [Mckay, Buzacott, & Safayeni, 1988]? Nevertheless these factors, which an experienced human scheduler learns to take into consideration, could have a big influence on schedule quality but it is very difficult to represent them in a principled manner so they can be used by an automated scheduling system. In other words, constructive scheduling systems with an *explicit* problem model will fail to meet real-world requirements.

The job shop scheduling problem is a well-known NP-complete problem [Garey & Johnson, 1979]. Typically the limited resource capacity induces interactions between activities competing for the possession of the same resource during overlapping time intervals. Hence, in the job shop scheduling problem, except for artificial or simplistic situations, it is generally not possible to impose rigid bounds on the quality of a solution (e.g. amount of weighted tardiness) with assurance that a solution actually exists. Moreover, it is not possible to bound the scope of a repair in advance. In scheduling, given the tightly coupled nature of scheduling decisions, a change in one part of a schedule often has ripple effects that can span the time horizon of the rest of the schedule from the point of change. For example, in Fig. 1.2 moving forward the last activity of ORDER3 creates downstream cascading constraint violations. The use of heuristics in selecting an appropriate repair action

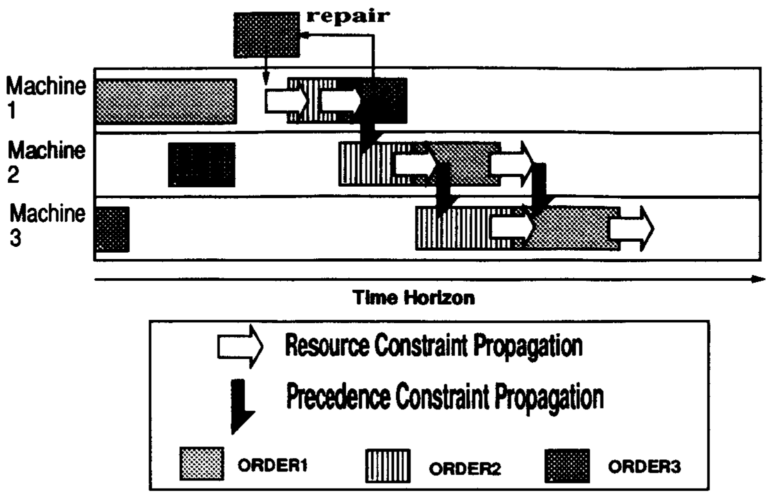
repair
Machine 1
Machine 2
Machine 3
Time Horizon
Resource Constraint Propagation
Precedence Constraint Propagation
ORDER1
ORDER2
ORDER3

Fig. 1.2: An example of tight constraint interactions in a scheduling problem

can help prevent such unrestricted propagation. But, considering the combinatorics of the job shop scheduling problem, even human experts are not believed to have the generalized heuristic knowledge for selecting the right repair action at the right situation. CABINS inductively learns the concept of an appropriate repair action in particular problems from successful and failed repair cases.

1.2 CABINS: Case-Based Repair Approach

The consideration of the job shop schedule optimization task and the demands of the knowledge acquisition task in the domain, make it clear that (a) an iterative revision optimization technique would be most suitable, (b) recording the user's judgments in a case base is an effective and flexible way of eliciting user optimization preferences, and (c) recording successful and failed repair trials in a case base is an appropriate way of learning to improve search efficiency. The author hypothesizes that these observations hold true in most of the real-world ill-structured optimization problems (e.g. VLSI layout and circuit design, transportation planning). This thesis presents an iterative repair framework that addresses the above points in a uniform and domain-independent manner. The rest of this section briefly describes the architecture and methodologies of the CABINS system. More complete descriptions of CABINS will appear in Chapter 2 and Chapter 3.

1.2.1 Architecture

CABINS has three classes of repair decisions: *repair goals*, *repair strategies* and *repair tactics*. A repair goal is derived from a particular high level description of defects in the current solution of a problem and their significance to a user. A way to achieve a particular repair goal is designated by a selection of one of the associated repair strategies for it. Each repair strategy is executed by a successive application of a variety of repair tactics associated with it.

The author has identified two general types of repair strategies that control the extent of repair effects : *local patching* and *model modification.* Local patching forces CABINS to select the repair actions that result in changing value assignments in the solution within a given set of constraints. Local patching is in general less costly and disruptive for execution. In a factory scheduling problem, for example, if the repair goal is "reduce order tardiness ", specific strategies by local patching may include "reduce the slack between activities in the tardy order", and "reduce the idle-time of resources needed by activities in the tardy order". Model modification allows CABINS to re-formulate the problem by changing model parameters, such as the number of orders to be scheduled, or global constraints such as changing release or due dates, increasing resource capacity or increasing number of shifts. Model modification facilitates the solution of the problem, since it amounts to global constraint relaxations. However, in practice, model modification is costly to implement (e.g. "buy new equipment", "pay for extra shifts in a factory", "subcontract orders to outside contractors"). The default repair strategies in CABINS are of local patching type. Repairing with local patching strategies is a computationally more challenging task, since the system must improve the solution without relaxing the already imposed constraints. If local patching is unsuccessful in fulfilling the repair goal, the repair episode is considered a failure. The experiments in this thesis were run within these more stringent assumptions.

CABINS can operate in different modes that exhibit various levels of autonomy.

- *User-directed* mode

  The user selects a repair decision (i.e. goal, strategy and tactic) and evaluates the results of its application.

- *Interactive assistance* mode

  CABINS suggests repairs and evaluations of repair results, but the user can override the suggestions and make new selections. Both the user-directed and interactive assistance modes are used for acquisition of the cases.

- *Autonomous* mode

  Without user intervention, CABINS uses the case base that was acquired in the training phase for repair selection and evaluation of repair results.

In the experiments reported in the thesis, CABINS operated autonomously.

Fig. 1.3 shows the schematic diagram of the overall architecture of the CABINS implementation. CABINS is composed of three modules: (1) an initial solution builder, (2) an interactive repair (case acquisition) module and (3) an automated repair (case re-use) module.

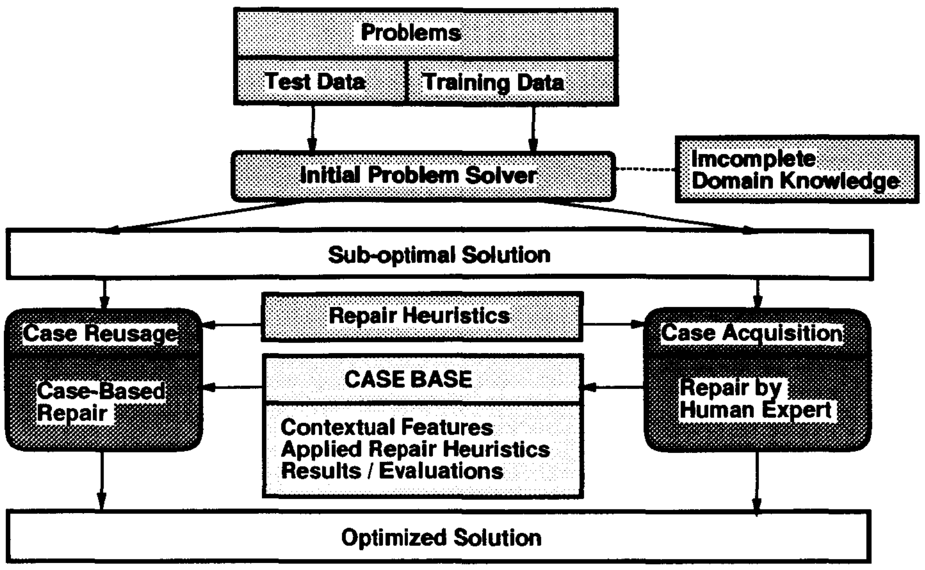

Fig. 1.3: CABINS architecture

To generate an initial solution, CABINS can use one of several problem solving methods (e.g. use of dispatch rules) available in CABINS. But, in general any initial problem solver cannot always produce an optimal solution, because the complete knowledge of the domain and user's preferences are not available to the problem solver.

In order to compensate for the lack of these types of knowledge, CABINS gathers the following information in the form of cases through interaction with a domain expert in its training phase.

- A suggestion of which defect to be repaired : a user's selection of the most critical defect in a given situation
- A suggestion of which repair heuristic to apply : a user's decision on what repair heuristic to be applied to a given solution for quality improvement.
- An evaluation of a repair result : a user's overall evaluation of a modification result.

A basic assumption of the case-based approach is that, in spite of ill-structuredness of the problem, the following three types of domain knowledge are available and constitute useful case features.

- Repair heuristics : a set of repair heuristics that can be applied to a problem.
- Descriptive features : attributes of a problem that describe a particular problem situation and might be useful in estimating the effects of applying repair heuristics to the problem. These features will be explained in detail in Section 2.2 for the job shop scheduling problem.

In formulating cases, the author assumes the existence of effective indexing vocabularies of the cases for domain-independent repair task and hypothesizes that it is relatively easy to cast these domain independent vocabularies into a particular application domain such as a scheduling problem. This will be explained in more details in Section 2.1.

- Evaluation criteria : quantification of different aspects of the effects of applying repair heuristics to the problem. The degree of importance on these criteria is in general user- and state-dependent.

Once enough number of the cases have been accumulated, CABINS can efficiently repair the solutions created by the initial problem solver to user's satisfaction by invoking CBR process without further interactions with the user.

1.2.2 Schedule Repair by CABINS

The approach to incremental schedule repair, implemented in the CABINS system, uses an integration of case-based reasoning (CBR) and fine granularity constraint propagation mechanisms [Miyashita & Sycara, 1994a]. Integrating CBR with constraint propagation stems from a variety of motivations. Because a case describes a particular specific experience, the factors that were deemed relevant to this experience can be recorded in the case. This description captures the dependencies among schedule features, the repair context and a suitable repair action. CBR allows capture and re-use of this dependency knowledge to dynamically control the search process and differentially bias schedule repair decisions in future similar situations. On the other hand, because of the tightly coupled nature of schedule repair decisions, a revision in one part of the schedule may cause constraint violations in other parts. It is in general impossible to predict in advance either the extent of the constraint violations resulting from a repair action, or the nature of the conflicts. As a result, constraint propagation techniques are necessary to determine the *ripple effects* that spread conflicts to other parts of the schedule as repair actions are applied and specific revisions are made.

In CABINS, the repair method guarantees a conflict free schedule at the end of each repair iteration, thus exhibiting *anytime executable* behavior [Dean & Boddy, 1988]. This is especially important in time-critical contexts since there could only be a certain limited amount of time for the system to solve a scheduling problem. Unlike other systems that utilize iterative repair to find a feasible schedule (e.g. [Zweben, Deale, & Gargan, 1990; Minton *et al.*, 1990]), 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 better quality with a increase of the time allowed for repair.

CABINS acquires from cases a category of concepts that reflect user preferences: what combination of effects produced by an application of a particular local optimization action on a schedule constitutes an acceptable or unacceptable outcome. In CABINS, the optimization criteria are not explicitly represented as case features or in terms of a cost function, but they are implicitly and extensionally represented in the case base. CABINS learns three additional categories of concepts that reflect control knowledge for quality enhancement and efficiency improvement: (1) what aspect of a schedule to be repaired, (2) what heuristic repair action to choose in a particular repair context, and (3) when to give up further repair. These concepts are recorded in the case base and are used by CABINS to guide iterative optimization and infer optimization tradeoffs in evaluating the current solution. In this way, the acquired knowledge is exploited to enhance the incomplete domain model in CABINS and improve efficiency of problem solving and quality of resulting solutions according to the user preferences.

1.2.3 CBR for Schedule Optimization

Because of the characteristics of the scheduling domain described in Section 1.1 and the interest in capturing context dependent user preferences and situation sensitive search control knowledge, CBR appears to be a natural method for knowledge acquisition. However, applying CBR to schedule improvement, a numerical optimization problem, is very challenging. In general, CBR has been used for ill-structured symbolic problems, such as planning [Hammond, 1989; Kambhampati & Hendler, 1992; Veloso, 1992], legal reasoning [Ashley, 1987; Rissland & Ashley, 1988], argumentation [Sycara, 1989], conceptual design [Sycara *et al.*, 1991], medical diagnosis [Koton, 1988] where the primary concern has been plausibility or correctness of resulting artifacts (plan, argument, design) and computational efficiency of problem solving process rather than artifact quality.

The challenges in applying CBR to schedule optimization were to determine what constitutes a case in the domain of schedule optimization and what the case indices should be. The intuitive answer would be to consider a whole schedule as a case [Koton, 1989; Mark, 1989]. This solution is attractive since, if the right information could be transferred from one scheduling scenario to another, or with little adaptation, a new problem would be solved with relative ease. In the traditional planning problem, the plan operators capture some form of domain causality in their preconditions and effects. Saving a plan and the derivational trace of how the plan was generated, captures pretty much the planning process and can be easily utilized to solve future similar problems (e.g. [Hammond, 1989; Kambhampati & Hendler, 1992; Veloso, 1992]). However, it is not true in the scheduling problem. Because of the high degree of nonlinearity of scheduling constraints and objectives, a very small difference between an input problem specification and the problems in the case base can in general result in large variations in the results both in terms of the amount of modifications needed and the quality of a resulting schedule. A second difficulty with respect to having a whole schedule as a case came in the form of what indices to choose. Indexing a case in terms of the goals that must be achieved and the problems that must be avoided [Hammond, 1989] is a good guideline and has served many CBR systems well. However, in the scheduling domain, the goals to be achieved (the optimization criteria) cannot be explicitly stated, since they reflect context-dependent user preferences and tradeoffs. Even if the optimization objectives were explicit, because of the nonlinearities of the problem, retrieving a schedule in which the achieved objectives were the same as the desired ones in the current problem would give little or no help in adapting the retrieved schedule to the current problem specifications. Moreover, because of unpredictable ripple effects of constraint propagation and tight constraint interactions, the problems to be avoided are not at all obvious, neither can they be discovered since a causal model for scheduling cannot be assumed.

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

Hence in CABINS, a case describes the *application of a schedule revision decision on a single scheduling entity.* Operationalization of a schedule revision decision is done by means of a *schedule repair action.* The analysis have identified three classes of schedule repair actions (i.e. goal, strategy and tactic), which will be described in detail in Chapter 2. A scheduling entity for a repair goal is a whole schedule, a scheduling entity for a repair strategy is an order and a scheduling entity for a repair tactic is an activity. Each application of a schedule repair action results in a new schedule. The search space of CABINS is the space of complete schedules that incorporate acceptable user optimization tradeoffs. Hence the predictive case features that are suitable for case indexing should be ones that capture good tradeoffs. Although schedule optimization is ill-structured, the author makes the hypothesis that there are regularities of the domain that can be captured, albeit in an approximate manner, in these features. In CABINS, indices are divided into two categories. The first category consists of the *descriptive features.* Since the results of schedule revision associated with a single scheduling entity pertain to the whole schedule, it is impossible to make a precise prediction of repair effects in advance of revisions. Descriptive features that express characteristics of a scheduling entity operate as contextual information for selection of a particular repair action and allow CABINS to estimate the effects of each repair action in advance.

The schedule resulting from a repair action application must be evaluated in terms of user-defined tradeoffs. The user cannot predict the effects of modification actions on schedule correctness or quality since a modification could result in worsening schedule quality or introducing constraint violations. Nevertheless, the user can perform consistent evaluation of the results of schedule revisions. This evaluation is recorded in the case as part of the case's repair history. The *repair history* constitutes the second category of case features. Thus, the case base incorporates a distribution of examples that collectively capture repair performance tradeoffs under diverse scheduling circumstances.

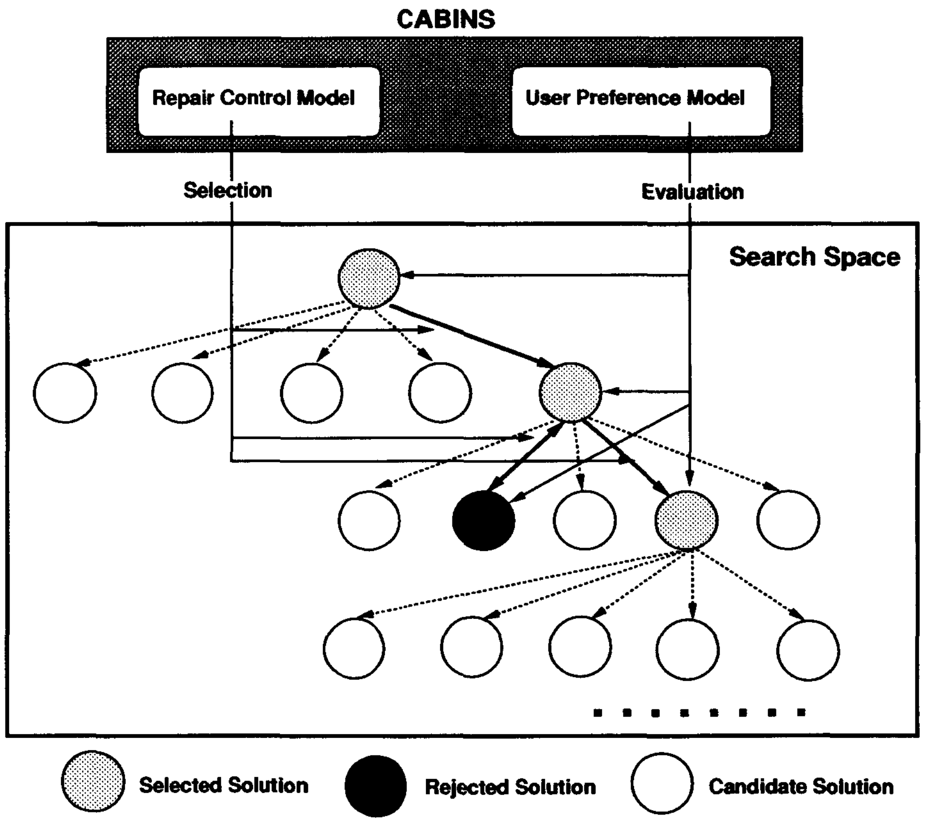

Fig. 1.4: Search space and search control in CABINS

CABINS searches for an "optimal" schedule over the space of complete schedules. Fig. 1.4 shows the schematic diagram of the search space and search control in the CABINS system. CABINS revises the current solution iteratively to improve the solution quality. For each step of the search, CABINS selects a solution among the neighbors of the current solution.

The neighborhood size for the current solution (i.e. the number of potential solutions for each revision) is equal to *Number_of_Repair_Actions* × *Number_of_Repair_Objects*. In a scheduling problem, *Repair_Actions* are several heuristics that modify the assignments of resources to activities in the schedule and *Repair_Objects* are typically the activities in the schedule. The number of revision cycles required to obtain a final solution cannot in general be predicted in advance because of tight constraint interactions in the scheduling problem. Hence the search space for a large scheduling problem can be intractably big. To reduce the required search efforts, CABINS has the following mechanisms of search control using CBR: A *repair control model* provides the search control through case-based selection of the next repair action to be applied, and a *user preference model* provides the search control through case-based evaluation of the result of the application of a selected repair action. The descriptive features are the indices that are used to retrieve a case that suggests the next repair action to be applied. The features associated with the repair history are used to retrieve cases that suggest evaluations of a repair outcome. For a more detailed description of case representation and indexing, see Chapter 2 and Chapter 3.

1.3 A Guide to this Thesis

This thesis presents the methodology and initial experimental results to test the following hypotheses: (1) CBR-based incremental revision methodology shows good potential for capturing user optimization preferences in ill-structured domains, such as job shop scheduling, and re-use them to guide optimization, (2) CABINS can learn control knowledge for improving solution quality and problem solving efficiency and improve its own competence through case accumulation, and (3) CABINS produces solutions of high quality as compared with other optimization techniques. To validate the above hypotheses, the empirical studies were done in the domain of the job shop scheduling. And to test the last hypothesis, the author compared the solutions produced by CABINS for the specific optimization criteria, with solutions produced by simulated annealing (a well known iterative optimization technique) for the same criteria.

The rest of the thesis is organized as follows: Chapter 2 presents case descriptions and case acquisition in CABINS. Chapter 3 explains how to use cases for schedule optimization problems. Chapter 4 describes the knowledge acquisition by CABINS for improving solution quality and presents the experimental results. Chapter 5 presents the learning methods of improving search efficiency by CABINS and some empirical observations. Chapter 6 presents additional experimental results to address the issue of case accumulation effects to solution quality and problem solving efficiency. Chapter 7 provides the comparison of this work with other related works. Chapter 8 presents concluding remarks and suggestions for future research.

Chapter 2

Building a Case Base

A knowledge-based system (or an expert system) has explicit representation of knowledge in addition to the inference mechanism that operates on the knowledge to achieve the system's goal. A knowledge base represents a *model* of how domain experts approach a complicated problem in the domain. It is an *operational model* which, hopefully, exhibits some desired behavior which has been specified or observed in the real-world — in exactly the same way as a *mathematical model* attempts to mirror real-world situations.

Builders of expert systems formulate the model, first by defining a model of the behavior that they wish to understand and then corroborating and extending that model with the aid of specific examples. For example, PROTÉGÉ [Musen, 1989] has two interrelated phases of knowledge base construction: (1) *model building* and (2) *model extension.* When building a model, developers must first perform a requirements analysis and identify the *task* that the expert system has to perform. Then, knowledge engineers and domain experts cooperate to construct a model of the proposed system's behavior. This model generally corresponds to the developer's theory of how the experts actually solve the problem. For extending a model, the model of the intended behavior of the expert system is validated by ascertaining how well the model applies to closely related application problems.

Much of the activities involved in the first stage of model formulation, model building, entails *knowledge-level* [Newell, 1982] analysis, which determines (1) the *goals* for a knowledge-based system, (2) the *actions* of which the system is capable, and (3) the *knowledge* that the system can use to determine the actions that attain the goal. In recent research of AI, there is a clear consensus in favor of knowledge-level analyses and its advantages for knowledge modeling and acquisition. Chandrasekaran and his colleagues advocated the *generic task* framework [Chandrasekaran, 1988] and identified a number of tasks of general utility (such as classification), methods for performing the tasks and the kinds of knowledge needed by the methods. Clancey proposed *heuristic classification* [Clancey, 1985] as an abstract inference pattern for a diagnosis task by examining some expert systems such as MYCIN. McDermott developed *half-weak methods* [McDermott, 1988], such as *cover-and-differentiate* and *propose-and-revise* methods, for solving general tasks that do not require domain specific search control knowledge. These methodologies of knowledge-level analyses have successfully been applied to the development of various expert systems.

However, the latter stage of model formulation, model extension, has no generic methodology corresponding to the knowledge-level analysis for the first stage of model formulation. It is generally believed that, though creating a knowledge model may be difficult, extending an existing model is less arduous for human experts. In other words, whereas domain experts may not be able to introspect and articulate the *process knowledge* that allows them to solve problems, these experts can easily supply the *content knowledge* that may or may not be consistent with a given model. To elicit consistent domain knowledge from human experts, several model-based knowledge acquisition tools have been developed such as MOLE [Eshelman, Ehret, & McDermott, 1987], SALT [Marcus & McDermott, 1987], KNACK [Klinker, 1988] and OPAL [Musen *et al.*, 1987].

Although these model extension (knowledge acquisition) tools are powerful in allowing domain experts to make large knowledge bases without help from knowledge engineers, such tools must be strongly tied to a specific problem solving method presupposed by the tools. For example, while SALT has been proved to be useful for acquiring knowledge of the expert system, called VT, which supports design of elevator systems, SALT could not acquire effective knowledge for solving scheduling problems. The failure of SALT was caused by the fact that the *propose-and-revise* problem solving method assumed by SALT was inappropriate for the scheduling problem in spite of its structural resemblance to the design problem [Stout *et al.*, 1988]. The main reasons for this were the following characteristics of the scheduling problem which did not appear in the design problem: (1) the dynamic nature of fix preferences, and (2) the high level conflict among fixes for different constraints. Hence, if a problem solving method is generic enough to be applicable to a wide variety of tasks and, at the same time, is capable of matching the nuances of particular applications, a model extension framework based upon such a problem solving method has a highly practical value.

The author believes that the case-based reasoning methodology gives the strong leverage for building a model-extending mechanism for ill-structured optimization problems such as the job shop scheduling. Case-based reasoning (CBR) is the problem solving paradigm where previous experiences are used to guide problem solving [Kolodner, Simpson, & Sycara, 1985]. Cases similar to the current problem are retrieved from memory according to a similarity metric, the best case is selected from those retrieved and compared to the current problem. If needed, the precedent case is adapted to fit the current situation based on identified differences between the precedent and the current cases. CBR allows a reasoner (1) to propose solutions in domains that are not completely understood by the reasoner, (2) to evaluate solutions when no algorithmic method is available for evaluation, and (3) to interpret open-ended and ill-defined concepts. CBR also helps a reasoner (1) take actions to avoid repeating past mistakes, and (2) focus its reasoning on important parts of a problem [Kolodner, 1993]. Owing to the above advantages, CBR has successfully been applied to many kinds of problems such as design, planning, diagnosis and instruction. Thus CBR can be regarded as an appropriate problem solving method for a large class of applications.

In comparison to other knowledge acquisition and learning paradigms, CBR has a number of practically desirable features which encourage CBR applications in many domains [Sycara & Miyashita, To be published]. In CBR, successful cases are stored in the case base so that they can be retrieved and re-used in the future. Failed cases are also stored so that they can warn the problem solver of potential difficulties and help recover from failures. After a problem is solved, the case base is updated with the new experience. Thus, *learning is an integral part of case-based problem solving.* Moreover, because cases express particular experiences, they are a better "cognitive match" for the expert.

It is easier for experts to collect a sufficient number of problem cases by actually solving sample problems (drawing from their experience with other similar cases) rather than try to abstract the particulars of one or more problem cases in order to formulate a consistent rule-set. Hence, CBR has been considered as a more natural and less time consuming method of knowledge acquisition. Since both successes and failures are recorded in the case base, each additional case helps refine and re-formulate the knowledge in the case base. This is done without additional computational overhead since each case is a "package of knowledge" that is independent of the other cases in the case base. On the other hand, while the refined or re-formulated knowledge is explicitly captured in rule-based systems; in CBR the refined knowledge is implicit and is extensionally stored in the case base.

One of the most important differences of CBR as compared with rule-based knowledge acquisition and system operation is that, since a rule-based system will usually give satisfactory performance only after all (or nearly all) the relevant rules are collected, the lengthy process of knowledge acquisition has to precede the system operation. During knowledge acquisition, the user has to invest time to provide the knowledge to the system without getting any problem solving advice from it. CBR addresses knowledge acquisition incrementally and can save the user significant overhead by collecting the cases *during routine system operation.* In terms of practical issues, because of its capability for partial matching, a CBR system can give potentially useful advice even if it has only an initial small set of cases in the case base. In this way, knowledge acquisition and re-use of the learned knowledge is effected smoothly and non-intrusively.

It should be noted, however, that CBR is not a panacea that obviates any overhead associated with knowledge acquisition. It defines new types of knowledge acquisition tasks, i.e., definition of appropriate case features and indices. Most often, however, the acquisition of these types of knowledge is simpler than acquiring rules for problem solving. The author believes this to be true for the following reasons:

- Since rules effect a mapping from features of a situation to an outcome action, writing rules requires understanding of causality in the domain. On the other hand, since case description and indexing pertain only to situations, causality of the domain does not need to be explicitly stated.

- Superfluous features or inconsistent cases, albeit taking up storage space, do not annihilate the accuracy of the system's performance. They can be weeded out, if desired, as the system operation is monitored. In other words, CBR is a more forgiving knowledge acquisition method tolerating noisy data quite well.

- Analysis of the application problem at a task-level provides a useful category of vocabularies for describing features of situations in the problem. These vocabularies can be mapped to the domain specific vocabularies for case descriptions without much effort since both vocabularies are represented at the knowledge-level.

This paper presents a case-based approach, implemented in the CABINS system, which formulates a model of the optimization task and uses it to guide iterative solution optimization in ill-structured domains. The model of an optimization task should have the following knowledge: (1) user context-dependent preferences and (2) situation-dependent search control. In CABINS, CBR method is used for extending the optimization task model created by the domain experts and knowledge engineers based on the *task structure* analysis [Chandrasekaran, Johnson, & Smith, 1992]. The thesis shows that when coupled with the task-level analysis, the case-based reasoning method can provide strong leverage for reducing the difficulty of knowledge acquisition. To validate the effectiveness of our approach, extensive experiments on CABINS' performance have been done in the domain of job shop scheduling, which is a widely known ill-structured optimization problem.

The rest of this chapter presents the methodology of creating a case base using the knowledge-level task analysis method, the detailed case descriptions of CABINS derived from the proposed methodology, and the case acquisition protocol in CABINS.

2.1 Modeling the Optimization Task

The analysis of the ill-structuredness of the job shop scheduling domain in Section 1.1 provides insights both about the type of the problem solving methods suitable for the ill-structured optimization problem and the type of knowledge required for solving the problem. These insights are used for the definition and enhancement of a knowledge-level model for the problem.

Recently a number of uniform knowledge-level analysis frameworks for describing systems have been developed by several research groups such as MULTIS [Mizoguchi, Tijerino, & Ikeda, 1992], KADS [Wielinga, Schreiber, & Breuker, 1993], SPARK [Klinker *et al.*, 1991] and PROTÉGÉ-II [Puerta *et al.*, 1992]. The author adopt the task structure analysis

[Chandrasekaran, Johnson, & Smith, 1992] for building the model of the optimization task. The task structure is the tree of tasks, methods and subtasks applied recursively until it reaches the tasks that are in some sense performed directly using available knowledge. "Task" is synonymous with types of problem-solving goals: for example, optimization is a task since it characterizes a family of the problems, all of which require achieving the goal of generating a solution that maximizes given evaluation criteria. "Method" is a process used to achieve the goals in the task: for example, the task of optimization can be accomplished either by constructive methods or by repair-based methods.

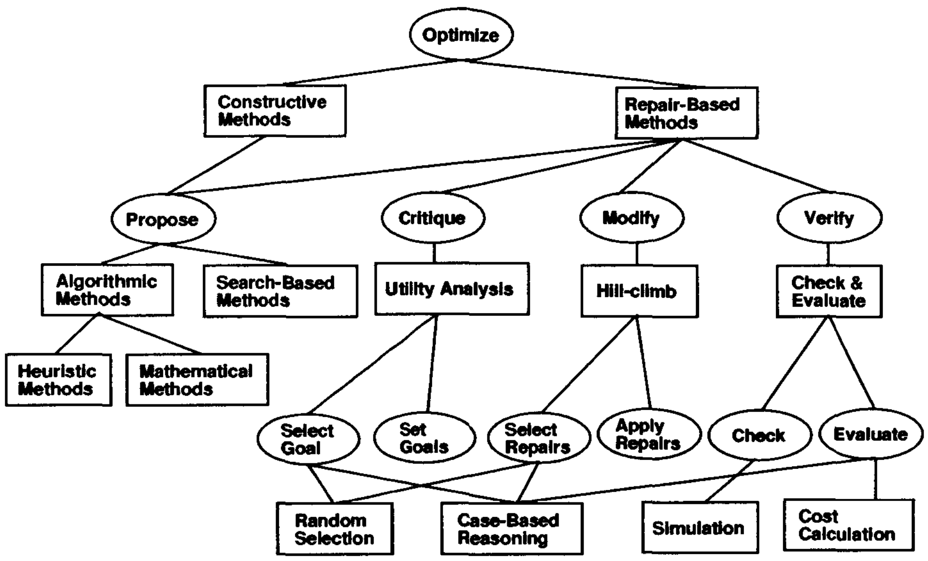

Fig. 2.1: Task structure of optimization

Fig. 2.1 shows the task structure for an optimization task. Since design can be considered as a class of the optimization task, this task structure is constructed based on the task structure of design [Chandrasekaran, 1990; Mizoguchi, 1992]. In the task structure diagram, circles represent tasks and rectangles represent methods. This diagram is not intended to show a complete task structure for the optimization task: it, however, captures some methods and subtasks that are relevant in this paper.

The optimization task can be solved either by constructive methods or by repair-based methods. But, in ill-structured domains such as scheduling, since there is no complete domain knowledge available, the constructive methods cannot produce high quality solutions in an efficient way. The repair-based methods consist of four subtasks: propose, verify, critique and modify. For proposing a solution, two principal methods are available: algorithmic and search-based. The algorithmic methods are further categorized into two classes: heuristic and mathematical. The heuristic methods solve the problem using approximated algorithms. This methodology works only for problems with a restricted problem structure. The mathematical methods, such as linear and integer programming, can solve well-structured problems only after a precise mathematical model of the problem has been constructed. Search-based methods, such as constraint satisfaction and branch-and-bound, search for the optimal solution in the space of partially constructed solutions with the help of domain specific search control knowledge.

In critiquing a solution, quality of the solution is analyzed based on the utility function of domain experts. If the solution is judged as acceptable, repair process is terminated with the solution. Otherwise, the sources of unacceptability in the solution are identified as repair goals. In modifying a solution, the most effective way of achieving repair goals is selected and applied to the current problem. For selection of repair goals and repair actions, one possible method is to find the most similar past experiences to the current problem situation, which suggest the plausible repair action in the current context. In the ill-structured problems, the random selection method is often used for selecting a repair action, since it allows a solution to escape from local minima [Reeves, 1993]. Because the methods of goal-setting and repair-application in Fig. 2.1 are strongly domain-dependent, these methods have to be defined and developed by domain experts with the help from knowledge engineers.

In verifying solutions, a problem repair must be checked regarding feasibility of the result. This can be done using simulation methods, such as constraint propagation. If a feasible solution is achieved, it is then evaluated to see whether the repair improves quality of a solution by calculating an explicit cost function, or finding whether the most similar past repair results were evaluated as acceptable or not.

In CABINS, case-based reasoning is used as a method for three subtasks: evaluation, goal selection and repair selection. This is based on our hypothesis that the knowledge required for these subtasks can be acquired with small efforts by the approach shown in Section 2.1.1.

2.1.1 Case-Based Model Extension

The task structure used for model building is an analytical tool. A system that performs the task can be viewed as using some of the methods and subtasks in the task structure. Hence, it simply provides a generic vocabulary for describing how systems work. In order to formulate a model for a specific application problem, a developer of the application program needs to extend the model using a task structure.

Fig. 2.2 shows the schematic diagram of the model formulation process in CABINS. In order to develop the CABINS system, the model of the optimization task in Fig. 2.1 needs to be extended because several kinds of domain-dependent knowledge are required for performing the task. Since CABINS uses case-based reasoning as a principal problem solving method for the optimization task, the model extension process in CABINS is composed of the two phases: *specialization* and *operationalization and progressive enhancement.*

The first phase, specialization, is carried out at the knowledge-level. In specialization, the generic vocabulary found in the task structure analysis is transformed and refined

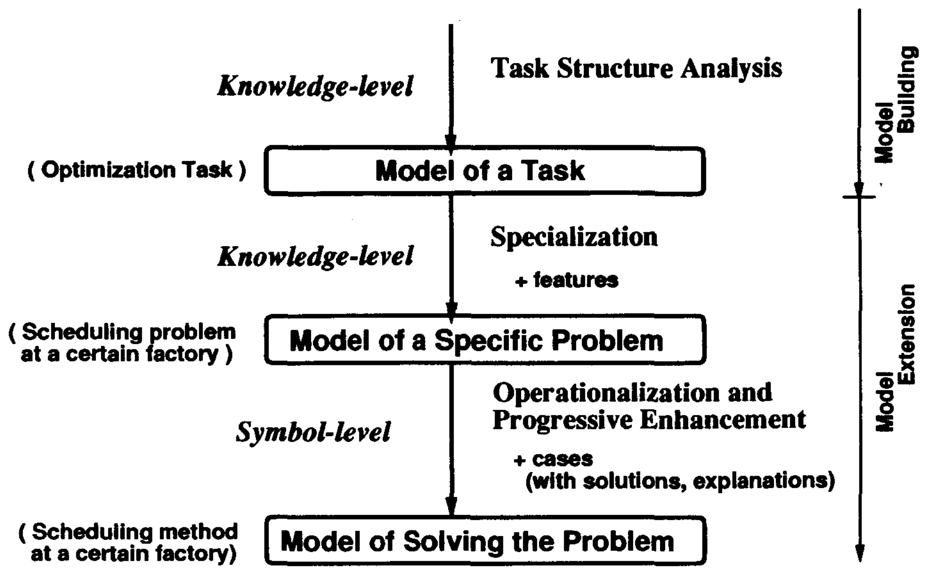
Knowledge-level
Task Structure Analysis
(Optimization Task)
Model of a Task
Knowledge-level
Specialization
+ features
(Scheduling problem at a certain factory)
Model of a Specific Problem
Symbol-level
Operationalization and Progressive Enhancement
+ cases (with solutions, explanations)
(Scheduling method at a certain factory)
Model of Solving the Problem
Model Building
Model Extension

Fig. 2.2: Model formulation in CABINS

into the vocabulary of the specific problem domain. From the CBR point of view, task structure analysis can provide the abstract case feature categories, and the specialization process maps out the feature descriptions of cases in the application domain. For example, task-level analysis suggests that a case feature category for solution quality description is necessary for the goal selection subtask, and the specialization process in the scheduling domain transforms the feature descriptions into more specific ones such as tardiness and WIP of a schedule. Because both the generic vocabulary found in the task structure analysis and the domain-specific vocabulary to be specified at specialization are described in the knowledge-level language, specialization can be accomplished by domain experts who can enumerate the appropriate feature descriptions in the application domain with the help from a knowledge engineer who can explain the meaning of the generic vocabularies.

The second phase, operationalization and progressive enhancement, is the symbol-level process. In this phase, specific cases are accumulated by domain experts in a case base according to the case descriptions defined at the previous stage. These cases contain *content knowledge* of the domain such as the judgments and explanations by domain experts in the particular problem context. In ill-structured problems such as job shop scheduling where even human experts are not expected to have sufficient understanding of the problem, acquiring content knowledge as cases is easier than acquiring it in other forms such as rules because of the following reasons: (1) no explicit understanding of domain causality is necessary since cases implicitly map problem features to solutions, (2) knowledge does not need to be abstracted since an indexing mechanism abstracts the content of cases when a case is retrieved, (3) no consistency check of knowledge needs to be done since inconsistent or wrong cases are neglected in retrieval, and (4) no meta-control information over knowledge usage (such as certainty factors) needs to be described since population and relevance of the cases implicitly defines such control. Cases not only operationalize the model to be executable but also improve the capability of the operationalized model incrementally. Accumulation of cases that successfully achieved the problem goals creates the implicit model of how domain experts solve the problem and make high quality solutions. Accumulation of cases that failed in solving the problem extends the model inductively with useful search control knowledge for avoiding similar failures, which may not be recognized even by the experts. Thus, accumulation of cases can progressively enhance the problem solving capability in terms of both solution quality and problem solving efficiency. The experimental results by CABINS in Chapter 6 validate the effectiveness of the above approach in job shop scheduling problems.

2.2 Case Representation

Corresponding to the task structure in Fig. 2.1, CABINS has three classes of classification decisions: goal selection, repair selection and evaluation. In CABINS, repair methods are further divided into strategies and tactics. A repair goal is derived from a particular high level description of defects in the current solution of a problem and their significance to a user. A way to achieve a particular repair goal is designated by selection of one of the associated repair strategies. Each repair strategy is executed by a successive application of a variety of repair tactics associated with it. A result of a tactic application is evaluated to check whether it is acceptable or not. In CABINS' repair process, all decisions (i.e. goal/strategy/tactic selection and result evaluation) are made using case-based reasoning. Hence, the content of a case must be able to represent all the decision criteria of human experts in the repair process.

A case in CABINS describes the application of a particular repair action to the problem. CABINS has three general types of cases corresponding to three hierarchical classes of repair actions: *goal_case*, *strategy_case* and *tactic_case*. Each case type is delineated with *descriptive features*, which are heuristic approximations that reflect problem space characteristics, and a *repair history*, which records the sequence of applications of successive repair actions, the repair effects and the repair outcome. Each case type has the different sets of the categories that are derived from task-level analysis and characterize the features to be described in the case. The hierarchy of repair actions and the categorization of case features give strong semantics for helping a user of CABINS understand and organize her/his expertise to be represented in a case. As a result, domain experts can easily define their own specific features and repair actions, which can be implemented by a programmer without knowledge of CABINS' internal structure and process.

Feature categories in CABINS are derived from the task structure of the optimization task (see Section 2.1). The task structure analysis has identified the following feature categories for each type of case:

- In a goal_case, information necessary for selecting a repair goal is stored. From a domain-independent point of view, the repair goal is selected by identifying the most critical defect of a current problem taking into consideration the problem context. Accordingly, the goal_case has two categories of features: *Quality* and *Situation*. In Quality category, a user can define the features which constitute the evaluation of the current problem and whose unsatisfactory values cause defects. In Situation category, a user describes external factors which could influence the evaluation of the current problem. A repair action recorded in the goal_case is the repair goal. The repair goal is to evaluate the current problem from one aspect of the objectives and sort repair targets according to the evaluation. A user's expertise captured in the goal_case is that of detecting the defects of a solution in a given context (i.e. quality and situation).

- A strategy_case records the information necessary to select a repair strategy. Since the repair strategy is selected based upon global characteristics of a repair target, the strategy_case has a single category for features: *Global-context*. Features in Global-context represent potential repair flexibility of the repair target as a whole. A repair action stored in the strategy_case is the repair strategy. The repair strategy is to determine the degree of possible change by a repair and sort changeable components of the repair target appropriately in the order of tactic applications so that ripple effects of tactic applications are minimized. The strategy_case captures user's knowledge about the tradeoffs between possible effects and allowable global disruptions for repairing the target.

- A tactic_case represents the information required for selecting a repair tactic. The repair tactic is selected based on local characteristics of each component of a current repair target. Features in the tactic_case belong to the category of *Local-context*. Local-context features reflect flexibility for revision of a repair target component within limited bounds allowed by a repair strategy. A repair action stored in the tactic_case is the repair tactic that executes revision on components of a repair target.

  Another important piece of information stored in the tactic_case is evaluation of a repair result. Features used for evaluating a repair result belong to *Repair-effect* category. Since a repair tactic is the only repair action that can make actual changes on the repair targets, the effects of the changes are evaluated by a domain expert and stored in the tactic_case with the evaluation. Thus, the tactic_case captures a user's knowledge about (a) prediction of local effects by a possible tactic application, and (b) the tradeoffs between favorable and unfavorable repair effects (both in local and global perspectives).

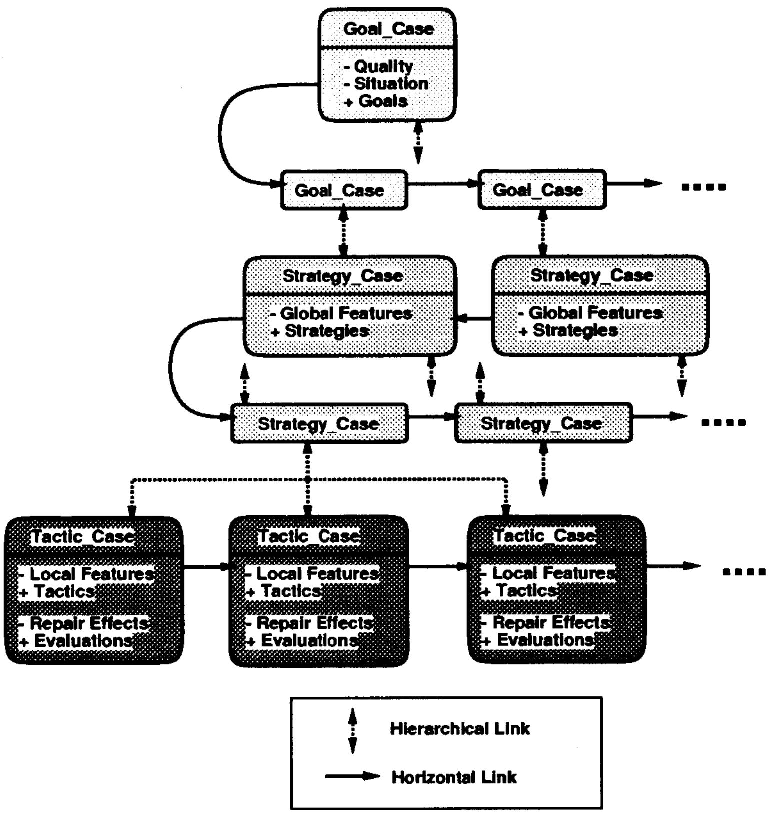

Fig. 2.3: CABINS case memory structure

Fig. 2.3 depicts the case memory structure of CABINS. Cases are connected with each other by the links of the two different types. By the hierarchical link, a case is connected with its upper case that derived the case, and also with its lower case(s) that was(were) derived from the case. Thus, goal_cases, strategy_cases and tactic_cases are connected with each other. Consequently the entire (or a portion of) repair process structure can be easily re-constructed from the case base. And the cases of the same type are connected by the horizontal link. As a result, the case with the closest match to the current problem can be retrieved efficiently. Goal_cases, strategy_cases and tactic_cases are subparts of the episode of repairing one repair target, which is again a subpart of the entire episode of repairing the current problem. The distributed representation and memory structure of the cases make it possible for CBR process to notice the similarity between parts of the problems, create generalization about the parts of the problems and use this generalization to make predictions in new unknown situations sharing some parts.

2.2.1 Attributes for a Case Feature

As will be shown in the following examples, each feature in CABINS' case is described by a set of attributes such as `Feature`, `Value`, `Filtering`, `Importance` and `Similarity`. `Feature` attribute specifies a name of the feature to be considered. `Value` attribute records a value of the feature. Values of `Filtering`, `Importance` and `Similarity` attributes are used for case retrieval purpose (Section 3.1) and assigned their values subjectively by a user during case acquisition (Section 2.3).

If a value of `Filtering` attribute in a feature is "ON", exact match of the feature is required for the case to be retrieved. If the value is "OFF", partial match can contribute to the case retrieval. A value of `Importance` attribute designates importance of the feature in a particular case. Thus, `Filtering` and `Importance` attributes allow the user to express the uniqueness of the feature in a particular case in different ways. These attributes can be used to represent an exceptional case in which a value of the feature in the case has a special meaning to the user.

`Similarity` attribute denotes the function used to calculate feature similarity. Current implementation of CABINS has three similarity functions: "Normal", "HighCut" and "LowCut". Details of these functions are described in Section 3.1.

2.2.2 Cases for Schedule Repair Task

Fig. 2.4 presents a schematic diagram showing the abstracted content of CABINS case base. The content is derived from the transformation of the task structure analysis results on the optimization task to the schedule problem. In the scheduling problem, an order in a schedule is selected as a repair target for a strategy application. For a tactic application, each activity in the selected order is designated as a repair target. Hence, features of an

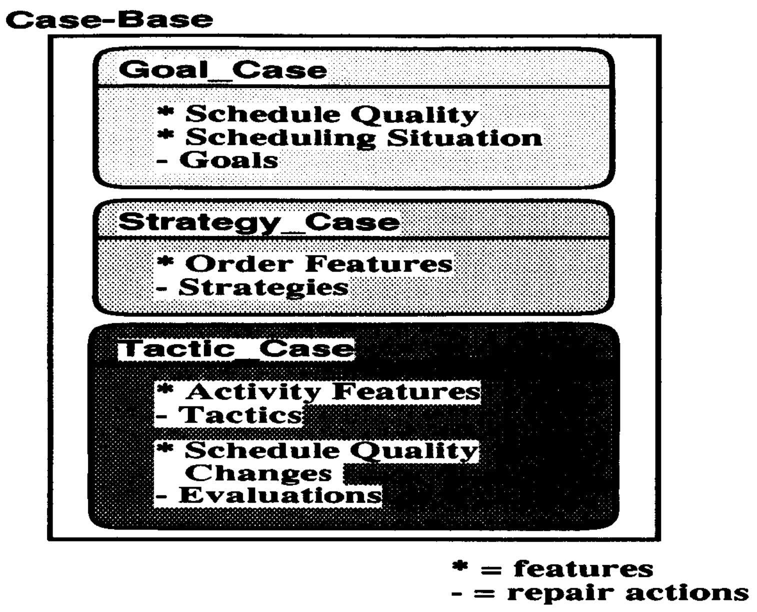

Fig. 2.4: Case base for schedule repair task

order are stored as global-context in strategy cases, and activity features are recorded as local-context in tactic cases. Also in tactic cases, schedule quality changes are recorded as repair-effect with the result evaluation by domain experts. This section explains the case features for the schedule repair task in each type of the case. The details of repair actions for schedule repair will be presented in Section 3.2.1.

Fig. 2.5 shows an example of a goal_case used in the experiments of this thesis. In the example, features belonging to Schedule_Quality are "weighted_tardiness" and "in-process_inventory", since they are major concerns in the experiments. For features in Scheduling_Situation, the case has "year", "month" and "economy" (i.e. boom or depression), which are considered to affect the judgment on the appropriateness of the current schedule. For example, "month" may imply several seasonal factors which influence the production planning such as more strict due-date requirements that are widely observed at the end of a year. The case shows that "reduce_inprocess_inventory" is selected as a critical goal and successfully achieved after a failure of attaining "reduce_weighted_tardiness" goal.

Fig. 2.6 is an example of a strategy_case in CABINS. In the example, Order_Features are composed of five features describing Global-context of an order. "Slack_ratio" is, for example, the total waiting time divided by the length of an allowable time window for completion of the order (i.e. from its release_date to due_date). High "slack_ratio" often shows a loose schedule with much repair flexibility. "Resource_busy_deviation" is the standard deviation of utilization of all the resources that activities of the order can be assigned to. High "resource_busy_deviation" indicates the presence of highly contended-for resources (bottleneck resources) which in turn makes repair less flexible. And the example case

```
G_Case {
  Name = "exp-data/exp_0_0_5:G1";
  Schedule_Quality = (
    Slot {                                 Slot {
      Feature = weighted_tardiness;          Feature = inprocess_inventory;
      Value = 2370;                          Value = 7270;
      Filtering = OFF;                       Filtering = OFF;
      Importance = 1.0;                      Importance = 1.0;
      Similarity = NORMAL;                   Similarity = NORMAL;
    }                                      }
  )
  Scheduling_Situation = (
    Slot {                                 Slot {
      Feature = year;                        Feature = month;
      Value = 1993;                          Value = 10;
      Filtering = OFF;                       Filtering = OFF;
      Importance = 1.0;                      Importance = 1.0;
      Similarity = NORMAL;                   Similarity = NORMAL;
    }                                      }
    Slot {
      Feature = economy;
      Value = VERY_BAD;
      Filtering = ON;
      Importance = 1.0;
      Similarity = NORMAL;
    }
  )
  Goals = (
    G_Solution {
      Goal = reduce_weighted_tardiness;
      Result = FAILED;
    }
    G_Solution {
      Goal = reduce_inprocess_inventory;
      Result = SUCCEEDED;
    }
  )
}
```

Fig. 2.5: An example of CABINS' goal_case for a schedule repair problem

```
S_Case {
  Name = "exp_0_1_1:G34:GS2:O8";
  Goal = reduce_inprocess_inventory;
  Order_Features = (
    Slot {                                 Slot {
      Feature = slack_ratio;                 Feature = tardiness_ratio;
      Value = 0.73529;                       Value = 0.0;
      Filtering = OFF;                       Filtering = OFF;
      Importance = 1.0;                      Importance = 1.0;
      Similarity = NORMAL;                   Similarity = NORMAL;
    }                                      }
    Slot {                                 Slot {
      Feature = inventory_ratio;             Feature = resource_idle_ratio;
      Value = 0.125;                         Value = 0.247059;
      Filtering = OFF;                       Filtering = OFF;
      Importance = 1.0;                      Importance = 1.0;
      Similarity = NORMAL;                   Similarity = NORMAL;
    }                                      }
    Slot {
      Feature = resource_busy_deviation;
      Value = 2.387823;
      Filtering = OFF;
      Importance = 1.0;
      Similarity = NORMAL;
    }
  )
  Strategies = (
    S_Solution {
      Strategy = shift_right_all;
      Result = SUCCEEDED;
    }
  )
}
```

Fig. 2.6: An example of CABINS' strategy_case for a schedule repair problem

records that "shift_right_all" strategy, which means to move *all the activities* of the order to the right on the timeline, is selected as a repair strategy and succeeded in achieving a goal.

Fig. 2.7 is an example of a tactic_case in CABINS. `Activity_Features` includes the features of an activity within limited bound. In particular, the bound that CABINS uses is a time interval called *repair time horizon.* The repair time horizon of the activity is the time interval between the end of the activity preceding the activity in the same order and the start of the activity succeeding the activity in the same order (see Fig. 2.8).

Associated with the repair time horizon are the features which potentially are predictive of the effectiveness of applying a particular repair tactic. These features are in the same spirit as those utilized in [Ow, Smith, & Thiriez, 1988]. "Left_slack_ratio" and "right_slack_ratio" roughly estimate the flexibility of the activity in its time horizon *without* considering resource contention. And "alternative_resources" shows the number of alternative resources to which the activity can be assigned. The other features in `Activity_Features` predict how much overall gain will be achieved by applying a corresponding repair tactic to the activity in its time horizon. For example, "imm_left_idle_ratio" predicts the possible effects of applying "slide_left" tactic to the activity.

In the example case, "jump_left" tactic, which moves the activity on the same resource as much to the left on the timeline as possible within the repair time horizon, is applied and the effects of the repair are recorded in the features of `Schedule_Quality_Changes`. Features in `Schedule_Quality_Changes` describe the impacts of a repair action application on schedule optimization objectives (e.g. tardiness, inventory). Typically these effects reflect a diverse set of objectives to be considered and heavily related to `Schedule_Quality` features in a goal_case. To be noted is that there are two perspectives in recording these effects. One is the local perspective that describes the effects that occurred to the activity. The other is the global perspective, which represents the effects of a tactic application in the overall schedule. Since the effects caused by a tactic application are not determined until a final tactic in the strategy is applied, a human expert has to guess the acceptability of a tactic application result by considering the trade-off of global and local repair effects. As a result, the repair effects in both perspectives need to be recorded

`Result` is the evaluation assigned to the set of effects of a repair action and takes a value in the set ["SUCCEEDED", "FAILED"]. This judgment of a repair outcome must be made by a domain expert in the training phase and gets recorded in the case base. An outcome is "SUCCEEDED" if the tradeoff involved in the set of effects for the current application of a repair action is judged acceptable. `Effect` is approximation of the aggregated effects by a repair and is determined subjectively by the expert. This value might be given in terms of a numeric as shown in the example, or using a user-defined ordinal category (e.g. ["VERY GOOD", "GOOD", "OK", "BAD", "VERY BAD"]). When rational value assignments of `Effect` are given to the cases, CABINS can retrieve more effective cases for selecting repair actions (see Section 4.3.2).

```
T_Case {
   Name = "exp_2_1_9:G16:GS1:O1:OS1:A2";
   Goal = reduce_weighted_tardiness;
   Strategy = shift_left_all;
   Final = NO;
   Activity_Features = (
      Slot {                                    Slot {
         Feature = left_slack_ratio;               Feature = right_slack_ratio;
         Value = 13.571429;                        Value = 0.0;
         Filtering = OFF;                          Filtering = OFF;
         Importance = 1.0;                         Importance = 1.0;
         Similarity = NORMAL; }                    Similarity = NORMAL; }
      Slot {                                   Slot {
         Feature = imm_left_idle_ratio;           Feature = imm_right_idle_ratio;
         Value = 25.714286;                       Value = 1885.714286;
         Filtering = OFF;                         Filtering = OFF;
         Importance = 1.0;                        Importance = 1.0;
         Similarity = NORMAL; }                   Similarity = NORMAL; }
      Slot {                                   Slot {
         Feature = aggr_left_idle_ratio;         Feature = aggr_right_idle_ratio;
         Value = 6242.857143;                    Value = 0.0;
         Filtering = OFF;                        Filtering = OFF;
         Importance = 1.0;                       Importance = 1.0;
         Similarity = NORMAL; }                  Similarity = NORMAL; }
      Slot {                                  Slot {
         Feature = left_swappability;           Feature = right_swappability;
         Value = 150.769231;                    Value = 0.0;
         Filtering = OFF;                       Filtering = OFF;
         Importance = 1.0;                      Importance = 1.0;
         Similarity = NORMAL; }                 Similarity = NORMAL; }
       Slot {                                 Slot {
         Feature = left_alt_idle_ratio;         Feature = right_alt_idle_ratio;
         Value = 271.428571;                    Value = 0.0;
         Filtering = OFF;                       Filtering = OFF;
         Importance = 1.0;                      Importance = 1.0;
         Simialrity = NORMAL; }                 Similarity = NORMAL; }
       Slot {                                 Slot {
          Feature = left_alt_swappability;      Feature = right_alt_swappability;
          Value = 0.0;                          Value = 0.0;
          Filtering = OFF;                      Filtering = OFF;
          Importance = 1.0;                     Importance = 1.0;
          Simialrity = NORMAL; }                Similarity = NORMAL; }
        Slot {
          Feature = alternative_resource;
          Value = 3;
          Filtering = ON;
          Importance = 1.0;
          Simialrity = NORMAL; }
   )
   Tactics = (
      T_Solution = {
         Tactic = jump_left;
         Schedule_Quality_Changes = (
            Slot {                                   Slot {
               Feature = local_weighted_tardiness;      Feature = local_inprocess_inventory;
               Value = 280.0;                           Value = -950.0;
               Filtering = OFF;                         Filtering = OFF;
               Importance = 1.0;                        Importance = 1.0;
               Similarity = NORMAL; }                   Similarity = NORMAL; }
            Slot {                                   Slot {
               Feature = global_weighted_tardiness;     Feature = global_inprocess_inventory;
               Value = 0.0;                             Value = 0.0;
               Filtering = OFF;                         Filtering = OFF;
               Importance = 1.0;                        Importance = 1.0;
               Similarity = NORMAL; }                   Similarity = NORMAL; }
         )
         Effect = 280.0;
         Result = SUCCEEDED; }
   )
}
```

Fig. 2.7: An example of CABINS' tactic_case for a schedule repair problem

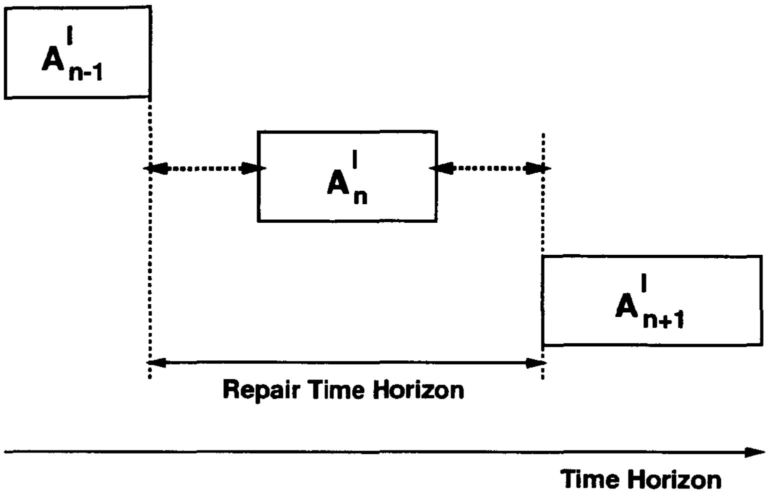

Fig. 2.8: Repair time horizon of the activity (A^l_n)

2.3 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 the problems by hand. The interactions and the results are stored in cases with the information of the problem contexts. Through the case acquisition, the user can operationalize the model of solving schedule optimization problems.

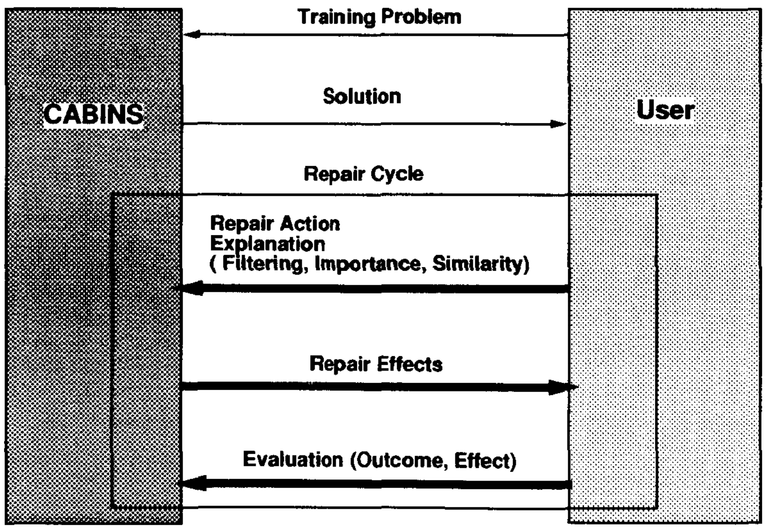

Fig. 2.9: Interactions between CABINS and a user for case acquisition

Fig. 2.9 shows interactions between CABINS and the user for case acquisition. In iteratively repairing a solution of a training problem, the user has to select the repair action that is deemed to be appropriate in a given particular problem situation, apply it to the current, and evaluate the result repeatedly. User's decisions in the course of a repair along with the problem context are recorded in a case. In the selection of the above repair actions (i.e. goal, strategy and tactic), the user can assign values to the attributes of case features, such as `Filtering`, `Importance` and `Similarity` as explanations to his/her decisions.

2.3.1 Case Acquisition for Schedule Repair Task

In a schedule repair task, the user first selects the most urgent repair goal for a given schedule from the list of user defined goals to be achieved in the schedule. User's selection of a repair goal, overall quality of the current schedule and situations influencing user's scheduling decisions are recorded in a goal_case. An application of the repair goal to the current schedule produces the sorted list of orders according to significance of the defect in the given repair goal. Then, the user selects a repair strategy from a set of user defined repair strategies for repairing an order according to the sorting. User's selection of the repair strategy and global characteristics of the order constitute the content of a strategy_case. When the repair strategy is applied to the order, some activities of the order are picked up and sorted so as to avoid unnecessary computations and unbounded ripple effects. Finally, the user selects a repair tactic from a set of user defined repair tactics for repairing an activity in the sorted queue.

A repair tactic application causes changes in the schedule by executing an repair by applying constraint propagation to re-schedule the activities. The repair tactic application may result in an infeasible schedule. An infeasible schedule will occur when constraints are propagated beyond the fixed time window boundary of any order. The details of the constraint propagation techniques will be described in Section 3.2.1.

If the outcome of the repair tactic application is feasible, the effects of the repair are calculated and shown to the user. An effect describes the result of the repair with respect to one of the repair objectives defined as `Schedule_Quality_Changes` features in a tactic_case. On account of tight constraint interactions, these effects are ubiquitous in job shop scheduling and make schedule optimization extremely hard. When the application of a repair tactic produces a feasible result, the user must determine whether the resultant schedule is acceptable or not based upon the calculated effects. The outcome is judged as unacceptable, if the schedule resulting from the application of the revision heuristic is feasible but the revision result does not make any improvement with respect to the user's criteria. This could happen because harmful effects might outweigh, in the user's judgment, the effected improvement. For example, if reduction of an order tardiness enforces increased utilization of low-quality machines, total cost incurred by this repair might eventually be increased, but not decreased for the user who would dislike the possible low quality of products. Therefore such a repair might be judged as unacceptable by the user. The user's judgment as to balancing favorable and unfavorable effects related to a particular optimization objective constitutes the explanations of the repair outcome. The user can supply a supplemental explanation of the judgment by assigning a value to `Effect` attribute of a repair tactic description in a tactic_case. This gives a case the additional information about to what extent the case is acceptable or unacceptable and this information might be utilized to retrieve a more effective case for selecting repair tactics. At the end of each repair tactic application, 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 activity features.

The repair process continues until an acceptable outcome is reached, or failure is declared. Failure is declared when there is no more repair action available. The sequence of applications of successive repair actions, the effects and user's evaluation of the results are recorded in the case. In this way, a number of cases are accumulated in the case base.

2.4 Summary

This chapter has presented the detailed methodology for creating a case base in the CABINS system. First, knowledge-level task analysis method, task structure analysis, has been adopted. Then, CBR is used as a model extending method, which comprises of transformation and enhancement processes. The detailed case descriptions for schedule optimization problems in CABINS are derived from the analysis of the optimization task.

And, the chapter has shown the case acquisition protocol between CABINS and a user. The user's decisions and explanations in the course of problem solving are collectively stored as cases in CABINS' case base.

Chapter 3

Utilizing Cases

CABINS has three classes of repair actions (i.e. goal setting, and strategy/tactic applications), each corresponding to the different level of decision makings in the repair-based optimization method.

Fig. 3.1 shows the case-based repair process of CABINS for the job shop scheduling problems, which has the following basic steps:

1. Suboptimalities in a schedule are identified and one of them is recognized to be most urgent (i.e. a repair goal) based upon past experiences using CBR.

2. If the selected repair goal is to "give-up", CABINS terminates.

3. Defective orders are sorted (in decreasing sequence) according to the degree of the defect in the repair goal and are repaired based upon this sorting.

4. The defective order under current repair consideration is called the *focal_order* and a repair strategy for the current focal_order is selected using CBR.

5. If the selected repair strategy is to "give-up", next order in the sorted defective order queue is selected as a focal_order for further repair.

6. Activities in the focal_order are selected and sorted by the repair strategy so as to (a) avoid unnecessary computations, and (b) limit the amount of ripple effects (schedule disruption) that could be caused by moving activities that are too tightly scheduled and whose move would cause many constraint violations. Activities are repaired one by one according to this ordering.

7. The activity under current repair consideration is called the *focal_activity.* A repair tactic is selected for the current focal_activity using CBR.

8. If the selected repair tactic is to "give-up", next activity in the sorted activity queue is selected as a focal_activity for further repair.

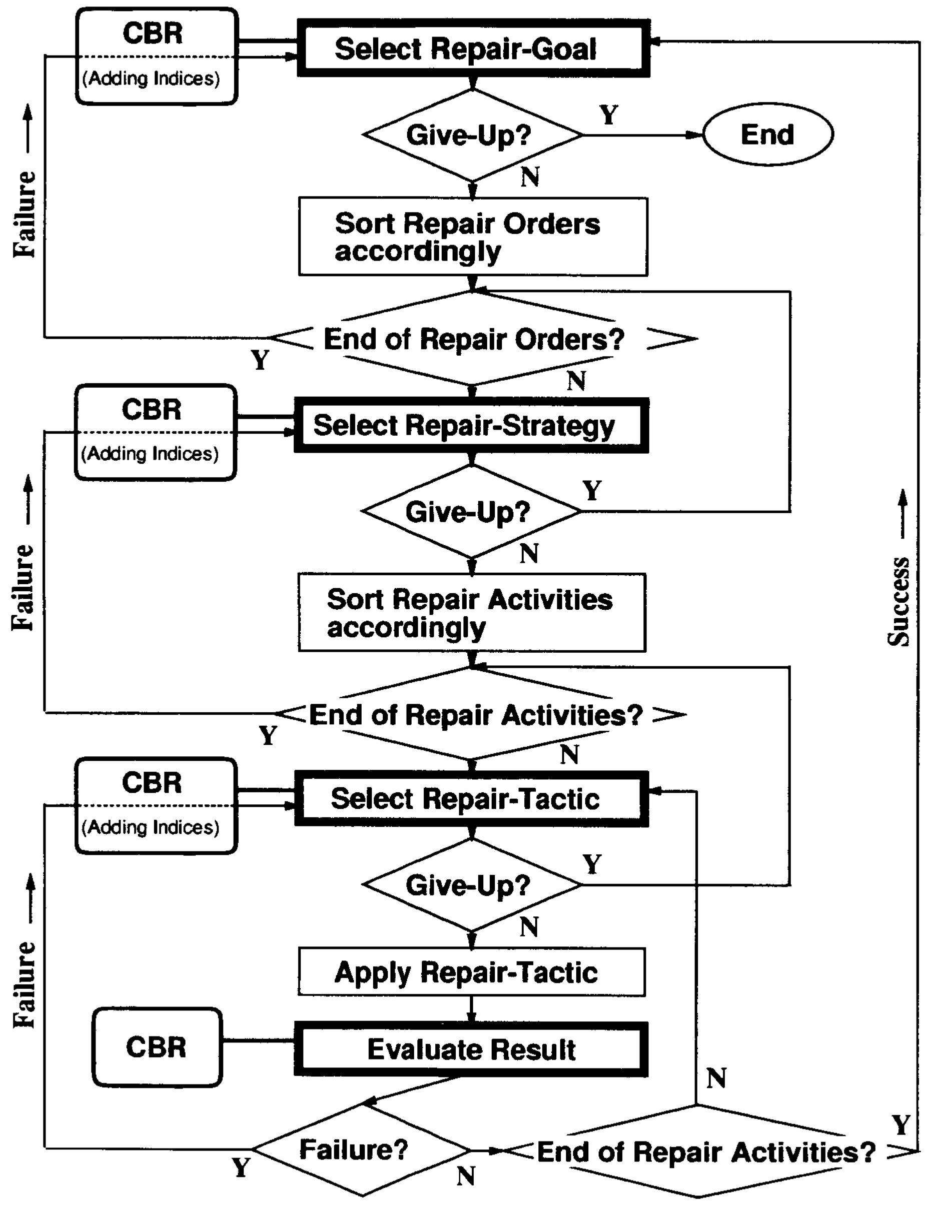

Fig. 3.1: CABINS repair-based optimization procedure

9. The repair tactic is applied to the focal_activity as follows: (a) changing the assignment of the focal_activity as specified by the selected repair tactic, and (b) resolving constraint violations caused in (a) by means of the constraint propagation mechanism in CABINS.

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

11. According to the repair outcome, CABINS proceeds to one of the following steps.

    (a) If a repair of the current focal_activity is evaluated as a success, a next activity in the sorted activity queue is selected as a next repair target. If no activity is left in the queue, it means that the focal_order is successfully repaired. If a repair of the current focal_order is successfully done, a next repair goal is selected even if there still remain other orders to be repaired in terms of the current repair goal. This is because the effects of repairing one focal_order may change the nature of the problem drastically, so that it may be possible to take advantage of the opportunities for repairing more critical defects which were difficult before.

    (b) If an application of a repair tactic is evaluated as a failure, or accomplishment of a selected repair goal/strategy is given up for all the repair targets, the selection of such goal/strategy/tactic is deemed as a failure. When a failure is detected, CBR process selects a next repair action *using information of a failure as additional indices.* This CBR invocation retrieves similar past failures of the repair action that were successfully repaired and the repair action that was eventually successful in fixing the past failure. The assumption here is that a similar outcome for the same repair action implies similarity of causal structure between the past and current case. Thus, the eventually successful repair action of a similar failure can potentially be successful in the current problem.

A more detailed trace of CABINS' repair process in the job shop scheduling problem will be described in Section 3.2.

3.1 Case Retrieval

In CABINS, all the selections of repair actions and the evaluations of repair results are accomplished by the case-based classification method, which retrieves the past similar cases in the case base to the current problem situation (i.e. in other words, classifies the current problem into the class of past similar cases) and re-uses the past decisions made in the cases. As a case retrieval mechanism, CABINS uses a *k-Nearest Neighbor* method (k-NN) [Dasarathy, 1990]. A k-NN finds the k-nearest neighbors, where k is some constant, of the current problem from the training data based on the pre-determined matching measures, and in its simplest form, the single nearest neighbor is found and chosen as a classification result.

In domains, such as scheduling that do not have clear predictive features owing to lack of the causal structure, there can be many matches other than the nearest match that can potentially contribute to accurate classification. For example, if a large number of near neighbor cases are of the same category, a higher confidence can be given to the classification result than if the near neighbors are of many different categories. For example, in determining the repair tactic to be applied to the current problem, suppose that five nearest neighbors are selected. Three of them are of "jump_left" cases, whose matching value is 0.9, 0.2 and 0.1 and the other two are of "swap_left" cases, whose matching value is 0.8 and 0.75. In 1-NN method, "jump_left" is selected as a repair tactic because the nearest retrieved case uses "jump_left" as a successful modification heuristic. In this method, the cases with relatively high matching value are ignored in the classification. Hence, the author used five nearest neighbors empirically in the current implementation of CABINS. The author used the sum of the matching value in k-nearest neighbors as a selection criterion, instead of using the frequency of appearance of a class among k-nearest neighbors, to avoid that dissimilar cases may have an undue influence on the classification result. In the previous example, "swap_left" is selected as a repair tactic by CABINS (since its total matching value is 1.55 vs. 1.2 of "jump_left"). The similar methods have been successfully applied in the different domains without clear causal structure such as English word pronunciation and text classification in [Stanfill & Waltz, 1986; Creecy *et al.*, 1992].

3.1.1 Similarity Functions

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

$$\begin{aligned}
Distance^i_j &= (Importance^i_j \times DisFunc^i_j(CF^i_j, PF_j))^2 \\
Similarity_i &= (1.0 - \sqrt{\frac{\sum_{j=1}^{N} Distance^i_j}{N}})^\alpha \\
Match_i &= \begin{cases} Similarity_i \times Effect_i & \text{(for repair action selection)} \\ Similarity_i & \text{(for result evaluation)} \end{cases}
\end{aligned}$$

where $Distance^i_j$ is dissimilarity of the j-th feature between the i-th case and the current problem. $Importance^i_j$ is the value of `Importance` attribute of the j-th feature of the i-th case in the case base, CF^i_j is the value of `Value` attribute of the j-th feature in the i-th case, PF_j is the value of `Value` attribute of the j-th feature in the current problem, $DisFunc^i_j$ is the distance function for the j-th feature of the i-th case, stored as the value of Similarity attribute. *Similarity*$_i$ is similarity between the i-th case and the current problem. α is a parameter to balance the tradeoff between case's similarity and its possible effect in calculating a matching value. And *Match*$_i$ is the matching value of the i-th case to the current problem. *Effect*$_i$ is the value of Effect attribute of the i-th case. For goal/strategy selection, the Effect value of last applied tactic_case associated with the goal_case/strategy_case is used for calculation.

For selecting repair actions, the author takes into consideration not only the similarity but also the past repair effect of the retrieved case. This reflects the general user's preference to the more effective repair actions. As an example of a repair tactic selection, if a case that used "jump_left" tactic and another case with "swap_left" tactic are considered equally similar to the current problem, a user prefers the case with "jump_left" tactic because "swap_left tactic", which moves back a swapped activity, usually gains less effects than "jump_left" tactic due to its possible side-effects. A parameter α is used to control sensitivity of a similarity factor to the matching value calculation. A larger value of α means that the similarity factor has more influence to the matching value when there are some similar cases to the problem, but it has less influence when no similar enough case is available and instead the possible effect of the case becomes a major concern. In the current implementation of CABINS, the author heuristically fixed its value as 2.0. See the experiment results in Section 4.3.2 for the analysis on the effectiveness of this case retrieval scheme.

As for a distance function, the current implementation of CABINS has the following three types:

$$\begin{aligned}
Normal(CF^i_j, PF_j) &= Sal^i_j \times |CF^i_j - PF_j|/D_j \\
HighCut(CF^i_j, PF_j) &= Sal^i_j \times \max(0.0, (CF^i_j - PF_j)/D_j) \\
LowCut(CF^i_j, PF_j) &= Sal^i_j \times \max(0.0, (PF^i_j - CF_j)/D_j)
\end{aligned}$$

where D_j is a normalization factor for the j-th feature, so that a distance function yields a value in [0.0, 1.0]. Sal^i_j is the salience value of the j-th feature for the repair action used in the i-th case, which takes a value in the range [0.0, 1.0]. This value represents feature relevance to each repair action. For example, a value of "left_slack_ratio" has significant influence in selecting "jump_left" repair tactic, but not in selecting "jump_right" repair tactic because "jump_right" tactic can be safely applied regardless of the value of "left_slack_ratio". A value of Sal^i_j can be assigned subjectively by a user a priori, or calculated after enough number of cases are accumulated with the help of the statistical algorithms such as Relief [Kira & Rendell, 1992] and IBL4 [Aha, 1992]. See the empirical results in Section 4.3.1 for the discussion on the usefulness of this salience value.

"Normal" is a weighted normalized Euclidean distance function, which is the most popularly used distance measure in the research of classification. "HighCut" and "LowCut" distance functions are used for a case feature whose value is, a domain expert thinks, big/small enough to classify the case into a certain class, so that the problem feature with the greater/smaller value is equally considered as a perfect match to the case feature.

3.2 Repair by CABINS

Once CABINS has constructed a case base from training data, CABINS can carry out all the necessary works for repair without any interaction with its user. Invocation of CBR suggests a repair action to be applied. Case features are used as indices to retrieve past repair episodes in selecting repair actions (i.e. goal, strategy and tactic). These repair actions can be defined based upon the particular needs in the domain.

After a repair tactic has been applied to the problem and if the result is feasible, repair evaluation is performed. Unlike the traditional repair-based approaches (e.g. simulated annealing, tabu search) where the effects of revision are evaluated in terms of the explicitly defined cost function, CABINS evaluates the result using CBR, thus obviating the need for the presence of an explicit cost function. Using the repair effects as indices, CBR is invoked and returns an outcome in the set ["SUCCEEDED", "FAILED"].

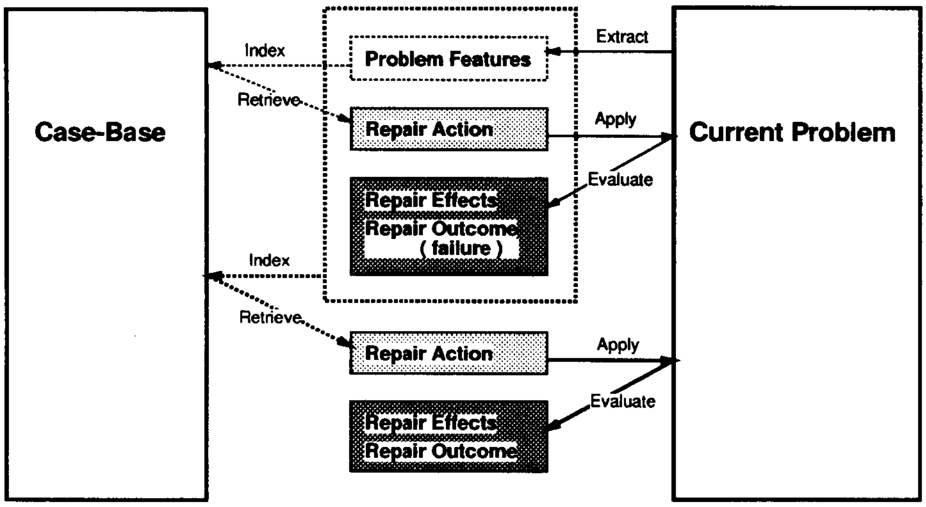

Fig. 3.2: Failure recovery by CABINS using failure information as additional case indices

If the result is acceptable ("SUCCEEDED"), then CABINS proceeds to repair another target. But, as shown in Fig. 3.2, if the outcome of current revision is determined as either infeasible or unacceptable ("FAILED"), CABINS performs another CBR invocation using as indices the conjunction of the current repair effects and outcome (infeasible or unacceptable), the failed repair heuristic, and the case 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 [Sycara, 1988; Simpson, 1985], and allows the problem solver to access past solutions to the failure.

The following sections explain the sets of repair actions that are utilized in the experiments reported in this thesis and present an example session of CABINS repair process in a job shop scheduling problem.

3.2.1 Schedule Repair Actions

The repair goal is selected to focus the attention of repair to the most critical defect in the current schedule. The repair goals used in the experiments are:

reduce_weighted_tardiness : try to reduce the weighted tardiness of the current schedule.

reduce_inprocess_inventory : try to reduce the inprocess inventory (WIP) of the current schedule.

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

When "reduce_weighted_tardiness" is selected as a repair goal, all the tardy orders in the schedule are sorted in the descending order of their weighted tardiness value. When "reduce_inprocess_inventory" is selected, all the orders in the schedule are sorted in the descending order of their WIP value. When "give_up" is selected as a repair goal, CABINS repair process is terminated.

The first order of the sorted order list becomes a current focal_order, and the repair strategy is selected to repair the focal_order according to the selected repair goal. The repair strategies used in the experiments are:

shift_left_all : try to move *all the activities* of the current focal_order to the left on the timeline.

shift_right_all : try to move *all the activities* of the current focal_order to the right on the timeline.

shift_left_least : try to move *the least number of the current focal_order's activities enough to attain a goal* to the left on the timeline.

shift_right_least : try to move *the least number of the current focal_order's activities enough to attain a goal* to the right on the timeline.

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

"Shift_left_all" strategy sorts all the activities of the focal_order in the descending order of their start time. "Shift_right_all" strategy sorts all the activities of the focal_order in the ascending order of their start time. "Shift_left_least" strategy picks up the activity of the focal_order sequentially from its last activity to its first and adds its waiting time until the added waiting time exceeds the desired value designated by the repair goal, and sorts these activities in the descending order of their start time. "Shift_right_least" strategy picks up the activity of the focal_order sequentially from its first activity to its last and adds its succeeding activity's waiting time until the added waiting time exceeds the desired value, and sorts these activities in the ascending order of their start time. When "give_up" is selected, there are two possible actions to be taken: (1) if there remain other orders in the list sorted by a repair goal, a repair strategy selection is tried to the next order in the sorted order list, and (2) if no order remains untried in the sorted order list, another repair goal is to be selected.

Repair by CABINS is tried to one activity after another in the sorted activity list created by a repair strategy. The first activity in the list is a current focal_activity. Repair of the focal_activity is performed by applying a repair tactic. The tactics currently available in CABINS are:

slide_left : try to move the 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.

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

jump_alt_left : try to move the 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_left : swap the focal_activity with the activity on its left on the *same resource* within the repair time horizon that causes the least disruptions.

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

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

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

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

swap_right : swap the focal_activity with the activity on its right on the *same resource* within the repair time horizon that causes the least disruptions.

swap_alt_right : swap the focal_activity with the activity on its right 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.

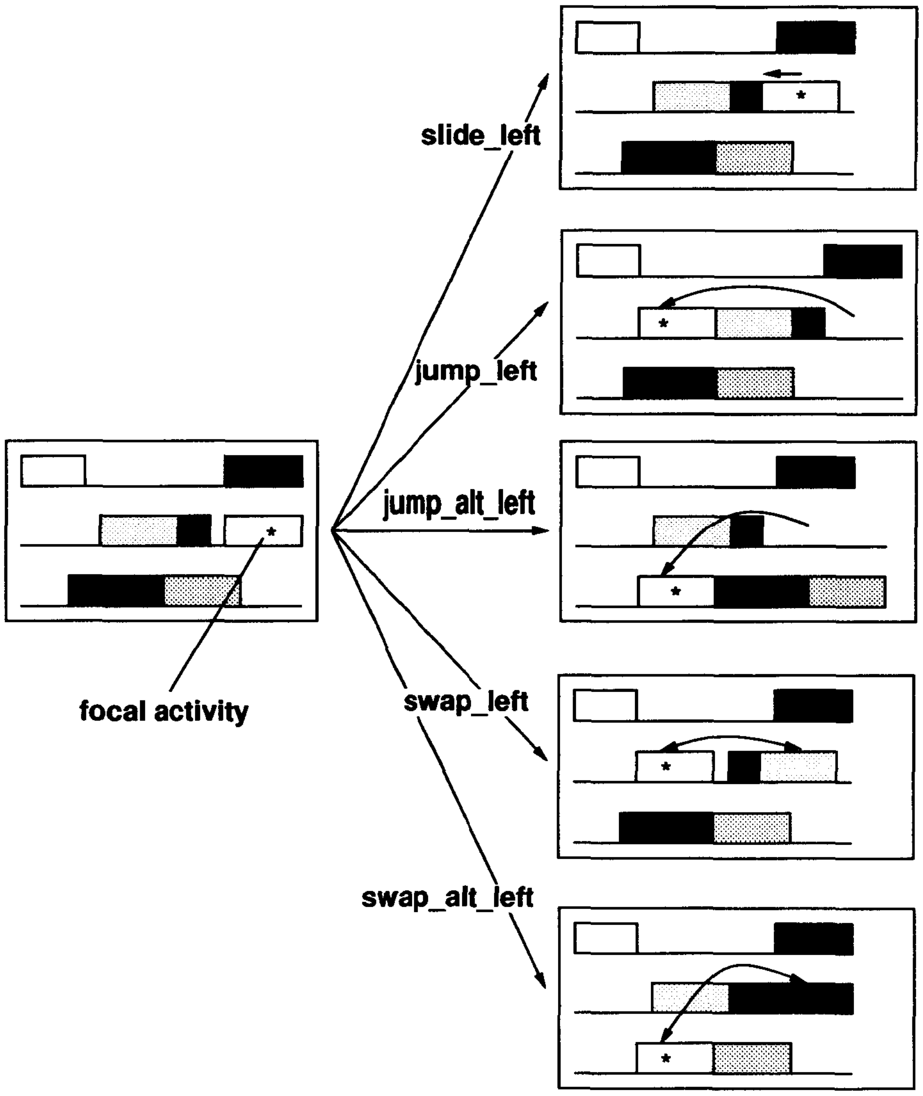

Fig. 3.3: Schedule repair tactics in CABINS (1)

Fig. 3.3 and Fig. 3.4 show the simple pictorial explanations of the above repair tactics. When "give_up" is selected as a repair tactic, the next activity in the sorted list, if any, is selected as a next repair target. If such an activity does not exist, another repair strategy is selected for repairing the focal_order.

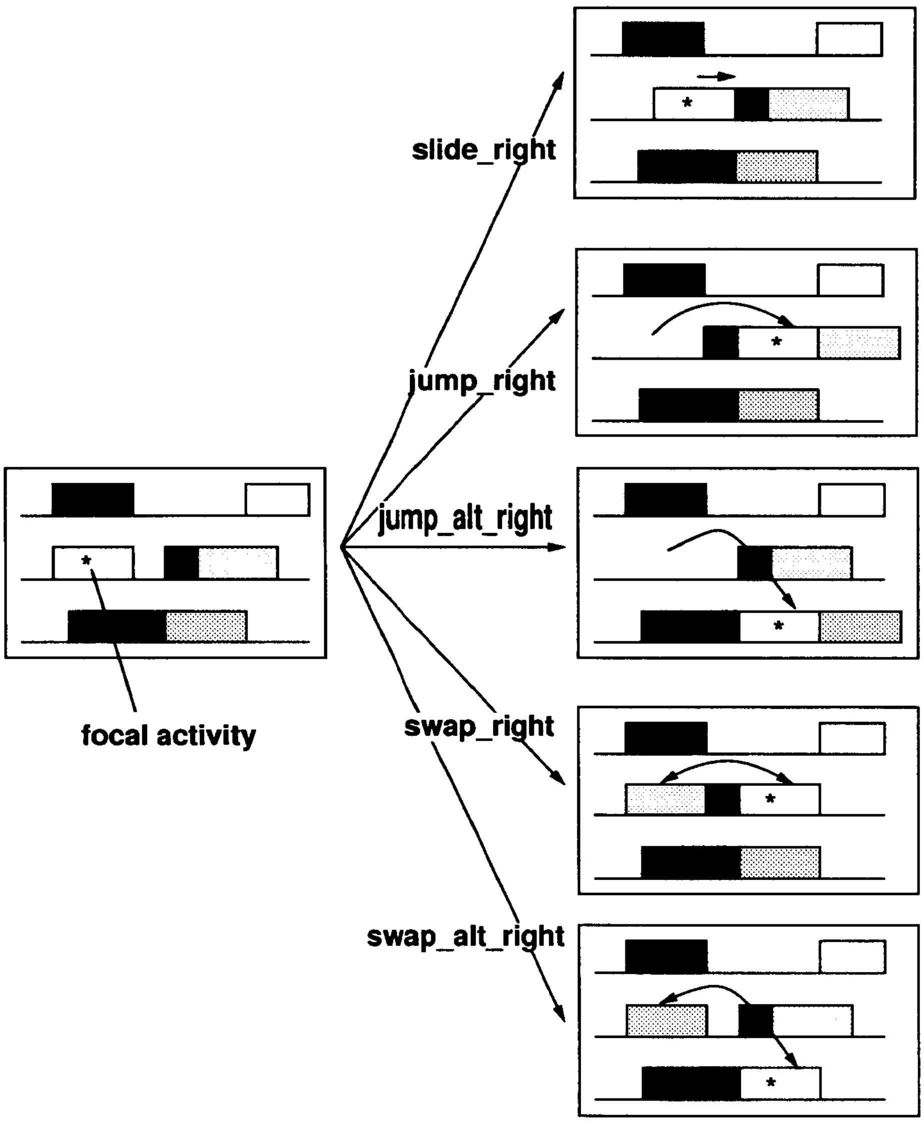

Fig. 3.4: Schedule repair tactics in CABINS (2)

For the other repair tactics, the process of a tactic application has the following steps:

1. Determine the *predictive* start time and designated resource 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 designated resource is a temporary resource to which the focal_activity is moved by a repair tactic. When a repair result is evaluated as acceptable, the predictive start time and designated resource are fixed as assignments to the focal_activity.

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 and resolve capacity and temporal constraint violations. Simple *left-shifting* and *right-shifting* heuristics are used for constraint propagation and the effects of applying the heuristics are also propagated recursively, thus creating the ripple effects on the overall schedule.

   Fig. 3.5 shows the result of consecutive applications of right-shifting heuristic to resolve a constraint conflict involving the activity A_1^1 by moving its scheduled start time on the resource $M1$. Fig. 3.6 shows the result of consecutive applications of left-shifting heuristic to resolve a constraint conflict involving the activity A_2^3 by moving its scheduled start time on the resource $M2$.

3. The propagation results in a feasible schedule when no activity is moved beyond the allowable time window of its order. If a feasible schedule is achieved, the application of the repair tactic is deemed successful and the effect is evaluated whether favorable or not. Otherwise, if an infeasible schedule is achieved, the application of the repair tactic is found failure and another repair tactic is tried if any tactic remains untried.

The above process results in a conflict-free revised schedule. The effects of a revision are calculated and CBR is invoked with the effects as the relevant indices to evaluate a repair outcome.

3.2.2 An Example

In this section, CABINS' repair process is illustrated with a very simple example. A schedule to be repaired is shown in Fig. 3.7 in the form of a gantt chart. In the gantt chart, each row represents assignments on each resource along the timeline, each white box corresponds to an assignment of an activity, and a number in a white box identifies the order which the activity belongs to. The example has ten orders $(O_1, \ldots, O_{10})$ and each order 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.

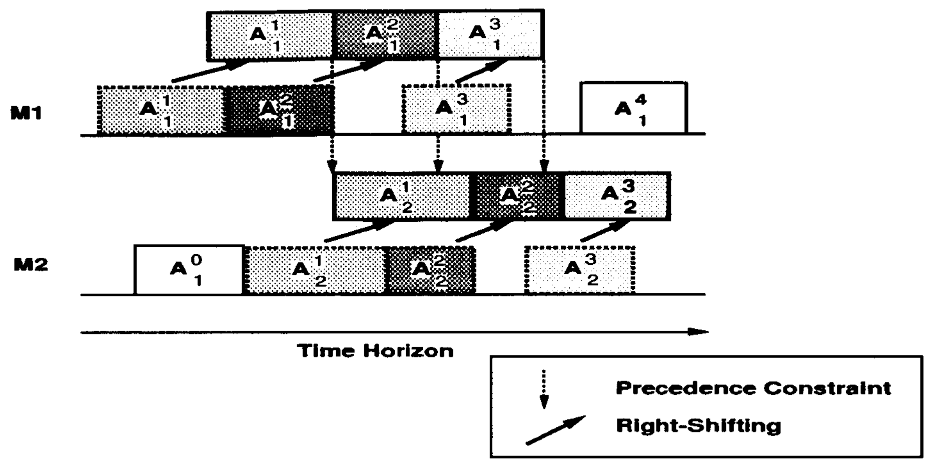

Fig. 3.5: An example of right-shifting heuristic applications

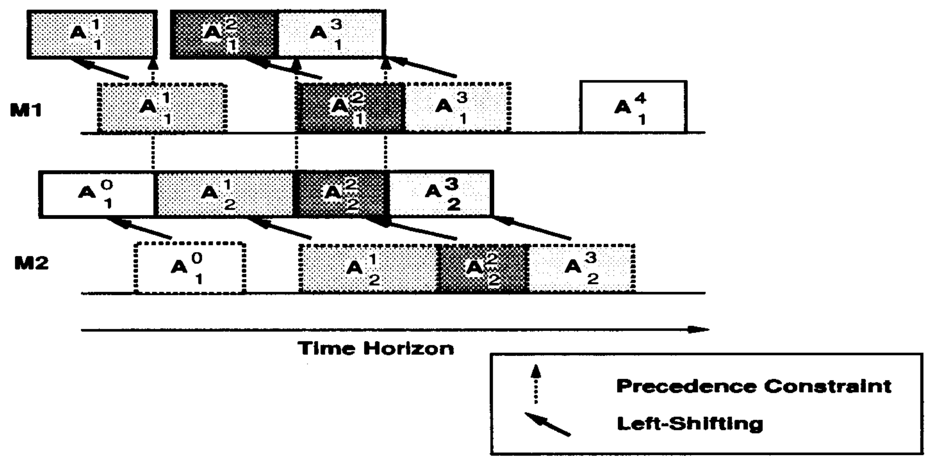

Fig. 3.6: An example of left-shifting heuristic applications

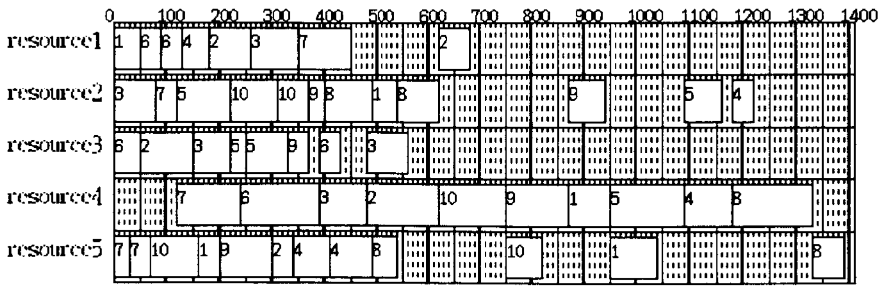

Fig. 3.7: Original schedule results before repair

The schedule in Fig. 3.7 has weighted tardiness of 350 and WIP of 2510. Suppose the CBR is invoked with these indices and selects "reduce_weighted_tardiness" as a repair goal, reflecting the user's severer concern on weighted_tardiness than WIP in the current scheduling situation. Note that this user's concern is represented extensionally in the set of accumulated cases not explicitly in the features of a case. The selected repair goal sorts the orders in the decreasing sequence according to their weighted tardiness value. Since O_8 has the biggest weighted tardiness of all the orders, it now becomes the focal_order. This order has a weight of 2, a due date of 1,250 and the scheduled end-time of its last activity is 1390. Hence it has a weighted tardiness of $2 \times (1390 - 1250) = 280$. Invoking CBR with `Order_Features` (i.e. "slack_ratio", "tardiness_ratio", "inventory_ratio", "resource_idle_ratio", and "resource_busy_deviation"), CABINS selects "shift_left_least" as a repair strategy to be applied, reflecting the order's characteristics of relatively big "slack_ratio ", which is created by a long waiting time in front of the bottleneck resource (i.e. R_4), compared with "tardiness_ratio". "Shift_left_least" strategy is applied to O_8 and creates a sorted activity list ({ A_4^8, A_5^8 }) because the waiting time of A_4^8 is big enough to repair the weighted tardiness of O_8. Suppose that the current focal_activity is A_4^8. CBR invocation with `Activity_Features` (see Fig. 2.7) as indices selects "swap_left" as a repair tactic for A_4^8. One can see from Fig. 3.7 that this is a good choice because the focal_activity is scheduled on the bottleneck resource R_4, which doesn't have any substitutable resource and any idle time in the repair time horizon (time between the end of A_3^8 and the start of A_5^8).

The swap repair tactic roughly calculates the effects of swapping the current focal_activity with each activity within the current focal_activity's time horizon and selects the activity that gives the biggest net gain (note that swapping an activity that is scheduled earlier with one that is scheduled later will now delay the earlier activity). In the example, suppose that activity A_4^4 is selected as the activity to be swapped with the current focal_activity A_4^8. 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) and A_4^4 is moved to start at time 1180 (the start time of activity A_4^8 prior to the swap). Because the new assignments of two activities overlap each other, constraint propagation is invoked and the assignment of A_4^4 is further delayed. Due to the delay of activity A_4^4, 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 3.2.1). Constraint propagation discovers this constraint conflict and shifts activity A_5^4 further to the right on resource R_2. Since Order O_4 has weight 3, its weighted tardiness is now $3 \times (1370 - 1320) = 150$. The repaired schedule result is shown in Fig. 3.8.

CABINS calculates both local effects (i.e. effects on the repair target, O_8) and global effects (i.e. effects on the whole schedule) for result evaluation. In this example, "local_weighted_tardiness" is estimated as +180 time units and "local_inprocess_inventory" is

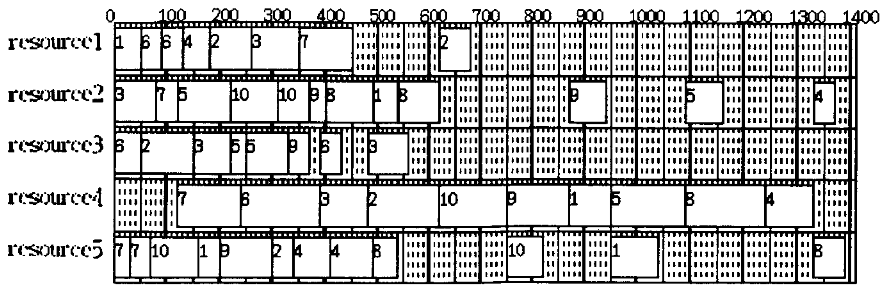

Fig. 3.8: Schedule results after repair on A_4^8

estimated as +200 units, both being improved by the change of A_4^8. And "global_weighted_tardiness" is +30 units (i.e. 180 − 150) and "global_inprocess_inventory" is −750 units (as the waiting time in O_4 increases by 950 units). CBR is invoked using these effect values as indices to determine whether this repair outcome is acceptable or not. If there are more significant "SUCCEEDED" cases than "FAILED" cases in the retrieved k-nearest neighbors, the repair is considered reflecting the tradeoffs of user's preference (in this example, little weight on "global_inprocess_inventory") and the outcome is considered as acceptable. Otherwise, the outcome is considered as unacceptable, thus showing that loss in "global_inprocess_inventory" is more critical than possible gain in weighted tardiness according to the user's preferences.

3.3 Summary

This chapter has presented the details of case-based repair approach implemented in the CABINS system. First, the case retrieval method has been devised for ill-structured problem solving. Then, the repair actions implemented in CABINS for the schedule optimization task have been explained. In addition, a detailed CABINS schedule repair process has been shown with an example.

Chapter 4

Enhancing Solution Quality

The author conducted a set of experiments to test the following hypotheses:

1. CBR-based incremental modification methodology could be effective in capturing user optimization preferences and re-using them to control optimization process.

2. As an iterative optimization method, CABINS' approach produces solutions of comparable quality to other revision-based optimization methods such as simulated annealing.

In this thesis, the job-shop scheduling problem is used as a domain of the experiments, since it is a widely known ill-structured optimization problem and there have been several methodologies developed for the problem, with which CABINS' performance can be compared in terms of solution quality and problem solving efficiency.

The above hypotheses are difficult to test since, owning 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, the author had to devise a method to test the hypotheses in a consistent manner. For performing a objective test, 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, the author used some different explicit criteria to reflect the imaginary user's optimization criteria — although they are widely accepted criteria in practical scheduling environments — and built a rule-based reasoner (RBR) that goes through a trial-and-error repair process to optimize a schedule based on the given criteria. Since the RBR was constructed not to select the same repair action after an application of the repair action was evaluated as unacceptable, it could go through all the available repair actions before giving up a further repair. For each repair, the repair effects were calculated and, on this basis, since the RBR had a pre-defined evaluation objective, the RBR could evaluate the repair outcome consistently. Thus, the RBR with different rules was used each time to generate different case bases for different explicit optimization objectives. 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. Since an objective was designed into the RBR so it could be reflected in the corresponding case base, RBR was used not only as a case base builder but also as an experimental baseline against which to evaluate the schedules generated by CABINS.

The author evaluated the approach on a benchmark suite of job shop scheduling problems where parameters, such as the number of bottlenecks and the range of due dates and activity durations, were varied to cover a broad range of job shop scheduling problem instances. In particular, the benchmark problems have the following structure: each problem has ten orders of five 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 knowledge of the problem had not been unintentionally hardwired into the solution strategies, the author used a problem generator function that embodied the overall problem structure described above to generate job shop scheduling instances where the problem parameters were varied in controlled ways. In particular, six classes of ten problems each — in all, sixty problems — were randomly generated by considering three different values of the range parameter (static, moderate, dynamic), and two values of the bottleneck configuration (one and two bottleneck problems). The slack was adjusted as a function of the range and bottleneck parameters to keep demand for bottleneck resources close to a hundred percent over the major part of each problem. Durations for activities in each order were also randomly generated.

Table 4.1: Characteristics of the six problem sets used in the experiments

	Bottleneck Parameter	Range Parameter
Class-1	1	Static
Class-2	2	Static
Class-3	1	Moderate
Class-4	2	Moderate
Class-5	1	Dynamic
Class-6	2	Dynamic

Table 4.1 shows the parameter settings for each class of the problem sets. Appendix A presents a sample scheduling problem in the Class-6 problem set.

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. [Sadeh, 1991; Smith & Cheng, 1993]). The problem instances, although randomly generated, shared features of problem structure (e.g. each problem has five machines, of which one or two machines are bottlenecks, and substitutable machines exist for the non bottleneck machines etc). This situation can be seen in a real-world scheduling problem such as a factory scheduling that routinely devises a schedule every week for producing the similar (if not same) set of products using the fixed set of resources. CBR can exploit the captured regularities in the structure of the problems for transfer to later problem solving.

The benchmark problems are variations of the problems originally reported in [Sadeh, 1991] and used as a benchmark by a number of researchers (e.g. [Smith & Cheng, 1993; Muscettola, 1993; Liu & Sycara, 1993]). The problem sets in this thesis are, however, different from the original problems in two respects: (a) the problems have substitutable resources for non-bottleneck resources, and (b) the due dates of orders in the problems are tighter by twenty percent than those in the original problems.

A cross-validation method was used to evaluate the capabilities of CABINS. Each problem set of each problem class was divided into two groups with the same size (i.e. each group contains thirty problems). All problems in one group were repaired by RBR as the training sets to gather cases. These cases were then used for case-based repair of the validation problems in the other group. The above process was repeated by interchanging the training sets. So, two case bases were made for each experiment.

4.1 Preference Acquisition in Cases

The hypothesis that CABINS can acquire user preferences was tested by the two experiments, each of which has a different objective function for optimization. The first experiment reflects the user's preference for repairs that minimize the biased combination of weighted tardiness (bias = 1.0) and WIP (bias = 0.05) (hereafter, called Experiment-A), and the second reflects the criterion of minimizing the weighted tardiness and WIP equally (Experiment-B). Thus, in the experiments, the user's objective function is a multi-objective function that is difficult to be optimized heuristically. WIP and weighted tardiness are not always compatible with each other. There are situations where WIP is reduced, but weighted tardiness increases. In general, when a schedule must be optimized according to multiple objectives (as is usually the case), tradeoff of these objectives must be taken into consideration. Tradeoff is context-dependent and therefore difficult to describe in a simple manner. CABINS infers the tradeoff from the cases which store the user's evaluation of schedule repair results. In this set of experiments, a case base that implicitly reflects the optimization criterion was generated through RBR and used for schedule repair.

A case base was constructed for each experiment through RBR. RBR is composed of the four modules, each for goal selection, strategy selection, tactic selection and repair result evaluation respectively.

For goal selection, RBR executes the following steps:

1. Calculate both weighted tardiness and WIP of the current schedule.
2. Bias the calculated objective values based on the user's preferences. (in the Experiment-A, this implies to multiply WIP value by 0.05.)
3. If the value of weighted tardiness is greater than that of WIP,
    (a) if this is the first repair trial, then select "reduce_weighted_tardiness" as a goal.
    (b) if the previous repair trial is "SUCCEEDED", then select "reduce_weighted_tardiness" as a goal.
    (c) if the previous repair trial is "FAILED" after "SUCCEEDED" repair trial, then select "reduce_inprocess_inventory" as a goal.
    (d) otherwise, select "give_up" (terminate a repair process).
4. If the value of WIP is greater than that of weighted tardiness,
    (a) if this is the first repair trial, then select "reduce_inprocess_inventory" as a goal.
    (b) if the previous repair trial is "SUCCEEDED", then select "reduce_inprocess_inventory" as a goal.
    (c) if the previous repair trial is "FAILED" after "SUCCEEDED" repair trial, then select "reduce_weighted_tardiness" as a goal.
    (d) otherwise, select "give_up" (terminate a repair process).

Thus, since RBR knows the user's preferences accurately, RBR pursues to set a repair goal which respects the user's preference as much as possible.

Then, RBR selects a repair strategy for the focal_order based on the selected repair goal as follows:

1. If a repair goal is "reduce_weighted_tardiness",
    (a) if the strategy "shift_left_all" is tried in the previous repair, then select "give_up" as a strategy.
    (b) otherwise, select "shift_left_all" as a strategy.
2. If a repair goal is "reduce_inprocess_inventory",
    (a) if the strategy "shift_right_all" is tried in the previous repair, then select "give_up" as a strategy.

(b) otherwise, select “shift_right_all” as a strategy.

Since only local patching strategies are used for the experiments, current rules for strategy selection are quite simple. But, in realistic situations that allow model modification, selection of repair strategies can be much more complicated.

RBR uses the following rules to determine a repair tactic for the focal_activity:

1. Calculate activity features of the focal_activity.

2. If a repair strategy is “shift_left_all”,

   (a) if “left_slack_ratio” is smaller than 0.0, then select “give_up” as a tactic.

   (b) if “aggr_left_idle_ratio” is greater than “imm_left_idle_ratio”, “left_swappability”, “left_alt_idle_ratio” and “left_alt_swappability”, and “jump_left” is not tried yet, then select “jump_left” as a tactic and set a value of “aggr_left_idle_ratio” as -1.0 to prevent a repetitive usage of the same tactic.

   (c) if “left_swappability” is greater than “imm_left_idle_ratio”, “aggr_left_idle_ratio”, “left_alt_idle_ratio” and “left_alt_swappability”, and “swap_left” is not tried yet, then select “swap_left” as a tactic and set a value of “left_swappability” as -1.0.

   (d) if “left_alt_idle_ratio” is greater than “imm_left_idle_ratio”, “left_swappability”, “aggr_left_idle_ratio” and “left_alt_swappability”, and “jump_alt_left” is not tried yet, then select “jump_alt_left” as a tactic and set a value of “left_alt_idle_ratio” as -1.0.

   (e) if “left_alt_swappability” is greater than “imm_left_idle_ratio”, “left_swappability ”, “left_alt_idle_ratio” and “aggr_left_idle_ratio”, and “swap_left_alt” is not tried yet, then select “swap_left_alt” as a tactic and set a value of “left_alt_swappability ” as -1.0.

   (f) if “imm_left_idle_ratio” is greater than “left_alt_swappability”, “left_swappability ”, “left_alt_idle_ratio” and “aggr_left_idle_ratio”, and “slide_left” is not tried yet, then select “slide_left” as a tactic and set a value of “imm_left_idle_ratio” as -1.0.

   (g) otherwise, select “give_up” as a tactic.

3. If a repair strategy is “shift_right_all”,

   (a) if “right_slack_ratio” is smaller than 0.0, then select “give_up” as a tactic.

   (b) if “aggr_right_idle_ratio” is greater than “imm_right_idle_ratio”, “right_swappability”, “right_alt_idle_ratio” and “right_alt_swappability”, and “jump_right” is not tried yet, then select “jump_right” as a tactic and set a value of “aggr_right_idle_ratio” as -1.0 to prevent a repetitive usage of the same tactic.

(c) if "right_swappability" is greater than "imm_right_idle_ratio", "aggr_right_idle_ratio", "right_alt_idle_ratio" and "right_alt_swappability", and "swap_right" is not tried yet, then select "swap_right" as a tactic and set a value of "right_swappability" as -1.0.

(d) if "right_alt_idle_ratio" is greater than "imm_right_idle_ratio", "right_swappability", "aggr_right_idle_ratio" and "right_alt_swappability", and "jump_alt_right" is not tried yet, then select "jump_alt_right" as a tactic and set a value of "right_alt_idle_ratio" as -1.0.

(e) if "right_alt_swappability" is greater than "imm_right_idle_ratio", "right_swappability", "right_alt_idle_ratio" and "aggr_right_idle_ratio", and "swap_right_alt" is not tried yet, then select "swap_right_alt" as a tactic and set a value of "right_alt_swappability" as -1.0.

(f) if "imm_right_idle_ratio" is greater than "right_alt_swappability", "right_swappability", "right_alt_idle_ratio" and "aggr_right_idle_ratio", and "slide_right" is not tried yet, then select "slide_right" as a tactic and set a value of "imm_right_idle_ratio" as -1.0.

(g) otherwise, select "give_up" as a tactic.

RBR infers the possible repair effect for the application of each repair tactic from values of activity_features of the current focal_activity. Then, RBR selects the most promising repair tactic of all the untried repair tactics for the next repair trial until no repair tactic remains untried.

After each application of a repair tactic, RBR calculates the change in the value of the objective function and evaluates the result of the repair as either "SUCCEEDED" (when the objective value is improved) or "FAILED" (otherwise). Thus, RBR uses greedy heuristics for selecting repair actions such as goals, strategies and tactics, and exploits the precise knowledge (i.e. an objective function) for evaluating the repair results.

Table 4.2: Sizes of acquired case bases for experiments

	Goal_Case	Strategy_Case	Tactic_Case
Experiment-A	848	2675	8376
	845	2614	8208
Experiment-B	943	2480	7816
	891	2370	7592

The number of cases created by RBR and stored in a case base for each experiment is shown in Table 4.2. 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 as a sole source of knowledge both for selecting suitable repairs, and also for evaluating repair results.

In every experiment in this thesis, an initial schedule is created by the EDD(Earliest Due Date) dispatching rule, which tries to allocate the resource to the activity whose order due date is closest. This rule is widely used as a convenient scheduling rule in real-world scheduling problems although quality of schedules by EDD rule is not always optimal. Then, the created schedule is further repaired by several methods such as RBR, CABINS and simulated annealing.

4.1.1 Experiment Results

A graph in Fig. 4.1 shows the comparison of the repair ratio, the ratio of improved objective, by CABINS and RBR for each of six problem sets in the Experiment-A. The repair ratio is calculated as follows:

$$RepairRatio = \frac{Objective\_Before\_Repair - Objective\_After\_Repair}{Objective\_Before\_Repair} \times 100.0$$

where *Objective_Before_Repair* represents the objective value of the initial scheduling result and *Objective_After_Repair* is the objective value of the repaired schedule. Table 4.3 shows the average and standard deviation values of schedule quality by CABINS and RBR over sixty problems. These results show that CABINS can produce slightly better schedules with smaller deviation than RBR, which means CABINS is more robust than RBR in creating high quality solutions.

A graph in Fig. 4.2 shows the comparison of the repair ratio by CABINS and RBR in the Experiment-B. Table 4.4 shows the average and standard deviation of schedule quality. These results show that in this experiment CABINS and RBR have little difference of the solution quality in terms of both average and deviation.

The results of the both experiments suggested that CABINS could generate as high quality schedules as RBR with respect to the two different objectives in all six problem classes. Fig. 4.3 presents subtraction of the repair ratio of CABINS from that of RBR for each problem in the Experiment-A and the Experiment-B. The graph shows that only for a few problems there are considerable differences in repair ratio between them. These results indicate that CABINS can acquire different and subjective user preferences on the tradeoffs of diverse scheduling objectives from the cases. Thus in CABINS' approach, unlike traditional heuristic scheduling approaches [French, 1982; Morton & Pentico, 1993], it is not necessary to devise a particular heuristic to suit the optimization criterion. Only the case base must be changed for different objectives to be optimized. And, unlike the traditional search-based scheduling approaches such as branch-and-bound, dynamic programming, tabu search, simulated annealing and so on, CABINS does not require the

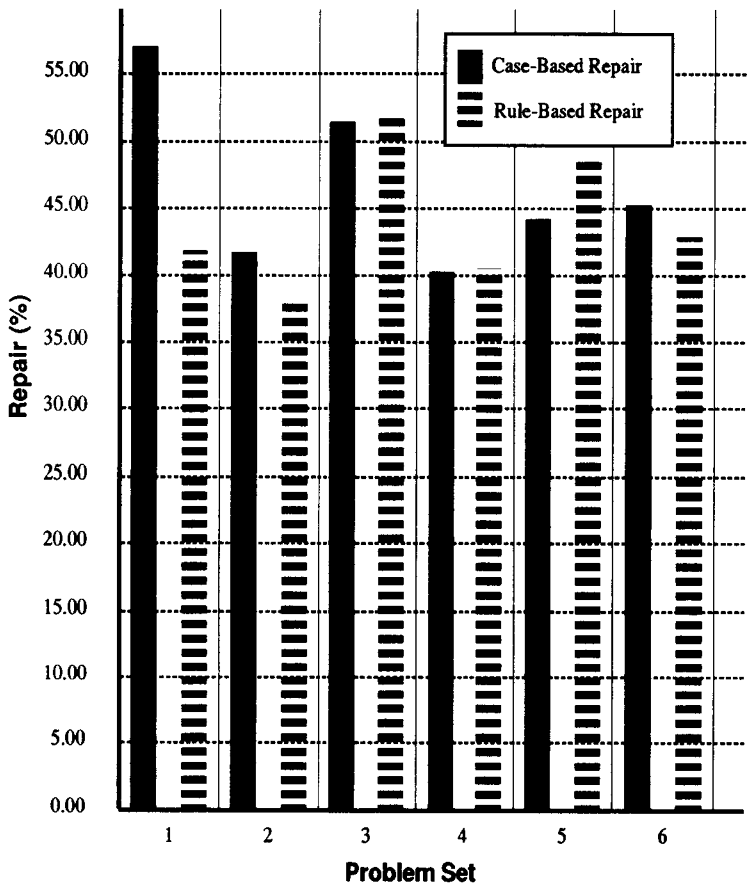

Fig. 4.1: Repair ratio of each problem class by CABINS and RBR in Experiment-A

Table 4.3: Average and standard deviation of schedule quality results by CABINS and RBR in Experiment-A

	Average	Standard Deviation
CABINS	698.8	867.2
RBR	752.4	955.4

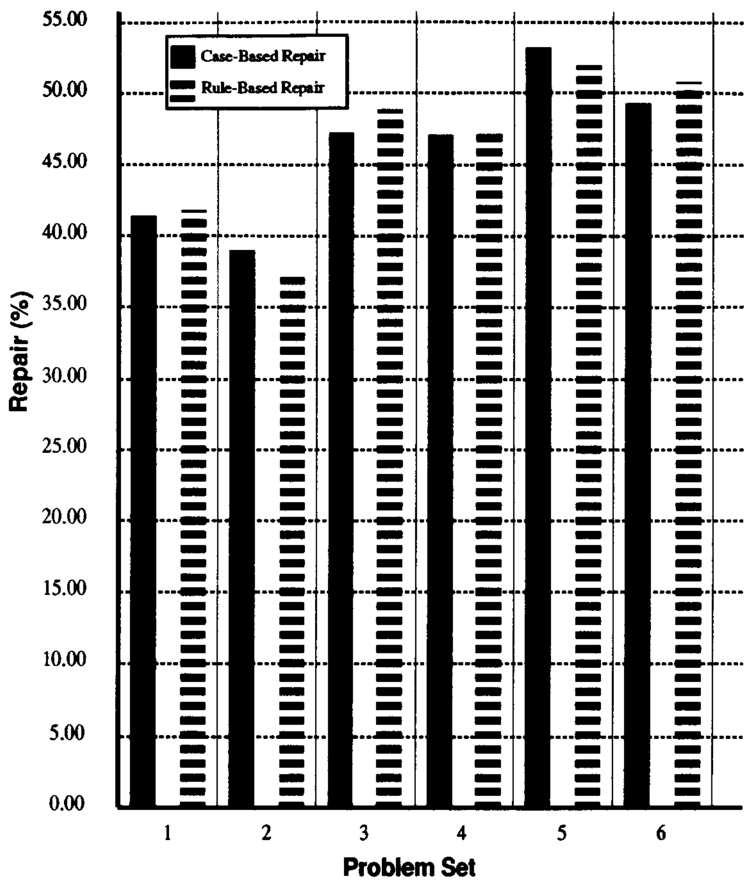

Fig. 4.2: Repair ratio of each problem class by CABINS and RBR in Experiment-B

Table 4.4: Average and standard deviation of schedule quality results by CABINS and RBR in Experiment-B

	Average	Standard Deviation
CABINS	4688.9	4738.4
RBR	4670.8	4721.3

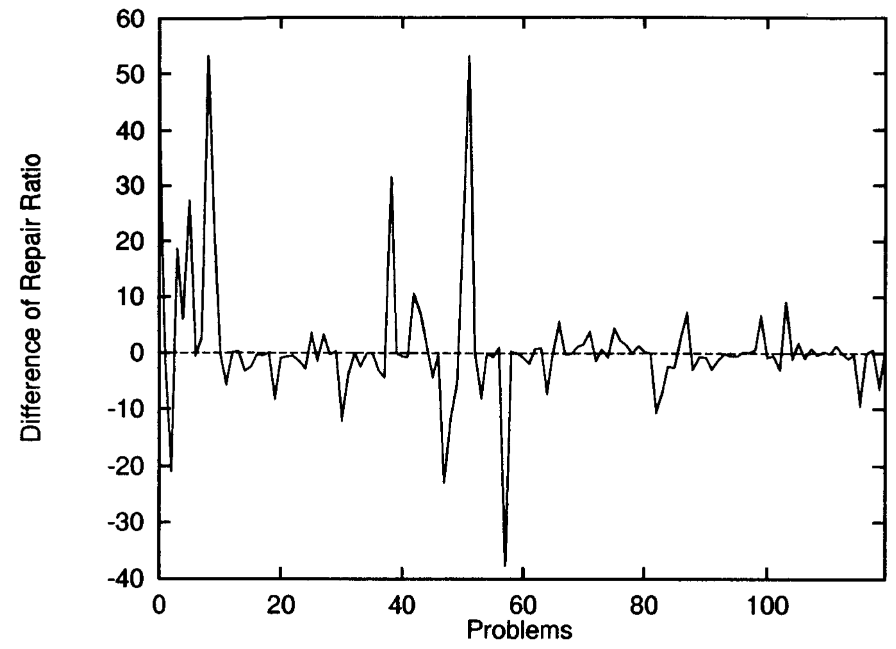

Fig. 4.3: Difference of repair ratio between CABINS and RBR (positive values mean that CABINS surpasses RBR)

objective function to be explicitly represented in the simple mathematical formula. CABINS has a potential of inducing more complicated form of user's objectives (e.g. allowing exceptional situation handling) from the cases. It is true that user's objectives can be elicited by doing intensive interviews with domain experts and represented in the form of rules as was done in making RBR modules for gathering cases in the experiments. But a scheduling problem is such an ill-structured problem that even a domain expert is not supposed to have a sufficient knowledge for making a good schedule efficiently [Kempf *et al.*, 1991]. In general, it is believed that more search efforts are required to find better schedules. Nevertheless, Chapter 5 will show that CBR methodology in CABINS can induce efficient control knowledge from the cases obtained through the applications of the rules and can find a schedule more efficiently than RBR while maintaining quality of the schedules.

4.2 Optimization through Cases

To investigate the hypotheses that CABINS can produce high quality solution based upon the acquired user preference knowledge, the author compared CABINS with simulated annealing method [Kirkpatrick, Gelatt, & Vecchi, 1983]. Simulated annealing (SA) is a widely known general purpose search procedure that generalizes iterative improvement approaches to combinatorial optimization by randomly accepting transitions to lower qual-

```
$T = T_0$;
$x = x_0$ ($\in S$);
$min = \infty$;
while ($T > T_1$) {
    for $i = 1, N$ {
        $x' = neighbor(x)$;
        if ($cost(x') < cost(x)$)   $x = x'$;
        else if ($rand() < \exp\{(cost(x) - cost(x'))/T\}$)   $x = x'$;
        if ($cost(x) < min$)   $min = cost(x)$, $s = x$;
    }
    if ($min$ was not modified in the above loop)   $T = T * \alpha$;
}
```

Fig. 4.4: Simulated annealing procedure

ity solutions so as to avoid being trapped in local minima. SA 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 [Dowsland, 1993]. SA has also been applied to the job shop scheduling domain for the makespan objective and is reported [Laarhoven, Aarts, & Lenstra, 1992] to have a potential of finding shorter makespans than the state-of-the-art tailored heuristic, e.g. the shifting bottleneck procedure [Adams, Balas, & Zawack, 1988].

The details of the current SA implementation are shown in Fig. 4.4. SA procedure is designed to find a schedule $x \in S$ that minimizes the cost function, $cost(x)$. The procedure starts with an initial schedule x_0 and iteratively moves to other neighboring schedules, while remembering the best schedule found so far, s. A neighbor of a schedule x is generated by selecting one activity in a schedule x randomly and applying one randomly chosen repair tactic to the selected activity in a schedule x. SA uses the same set of repair tactics used by CABINS (see Section 3.2.1 for the details of available repair tactics). Typically, the procedure only moves to neighboring schedules that are better than the current schedule. However, in SA, the probability of moving from a schedule x to an inferior schedule x', $\exp\{(cost(x) - cost(x'))/T\}$, is always greater than 0, thereby allowing the procedure to escape from local minima. The function $rand()$ generates a random number from a uniform distribution on the interval of [0,1]. The so-called temperature, T, is a parameter controlling the probability of accepting a transition to a low quality solution. The higher the temperature is, the greater the possibility of moving to a lower quality solution. At the beginning of the procedure, the temperature is set at a very high value,

T_0, therefore allowing frequent transitions to a lower quality schedule. If, after N cycles of iteration, the best schedule found by the procedure has not changed, the temperature parameter T is decremented by a factor α $(0 < \alpha < 1)$. When the temperature drops below a pre-defined level, T_1, the procedure stops and the best schedule it found is returned. In the current implementation, these parameter values were defined as $N = 250$, $\alpha = 0.90$ and $T_1 = 0.01$ and the initial temperature T_0 was calculated as ten times of the maximum cost change in the N cycles of test execution of *neighbor*() function.

4.2.1 Experiment Results

The schedule quality of CABINS and SA was compared using the same objectives and case bases used in Section 4.1. To collect the data of SA, each experiment was run ten times and the average results of these ten separate runs were reported (since in SA the probabilistic factor has been introduced, the results are not necessarily the same across the different experimental runs).

Fig. 4.5 shows the comparison of the repair ratio by CABINS and SA for each of six problem classes in the Experiment-A. Table 4.5 presents the average and standard deviation of the results of all sixty problems in the benchmark. These results suggested that CABINS generated schedules of about 14% higher quality with 12% smaller standard deviation than the average runs of simulated annealing.

Fig. 4.6 shows the comparison of the repair ratio by CABINS and SA for each of six problem classes in the Experiment-B. Table 4.6 presents the average and standard deviation of the results of all sixty problems in the same experiment. These results suggested that CABINS generated schedules of the similar quality to the average runs of simulated annealing in terms of average and standard deviation.

From these experiment results, CABINS' superiority to SA in terms of repair ratio was assumed and examined by hypothesis testing. A signed rank test suggested in [Etzioni & Etzioni, 1994] was used as a testing method. Details of the testing method is fully described in Appendix B. In a nutshell, the hypothesis test checks whether the data provide sufficient evidence against the null hypothesis (H_0) that is negation of the hypothesis to be established (H_a). The p-value is the probability, assuming H_0 holds, of encountering data that favors H_a as much as or more than a reference measure z which is calculated from data observed in the experiment. A small p-value leads one to reject H_0. If the *significance level* required for the experiment is α, then H_0 is rejected if the p-value is less than α. For experiments in the thesis, α is taken to be 0.05.

Table 4.7 shows the reference measure and p-value of the data obtained from all the results in the Experiment-A and the Experiment-B. The null hypothesis states that SA can attain at least as high repair ratio as CABINS. From the p-value in Table 4.7, H_0 can be rejected at the significance level of 0.05. Hence, it is confirmed that CABINS repaired the schedules significantly better than SA.

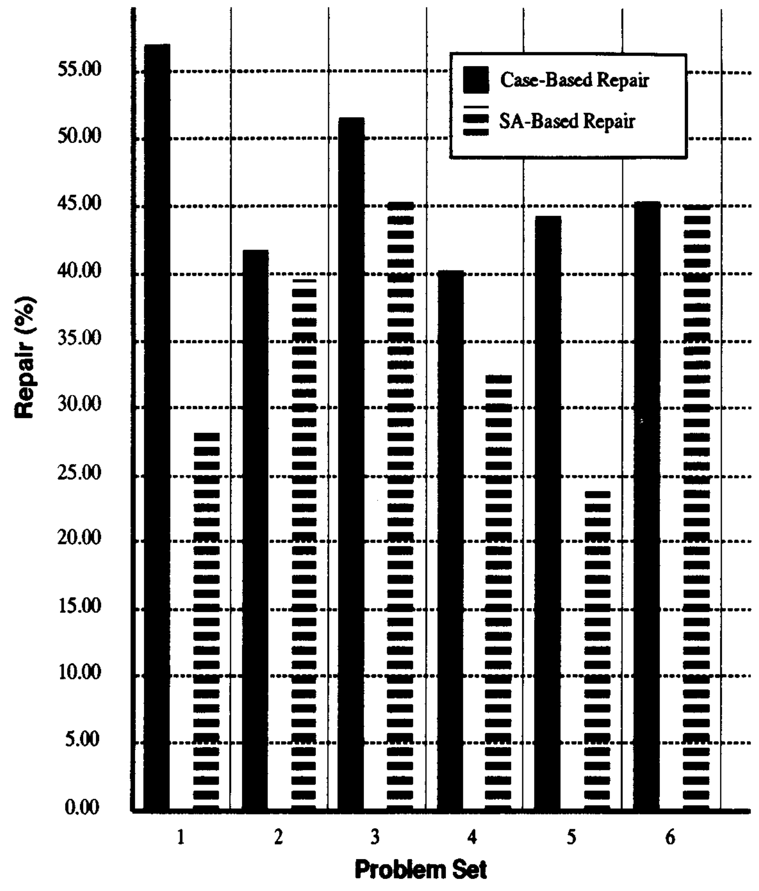

Fig. 4.5: Repair ratio of each problem class by CABINS and SA in Experiment-A

Table 4.5: Average and standard deviation of schedule quality results by CABINS and SA in Experiment-A

	Average	Standard Deviation
CABINS	698.8	867.2
SA	812.6 (best = 596.3, worst = 1061.7)	980.3 (best = 766.8, worst = 1249.9)

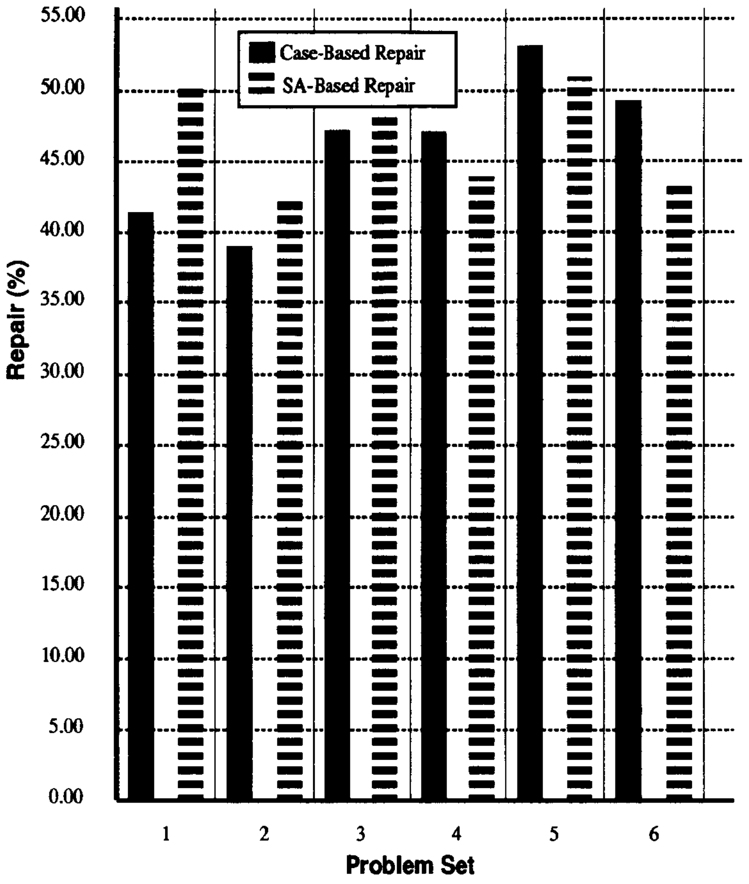

Fig. 4.6: Repair Ratio of each problem class by CABINS and SA in Experiment-B

Table 4.6: Average and standard deviation of schedule quality results by CABINS and SA in Experiment-B

	Average	Standard Deviation
CABINS	4688.9	4738.4
SA	4677.6 (best = 3918.2, worst = 5756.7)	4683.2 (best = 3939.2, worst = 5783.8)

Table 4.7: Repair ratio test between SA and CABINS by Etzioni's signed rank test

	z	p-value
SA vs CABINS	3.54	0.00

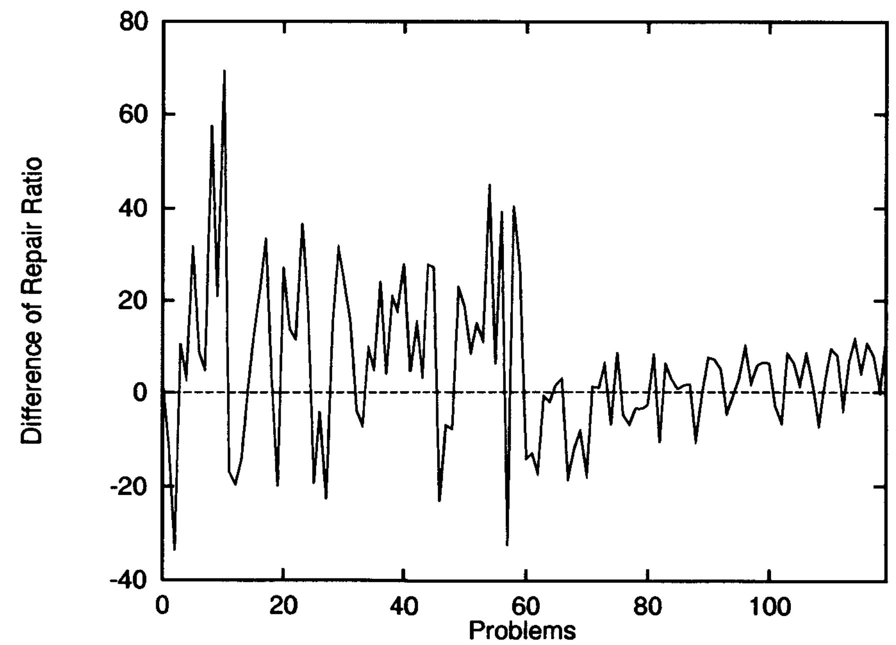

Fig. 4.7: Difference of repair ratio between CABINS and SA (positive values mean that CABINS surpasses SA)

Fig. 4.7 presents subtraction of the repair ratio of CABINS from that of SA for each problem in the Experiment-A and the Experiment-B. The graph shows that in considerably many problems CABINS outperforms SA in terms of the repair ratio. Especially for the first 60 problems (i.e. problems in the Experiment-A), CABINS has a significant superioty to SA. Considering that CABINS reached the solutions much faster than SA, the results seems even more impressive. The search efficiency issues of CABINS will be discussed fully in Chapter 5.

Table 4.5 and Table 4.6 also show that the best results of ten SA runs have about 15% better quality than CABINS. This is because the random search mechanism in SA sometimes succeeds in escaping from deep local minima. In the situations that require the ultimately high quality schedule regardless of the computational cost, SA is thought to be a better candidate than CABINS. But, since defining the neighborhood and the cost function of the problem requires extensive analysis and study of the domain, the application of SA is often difficult in the real-world problems. To reduce the burden of precise definition of the problem neighborhood structure and the cost function, the case-based approach of CABINS can be applied to the implementation of SA. Since CABINS is based on the most general hill-climb search mechanism, integration of CABINS and SA is straightforward. Two functions in Fig. 4.4, *neighbor*() and *cost*(), need to be adapted for the purpose. In *neighbor*(), repair actions are selected through CBR and applied to the current schedule. And in *cost*(), a repair result is evaluated as either "SUCCEEDED" or "FAILED" by retrieving the similar cases. The function returns 0 when the repair is evaluated as a success, and it returns the matching distance from the most similar "SUCCEEDED" case (i.e. the near-miss case) when the repair is evaluated as a failure. The intuition behind this formulation is that the more distant the current repair result is from the success cases, the more likely the result should be rejected. And the best schedule m is always updated when *cost*() returns 0. Actual implementation and experimentation of this integrated method is left for a future research project.

4.3 Analysis of CABINS Performance

From the results shown so far, it is convincing that CABINS should be able to acquire control knowledge to meet user preferences and exploit it for making high quality solution. But, what underlies the power of the approach? The author's hypothesis is that carefully captured case indices and features based upon the task-level analysis are the main source of the power in CABINS. In other words, although a problem belongs to an ill-structured domain, the author assumes that it exhibits some domain regularities that could be captured, albeit only approximately, in a case. In CABINS, a case represents applications of a revision action to a part of the problem, thus expressing dependencies among features of the problem, the repair context and a suitable repair action. CBR allows capture and re-use of this dependency knowledge to dynamically adapt the search procedure and differentially bias repair decisions in future similar situations.

However, it is difficult to analyze how the content of the case representation (i.e. structures and indices/features) affects the performance of CABINS because they are heavily domain-dependent. In this section, empirical analysis is made on how other aspects of case representation can affect the accuracy of case retrieval and the resultant solution quality. And Chapter 6 will discuss the influence of a case base size to the solution quality and problem solving efficiency. The experiments in this section used the same problems and the objective function used in the Experiment-A in Section 4.1.

4.3.1 Case Retrieval with/without Salience

In Section 3.1.1, a set of distance functions were defined as:

$$\begin{aligned}
Normal(CF^i_j, PF_j) &= Sal^i_j \times |CF^i_j - PF_j|/D_j \\
HighCut(CF^i_j, PF_j) &= Sal^i_j \times \max(0.0, (CF^i_j - PF_j)/D_j) \\
LowCut(CF^i_j, PF_j) &= Sal^i_j \times \max(0.0, (PF^i_j - CF_j)/D_j)
\end{aligned}$$

where CF^i_j is the value of `Value` attribute of the j-th feature of the i-th case, PF_j is the value of `Value` attribute of the j-th feature in the current problem, and D_j is the normalization factor for the j-th feature, so that a distance function yields a value in [0.0, 1.0]. Sal^i_j is the salience value of the j-th feature for the repair action used in the i-th case.

Sal^i_j represents the feature relevance to each concept (i.e. the selected repair action). By adjusting a value of a salience and using it for distance calculation, CABINS is believed to become less sensitive to irrelevant features. For example, a value of "left_slack_ratio" feature in a tactic_case has significant influence in selecting one of repair tactics which move a focal_activity to its left (such as "jump_left" repair tactic), but not in selecting a repair tactic which moves a focal_activity to the right, since such a tactic can be safely applied regardless of the value of a slack in the left side of the activity's repair time horizon, which "left_slack_ratio" feature stands for.

In the following experiments, in one experiment the values of feature saliences in tactic_cases were hand-tuned, so that the features relating to the left-side of the repair time horizon have salience of 0.0 in the cases for selecting a tactic moving a focal_activity to the right and vice versa. In the other experiment, the salience value of all the features in tactic_cases was set as 1.0.

Fig. 4.8 shows the comparison of the repair ratio by CABINS with and without tuned salience values for each of six problem sets in the experiment. Table 4.8 presents the average and standard deviation of the results of all sixty problems in the benchmark. These results suggested that CABINS with tuned salience values generated schedules of slightly higher quality than CABINS without tuned salience values. The suggestion was examined by hypothesis testing as was done in Section 4.2.1.

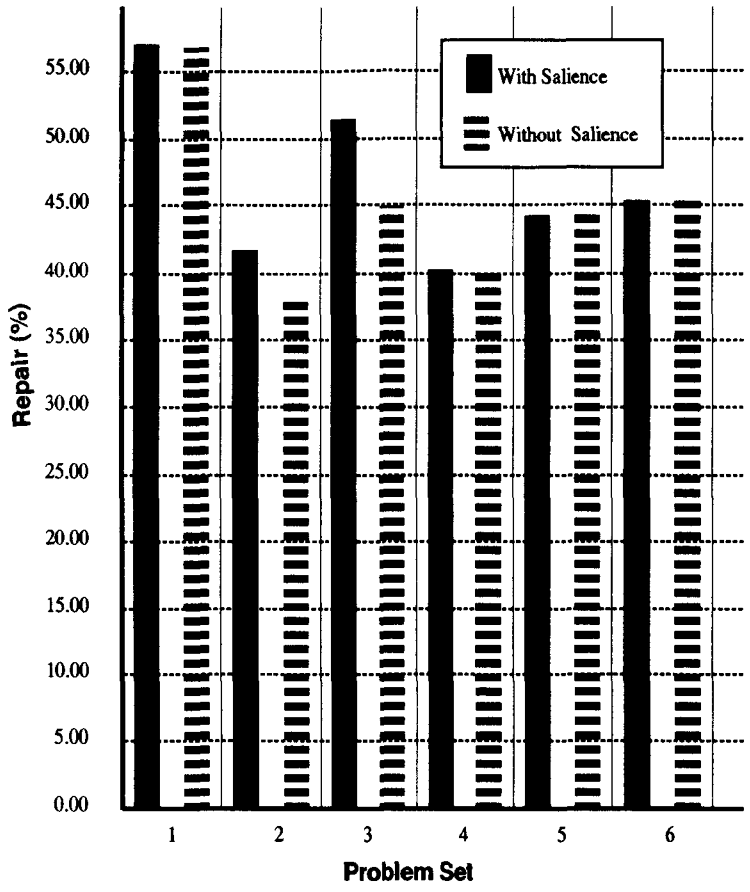

Fig. 4.8: Repair ratio of each problem class by CABINS with/without salience information

Table 4.8: Average and standard deviation of schedule quality results by CABINS with/without salience information

	Average	Standard Deviation
Without Salience	729.6	900.8
With Salience	698.8	867.2

Table 4.9: Repair ratio test between different indexing mechanisms in CABINS by Etzioni's signed rank test; evaluation of effectiveness of salience information

	z	p-value
CABINS w/o salience vs CABINS w/ salience	0.29	0.39

Table 4.9 shows the reference measure and p-value of the experiment results. The null hypothesis states that CABINS without tuned salience information can attain at least as high repair ratio as CABINS with tuned salience information. From the p-value in Table 4.9, H_0 cannot be rejected at the significance level of 0.05. Hence, it cannot be concluded that CABINS with tuned salience information can repair schedules better than CABINS without tuned salience information.

The above result is contradictory to the author's hypothesis that salience information should contribute to the accuracy of case retrieval. To explain the result of this experiment, subtraction of the repair ratio of CABINS without tuned salience information from that of CABINS with tuned salience information is calculated for each problem in the experiment and plotted in the graph shown in Fig. 4.9.

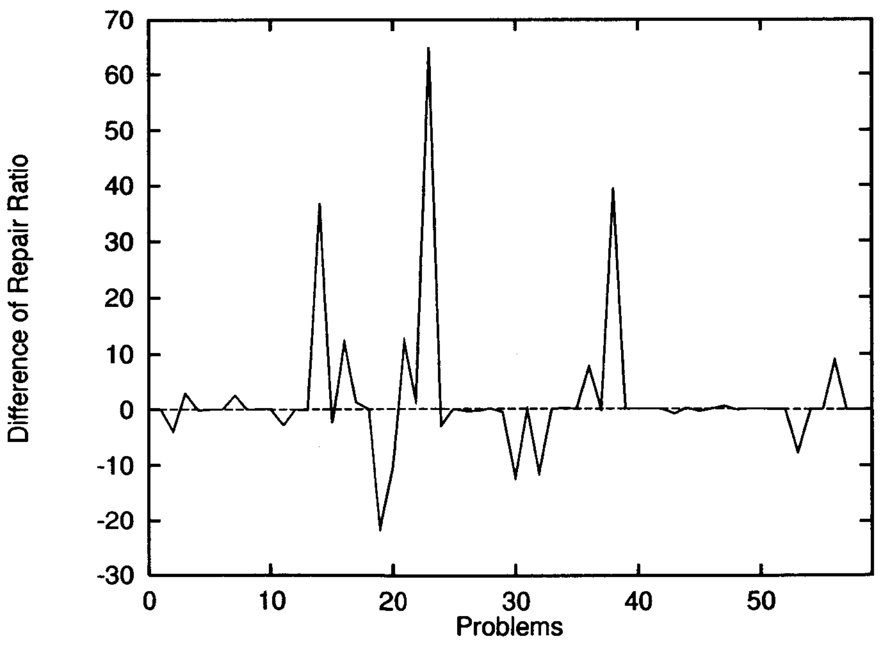

Fig. 4.9: Difference of repair ratio between CABINS with/without tuned salience information (positive values mean that CABINS with tuned saliecne information surpasses CABINS without tuned salience information)

The graph shows that only for a few problems CABINS with tuned salience information had a superiority to CABINS without tuned salience information in repair ratio. Hence, difference between CABINS with/without tuned salience information in repair ratio was not regarded as significant by the hypothesis test. The author assumes that high quality performance by CABINS without tuned salience derives from an accumulation of a large number of cases available. Chapter 6 will show the effects of the number of cases to the performance of CABINS with/without tuned feature salience.

4.3.2 Case Retrieval with/without Effect

In Section 3.1.1, a matching value of cases in selecting a repair action was defined as:

$$Match_i = Similarity_i \times Effect_i$$

where $Match_i$ is the matching value of the i-th case to the current problem. And, $Similarity_i$ is the similarity between the i-th case and the current problem. $Effect_i$ is the value of `Effect` attribute of the i-th case. In case of goal/strategy selection, the `Effect` value of last applied tactic_case associated to the goal_case/strategy_case is used for retrieval.

Intuitive interpretation of using an effect value for selecting a repair action is that if several equally similar cases are available for the current problem, one of the most attractive rules for breaking a tie is to select the most successful case, since one can expect such a case may help produce a good result in the similar problem context.

The following experiments examined usefulness of the effect value for calculating a case matching value in repair action selection; in one experiment the effect values were calculated accurately by means of the objective function, and in the other experiment the effect values were set as 1.0.

Fig. 4.10 shows the comparison of the repair ratio by CABINS with and without effect information for each of six problem classes in the experiment. Table 4.10 presents the average and standard deviation of the results of all sixty problems in the benchmark. These results suggested that CABINS with effect information generated schedules of higher quality than CABINS without effect information. The suggestion was examined again by hypothesis testing.

Table 4.11: Repair ratio test between different indexing mechanisms in CABINS by Etzioni's signed rank test; evaluation of effectiveness of effect information

	z	p-value
CABINS w/o effect vs CABINS w/ effect	3.40	0.00

Table 4.11 shows the reference measure and p-value of the experiment results. The null hypothesis states that CABINS without effect information can attain at least as high repair

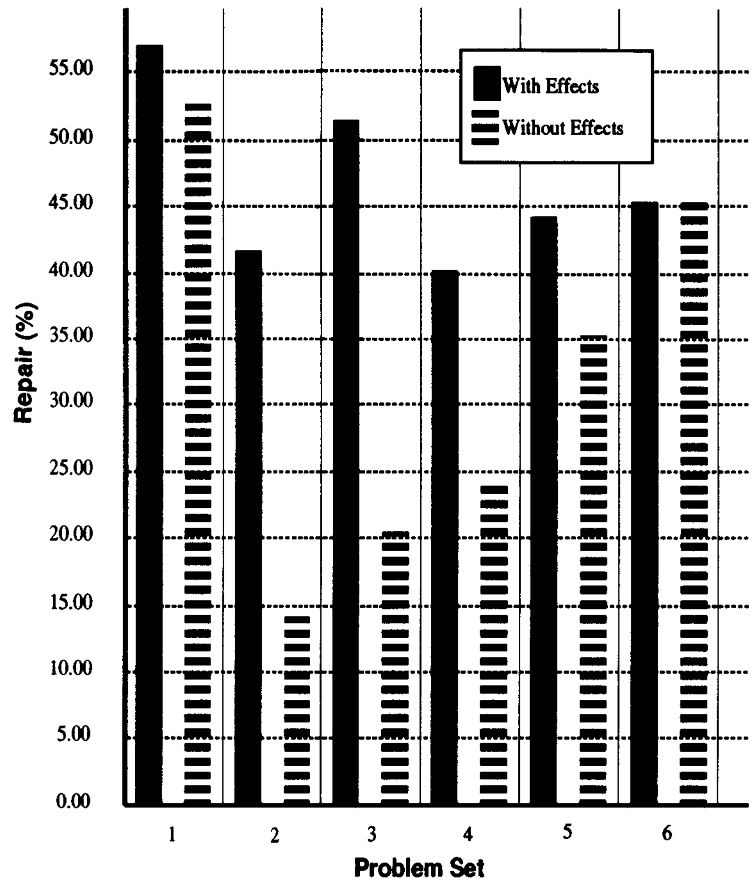

Fig. 4.10: Repair ratio of each problem class by CABINS with/without effect information

Table 4.10: Average and standard deviation of schedule quality results by CABINS with/without effect information

	Average	Standard Deviation
Without Effect	923.3	1128.4
With Effect	698.8	867.2

ratio as CABINS with effect information. From the p-value in Table 4.11, H_0 can be rejected at the significance level of 0.05. Hence, it is affirmed that CABINS with effect information can repair the schedules significantly better than CABINS without effect information.

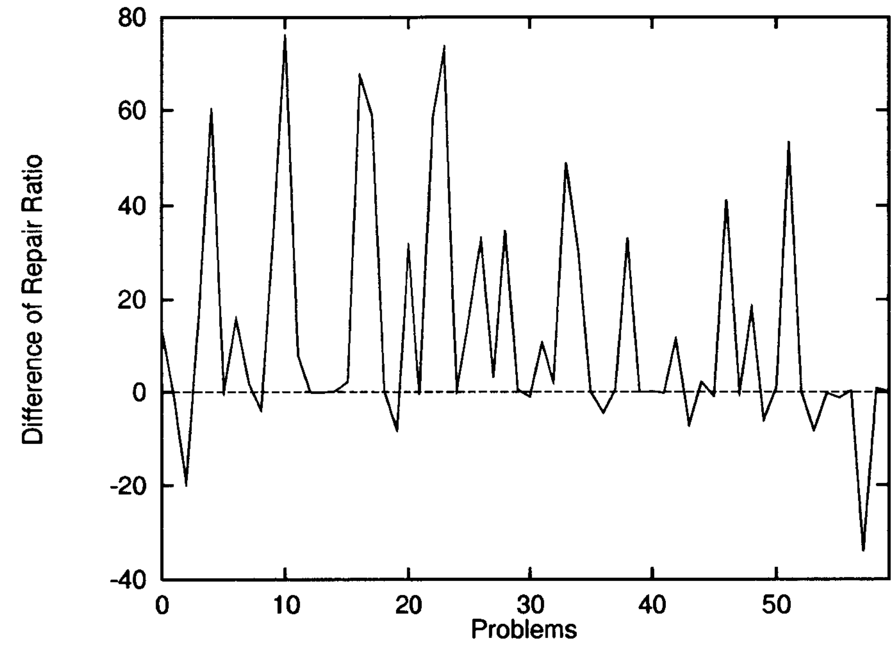

Fig. 4.11: Difference of repair ratio between CABINS with/without effect information (positive values show that CABINS with effect information surpasses CABINS without effect information)

The graph in Fig. 4.11 shows the subtraction of the repair ratio of CABINS without effect information from that of CABINS with effect information for each problem in the experiment. The graph presents that in a considerable number of problems CABINS with effect information was superior to CABINS without effect information in terms of repair ratio.

Although the effect information turns out to be useful for repairs, it is worth noting that the use of effect information for case retrieval depends upon the assumption that domain experts can provide rational evaluation on the effects of repairs, thus making correct effect information available.

4.4 Summary

This chapter has shown the following results from a set of empirical studies:

- CABINS can acquire the control knowledge for enhancing solution quality by storing the history of repair trials in the form of cases and re-use it in making a high quality solution for new problems. CABINS has achieved as high performance as its teacher, heuristically encoded Rule-Based Reasoner (RBR).

- The solution quality by CABINS is significantly better than, or at least as good as the solution quality by simulated annealing method (SA), which is a well recognized revision-based optimization methodology.
- The performance of CABINS does derive partly from its use of past effect information in cases, but not from its use of feature salience information for case retrieval.

Chapter 5

Improving Search Efficiency

Chapter 4 has empirically demonstrated that CABINS can acquire context-dependent user preferences and exploit them for making a high quality solution through case-based reasoning. However, no concern has been shown about the efficiency issue in the chapter. In the real-life problem solving situations, problem solving efficiency should be a big concern. To improve the efficiency of problem solving, the problem solvers need to have effective search control knowledge that can efficiently guide the search process to the successful directions.

Table 5.1: Comparison of averaged quality and efficiency results in Experiment-A by several repair methods

	WT + WIP	Tactic Applications
CABINS	698.8	2206.2
RBR	752.4	1229.8
SA (average)	812.6	12568.8
SA (best)	596.3	125688.2

Table 5.1 shows the average values of the schedule quality and the number of tactic applications required to repair schedules by CABINS, RBR and SA in the Experiment-A of Section 4.1. CPU time is not used as a comparison metric since only CABINS relies on extensive matching for selecting repair actions, which is computationally expensive in terms of CPU time. In the experiment, the computational cost of executing repair goals and strategies is negligible since the only thing they do is sorting focal_orders and focal_activities to regulate the sequence of repair tactic applications. The repair tactic application involves a real calculation of focal_activity movement, constraint propagation and an estimation of repair effects. Hence, the number of repair tactic applications is a major factor of the computational efficiency for the schedule repair task in the experiment.

The table shows that CABINS applied repair tactics about 80% more than RBR to achieve similar quality of schedules. It indicates that CABINS repeatedly selected ineffective repair tactics, applied them, restored schedules and selected a new repair tactic for another repair trial. The random search procedure in SA makes it possible to produce the very high quality of schedules at its best by the efforts of numerous tactic applications. But, the average results of SA show that SA is not comparable to RBR and CABINS in terms of tradeoff between quality and efficiency.

In ill-structured domains such as job shop schedule repair, even experts do not know how many different problem features are relevant and necessary to precisely predict the most successful repair action to be applied to the problem. But, since scheduling constraints are tightly interconnected, the necessary number of features for fully representing a problem must be very large. As a result, the number of features in the current case representation of CABINS could be insufficient. But the number of features needs to be kept moderate, because if a case has a large number of features, the number of training cases required for inducing the correct concepts increases drastically (*dimensionality problem*) [Weiss & Kulikowski, 1990]. Consequently, with a moderate number of features and training cases, any inductive learning methods, including case-based learning, cannot avoid making some wrong predictions. This explains the inefficiency of CABINS in Table 5.1.

To compensate for the lack of a large number of case descriptive features, repair failure experiences in cases can be used for improving predictive accuracy (i.e. retrieve more relevant cases from the case base). The author's hypothesis is that with the help of the past failure experiences, CBR enables CABINS the followings:

1. Learning to avoid repeating the similar failures that have been experienced before.

2. Learning to avoid wasting lots of efforts for trying to solve too difficult problems.

In other words, the author assumes that from failure cases CABINS can learn search control knowledge that shows the above capabilities.

To analyze the correctness of the above hypothesis, the following three repair methods were experimentally implemented in CABINS: *validated repair*, *interruptive repair* and *hybrid repair*. These repair methods were tested using the same problems, objectives and case bases used for the Experiment-A in Section 4.1.

5.1 Validated Repair

The validated repair method allows CABINS to apply a repair action only after the repair action is validated as possibly effective for repairing the current problem. Validation and justification techniques are widely used in many case-based reasoning systems (e.g. [Simoudis & Miller, 1990; Hammond, 1989]). In CHEF [Hammond, 1989], whose task is making a recipe for Chinese dishes, the MODIFIER module checks and alters the retrieved plan using a *causal model* of the domain (i.e. modification rules, critics with knowledge of goal specific requirements and general plan specifications). Hence, a modified recipe is less likely to fail in cooking. In CASCADE and COAST [Simoudis & Miller, 1990; Simoudis & Miller, 1991], which are help desk systems for diagnosis of DEC's software products, validation process extracts from the case base only those cases that appear to be closely relevant to the current problem, i.e. improves the accuracy of retrieval using a *validation model* of the domain. The validation model comprises the knowledge about the individual diagnostic test and its expected value for the case to match the current problem, and the knowledge about interrelations among the tests. The richness of the validation model enables the efficient directed search of the vast test space in the above systems and reduce the cost of search (i.e. number of expensive testings).

The above systems utilized the established models based on the deep understanding of their domain for validation. However, in ill-structured domains like a job shop scheduling, neither causal model nor validation model of the domain are readily available. For example, in the job shop scheduling problem, non-linearity of objectives and tight interactions of constraints make it hard to predict the effects of a local optimization decision on global optimality, thus making the analysis of the domain structure required to build a causal model and a validation model extremely difficult. Thus, CABINS has to make a validation of the retrieved case without such models. Compensating for lack of these models, CABINS utilizes past repair failure experiences to evaluate validity of selected repair actions.

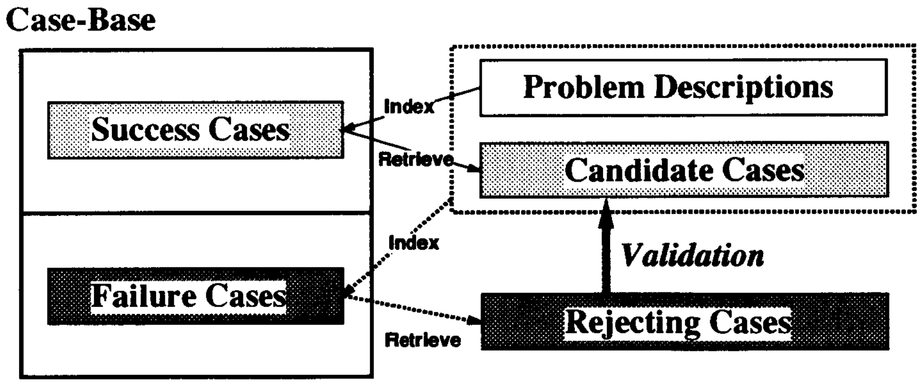

Fig. 5.1: The validation process in CABINS

Fig. 5.1 shows the schematic diagram of the validation process in CABINS. First, candidate cases are retrieved from *success cases* in a case base, which match the current problem situation. Then, rejecting cases are retrieved from *failure cases* which match the current problem and use the same repair actions used in the candidate cases. If the rejecting cases have little credibility (i.e. low similarity to the current problem), the candidate cases are validated to be possibly effective.

The validated repair method was implemented and applied to the tactic selection that is computationally most expensive in CABINS schedule repair process. The repair tactic selection procedure of the validated repair in CABINS is as follows:

1. Select k nearest tactic cases of the current problem which succeeded in repair (i.e. candidate cases in Fig. 5.1).
2. Accumulate the case similarity in the selected k success_cases for each repair tactic that was successfully applied in the case.
3. Sort the repair tactic according to the accumulated similarity.
4. Until no repair tactic remains in the sorted queue, do the followings:
   (a) Pick up the next repair tactic in the sorted repair tactics.
   (b) Select k nearest tactic cases of the current problem, in which an application of the picked-up tactic resulted in failure (rejecting cases in Fig. 5.1).
   (c) Sum up the similarity of the selected k cases.
   (d) If the sum is smaller than pre-define threshold value, return the tactic as a validated selection.
5. Return "give_up" as a repair tactic.

In the above procedure, CABINS judges the likelihood that a selected repair tactic will fail in the current problem from the past failure experiences of the same tactic. If CABINS judges that it is not likely to fail in the current problem (i.e. there are not enough numbers of similar failure experiences), CABINS validates the application of the selected repair tactic. Since in difficult problems such as schedule repair, failures usually outnumber successes, if the value of the threshold (in step 4(d) of the above procedure) is defined moderately, overly pessimistic results could be produced (i.e. CABINS seldom validates the tactic). To avoid this, a threshold value was set as 4.99. Since a value of k was 5 in the experiments (see Section 3.1), validation rejection was unlikely to occur.

5.1.1 Experiment Results

Graphs in Fig. 5.2 show the comparison of the repair ratio and the number of tactic applications for each class of the problems by RBR, CABINS and CABINS with the validated repair. Table 5.2 and Table 5.3 show the average and standard deviation of schedule quality and tactic applications number over sixty problems respectively. The results in the graphs and the tables show that CABINS with the validated repair improved its efficiency about 58% from original CABINS without unduly sacrificing the schedule quality. And it seemed that CABINS with the valid repair was even faster than RBR. The superiority

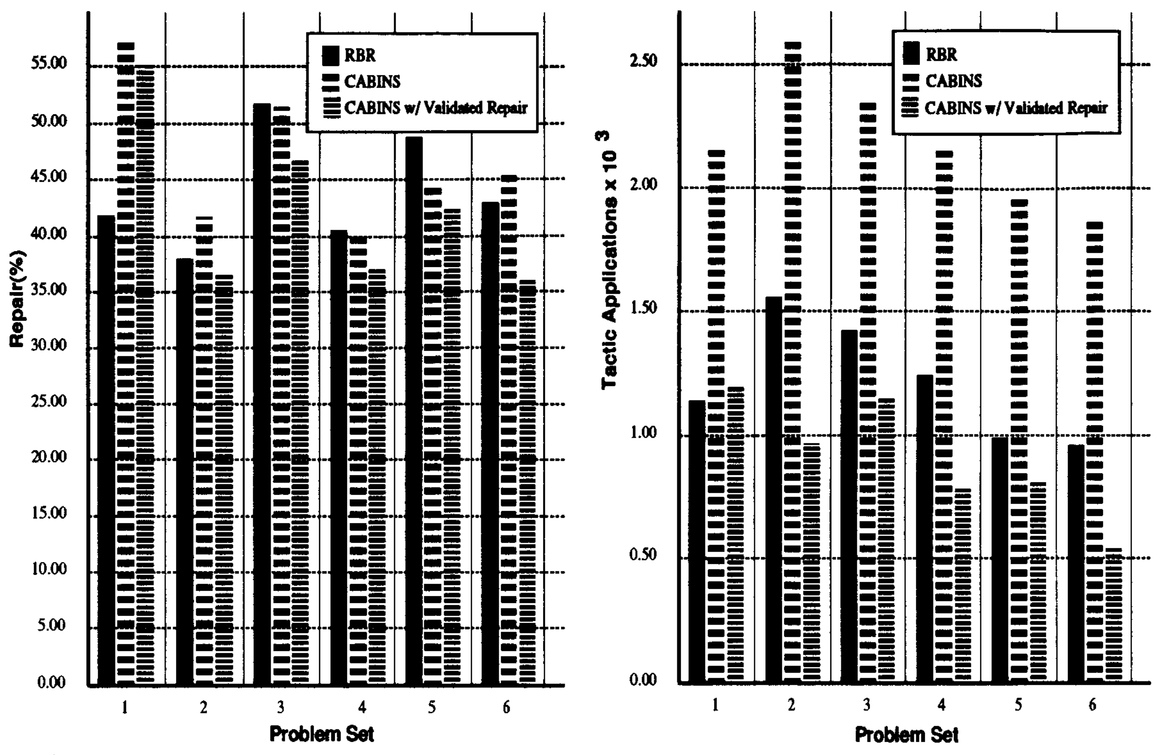

Fig. 5.2: Effects on repair ratio and efficiency results of each problem class in Experiment-A by the validated repair in CABINS

Table 5.2: Average and standard deviation of quality results in Experiment-A by the validated repair in CABINS

	Average	Standard Deviation
RBR	752.4	955.4
CABINS	698.8	867.2
CABINS w/ Validated Repair	751.1	923.1

Table 5.3: Average and standard deviation of efficiency results in Experiment-A by the validated repair in CABINS

	Average	Standard Deviation
RBR	1229.8	1288.1
CABINS	2206.2	2287.0
CABINS w/ Validated Repair	920.3	976.2

Table 5.4: Problem solving efficiency test between CABINS with several repair methods and RBR by Etzioni's signed rank test; evaluation of the validated repair method

	z	p-value
CABINS w/ Validated Repair vs RBR	4.44	0.00

of CABINS with the validated repair to RBR in repair efficiency is examined by Etzioni's hypothesis test.

Table 5.4 shows the reference measure and p-value of the experiment results. The null hypothesis states that CABINS with the validated repair can repair schedules at least as fast as RBR. From the p-value in Table 5.4, H_0 can be rejected at the significance level of 0.05. Hence, it is affirmed that there is a significant evidence in favor of the observation that CABINS with the validated repair can repair the schedules faster than RBR.

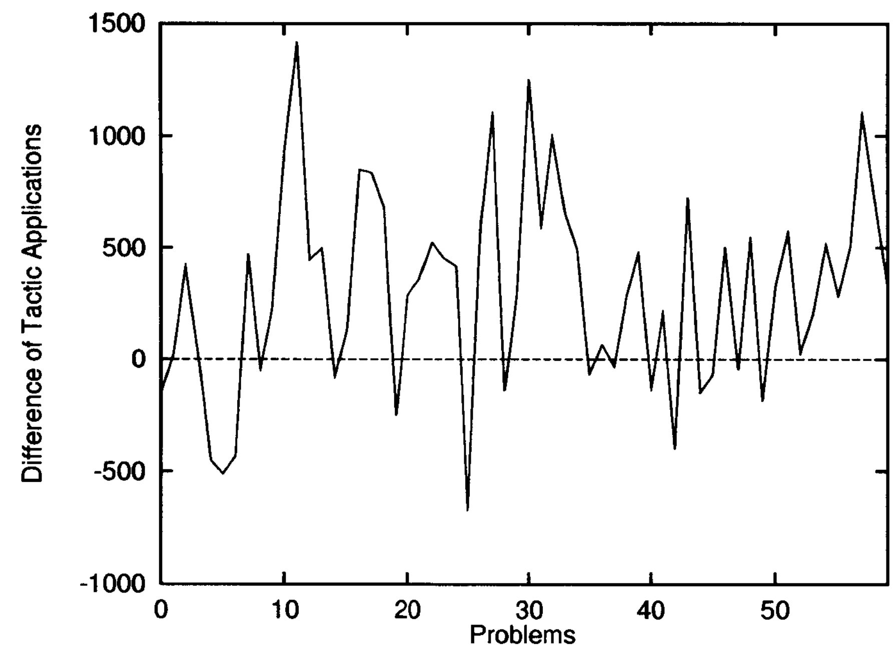

Fig. 5.3: Difference of tactic application numbers between RBR and CABINS with the validated repair (positive values show that RBR applied more tactics than CABINS with the validated repair)

The graph in Fig. 5.3 shows the subtraction of the tactic application number of CABINS with the validated repair from that of RBR for each problem in the experiment. The graph presents that in a considerable number of problems CABINS with the validated repair was superior to RBR in terms of the number of tactic applications.

5.2 Interruptive Repair

The interruptive repair method allows CABINS to shift its repair attention to another problem when the current problem seems too difficult to be solved. CABINS gives up a further repair when it determines that it would be a waste of time. To determine whether to give up a further repair, failure cases are utilized again. If there are enough failure cases in the past similar situations to the current one, CABINS gives up repairing the current problem, and switches its attention to another problem.

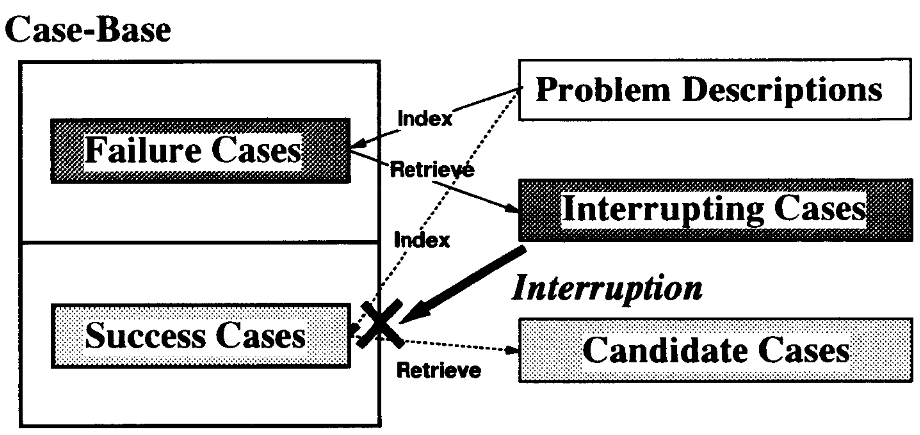

Fig. 5.4: The interruption process in CABINS

Fig. 5.4 shows the schematic diagram of the interruption process in CABINS. In the process, first interrupting cases are retrieved from *failure cases* which match the current problem situations. If the interruptive cases have high credibility (i.e. high similarity to the current problem), the succeeding process of retrieving candidate cases for finding a possible solution to the problem is not allowed to proceed. Then the problem solver seeks for another problem to be solved.

The repair tactic selection procedure of the interruptive repair in CABINS is as follows:

1. Select k nearest tactic cases of the current problem which failed in repair (i.e. interrupting cases in Fig. 5.4).

2. Accumulate the similarity of the the selected k failure_cases.

3. If the sum exceeds the pre-defined threshold value, return "give_up" as a repair tactic.

   Otherwise, select k nearest tactic cases of the current problem which succeeded in repair (candidate cases in Fig. 5.4 — these are same with the candidate cases in Fig. 5.1).

4. Accumulate the similarity of the case in the selected k success_cases for the repair tactic that was successfully applied in the case.

5. Sort the repair tactic according to the accumulated similarity.
6. Return the first repair tactic in the sorted repair tactics as a selected repair tactic.

In the above procedure, CABINS judges the likelihood that a current problem cannot be repaired from the failure experiences in the past similar problems. If CABINS judges that the current problem is not likely to be repaired (i.e. there are enough numbers of failure experiences in the similar problems), CABINS gives up repairing the current problem. As was discussed in the validated repair method, in difficult problems such as schedule repair, failures usually outnumber successes and if the value of the threshold (in step 3 of the above procedure) is defined moderately, overly pessimistic results could be produced (i.e. CABINS suggests giving up too often). To avoid this, a value of the threshold was set as 4.99.

5.2.1 Experiment Results

Graphs in Fig. 5.5 show the comparison of the repair ratio and the number of tactic applications for each class of the problems by RBR, CABINS and CABINS with the interruptive repair. Table 5.5 and Table 5.6 show the average and standard deviation of schedule quality and tactic application numbers over sixty problems respectively. The results in the graphs and the tables show that CABINS with the interruptive repair improved its efficiency about 46% from original CABINS while maintaining high schedule quality. One potential reason for these results is that, as described in section 1.1, the effects of schedule repair are pretty unpredictable and there is a good chance that another application of repair tactic may make the problem, which once judged difficult, easier by changing the existing schedule fundamentally so that CABINS can go back to the problem afterwards and repair it without wasting much effort.

And CABINS with the interruptive repair seemed to be slightly more efficient than RBR. This observation is tested by the hypothesis test.

Table 5.7: Problem solving efficiency test between CABINS with several repair methods and RBR by Etzioni's signed rank test; evaluation of the interruptive repair method

	z	p-value
CABINS w/ Interruptive Repair vs RBR	0.67	0.25

Table 5.7 shows the reference measure and p-value of the experiment results. The null hypothesis states that CABINS with the interruptive repair can repair schedules at least as fast as RBR. From the p-value in Table 5.7, H_0 cannot be rejected at the significance level of 0.05. It cannot be concluded that CABINS with the interruptive repair is more efficient

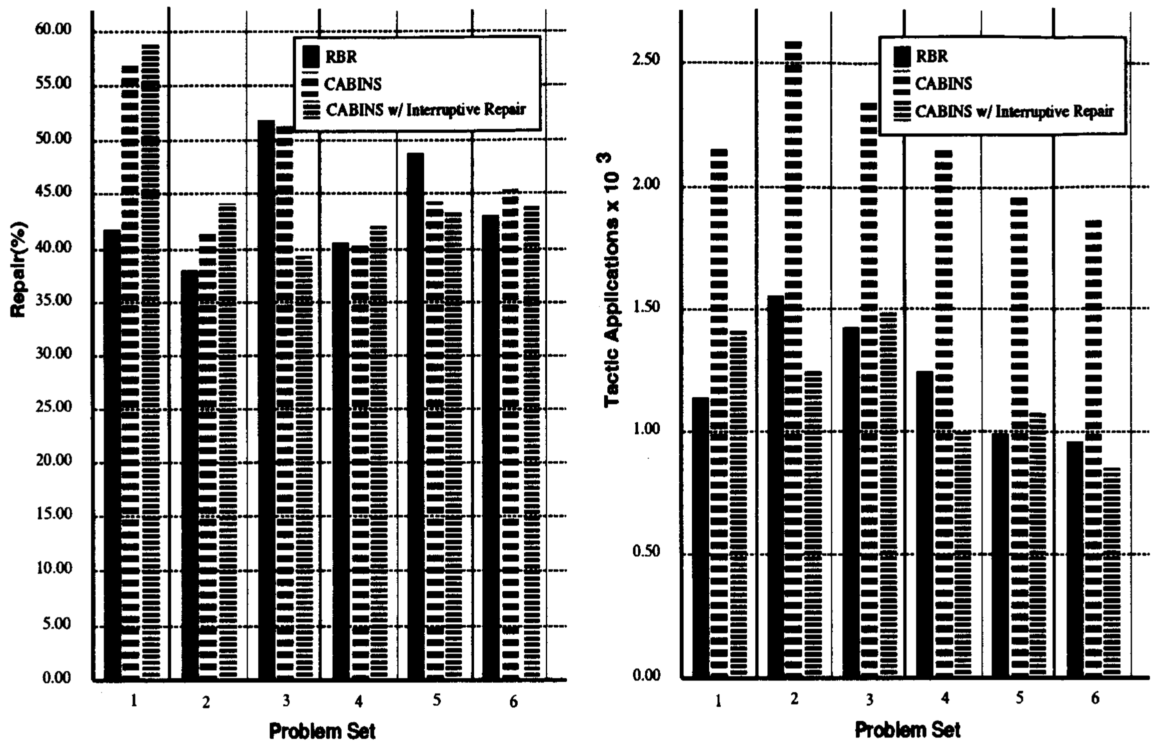

Fig. 5.5: Effects on repair ratio and efficiency results of each problem class in Experiment-A by the interruptive repair in CABINS

Table 5.5: Average and standard deviation of quality results in Experiment-A by the interruptive repair in CABINS

	Average	Standard Deviation
RBR	752.4	955.4
CABINS	698.8	867.2
CABINS w/ Interruptive Repair	724.3	907.2

Table 5.6: Average and standard deviation of efficiency results in Experiment-A by the interruptive repair in CABINS

	Average	Standard Deviation
RBR	1229.8	1288.1
CABINS	2206.2	2287.0
CABINS w/ Interruptive Repair	1186.8	1257.0

than RBR. (The opposite cannot be affirmed either.) Nevertheless, efficiency improvement of CABINS by the interruptive repair is apparent.

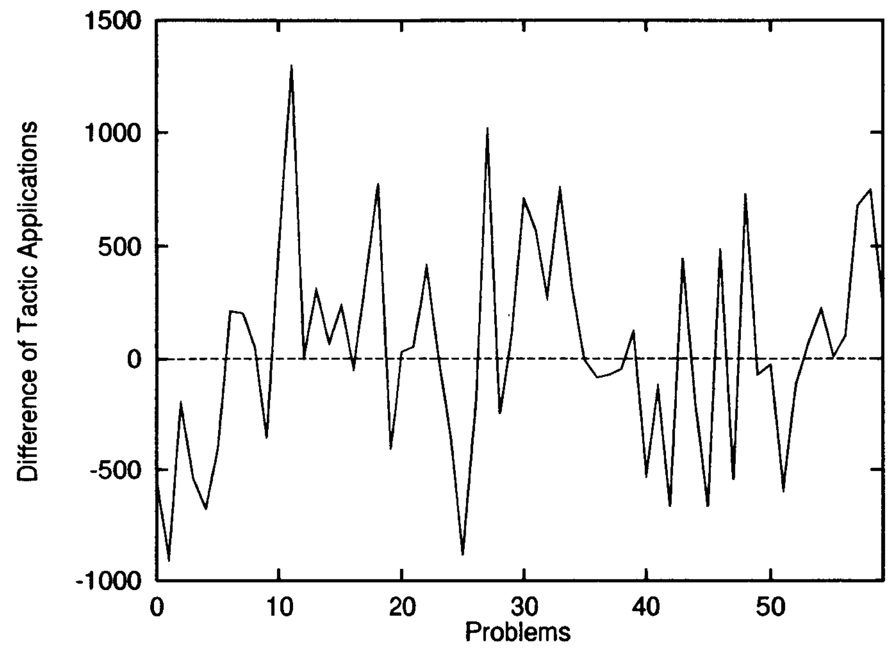

Fig. 5.6: Difference of tactic application numbers between RBR and CABINS with the interruptive repair (positive values mean that RBR applied more tactics than CABINS with the interruptive repair)

The graph in Fig. 5.6 shows the subtraction of the tactic application number of CABINS with the interruptive repair from that of RBR for each problem in the experiment. The graph presents that in some problems CABINS with the interruptive repair was superior to RBR in terms of the number of tactic applications, but in other problems RBR surpassed CABINS with the interruptive repair.

5.3 Hybrid Repair

As was explained in the previous sections, the validated repair method and the interruptive repair method exploit past failure experiences in the different phase of case retrieval for speeding up the repair process. Hence, the hybrid repair method can be devised in a straightforward fashion by combining these two repair methods. The repair tactic selection procedure of the hybrid repair in CABINS is as follows:

1. Select *k* nearest tactic cases of the current problem which failed in repair.

2. Accumulate the similarity of the the selected *k* failure_cases.

3. If the sum exceeds the pre-defined threshold value, return "give_up" as a repair tactic.

   Otherwise, select k nearest tactic cases of the current problem which succeeded in repair.

4. Accumulate the similarity of the case in the selected k success_cases for the repair tactic that was successfully applied in the case.

5. Sort the repair tactic according to the accumulated similarity.

6. Until no repair tactic remains in the sorted queue, do the followings:

   (a) Pick up the next repair tactic in the queue.

   (b) Select k nearest tactic cases of the current problem, in which an application of the picked-up tactic resulted in failure .

   (c) Sum up the similarity of the selected k cases.

   (d) If the sum is smaller than pre-define threshold value, return the tactic as a validated selection.

7. Return "give_up" as a repair tactic.

In the above procedure, the thresholds (in step 3 and step6(d)) are set with the same value as in the validated repair and the interruptive repair.

5.3.1 Experiment Results

Graphs in Fig. 5.7 show the comparison of the repair ratio and the number of tactic applications for each class of the problems by RBR, CABINS and CABINS with the hybrid repair. Table 5.8 and Table 5.9 show the average of schedule quality and tactic applications number over sixty problems respectively. The results in the graphs and the tables show that CABINS with the hybrid repair improved its efficiency about 65% from original CABINS without damaging the schedule quality.

The results also suggested that CABINS with the hybrid repair produced high quality schedules more efficiently than RBR. The superiority of CABINS with the hybrid repair to RBR in repair efficiency is examined by Etzioni's hypothesis test.

Table 5.10 shows the reference measure and p-value of the experiment results. The null hypothesis states that CABINS with the hybrid repair can repair schedules at least as fast as RBR. From the p-value in Table 5.10, H_0 can be rejected at the significance level of 0.05. Hence, it is affirmed that there is a significant evidence showing that CABINS with the hybrid repair can repair the schedules faster than RBR.

The graph in Fig. 5.8 shows the subtraction of the tactic application number of CABINS with the hybrid repair from that of RBR for each problem in the experiment. The graph

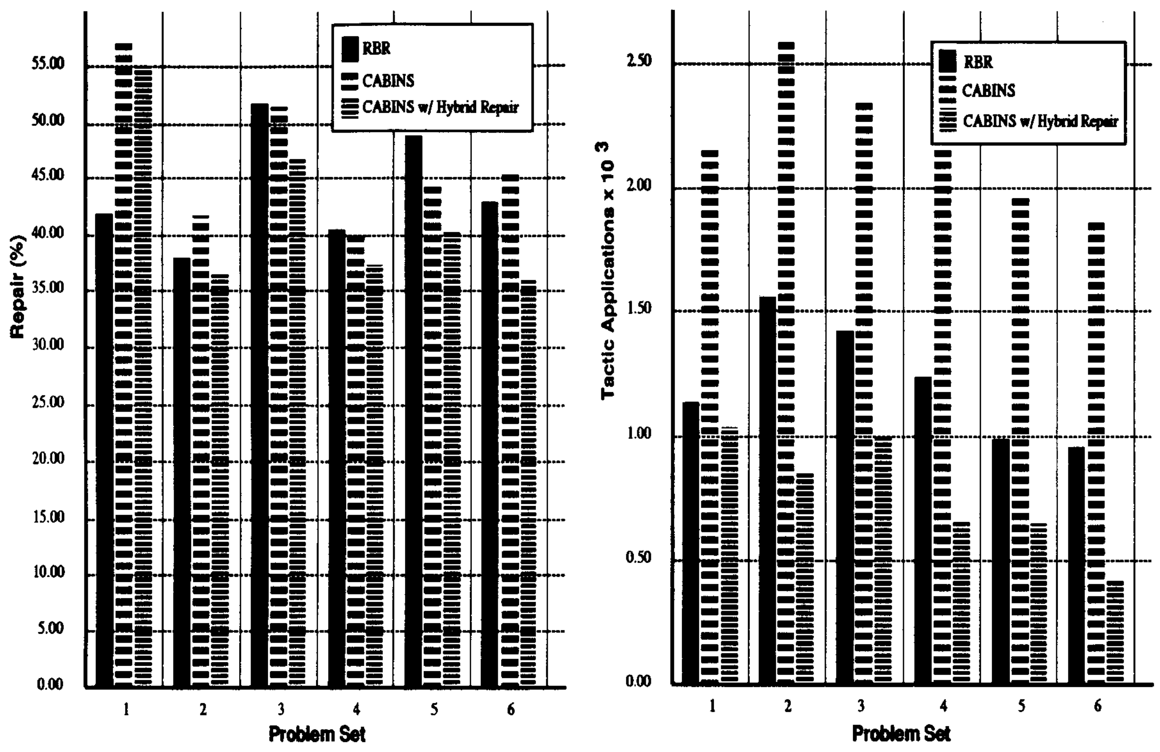

Fig. 5.7: Effects on repair ratio and efficiency results of each problem class in Experiment-A by the hybrid repair in CABINS

Table 5.8: Average and standard deviation of quality results in Experiment-A by the hybrid repair in CABINS

	Average	Standard Deviation
RBR	752.4	955.4
CABINS	698.8	867.2
CABINS w/ Hybrid Repair	753.4	924.1

Table 5.9: Average and standard deviation of efficiency results in Experiment-A by the hybrid repair in CABINS

	Average	Standard Deviation
RBR	1229.8	1288.1
CABINS	2206.2	2287.0
CABINS w/ Hybrid Repair	776.6	831.1

Table 5.10: Problem solving efficiency test between CABINS with several repair methods and RBR by Etzioni's signed rank test; evaluation of the hybrid repair method

	z	p-value
CABINS w/ Hybrid Repair vs RBR	6.10	0.00

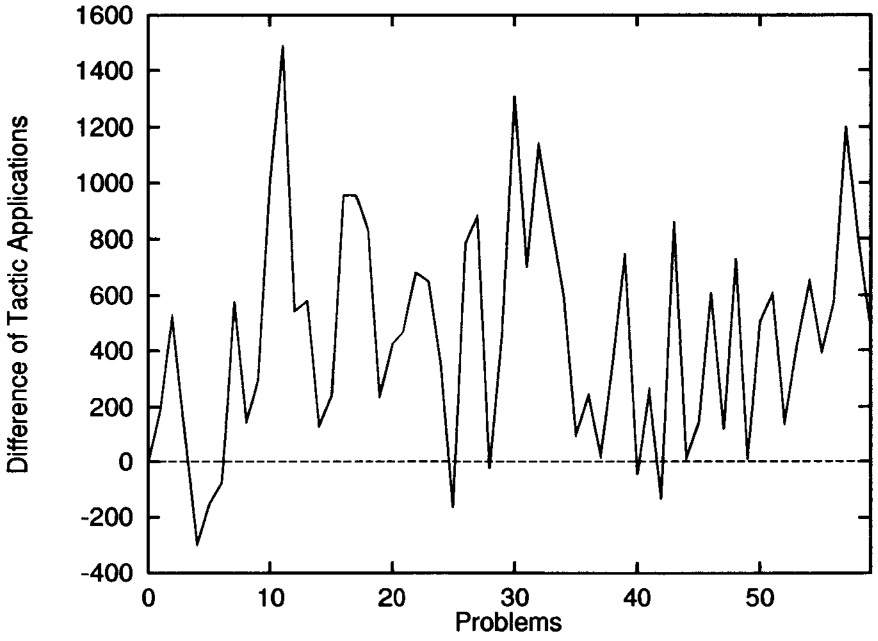

Fig. 5.8: Difference of tactic application numbers between RBR and CABINS with the hybrid repair (positive values mean that RBR applied more tactics than CABINS with the hybrid repair

presents that in a considerable number of problems CABINS with the hybrid repair was superior to RBR in terms of the number of tactic applications.

5.4 Effects of Using Failure Cases in CABINS

The ways to use failure cases in the validated repair and the interruptive repair seem to be independent to each other and compatible. To analyze the effects of combining two repair methods in the hybrid repair, the tactic application numbers of the validated repair and the interruptive repair were compared with that of the hybrid repair.

Table 5.11: Problem solving efficiency test between CABINS with several repair methods by Etzioni's signed rank test

	z	p-value
Hybrid Repair vs Validated Repair	6.29	0.00
Hybrid Repair vs Interruptive Repair	6.46	0.00

Table 5.11 shows the results of Etzioni's hypothesis test in which the null hypotheses state that CABINS with the validated repair and CABINS with the interruptive repair can repair schedules at least as efficiently as CABINS with the hybrid repair. From the p-values of Table 5.11, both H_0's can be rejected at the significance level of 0.05.

The graph in Fig. 5.9 shows the subtraction of the tactic application number of CABINS with the hybrid repair from that of CABINS with the validated repair for each problem in the experiment. And the graph in Fig. 5.10 shows the subtraction of the tactic application number of CABINS with the hybrid repair from that of CABINS with the interruptive repair for each problem in the experiment. The both graphs present that in a considerable number of problems CABINS with the hybrid repair was superior to both CABINS with the validated repair and CABINS with the interruptive repair in terms of the number of tactic applications. Consequently, it is acknowledged that CABINS with the hybrid repair is more efficient than CABINS with the interruptive repair and CABINS with the validated repair. Therefore, by combining the validated repair and the interruptive repair, a synergistic effect emerges in improving the repair process efficiency.

Experiment results in the previous sections show that failure cases can contribute to reducing the number of tactic applications in CABINS. But the reduction by the three repair methods (i.e. validated repair, interruptive repair and hybrid repair) is derived from the more elaborated case retrieval procedure of exploiting failure cases, which is computationally more expensive, than that of original CABINS. Hence, superiority of these repair methods of using failure cases to CABINS in efficiency needs to be evaluated in terms of CPU time spent on repairing schedules.

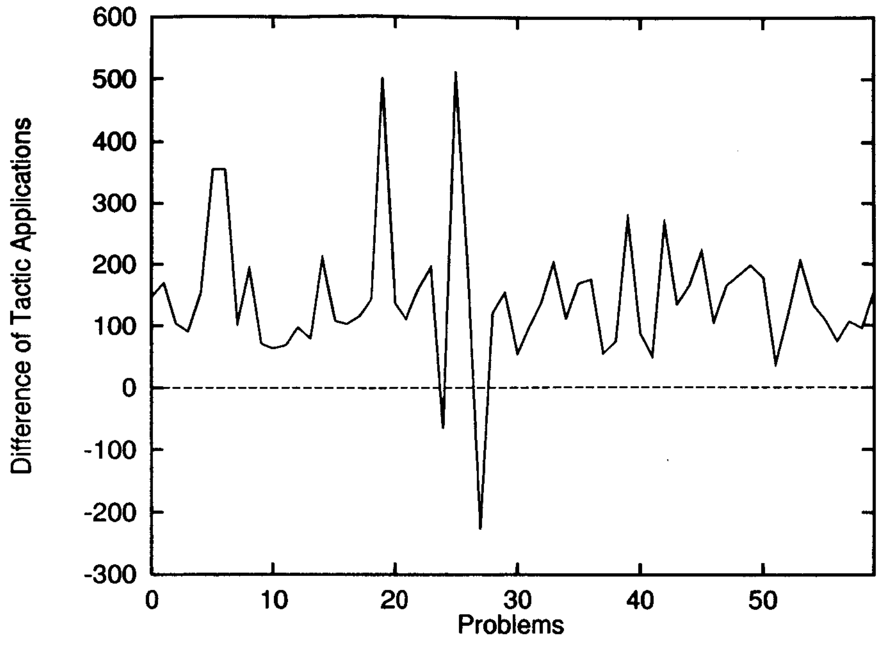

Fig. 5.9: Difference of tactic application numbers between CABINS with the validated repair and CABINS with the hybrid repair (positive values mean that CABINS with the validated repair applied more tactics than CABINS with the hybrid repair)

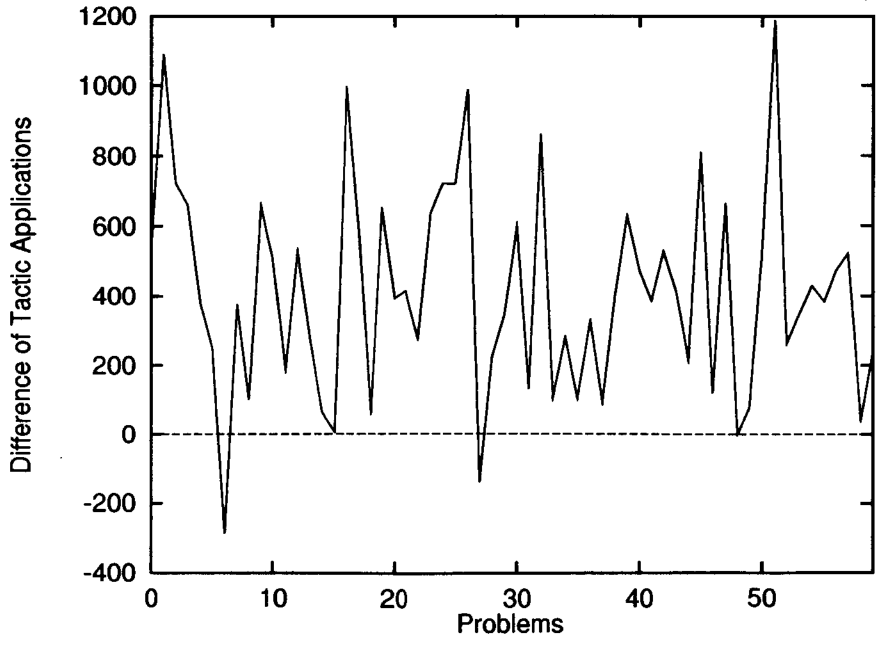

Fig. 5.10: Difference of tactic application numbers between CABINS with the interruptive repair and CABINS with the hybrid repair (positive values mean that CABINS with the interruptive repair applied more tactics than CABINS with the hybrid repair)

Table 5.12: Number of tactic applications and CPU time of CABINS with several repair methods

	Tactic Applications	CPU time (sec)
Validated Repair	920.3	435.9
Interruptive Repair	1186.8	528.1
Hybrid Repair	776.6	424.1
CABINS	2206.2	551.1

Table 5.12 presents that averaged CPU time results of repairing sixty scheduling problems by CABINS with three repair methods are smaller than that of CABINS. The ratio of CPU time reduction is much smaller than that of the tactic application number because the more complicated case retrieval processes of using failure cases kill the effect of tactic number reduction in terms of CPU time. This can be further explained by the fact that the number of failure cases is much greater than that of success cases, which makes the case retrieval process of utilizing failure cases even more expensive than that of using only success cases.

5.5 Summary

This chapter has shown the following results of a set of empirical studies in terms of learning search control knowledge for improving problem solving efficiency.

- CABINS can acquire the control knowledge for improving its problem solving efficiency by means of the past failure information for validating a use of a selected repair tactic to avoid repeating similar failures. CABINS with the validated repair has achieved high quality schedules more efficiently than RBR.
- Another repair method (the interruptive repair) that exploits the failure experiences to switch repair attentions when faced to the difficult problems. The experimental results have shown that CABINS with the interruptive repair has outperformed the repair efficiency of RBR.
- The hybrid repair method, which combines the above two repair methods, can further improve the repair efficiency of CABINS.

The author believes that the uses of CBR in the space of failures that have been shown in this chapter are domain independent methods of learning search control knowledge that allow the problem solver to improve its efficiency while preserving solution quality in domains without strong domain knowledge.

Chapter 6

Scaling-Up a Case Base

The source of power in CABINS definitely derives from its case base. With a sufficient number of cases in the case base, CABINS can produce high quality of solutions in an efficient way. In AI research, it is generally believed that "the power of an intelligent program to perform its task well depends primarily on the quantity and quality of knowledge it has about that task" [Buchanan & Feigenbaum, 1982]. But, how true is this proposition? Does the quality of solutions improve linearly with increase of the available knowledge? Or, how about the efficiency of problem solving? This chapter discusses about the cost and profit of having more cases in CABINS in terms of solution quality and problem solving efficiency. Although the costs of having more cases include the cost of acquiring cases (i.e. cost of human experts involved in case acquisition) and the cost of storing cases (i.e. cost of memory resource), they are not treated as critical issues in this thesis. The reasons are as follows: (1) In CABINS, no special interaction is required to a human expert for acquiring a new case, since all of the information elicited from a user for a new case (i.e. repair action selection, result evaluation and effect estimation) is acquired in the course of plain problem solving by the user. (2) The cost of memory device is decreasing drastically and becoming less critical in the design of the intelligent system.

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 guaranteeing overall satisfactory performance of the case-based system. Unfortunately, it is very difficult to answer this question in general owing to the ill-structuredness of the problem and the approximate nature of CBR (since no causal model is available). The author believes, however, that there exists some appropriate size of the case base which will give relatively satisfactory results in terms of solution quality and problem solving efficiency without excessive overhead for case acquisition and case retrieval from the case base.

In the following sections of this chapter, firstly the effect of case base size in CABINS is empirically investigated in terms of solution quality and problem solving efficiency. Then, the optimal size of a case base will be discussed from the viewpoint of overall problem solving performance. In the experiments of this chapter, the behavior of CABINS and that of CABINS with the hybrid repair are compared using the same problems, objectives and case bases that were used for the Experiment-A in Section 4.1.

6.1 Effect of Case Base Size on Quality

Increase of case base size might have beneficial or harmful effects on the CABINS performance in terms of solution quality. The possible explanations of these effects are as follows:

- Better quality solutions: After having new cases that successfully repaired the novel problems, CABINS increases possibility of improving solution quality by later reusing these cases for similar problem situations.
- Poorer quality solutions: There are two explanations that additional cases can have a deleterious effect on the quality of solutions found by CABINS. One is that incorrect cases may lead CABINS to produce poorer solutions. The other is that increase of the number of failure cases in the case base can force CABINS with the hybrid repair to give up further exploration for better solutions even when there is a good chance to find some.

The graph in Fig. 6.1 compares the performance of CABINS with different sized case bases in terms of solution quality. The comparison baseline is the performance of RBR. The comparison was calculated as:

$$\frac{Repair\_Ratio\_by\_CABINS}{Repair\_Ratio\_by\_RBR}$$

Thus, a value greater than 1.0 indicates that the solution produced by CABINS at the indicated case base size exceeded the solution quality produced by RBR. In the graph, the results of every 100 cases increment are plotted until the size of case base exceeds 2000 and then the results of every 1000 cases increment are plotted until 8000 are shown. And the graph in Fig. 6.2 compares the performance of CABINS with the hybrid repair in the same way. To get the case bases of different sizes, an appropriate number of cases for each problem class were randomly selected and deleted from the case base used in the Experiment-A in Section 4.1. 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 [Bareiss, 1989].

From the graphs, the schedule quality is found to improve with increased case base size. 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, 2000 cases) capture well

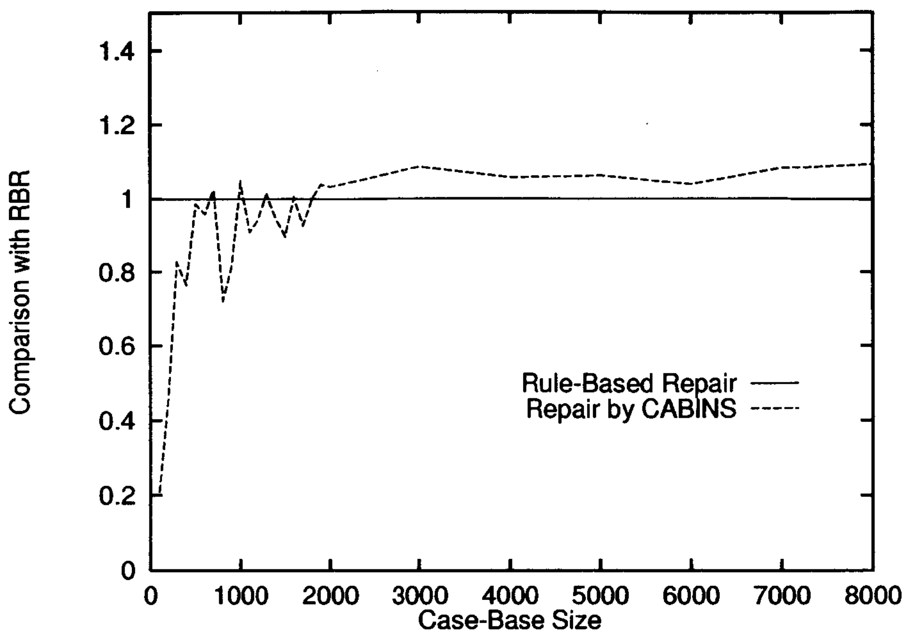

Fig. 6.1: Effect of case base size on solution quality by CABINS

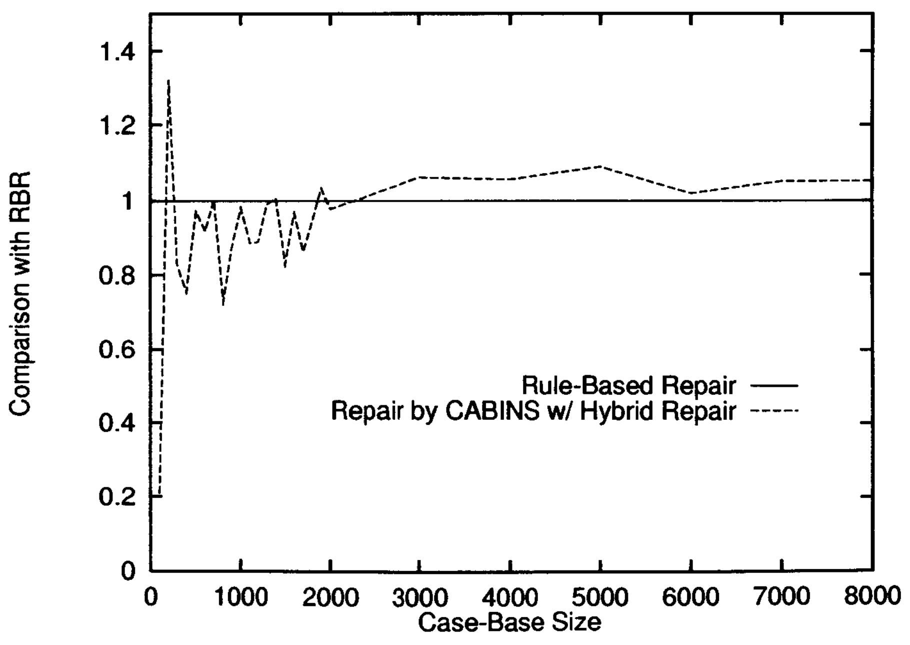

Fig. 6.2: Effect of case base size on solution quality by CABINS with the hybrid repair

characteristics of the *user preference model* in CABINS that is created by the records of solution evaluations by the user in the various problem contexts, 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, information about a different part of the model is added and improves the capability of CABINS producing solutions of better quality. In addition, the graphs indicate that the solution quality produced by CABINS does not degrade with increase of case base size because all the cases in CABINS are "correct" in the sense that human experts can always evaluate the result of repair action applications correctly in a consistent way. In other words, CABINS has no case which has the wrong evaluation of the repair result. And there is no significant difference in the behavior of CABINS and CABINS with hybrid repair. Therefore it is concluded that the size of case base in CABINS can improve the solution quality only at the early stage of case accumulation. Neither deleterious nor favorable effect on solution quality can be found by further accumulation of cases.

Section 4.3.1 compared the solution quality by two types of the CABINS implementation that calculates the feature distance with/without feature salience in a class of the selected repair action. In that section, the experiment results did not well verify the author's hypothesis: the results without feature salience were of comparable quality to the results with feature salience. In the section, to explain the results of the experiment, the author made an assumption that even CABINS without feature salience information could make high quality solutions because a large number of cases (more than 8000 cases) were available.

The graph in Fig. 6.3 shows the difference of the solution quality between CABINS with feature salience and CABINS without feature salience in terms of the comparison to RBR performance. The difference of the solution quality was calculated as:

$$\frac{Repair\_Ratio\_With\_Salience - Repair\_Ratio\_Without\_Salience}{Repair\_Ratio\_With\_Salience}$$

Hence, the positive values in the graph indicate that CABINS with feature salience produced the better solutions than CABINS without feature salience. The graph indicates that, when the size of a case base is small (i.e. about less than 1000), the performance of CABINS with feature salience is usually much better than CABINS without feature salience. Hence, it is concluded that the feature salience information is effective for improving the quality of solution especially when only a small number of cases are available.

6.2 Effect of Case Base Size on Efficiency

Increase of case base size might have beneficial or harmful effects on the CABINS performance in terms of problem solving efficiency, too. The possible explanations of these effects are as follows:

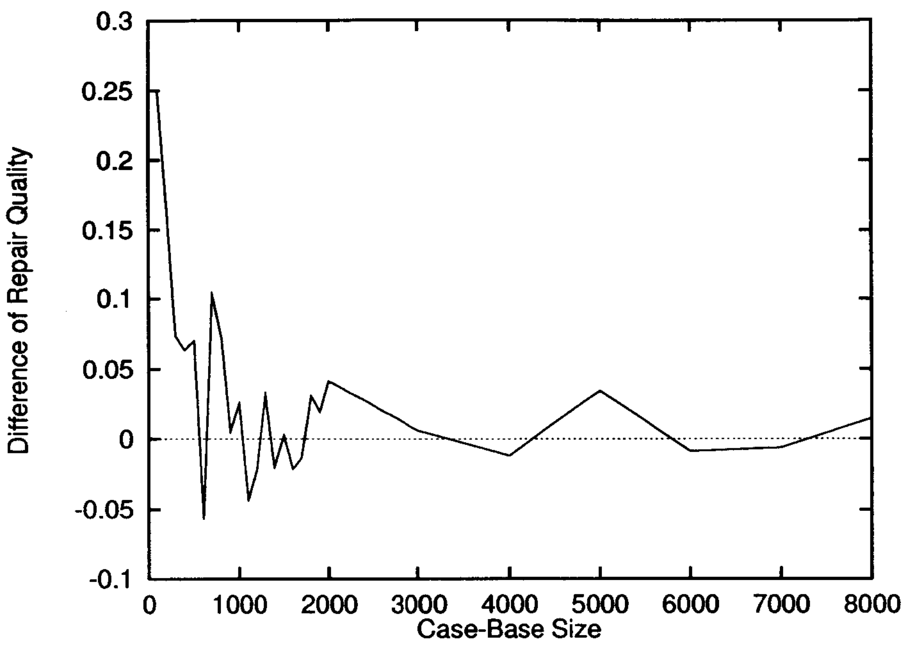

Fig. 6.3: Effect of case base size on quality difference by CABINS with/without salience information

- More efficient problem solving: By increasing the number of failure cases, the number of different failure types that become known to CABINS increases. Hence, CABINS with the hybrid repair can avoid repeating a large variety of failures, thus reducing the search time.
- Less efficient problem solving: There are two explanations that additional cases can worsen the search efficiency of CABINS. One is that irrelevant cases may suggest CABINS to expand the search tree in fruitless directions. The other is that more cases require more time for retrieval of the matching cases.

In this section, we take a number of repair tactic applications as an indicator of the problem solving efficiency by CABINS. In other words, we do not take into consideration of the case retrieval time for problem solving efficiency in this section. The problem of trading off the increased case retrieval cost and the reduced search cost from the viewpoint of the problem solving efficiency will be discussed in Section 6.3.

The graph in Fig. 6.4 compares the performance of CABINS with different sized case bases in terms of number of tactic applications. The comparison baseline is the performance of RBR. The comparison was calculated as:

$$\frac{Tactic\_Applications\_by\_CABINS}{Tactic\_Applications\_by\_RBR}$$

Thus, the value less than 1.0 in the graph indicates that problem solving by CABINS at the case size was more efficient than RBR. In the graph, the results of every 100 cases

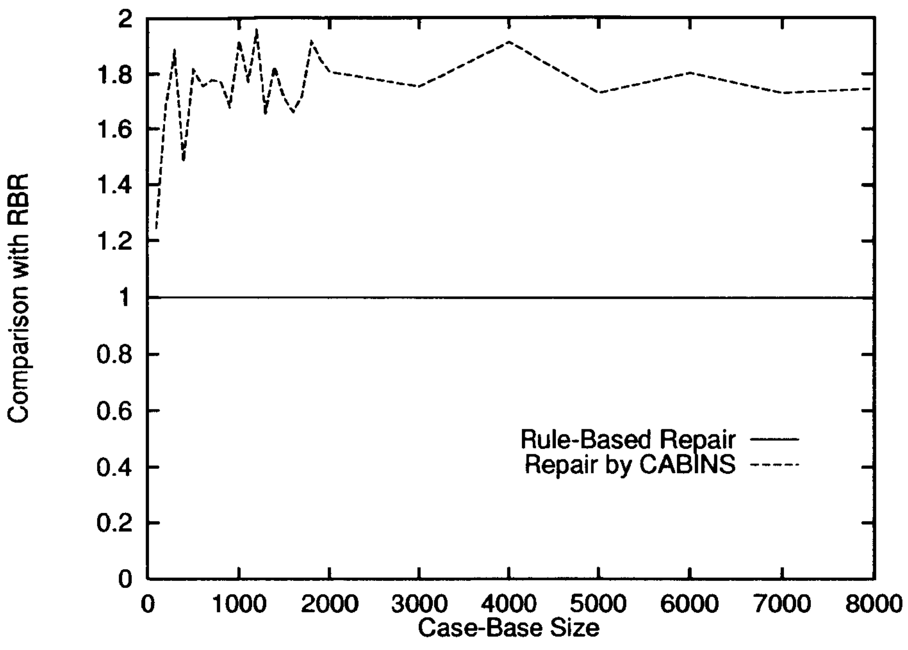

Fig. 6.4: Effect of case base size on problem solving efficiency by CABINS

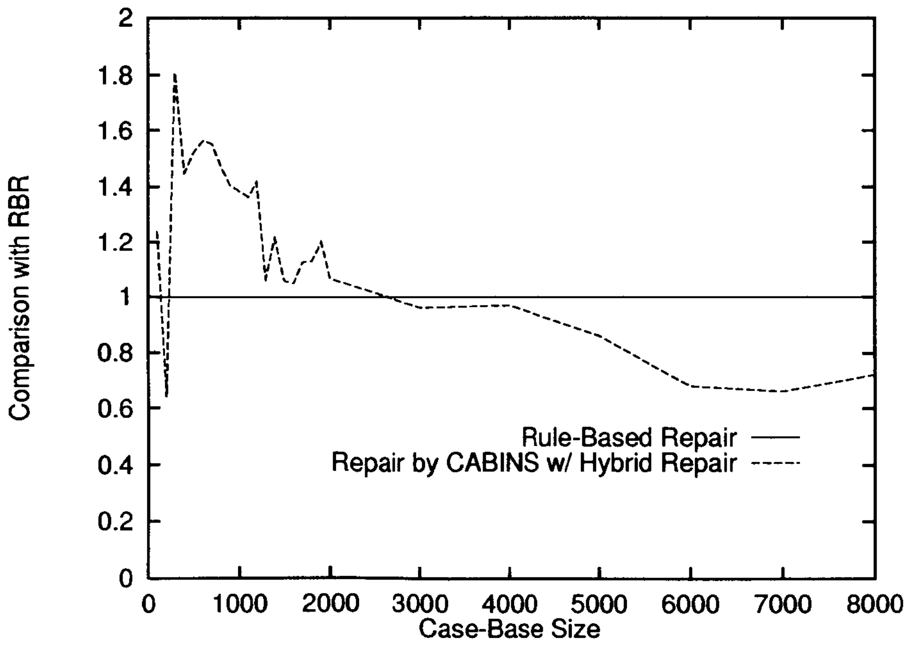

Fig. 6.5: Effect of case base size on problem solving efficiency by CABINS with the hybrid repair

increment are plotted until the size of case base exceeds 2000 and then the results of every 1000 cases increment are plotted until 8000 are shown. And the graph in Fig. 6.5 compares the performance of CABINS with the hybrid repair in the same way. The case bases of different sizes are created in the same manner with the experiments in Section 6.1.

The graph in Fig. 6.4 shows that CABINS does not improve nor deteriorate the problem solving efficiency with the increase of cases. This is because the acquired cases in CABINS are always "correct" and do not suggest any irrelevant search effort for solution improvement. However, the graph in Fig. 6.5 shows that CABINS with the hybrid repair can improve its problem solving efficiency drastically in accordance with the increase of case base size. This result indicates that CABINS with the hybrid repair acquires effective search control knowledge for speeding-up through the accumulation of cases. The search efficiency reaches a stationary value after a large amount of cases (about 6000 cases) are acquired. This can be explained by the more complicated nature of the *repair control model* which is created by the history of the repair action selections and their results in the different problem situations. More diversity of cases are required to create the repair control model than the cases that merely build the user preference model.

6.3 What Is Optimal Case Base Size?

The more cases are gathered, the more time is required for matching cases in case retrieval. Since, cases of the same class are basically structured flat in the form of a simple linear list in CABINS (see Fig. 2.3), the time for case retrieval increases linearly as the size of the case base grows. Fig. 6.6 shows the CPU time for a single case retrieval in CABINS with the hybrid repair method.

To find an optimal size for the case base, the tradeoff between increase of the case retrieval time and decrease of the repair action applications need to be balanced in accordance with growth of the case base size. The generalized form of this problem is called the *utility problem* [Minton, 1988] in machine learning research. As is experimentally shown in Fig. 6.6, the case retrieval time increases linearly with the increase of case base size. Thus, the case retrieval time can be formulated as:

$$Retrieve\_Time = Size \times T_r$$

where *Retrieve_Time* is the time required for each case retrieval, *Size* means the size of the case base and T_r is case retrieval time from a unit case base with a single case (i.e. a gradient of the graph in Fig. 6.6).

Since, in CABINS repair process, the total number of repair goal/strategy selection is less than 10% of the number of repair tactic applications and the process of repair goal/strategy is computationally negligible as compared with repair tactic process, it can be assumed that CABINS' execution time should mostly derive from the time for tactic selections, tactic

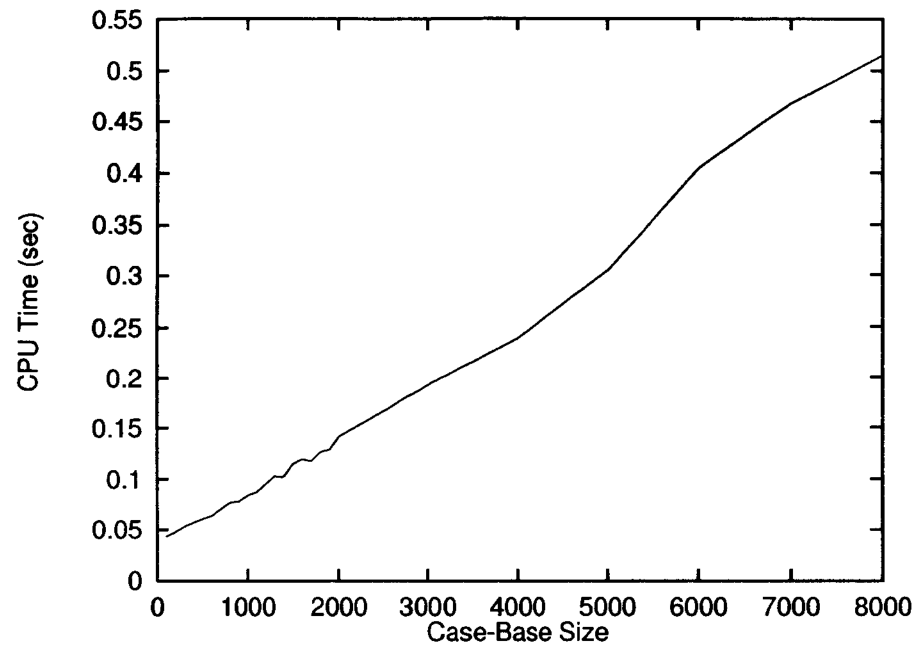

Fig. 6.6: Effect of case base size on case retrieval time in CABINS with the hybrid repair

applications and result evaluations. Since the time required to execute a repair tactic is irrelevant to the case base size and can be modeled as a constant, and a case needs to be retrieved once for a repair tactic selection and once for a result evaluation, the time of the entire repair process by CABINS is formulated as:

$$\begin{aligned} \textit{Repair\_Time} &= \textit{Tactic\_Applications} \times (T_a + 2 \times \textit{Retrieve\_Time}) \\ &= \textit{Tactic\_Applications} \times (T_a + 2 \times \textit{Size} \times T_r) \end{aligned}$$

where *Repair_Time* stands for the total time for repair by CABINS, *Tactic_Applications* means the number of repair tactic applications and T_a is the average time of one repair tactic application.

The graph in Fig. 6.7 shows the simulated CPU time results for CABINS with the hybrid repair, which are based on the experiment results in Fig. 6.5. Three lines in the graph depict the results under the different value assignments of T_a. The value of T_r was calculated from the graph in Fig. 6.6. When a repair tactic application is considered to be a computationally light process, the total execution time of CABINS is dominated by the case retrieval time and increases linearly with the increase of case base size. In this situation, the optimal case base should be a minimal case base that can produce high enough quality solutions (a case base with around 2000 cases in the experiments). If the time for a repair tactic application is moderate, the execution time of CABINS does not change much in accordance with the increase of cases. The best case base for this situation should be any case base that can produce high quality solutions without deteriorating the problem solving efficiency (a case base whose size is from 2000 to 6000 cases in the experiments). And, if the repair tactic is

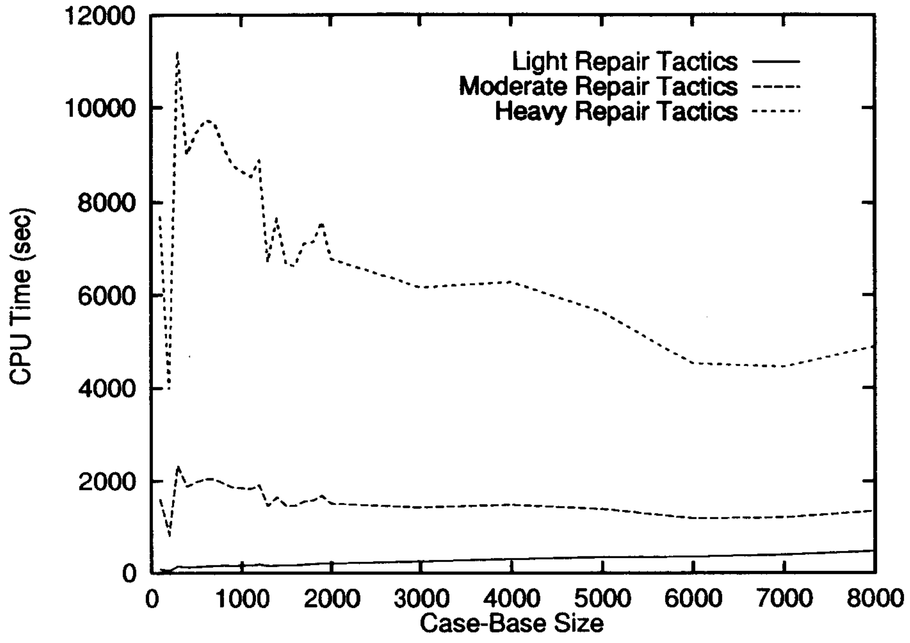

Fig. 6.7: Simulated CPU time results with different types of repair tactics in CABINS with the hybrid repair

a computationally expensive process, CABINS execution time is mainly influenced by the time for repair tactic applications. The optimal case base in this context should be the case base that has fully acquired search control knowledge for efficiency improvement (in the experiments, a case base with about 6000 cases). As a result, optimality of the case base size changes with the tradeoff between the computational cost of case retrieval and that of a repair action application.

The graph in Fig. 6.8 depicts the CPU time for execution of CABINS with the hybrid repair endowed with the different sizes of the case base. The graph shows that CPU time increases almost monotonously with the increase of case base size. This is because the repair tactic application in CABINS is computationally non-expensive. As was explained in Section 3.2.1, CABINS repair tactic invokes the simple constraint propagation process to move the focal_activity and resolve the constraint violations caused by the move. And in the small job-shop scheduling problem used in the experiments (i.e. scheduling of ten orders, fifty activities and five resources), the constraint propagation can be executed very efficiently.

The graph in Fig. 6.9 [Sycara & Miyashita, To be published] presents the CPU time for execution of the old version of CABINS [Miyashita & Sycara, 1994d; Miyashita & Sycara, 1994c]. The graph shows that CPU time is minimized when the case base has about 1000 cases in it. This can be explained from the fact that in the old version of the CABINS implementation, the repair tactic is implemented as the constraint satisfaction process based on the heuristics in [Sadeh & Fox, 1990] (see [Miyashita & Sycara, 1994b] for more

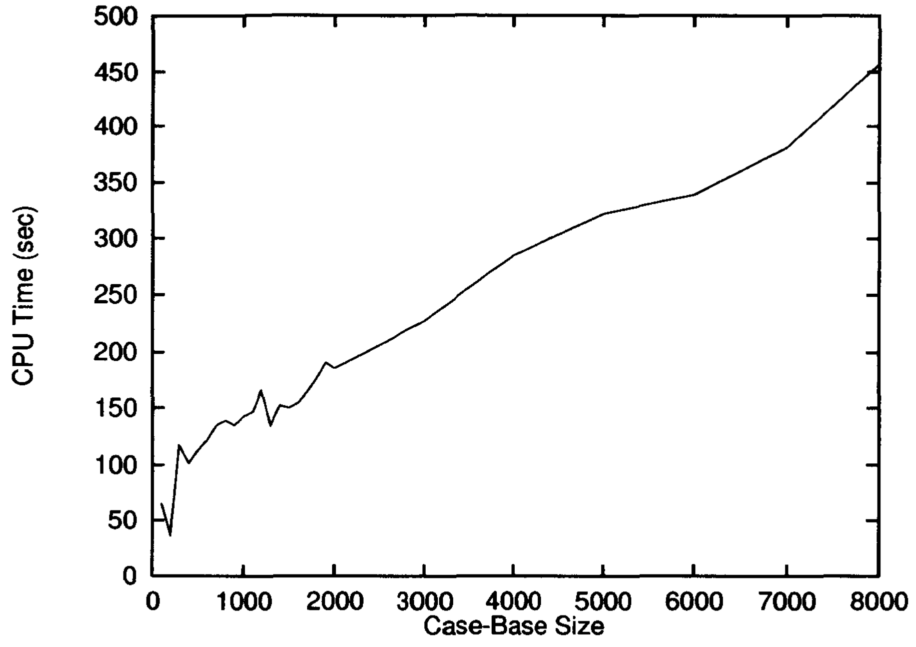

Fig. 6.8: Effect of case base size on execution time by CABINS with the hybrid repair

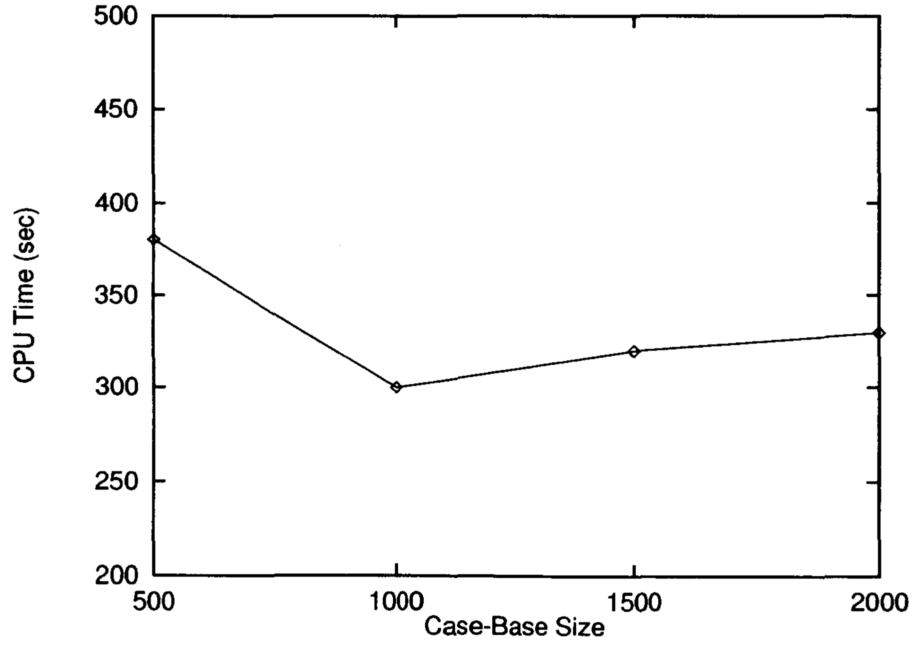

Fig. 6.9: Effect of case base size on execution time by the old version of CABINS

details about repair tactics in the old version of CABINS). Since the constraint satisfaction technique used in the old CABINS heavily relies on the backtrack based search, the repair tactic process was computationally expensive.

These experiment results could give good evidence that the author's hypothesis shown in Fig. 6.7 was correct concerning to the relation among the total execution efficiency, cost of repair tactics and the size of a case base.

6.4 Summary

This chapter has shown the effects of case accumulation in CABINS' problem solving performance. The following results have been empirically demonstrated:

- CABINS improves its solution quality with the increase of cases. But with a relatively small size of the case base (about 2000 cases), the quality improvement saturates.
- With the increase of cases, CABINS with the hybrid repair improves its problem solving efficiency in terms of the number of tactic application. For learning the control knowledge to improve its search efficiency to the utmost, CABINS with the hybrid repair requires a large number of cases (around 6000 cases).

Then, at the end of the chapter, the optimal size of the case base has been discussed. It has analytically been hypothesized that the optimal size of the case base should be determined to balance the tradeoff between case retrieval cost and repair action application cost. The quantitative analysis on the utility of the case base has been made in terms of solution quality and problem solving efficiency in the job shop scheduling problem and validated the hypothesis.

Chapter 7

Related Research

As was explained in the previous chapters, CABINS has novel contributions in the three areas of case-based reasoning, knowledge acquisition & machine learning, and scheduling & planning. This chapter analyzes the related research in these areas and compares it with CABINS.

7.1 Case-Based Reasoning

The repair-based method in CABINS is related to the repair-based methods that have been previously used in case-based planning systems (e.g. [Hammond, 1989; Kambhampati & Hendler, 1992; Veloso, 1992]). But, previous case-based systems for incremental solution revision have been motivated only by concerns of computational efficiency, preserving plan correctness rather than improving plan quality, and have assumed the existence of a strong domain model that provides feedback as to plan correctness.

CHEF [Hammond, 1989] is one of the first planning systems that utilize case-based reasoning methodology. CHEF's initial input is a set of goals to include different tastes and ingredients in a type of dish. At the first step, CHEF tries to anticipate any problems that might arise in a plan by means of a memory of past failures that is linked to the features that participated in causing the failures. And then, CHEF searches for a plan that satisfies as many of its current goals as possible while avoiding the problem that it has predicted. CHEF alters the plan it has found to satisfy any goals from the input that are not yet achieved with the help of the domain-specific modification rules (e.g. To use chicken as an ingredient, add boning step before chopping it). Once it completes a plan, CHEF runs a simulation using a set of domain causal rules. If any goal is unsatisfied or any unfavorable state is detected by the simulation, CHEF builds a causal explanation of why the failure has occurred using its understanding of the domain causality. The explanation of the failure is used to find the different strategies for repairing it. Along with repairing a plan, CHEF repairs its memory structure so that it will not make the same mistake again.

CHEF does this in two ways : one is to store the repaired plan so that it can be used again, and the other is to figure out which features should be blamed for a failure using the same causal explanation used for repairing the plan so that CHEF can anticipate this failure in later inputs. Thus, CHEF depends on the rich domain knowledge and the complete causal model of the domain for its repair capability.

PRIAR system [Kambhampati & Hendler, 1992] can modify existing plans, which are build by the hierarchical nonlinear planner based on NONLIN [Tate, 1977], to accommodate a variety of externally imposed constraints. Their method is formalized as a process of removing inconsistencies in the causal and teleological structure of the plans called *validation structure*. At the first step, PRIAR maps the objects of a plan in its plan library to the objects of a new planning problem and marks the inconsistencies in the mapped plan. And then these inconsistencies are resolved by a variety of repair methods. And finally, PRIAR *reduces* a resultant plan into an executable one. Since PRIAR is based upon formalization of the hierarchical nonlinear planner, its approach for plan modification is formal and domain-independent. And, Veloso [Veloso & Carbonell, 1992] has developed the similar system which combines the general problem solver based upon *means-ends analysis* (PRODIGY [Minton *et al.*, 1989]) and *derivational analogy* method [Carbonell, 1986]. While PRIAR puts its emphasis only on the systematic modification on the existing plan to meet the specifications of a new problem, her system learns to retrieve a better past case by improving similarity metrics from the interaction with the general problem solver.

Because both systems are based on the hypothesis that the plan built by their planner is causally and teleologically correct, failures arising from the incompleteness of the planner's domain knowledge cannot be rectified. And, in both systems there is no guarantee that the modified plan is (at least) as preferable as the plan that might have been found by planning from scratch. The goal of the systems is to find the *satisficing* plan efficiently rather than to find the *optimal* plan.

In the realistic problems such a factory scheduling problem, as any human expert cannot be expected to have the complete knowledge of the domain, the complete causal structure of the problem can't be deduced. And complete causal model, if any, might be too complicated to be efficiently exploited because of tight constraint interaction in the problem. And, since there are lots of favorable measures to evaluate "goodness" of the solutions, repairs are necessitated to compensate for the inadequacies of a generative method not only in satisfactoriness but also in optimality. From the above reasons, the primary motivation of CABINS is to automatically enhance the domain model and improve the quality of resultant solutions as well as improving the problem solving efficiency.

CYCLOPS [Navinchandra, 1991] is an integrated architecture of constraint satisfaction / optimization and case-based reasoning for multi-objective optimization problem solving to support a human designer in making an innovative conceptual design. In addition to searching an optimal solution, CYCLOPS explores to find some sub-optimal solutions by relaxing some of given design criteria. If a design alternative is found, CYCLOPS tries to find new criteria (such as exceptionally good view of the lake) in it, which may change the designer's view to the design, by reminding a similar case from the past experience. This emergence of new criteria is thought to be a source of innovation. And, if CYCLOPS finds that the alternative solution has "unfavorable" problems by matching with the precedents and the domain rules, the system seeks the precedent case which solved the similar problems and applies the adapted solution of the case to the current situation. CYCLOPS has many aspects common with CABINS in its concept. But, since CYCLOPS is intended to work as an interactive system with a human designer, the system requires lots of help from its user in selecting the best case, evaluating a repair result, dealing with a repair failure and avoiding an infinite loop of repair cycle.

CABINS is also related to to the previous case-based knowledge acquisition systems (e.g. SIZZLE [Offutt, 1988], Protos [Bareiss, 1989]). SIZZLE is a tool for building expert systems whose problem solving method is *extrapolate-from-a-similar-case*. SIZZLE elicits two kinds of knowledge from a domain expert: (1) a collection of problems and their associated expert-validated solutions, and (2) knowledge about how to vary a solution in an appropriate manner as a function of variation of the corresponding problem. These types of knowledge are exploited to find a solution of a input problem as follows: first the most similar past problem to the input problem is retrieved from a case base that stores a collection of problem-and-solution pairs, and then a solution of the retrieved problem is extrapolated based on differences between the input problem and the retrieved problem. But in many real-world problems with a complex causality, simple extrapolation of solutions cannot produce good solutions because mapping between case feature space and solution space cannot be well formulated and a size of an available case base is always too small to find a close enough case for input problems. In those ill-structured problems, a search procedure is required to derive a high quality solution of a given problem from a candidate solution. Exploiting past experiences, CABINS acquires the control knowledge for producing better solutions through search and improving search efficiency.

Protos is a case-based classifier system under the guidance of a human teacher. When presented with the description of an entity to be classified, Protos attempts to recall a past case and to explain its similarity to the given entity. When a human teacher decides that Protos fails to correctly classify or adequately explain a result, Protos interacts with the teacher to obtain the correct classification and an explanation of why it is correct. Protos learns by selectively retaining cases and the explanations in its knowledge base. Protos addressed many important aspects of case-based learning, such as learning indexing, learning additional domain knowledge, generalization of cases and selective retention of cases. But its strength is largely dependent upon its confinement of the domain to a classification task and its teacher's capability to formulate appropriate explanations and to present them to the system in the predefined explanation language. In the ill-structured domain, the approach of Protos cannot be applied because there is no explicit relation between case features and underlying reasons of user's judgment about modification results.

In CABINS' approach 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 ultimate expertise lies in her/his ability to perform a consistent evaluation of the results of problem solving.

7.2 Knowledge Acquisition and Machine Learning

Most of the expert systems developed so far were typically constructed with tremendous endeavors by knowledge engineers, and research in knowledge acquisition and machine learning is aimed at reducing the knowledge engineer's burdens in the development of expert systems.

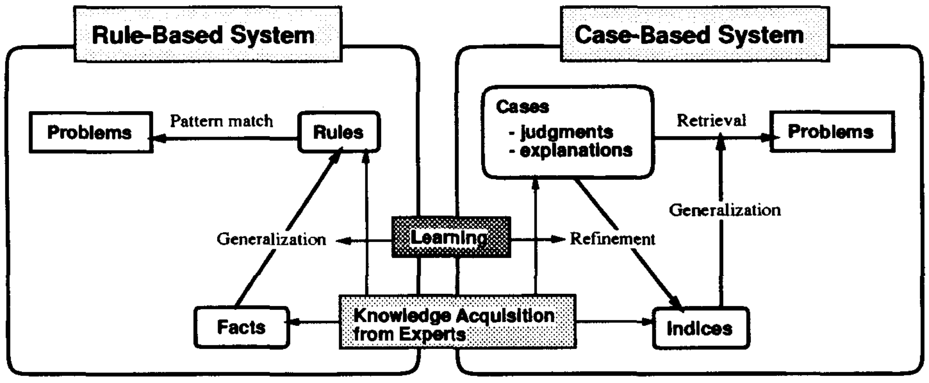

Fig. 7.1: Comparison of learning and knowledge acquisition in a rule-based system and a case-based system

A case-based system takes a quite different approach to knowledge acquisition and learning from that of a traditional rule-based system. Fig. 7.1 depicts the schematic comparison in knowledge acquisition and machine learning between a rule-based system and a case-based system. In the rule-based system, knowledge acquisition methods support an expert to define the vocabulary of a problem domain and tune the contents of existing rules. Machine learning methods such as decision tree induction and explanation-based learning can transform facts describing examples of expertise into more generalized rules, which are selected through the pattern matching procedure and applied to problems. To be noted, this generalization process always requires other sets of knowledge in the domain such as generalization hierarchy of domain concepts and complete domain theory, which are difficult to prepare in the ill-structured problems.

In the case-base system, knowledge acquisition methods aim not only to support definition of case description features and indices by an expert but also to extract his/her judgment and explanation in the process of actual problem solving. Learning in the case-based system takes place in the refinement of indexing contents with the increase of cases describing different problem solving episodes. Hence, learning in the case-base system requires no additional knowledge of application domains from the expert. To be applied to problems, cases are retrieved through approximate matching of selected indices among other features in the cases, thus generalization occurs when re-using cases even in the case-based system that has a flat case memory structure without abstraction hierarchy.

7.2.1 Knowledge Acquisition

Recently there have been a number of research activities of building a generic framework for acquiring domain knowledge to solve ill-structured problems.

KADS is a methodology for the structured and systematic development of knowledge-based systems, which aims to provide software engineering support for the knowledge-engineering process. KADS provides the following three important frameworks: (1) a framework to structure knowledge within a *four layered* model (i.e. *domain, inference, task, strategy* layers), (2) reusable modeling frameworks for partial models of expertise (i.e. *interpretation models*) and for primitive problem-solving actions (i.e. typologies of knowledge sources), and (3) a framework of the knowledge-based system development process (i.e. transition from generic models, to interpretation models, to conceptual models, to design models and finally to operational systems). To make a model for a particular application in KADS, first an interpretation model (i.e. *inference structure*) is selected from generic models based upon a taxonomy of task types. Then, the selected interpretation model is modified into a conceptual model through the applications of operators such as renaming, refinements, additions, simplifications and deletions. These operators are applied based on the three types of characteristics of the application domain: epistemic, pragmatic and computational task features [Schreiber & Wielinga, 1993]. Once a conceptual model is completed, it is transformed into a design model with the help of a *structure-preserving design* method. The structure-preserving method preserves the information content and structure present in the knowledge-level model (i.e. a conceptual model) in the final artifact. The method uses the conceptual model as a skeletal architecture of the artifact and extends it by considering inference methods, domain indexing and access functions, run-time data storage, and input/output functions [Schreiber, 1993]. On one hand, the KADS approach has comprehensible generality and flexibility in the development of knowledge-based system. But, on the other hand, users (both knowledge-engineers and domain experts) of KADS are required to do substantial amounts of analysis and design by themselves. The computational supports to the users on these aspects are current research topics in the KADS-II project.

PROTÉGÉ-II [Puerta *et al.*, 1992] is a knowledge-acquisition shell that provides a task-modeling environment for knowledge engineers and a knowledge-editing environment for application experts. In PROTÉGÉ-II, knowledge engineers can construct the model of application tasks by selecting and configuring appropriate methods from a library of role-limiting methods and smaller-grained methods called mechanisms. PROTÉGÉ-II generates knowledge-acquisition tools (i.e. knowledge-editors) that derive control strategies from those task models defined by knowledge engineers based on the needs of the particular task and domain. Application experts use the knowledge-editors to enter the knowledge of application problems. PROTÉGÉ-II is trying to overcome the following shortcomings in the role-limiting methods: (1) it assumes that problem-solving behavior can be defined in terms that are completely domain independent, which is not true in many applications, and (2) developers cannot always select appropriate problem-solving methods from the predefined set of the methods to model their application tasks. Although PROTÉGÉ-II has been successfully applies to the episodic skeletal-plan refinement task, more computerized supports to knowledge engineers in selecting appropriate mechanisms for their tasks need to be developed and the effectiveness of knowledge-editors that can be produced by PROTÉGÉ-II in other tasks and domains is still to be investigated.

McDermott and his group at DEC have been developing a framework for exploiting usable and reusable programming constructs called *mechanisms* [Klinker *et al.*, 1991]. A mechanism is usable if it can be used to automate a task by a non-programmer who understands and performs the task. A mechanism is a reusable if it can be employed for several domains and tasks. Mechanisms are identified by a developer's knowledge of a task with the help of a system called Spark. To map the mental model of a domain expert (i.e. the developer) and the computational model of mechanisms, Spark uses the *enterprise model* whose activities are represented in the developer's concept and terminology and explicitly mapped to corresponding mechanisms by an implementor of Spark. The identified mechanisms are integrated into a skeletal problem-solving method of the task using a handcrafted configurations. The domain specific knowledge which each mechanism needs to perform the task is elicited and encoded by a system called Burn. Burn manages a collection of specialized knowledge acquisition tools for each mechanism. The knowledge acquisition tool associated with a mechanism instantiates the mechanism into a situation-specific mechanism. Spark and Burn are similar in the concept to PROTÉGÉ-II. The biggest difference is the support of the enterprise model in Spark, which is aimed to ease the task model definition by a domain expert not by a knowledge engineer.

Chandrasekaran and his team have been doing a research on the *generic task* (GT) [Chandrasekaran, 1988] with the aim of identifying "building blocks" of reasoning strategies such that each of the types is both generic and widely useful as components of complex reasoning tasks. The generic task defines the functionality of the task, a vocabulary of knowledge constructs and a vocabulary for inference and control of the task. They have found a couple of generic tasks such as hierarchical classification, hypothesis matching and abductive hypothesis assembly, and designed high-level programming languages such as CSRL and DSPL to provide programmers with the primitives that allow the required knowledge to be directly described for any domain in which the task can be performed.

Thus, GT is a tool to help programmers develop the knowledge-based systems in the certain application domains.

The crux of above research activities is building a library of "role-limiting" problem solving methods involved in a complicated real-world problem and knowledge acquisition tools suitable for each of them. If such problem solving methods are sufficiently general, the corresponding knowledge acquisition tools can be re-used repeatedly for developing a variety of knowledge-based systems. In CABINS, task-level analysis found that a single problem solving method, case-based reasoning, was applicable to a large variety of subtasks in ill-structured optimization problems. The task structure identified by the analysis provides effective guidelines on case feature definitions, thus reducing necessary efforts of eliciting the knowledge required for applying case-base reasoning to specific problems. Since case-based reasoning can be used for several kinds of (sub)tasks such as classification, proposal, modification and evaluation, this method used in the development of CABINS has a wider range of applicability than traditional knowledge acquisition methods based on the notion of role-limitation.

7.2.2 Learning for Quality Enhancement

Until recently there have been only a few research efforts in acquiring useful knowledge to improve the problem solver's capability of producing "better" solutions.

ASK [Gruber, 1989] is an interactive knowledge acquisition tool that elicits *strategic knowledge* from the user in the form of justifications for action choices, and generates strategy rules that operationalize and generalize the expert's advises. ASK's knowledge acquisition process relies heavily on the interactions with the user : (1) eliciting the critique from the user, (2) performing credit assignment of the existing rules, (3) eliciting justifications from the user by asking relevant features of the current situations, (4) generating and generalizing a strategy rule, with user's guidance in generalization, and (5) verifying a rule by asking a user's approval. Since ASK can support only the reactive style of reasoning, the feature descriptions used in ASK's rules, which are invented by the user, have to be accurate in predicting the effects of actions and expecting the utility of the effects so that ASK can avoid global pitfalls. In CABINS the exploitation of failure experiences compensates for the incompleteness of the case features to improve its performance in terms of both solution quality and problem solving efficiency.

Learning apprentice systems such as LEAP [Mitchell, Mahadevan, & Steinberg, 1985; Mahadevan *et al.*, 1993] and CAP [Dent *et al.*, 1992] learn to improve their performance by observing and analyzing the problem solving steps of their users. LEAP learns the design rules for VLSI circuits using an *explanation-based learning* technique. CAP learns scheduling rules for arranging meetings during its normal use as an electric calendar by *inductive learning* methods (ID3 and backpropagation). Unlike CABINS, these systems compile training experiences into the generalized forms (i.e. rules and weight-tuned network). As a result, these systems suffer from the difficulty of dealing with exceptional data which are common in the real-world complicated problems. For example, the experimental results reported in [Miyashita & Sycara, 1993] have shown that keeping the case base rather than inducing rules from it and then utilizing these rules for problem solving results in superior performance.

Anapron [Golding & Rosenbloom, 1991] combines rule-based and case-based reasoning for improving the performance of a rule-based system. Cases are stored as negative (supplemental) exemplars of rules and used to override the decision made by the rules. Cases are retrieved based upon its *compellingness*, which is a combined measure of similarity and accuracy of a case. The experiment results in the domain of pronouncing English surnames show that the performance (i.e. accuracy of pronounce) of the rule-based system increased 12-17% by adding cases into the rule-base. Since the task of Anapron is a simple classification, the applicability of the approach to the more complicated task such as scheduling needs to be investigated.

EASe [Ruby & Kibler, 1992] can learn the *exceptions* not covered by an initial problem solver in order to further optimize its solutions. In the problem solving, EASe takes the following steps: (1) The goal decomposition component of EASe decomposes the problem of solving the goal to that of solving each of the individual subgoals and protecting them once solved, and orders these subgoals. (2) The constrained search component attempts to solve each subgoal successively. (3) The memory component is called when the constrained search component has failed to find a solution. The memory component takes as input a context and returns the impasse solution stored in an episode whose context matches with the current problem context. (4) When constrained search and memory have failed to improve the current impasse, the unconstrained search component searches for a solution. (5) If a solution is found by unconstrained search, the learning component stored the episode after determining the appropriate indices. EASe has been applied to the logic synthesis design problem and shown the significant improvement on the solution quality. But in EASe the tradeoffs between subgoals of the explicitly given optimization goal are encoded in the goal decomposition component, although CABINS induces these tradeoffs from cases. And there has been no report on the efficiency gain by EASe's learning capability and the quality comparison with other optimization methods.

COMPOSER [Gratch, Chien, & DeJong, 1993] automatically learns an effective domain-specific search strategy given a general problem solver with a flexible control architecture. The approach can be characterized as a hill-climbing search in the space of the possible strategies to optimize the expected *utility* of problem solving, using the statistics to evaluate performance over the expected problem distribution. Unlike CABINS, the utility must be given explicitly as a real valued function that is a measure of the goodness of the behavior of the problem solver. COMPOSER has been applied to the spacecraft communication scheduling problem, but so far there has been no report on the experiment results.

Pérez has been developing the learning mechanism of quality-enhancing control knowledge on the top of PRODIGY nonlinear planner [Pérez & Carbonell, 1994]. The learning algorithm is given a domain theory (operators and inference rules) and a domain-dependent objective function that describes the quality of the plans. The algorithm analyzes the problem-solving episodes by comparing the search trace of the planner solution given the current control knowledge and another search trace corresponding to a better solution (better according to the evaluation function). The latter search trace is obtained by letting the problem solver search further until a better solution is found, or by asking a human expert for a better solution and then producing a search trace that leads to that solution. The algorithm explains why one solution is better than the other and outputs the search control knowledge that leads future problem solving towards better quality plans. The learning process translates knowledge about plan quality encoded in a domain-dependent plan evaluation function into control knowledge that the planner can use at problem solving time. Unlike CABINS, the work has a limitation in the form of an evaluation function such that it is additive on the cost of the individual operators and does not capture the existence of tradeoffs between different quality factors.

7.2.3 Learning for Efficiency Improvement

In many domains, finding a solution at all requires a considerable amount of search. In these domains, the key information needed to create a useful problem solver is the knowledge of how to search through the problem space efficiently.

The STRIP system [Fikes, Hart, & Nilsson, 1972] learns *macro operators* that encapsulate successful operator sequences. When the system reaches a goal, it analyzes the successful operator sequence in order to identify a general set of conditions under which the sequence is guaranteed to apply. The generalized effects of the operator sequence become the effects of the macro operator. The addition of macro operators to the system can improve planning efficiency in two ways: (1) the system no longer needs to reason about the macro operator's implicitly encoded intermediate steps, and (2) the macro operators encode *experiential bias*, since a system that uses macro operators in preference to primitive operators will tend to explore previously successful operator sequence first. However, in the ill-structured problems which CABINS has been applied to, it is difficult to apply the above approach since the tight constraint interactions of the domain make the interactions of operators unpredictable.

HACKER [Sussman, 1975] is one of the first repair-based planning systems that explicitly reasoned about goal interactions. And more importantly, it has the ability to learn *generalized bug*. HACKER plans by solving subgoals independently and then simulating the resulting plan to determine how plan components interact. If the plan is flawed, HACKER employs its library of generalized bugs to correct the problem. A generalized bug associates a set of conditions that describe a problem in a plan with a repair that rectifies the problem. HACKER learns generalized bug by observing problems that occur during the construction of plans and then generalizing the problems and the repair plans. However, HACKER must repeat its failures to recognize that it was in a similar situation before and then make use of the learned generalized bug because HACKER's indexing does not reflect the fact that a particular problem having to do with a goal interaction was solved by a particular repair plan. Hence, unlike CABINS, HACKER cannot avoid repeating the similar problems.

LEX [Mitchell, Ugoff, & Banerji, 1983] applies a *version space algorithm* in the domain of symbolic integration to induce search control knowledge (operator selection rule) from positive and negative instances of states in which an operator should be applied, which the system generated during problem solving. LEX has limitations in that (1) the learning capability is strongly tied to the pre-defined language used to describe heuristics, and (2) it lacks in the ability to analyze the role of a particular search step in leading to a problem solution. The efforts of overcoming the above limitations have led to the research of the analytic learning method called *explanation-based learning (EBL)* [Mitchell, Keller, & Kedar-Cabelli, 1986].

SOAR [Laird, Rosenbloom, & Newell, 1986] employs a learning method called *chunking*. Chunking operates by summarizing the information examined while processing a subgoal for resolving an *impasse* and acquires control knowledge to help make decisions in similar subgoal processing. The chunking mechanism bears a strong resemblance to EBL in analyzing the aspects of the state that were accessed during the lookahead search in subgoal processing and forming the left-hand side of the chunk (i.e. the acquired rule).

Minton has developed one of the learning components of the PRODIGY planning system based on EBL [Minton, 1988]. The PRODIGY system combines learning from success, failure and goal interactions in a single explanation-based architecture. The system can learn from these different types of examples because it has multiple theories. After each planning episode, PRODIGY's EBL module explains the failures, successes and goal interactions that the planner encountered, and for each example, it learns a search control that lets the system select, reject or prefer the alternative the next time it encounters a similar problem. PRODIGY also explicitly addresses the utility problem, which occurs when the overhead of testing the applicability of learned knowledge degrades a system's performance rather than improves it. To determine the utility of its search control rules, PRODIGY monitors their average match cost and match frequency. It discards rules that are determined to be useless or harmful.

Although EBL is a strong learning method, it has the drawbacks as follows: (1) the method requires a *complete* domain theory to ground the generalization of the failures and successes encountered in the unique example to be analyzed, (2) the EBL learner performs an *eager* efforts of understanding and generalizing completely and correctly the *local and individual* decisions of the problem solving episode, and (3) EBL applies its learned knowledge only when the new decision making situation *exactly* matches the learned operationalized control knowledge. From the above reasons, it can be said that the some domains may be incompletely specified for which EBL is not able to generate deductive proofs. And, in complex domains, EBL can become very inefficient with long deductive chains producing complex rules for situations that may seldom, if any, be exactly repeated. Finally, the localized character of the learned knowledge in EBL is a source for an increase of the control knowledge available to match and select from at decision-making time. Case-based (or analogical) reasoning can be seen as a major relaxation of the above EBL's drawbacks: (1) the domain theory does not have to be completely specified as in CBR the problem solving episodes are loosely interpreted and not fully generalized, (2) the learning efforts are performed incrementally on an "if needed" basis at storage, retrieval and adaptation stage in CBR when new similar problems occur, (3) the complete problem solving episode can be interpreted as a global decision-making experience where subparts can be reused as a whole, and (4) CBR can reuse partially matched learned or accumulated experiences.

Recently there have been a few research activities that use the analogical reasoning method to learn search control knowledge for problem solving (e.g. EUREKA [Jones, 1993] and DÆDALUS [Langley & Allen, 1993]). The DÆDALUS system generates a plan based on the means-ends analysis and uses domain-specific knowledge to constrain and direct the search. DÆDALUS stores this knowledge in a *probabilistic concept hierarchy*, a tree of concepts that summarizes experience at different levels of abstraction using probabilistic descriptions. The learning process of the system alters the structure of the concept hierarchy and the probabilities stored therein. DÆDALUS stores each node of a problem solving trace as a case in the concept hierarchy with a description of a set of predicates, a set of differences and the operator used to solve it, organizing it via internal node that abstracts the cases which occur below it in the hierarchy. Given a new problem, the system invokes a variant of COBWEB [Fisher, 1987] to find the stored case with a similar structure and uses it to direct search. From the experiments in the blocks-world problem, it was shown that DÆDALUS could improve its problem solving efficiency. However, unlike CABINS, since DÆDALUS learns only from successful experiences, the feasibility of DÆDALUS in the ill-structured domain, where clear definitions of predicates and differences of operators are not available, has to be further investigated. Although EUREKA uses a semantic network as memory structure and employs the more sophisticated retrieval mechanism (called the *spreading-activation* algorithm) than that of DÆDALUS, the methodology is similar to DÆDALUS and shares the same problems.

7.3 Scheduling and Planning

The application of CABINS to the scheduling problem shares the same motivations and goals with the work in [Mckay, Buzacott, & Safayeni, 1988] where the motivations for interactive user manipulation of schedules is presented. In that work, the system monitors 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 they have not developed any deliberation on knowledge representation, communication protocol between a user and the scheduling system and mechanism to keep knowledge base consistent.

CABINS is rooted on concepts and mechanisms of a long line of research in constraint-directed scheduling [Fox, 1983; Smith *et al.*, 1986; Sadeh, 1991], which generates schedules by incrementally constructing and merging partial schedules. In the research, various properties and aspects of the scheduling problem have been extensively investigated in the framework of a constraint satisfaction problem and sophisticated procedures and techniques for constraint-directed scheduling have been proposed. 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 [Sadeh, 1991], or statically determined control level model for application of repair actions [Ow, Smith, & Thiriez, 1988]). CABINS' 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.

CABINS generates schedules by repair based method in the space of complete schedules. In this respect it is similar to [Zweben, Deale, & Gargan, 1990; Zweben *et al.*, 1993; Minton *et al.*, 1990; Biefeld & Cooper, 1991]. In [Zweben, Deale, & Gargan, 1990; Zweben *et al.*, 1993] 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 [Zweben *et al.*, 1993]. [Zweben *et al.*, 1992b] has studied the tradeoff among the amount of perturbations, speed of convergence to a conflict free schedule and schedule quality measured in terms of number of violated resource constraints. In [Minton *et al.*, 1990] 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. 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 problems [Muscettola, 1993]. This is because the min-conflict heuristic doesn't use any knowledge from the domain to focus its attention to a certain area of the schedule, it becomes very inefficient when the number of possible repair choices is large (such as a detail scheduling with the fine granularity of time). In [Biefeld & Cooper, 1991] schedule modifications are procedurally encoded. Small snapshots of the scheduling process, called chronologies, are used to focus the search using information gained incrementally during the scheduling process to locate, classify and resolve bottlenecks.

In [Zweben *et al.*, 1992a] 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 does not 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 scheduled based on the acquired objectives.

In more general context of the planning problems, GORDIUS [Simmons, 1992] aims to compensate for inadequacies of its initial problem solving method using Generate-Test-Debug (GTD) methodology. GORDIUS uses associational reasoning (rule-based reasoning) to efficiently generate initial hypotheses, then the tester simulates hypotheses to determine whether they are correct, and finally the debugger uses detailed causal reasoning to repair faulty hypotheses. Hence, GORDIUS requires rich domain knowledge such as encapsulated nearly independent association rules and the causal model of domain. The goal of GORDIUS is not to pursue an optimal solution, but to find one of plausible solutions. And in GORDIUS, the past experiences of debugging are not exploited to refine incomplete association rules which generate initial hypotheses. Consequently, GORDIUS cannot improve its problem solving efficiency by reducing the cost of expensive causal reasoning in the debugging stage.

Howe has suggested FRA (Failure Recovery Analysis) [Howe, 1992] to debug an incomplete planner in the problems that does not have a strong domain model. The objective of her research is not repairing each plan produced by a planner but repairing the planner to avoid making similar failures. FRA's repair process comprises the following four steps. The first step searches significant dependencies between plan failure recovery efforts and subsequent plan failures by means of a statistical method. The second step maps these dependencies to the structures in the planner's knowledge known to be susceptible to failures. The third step constructs explanations of how the observed dependencies might have been produced using knowledge of the structures. And finally, the fourth step recommends a set of possible modifications on a planner based upon the explanations. This recommendation is not intended to be implemented by the system itself, but a (human) designer of a planner decides what would be the best modification and whether the failure is worth avoiding at all. This method is being examined in the field which has simple causal interactions (simulation of forest fire fighting), but an application to a more complicated domain is still open to further investigation.

7.4 Summary

This chapter compared the work in this thesis with several other research efforts in the three research areas. The comparison made here is certainly neither exhaustive nor complete. Instead it tries to distinguish the characteristic aspects of CABINS.

In general, a comparison between CABINS and other systems has revealed its uniqueness in the following senses:

- CABINS designates the usage of case-based reasoning in ill-structured problems and provides the methodology to support the description of case features through task-level analysis, thus extending the feasibility of case-based reasoning to the domains that has been avoided by the traditional case-based reasoning research.
- CABINS learns the knowledge for solution quality enhancement by accumulating the repair-based optimization experiences. Neither evaluation functions nor extensive explanations from its user are necessary for CABINS to learn such knowledge.
- In the problems without a strong domain theory, CABINS can analogically learn the control knowledge for problem solving efficiency improvement through past repair failure episodes.
- CABINS provides a generic framework of schedule optimization with learning capabilities, thus eliminating the necessity of modeling a user's objective function and developing elaborated heuristics for each scheduling problem.

Chapter 8

Concluding Remarks

The experimental results in this thesis have shown that the CBR-based repair method has the potential to capture different user optimization preferences and performs well in terms of producing high quality solutions as compared with other general-purpose optimization methods. As compared with simulated annealing, one of the widely used repair-based optimization methods, CBR-based repair produces solutions of comparable quality with substantial computational savings. In addition, CBR-based repair exhibits desirable speed-up effects by learning from its past failure experiences while maintaining high solution quality.

To conclude the thesis, the author will attempt to answer the question "what makes the approach powerful?". The author believes the power of the approach stems from the following four factors:

1. Revision-based approaches by making available a complete assignment (a complete schedule for the experiment domain in this thesis) provide more information that can guide search as compared with constructive methods where only a partial assignment is available [Minton *et al.*, 1992]. CBR-based revision method in CABINS captures such relevant information in case features and exploits it as contextual information during case retrieval.

2. The case features were able to capture some important domain regularities, such as repair flexibility, through task-level analysis of the indexing vocabularies. 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.

3. 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. The abstraction hierarchy of repair actions was useful to generalize the problem structure of the current problem and map it into the space of the past cases.

4. Experimental results and discussions presented in Chapter 6 support the hypothesis that the cases CABINS acquired and used in the reported experiments appear to cover the whole problem space in a substantial way, thus allowing CBR-based repair to take advantage of this coverage.

8.1 Summary

In this thesis, the author has advocated a framework for knowledge acquisition and iterative revision for solving the ill-structured optimization problems. The approach utilizes CBR-based mechanisms for recording user's decisions in the course of repairs. The approach is predicated on the existence of (1) a set of vocabularies, which can extract and represent an aspect of the problems, and (2) repair heuristics, each of which operates with respect to a particular local view of the problem and offers selective advantages for improving solution quality. The author believes that these types of knowledge can be easily elicited from a domain expert by preparing adequate ontologies and abstraction hierarchy of the task and presenting them to the expert. The approach aims to capture and re-use user preferences and judgments for improving the solution quality and problem solving efficiency. The capability of acquiring user optimization preferences is important in the 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. And learning control knowledge for speeding-up problem solving is also important in the ill-structured domains, because in such domains even human experts lack effective heuristics to solve problems efficiently while maintaining the high solution quality. More importantly, CABINS can acquire the cases through user interaction during the process of solution improvement without imposing undue overhead on the user.

The scientific contributions of this research are as follows:

1. Contributions to Case-Based Reasoning:

   - Expanding the feasibility:

     Case-based reasoning has been applied to the problems, such as diagnosis, legal judgment and design, where a well-defined form of a "case" has naturally been used in problem solving and/or causality of the domain is clearly understood. The thesis designates the usage of case-based reasoning in ill-structured problems and provides the methodology to support the description of case features through task-level analysis and improve its performance via failure experiences, thus extending the feasibility of case-based reasoning to the domains that do not have clear understandings of "what is the case for the problem".

- Scale-up in the case base size:

  The thesis consists of empirical validation of the hypothesis that accumulation of cases improve the performance of the case-based reasoner in terms of both solution quality and problem solving efficiency.

2. Contributions to Machine Learning:

   - Preference learning:

     The thesis provides the method of learning the context-dependent preferences of users and producing the high quality solutions based on the learned preferences.

   - Speed-up learning:

     The thesis advocates the methods of utilizing the failure experiences for validating search decisions and avoiding similar failures, thus enabling to improve the search efficiency in the ill-structured domains without a complete domain theory.

3. Contributions to Scheduling:

   - Optimization without explicit objectives:

     The thesis provides the generic framework of case-based iterative revision that can solve a complicated schedule optimization problem without explicitly define objectives, which cannot be solved by the traditional optimization methods such as simulated annealing.

The experiments in the job shop scheduling domain have shown the following results:

1. Solution quality enhancement

   The experiment results indicate that different scheduling objectives implicitly reflected in the case base differentially bias the schedule repair procedure and CABINS can achieve as high quality of solutions as its teacher (i.e. the rule-based system in the experiments). Further experimental results show that for well defined objectives reflected in the case base, CABINS produces schedules with higher, or at least as good, quality as compared with other repair-based optimization methods, such as simulated annealing.

   The quality of solutions are improved by exploiting the effect information of each repair action application for effective case retrieval. But little enhancement is achieved by considering salience of case features for each repair action.

2. Problem solving efficiency improvement

   The experiment results show that CABINS can improve its problem solving efficiency by exploiting repair failure experiences in two ways: One ways is called the validated repair strategy, which allows CABINS to apply a repair action only after the repair action is validated as possibly effective for repairing the current problem. The other way is called the interruptive repair strategy, which allows CABINS to shift its attention to another problem when the current problem seems too difficult to be solved.

   And further improvement is achieved by integrating the above two repair strategies (called the hybrid repair strategy), thus making CABINS about 37% more efficient than its teacher (i.e. the rule-based system) while maintaining its solution quality.

3. Case accumulation effects

   The experiments demonstrate that CABINS improves its solution quality with the increase of cases. But with relatively small size of the case base (about 2000 cases), the quality improvement saturates. And, CABINS with the hybrid repair strategy improves its problem solving efficiency in terms of the number of tactic applications with the increase of cases. For learning the control knowledge to improve its search efficiency to the utmost, CABINS with the hybrid repair strategy requires a large number of cases (around 6000 cases).

8.2 Limitations and Future Work

From a theoretical point of view, there remain two questions in the present research on CABINS. The first question is, in evaluating the performance of CABINS, to what extent the information captured in cases from one set of problems can transfer to another set of 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. [Musick & Russell, 1992; Cheeseman, Kanefsky, & Taylor, 1991]) as to their structure and properties that characterize "easy" or "hard" problem instances, similar characterization of the ill-structured problem such as a job shop schedule optimization problems is currently an open research problem (e.g. [Cheeseman, Kanefsky, & Taylor, 1991; Baker, 1974]). Due to the tight constraint inter-dependencies in job shop scheduling 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 SPT 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 [Adams, Balas, & Zawack, 1988; Baker, 1974]. 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 the analysis in the thesis since CABINS does not have an explicit objective function, but instead aims at capturing implicitly context-dependent user preferences.

The other question on CABINS is how to construct and maintain optimal case base. CABINS suffers from a clear utility problem in that, its retrieval cost per a repair action and overall problem solving cost increase with more experiences accumulated beyond its optimal size, rather than decreasing as desired. Although several methodologies have been proposed for controlling the content of the knowledge base (including the case base) to maximize its *utility function* through *filtering* knowledge (e.g. [Minton, 1988; Bareiss, 1989; Aha, Kibler, & Albert, 1991; Markovitch & Scott, 1993]), it is difficult to apply these methodologies to the ill-structured problems without strong causal knowledge of the domain. Although the experiment results in Chapter 6 show that the cases acquired in CABINS cover the problem space in a fairly evenly distributed fashion, this might not be true in the more complicated problems that have many exceptional situations. In such problems, keeping the case base size optimal is not sufficient for maintaining the performance of the system optimal. Further analysis should be made to identify a priori the portion of a case base which will contribute to problem solving so that one can reduce the harm caused by increased case retrieval time while keeping high solution quality. For that purpose, it is required to develop the indexing vocabularies to capture not only the regularities of the problem structure but also the exceptionalities of the problems.

From a practical point of view, there remain much work to be done in CABINS, since the current status of the CABINS system is still a research prototype. First, the feasibility of the CABINS approach in the scaled-up problems has to be tested. Current problem sets used in the experiments of this thesis are much smaller in the size than usual real-world factory scheduling problems. A more efficient case retrieval mechanism such as the one using a discrimination network will be required to apply CABINS to larger problems without spoiling its problem solving efficiency. Second, the applicability of CABINS to other problem domains than scheduling problems also needs to be examined. Although the author thinks that CABINS' methodology of using success and failure cases in the repair-based methods for an optimization task is domain-independent, it must be verified by extensive applications of CABINS to different problems such as design and planning. Lastly but not least, in order to accelerate the application of CABINS, the more elaborated human-computer interface must be devised in CABINS for easing the expert's burdens of defining case features and adding new cases in the case base.

Appendix A

Scheduling Problem Instance

The following is an example of the scheduling problem used in the experiments. The problem belongs to the Class-6 problem set which has two bottlenecks and dynamic range parameters.

```
Scheduling_time_granularity = 10;

Resource resource1 = {
  Efficiency = 9;
  Quality    = 8;
  Price      = 14000000;
};

Resource resource2 = {
  Efficiency = 7;
  Quality    = 1;
  Price      = 10000000;
};

Resource resource3 = {
  Efficiency = 7;
  Quality    = 4;
  Price      = 7000000;
};

Resource resource4 = {
  Efficiency = 8;
  Quality    = 9;
  Price      = 4000000;
};

Resource resource5 = {
  Efficiency = 10;
  Quality    = 6;
  Price      = 2000000;
};

Client client1 = {
  Time_severity    = 1;
  Quality_severity = 1;
};

Client client2 = {
  Time_severity    = 6;
  Quality_severity = 8;
};

Client client3 = {
  Time_severity    = 6;
  Quality_severity = 8;
};

Client client4 = {
  Time_severity    = 1;
  Quality_severity = 2;
};

Client client5 = {
  Time_severity    = 6;
  Quality_severity = 5;
};

Order order1 = {
  Release_date = 300;
  Due_date     = 1540;
  Customer     = "client3";
  Activities   = (
    Activity activity1_1 = {
      Inventory_cost = 10;
      Duration = 60;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ( );
      Next_activities = ( "activity1_2" );
    };
    Activity activity1_2 = {
      Inventory_cost = 2;
      Duration = 40;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ("activity1_1" );
      Next_activities = ( "activity1_3" );
    };
    Activity activity1_3 = {
      Inventory_cost = 8;
      Duration = 90;
      Machinery = ( "resource3" );
      Prev_activities = ("activity1_2" );
      Next_activities = ( "activity1_4" );
    };
    Activity activity1_4 = {
      Inventory_cost = 2;
      Duration = 30;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ("activity1_3" );
      Next_activities = ( "activity1_5" );
    };
    Activity activity1_5 = {
      Inventory_cost = 5;
      Duration = 80;
      Machinery = ( "resource5" );
      Prev_activities = ("activity1_4" );
      Next_activities = (  );
    };
  );
};

Order order2 = {
  Release_date = 310;
  Due_date     = 1340;
  Customer     = "client3";
  Activities   = (
    Activity activity2_1 = {
      Inventory_cost = 9;
      Duration = 90;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ( );
      Next_activities = ( "activity2_2" );
    };
    Activity activity2_2 = {
      Inventory_cost = 3;
      Duration = 100;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ("activity2_1" );
      Next_activities = ( "activity2_3" );
    };
    Activity activity2_3 = {
      Inventory_cost = 2;
      Duration = 120;
      Machinery = ( "resource3" );
      Prev_activities = ("activity2_2" );
      Next_activities = ( "activity2_4" );
    };
    Activity activity2_4 = {
      Inventory_cost = 4;
      Duration = 50;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ("activity2_3" );
      Next_activities = ( "activity2_5" );
    };
    Activity activity2_5 = {
      Inventory_cost = 7;
      Duration = 100;
      Machinery = ( "resource5" );
      Prev_activities = ("activity2_4" );
      Next_activities = (  );
    };
  );
};
```

```
Order order3 = {
  Release_date = 40;
  Due_date     = 1350;
  Customer     = "client4";
  Activities   = (
    Activity activity3_1 = {
      Inventory_cost = 1;
      Duration = 70;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ( );
      Next_activities = ( "activity3_2" );
    };
    Activity activity3_2 = {
      Inventory_cost = 7;
      Duration = 110;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ("activity3_1" );
      Next_activities = ( "activity3_3" );
    };
    Activity activity3_3 = {
      Inventory_cost = 3;
      Duration = 130;
      Machinery = ( "resource3" );
      Prev_activities = ("activity3_2" );
      Next_activities = ( "activity3_4" );
    };
    Activity activity3_4 = {
      Inventory_cost = 3;
      Duration = 40;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ("activity3_3" );
      Next_activities = ( "activity3_5" );
    };
    Activity activity3_5 = {
      Inventory_cost = 5;
      Duration = 100;
      Machinery = ( "resource5" );
      Prev_activities = ("activity3_4" );
      Next_activities = (  );
    };
  );
};

Order order4 = {
  Release_date = 160;
  Due_date     = 1340;
  Customer     = "client3";
  Activities   = (
    Activity activity4_1 = {
      Inventory_cost = 3;
      Duration = 90;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ( );
      Next_activities = ( "activity4_2" );
    };
    Activity activity4_2 = {
      Inventory_cost = 3;
      Duration = 50;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ("activity4_1" );
      Next_activities = ( "activity4_3" );
    };
    Activity activity4_3 = {
      Inventory_cost = 2;
      Duration = 150;
      Machinery = ( "resource3" );
      Prev_activities = ("activity4_2" );
      Next_activities = ( "activity4_4" );
    };
    Activity activity4_4 = {
      Inventory_cost = 3;
      Duration = 90;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ("activity4_3" );
      Next_activities = ( "activity4_5" );
    };
    Activity activity4_5 = {
      Inventory_cost = 7;
      Duration = 140;
      Machinery = ( "resource5" );
      Prev_activities = ("activity4_4" );
      Next_activities = (  );
    };
  );
};

Order order5 = {
  Release_date = 180;
  Due_date     = 1550;
  Customer     = "client2";
  Activities   = (
    Activity activity5_1 = {
      Inventory_cost = 7;
      Duration = 50;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ( );
      Next_activities = ( "activity5_2" );
    };
    Activity activity5_2 = {
      Inventory_cost = 2;
      Duration = 40;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ("activity5_1" );
      Next_activities = ( "activity5_3" );
    };
    Activity activity5_3 = {
      Inventory_cost = 10;
      Duration = 120;
      Machinery = ( "resource3" );
      Prev_activities = ("activity5_2" );
      Next_activities = ( "activity5_4" );
    };
    Activity activity5_4 = {
      Inventory_cost = 9;
      Duration = 40;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ("activity5_3" );
      Next_activities = ( "activity5_5" );
    };
    Activity activity5_5 = {
      Inventory_cost = 1;
      Duration = 140;
      Machinery = ( "resource5" );
      Prev_activities = ("activity5_4" );
      Next_activities = (  );
    };
  );
};

Order order6 = {
  Release_date = 270;
  Due_date     = 1390;
  Customer     = "client2";
  Activities   = (
    Activity activity6_1 = {
      Inventory_cost = 6;
      Duration = 60;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ( );
      Next_activities = ( "activity6_2" );
    };
    Activity activity6_2 = {
      Inventory_cost = 4;
      Duration = 30;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ("activity6_1" );
      Next_activities = ( "activity6_3" );
    };
    Activity activity6_3 = {
      Inventory_cost = 7;
      Duration = 150;
      Machinery = ( "resource3" );
      Prev_activities = ("activity6_2" );
      Next_activities = ( "activity6_4" );
    };
    Activity activity6_4 = {
      Inventory_cost = 1;
      Duration = 60;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ("activity6_3" );
      Next_activities = ( "activity6_5" );
    };
    Activity activity6_5 = {
      Inventory_cost = 9;
      Duration = 100;
      Machinery = ( "resource5" );
      Prev_activities = ("activity6_4" );
      Next_activities = (  );
    };
  );
};

Order order7 = {
  Release_date = 220;
  Due_date     = 1480;
  Customer     = "client2";
  Activities   = (
    Activity activity7_1 = {
      Inventory_cost = 2;
      Duration = 80;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ( );
      Next_activities = ( "activity7_2" );
    };
    Activity activity7_2 = {
      Inventory_cost = 3;
      Duration = 110;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ("activity7_1" );
      Next_activities = ( "activity7_3" );
    };
    Activity activity7_3 = {
      Inventory_cost = 10;
      Duration = 80;
      Machinery = ( "resource3" );
      Prev_activities = ("activity7_2" );
      Next_activities = ( "activity7_4" );
    };
    Activity activity7_4 = {
      Inventory_cost = 7;
      Duration = 50;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ("activity7_3" );
      Next_activities = ( "activity7_5" );
    };
    Activity activity7_5 = {
      Inventory_cost = 4;
      Duration = 90;
      Machinery = ( "resource5" );
      Prev_activities = ("activity7_4" );
      Next_activities = (  );
    };
  );
};

Order order8 = {
  Release_date = 90;
  Due_date     = 1440;
  Customer     = "client4";
  Activities   = (
    Activity activity8_1 = {
      Inventory_cost = 6;
      Duration = 40;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ( );
      Next_activities = ( "activity8_2" );
    };
    Activity activity8_2 = {
      Inventory_cost = 5;
      Duration = 110;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ("activity8_1" );
      Next_activities = ( "activity8_3" );
    };
    Activity activity8_3 = {
      Inventory_cost = 1;
      Duration = 140;
      Machinery = ( "resource3" );
      Prev_activities = ("activity8_2" );
      Next_activities = ( "activity8_4" );
    };
    Activity activity8_4 = {
      Inventory_cost = 5;
      Duration = 80;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ("activity8_3" );
      Next_activities = ( "activity8_5" );
    };
    Activity activity8_5 = {
      Inventory_cost = 6;
      Duration = 140;
      Machinery = ( "resource5" );
      Prev_activities = ("activity8_4" );
      Next_activities = (  );
    };
  );
};

Order order9 = {
  Release_date = 150;
  Due_date     = 1410;
  Customer     = "client1";
  Activities   = (
    Activity activity9_1 = {
      Inventory_cost = 3;
      Duration = 70;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ( );
      Next_activities = ( "activity9_2" );
    };
    Activity activity9_2 = {
      Inventory_cost = 6;
      Duration = 100;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ("activity9_1" );
      Next_activities = ( "activity9_3" );
    };
    Activity activity9_3 = {
      Inventory_cost = 2;
      Duration = 100;
      Machinery = ( "resource3" );
      Prev_activities = ("activity9_2" );
      Next_activities = ( "activity9_4" );
    };
    Activity activity9_4 = {
      Inventory_cost = 9;
      Duration = 80;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ("activity9_3" );
      Next_activities = ( "activity9_5" );
    };
    Activity activity9_5 = {
      Inventory_cost = 10;
      Duration = 150;
      Machinery = ( "resource5" );
      Prev_activities = ("activity9_4" );
      Next_activities = (  );
    };
  );
};

Order order10 = {
  Release_date = 170;
  Due_date     = 1510;
  Customer     = "client4";
  Activities   = (
    Activity activity10_1 = {
      Inventory_cost = 6;
      Duration = 50;
      Machinery = ( "resource2" "resource1" "resource4" );
      Prev_activities = ( );
      Next_activities = ( "activity10_2" );
    };
    Activity activity10_2 = {
      Inventory_cost = 8;
      Duration = 60;
      Machinery = ( "resource4" "resource1" "resource2" );
      Prev_activities = ("activity10_1" );
      Next_activities = ( "activity10_3" );
    };
    Activity activity10_3 = {
      Inventory_cost = 5;
      Duration = 80;
      Machinery = ( "resource3" );
      Prev_activities = ("activity10_2" );
      Next_activities = ( "activity10_4" );
    };
    Activity activity10_4 = {
      Inventory_cost = 4;
      Duration = 80;
      Machinery = ( "resource1" "resource2" "resource4" );
      Prev_activities = ("activity10_3" );
      Next_activities = ( "activity10_5" );
    };
    Activity activity10_5 = {
      Inventory_cost = 10;
      Duration = 110;
      Machinery = ( "resource5" );
      Prev_activities = ("activity10_4" );
      Next_activities = (  );
    };
  );
};
```

Appendix B

Hypothesis Test

Hypothesis testing is an important and convenient tool for statistical treatment of experimental data. The goal of hypothesis testing is to estimate a *state of nature*, or underlying data-generating mechanism from a finite space of possibilities by analyzing available data. In formulating a hypothesis test, one designates the hypothesis that s/he likes to establish as the *alternate hypothesis* (H_a); its negation is called the *null hypothesis* (H_0). In the experiments of machine learning studies, H_0 might be that the problem solver without learning performs at least as well as the problem solver with learning, and H_a would be that the problem solver with learning is superior.

The crux of the hypothesis test is the decision whether the data provide sufficient evidence against H_0 to allow one to reject it. In essence, a hypothesis test is the statistical analog of a "proof by contradiction". The idea is to assume, tentatively, that H_0 holds true and to ask how *unlikely* is the experimental results observed, or ones that favor H_a even more. If the likelihood of observing such extreme experimental outcomes is very low, then there is strong evidence for rejecting H_0. The *p-value* is the probability, assuming H_0 is true, of encountering data that favors H_a as much as or more than the data observed in the experiment. Thus, a small p-value leads one to reject H_0.

If H_0 is rejected, the p-value is the probability that it has been rejected in error. The threshold for the p-value is called the *significant level.* If a hypothesis test is performed at a significant level α, then H_0 is rejected if the p-value is less than α, and the test is said to be statistically significant at level α. This means that the null hypothesis is rejected with the caveat of making error with probability at most α. Note that failure of rejecting H_0 is inconclusive; it does not mean H_0 is true. It only suggests that H_0 is probably false.

Naturally, the experiment data for analyzing the effects of learning are *paired*; problems are generated in a randomized manner and both problem solvers with/without learning attempt to solve each problem. Statistical methods for analyzing paired data are typically based upon the differences between the paired observations. And, if the distributions of the data is unknown, nonparametric tests need to be applied to the data as a method of hypothesis testing.

The author used the signed rank test, a variation of the Wilcoxon test [Green & Margerison, 1977], for testing paired data. The same method was used by Etzioni for analyzing the speedup learning effects [Etzioni & Etzioni, 1994]. The test procedure used in the thesis is as follows. The absolute values of the pair differences are ranked in increasing order (i.e. the smallest value is assigned the rank of one, the second smallest is assigned the rank of two, and so on). The signs of the differences are recorded along with the ranks. The null hypothesis is that the distribution of the pair differences is symmetric about zero. The alternate hypothesis is that the pair differences are slanted toward positive (or negative) values. Under the null hypothesis, the sum of the ranks corresponding to the positive differences is expected to be at least as large as the sum of the ranks corresponding to the negative differences. The p-value is equal to the probability that sum of the positive ranks is at least as large as that observed under the null hypothesis.

Suppose $x_1, x_2 \ldots, x_n$ are results of applying the problem solver with learning to a problem set (i.e., numeric values such as cost function and CPU time), and $y_1, y_2 \ldots, y_n$ are results of the problem solver without learning for the same problem set. And suppose the pair differences of these results are calculated as $d_i = x_i - y_i$. From the way in which the question is posed under consideration, the hypotheses to be tested are

$$\begin{aligned} H_0 : \mu_d &= 0.0 \\ H_a : \mu_d &< 0.0 \quad (or \quad H_a : \mu_d > 0.0) \end{aligned}$$

where μ_d are the means of the pair differences. These hypotheses mean that the results of the problem solver with learning are smaller (or greater) than those of the problem solver without learning.

The test criterion S is the sum of ranks corresponding to the positive differences in the ordered data of $d's$. The distribution of S is approximately normal for large n (at least $n > 25$), which is the case of the experiments in this thesis since every experiment has 60 data in the thesis. When this is true, the distribution of

$$Z = \frac{S - n(n+1)/4}{\sqrt{n(n+1)(2n+1)/24}}$$

is approximately equal to $\mathcal{N}(0,1)$.

Suppose the value of S for the current data is s and $z = \frac{s-n(n+1)/4}{\sqrt{n(n+1)(2n+1)/24}}$. If H_a asserts $\mu_d < 0.0$, that is $y's$ tend to be larger than $x's$, then using the normal approximation, H_0 can be rejected at the significant level α, if

$$P(S < s) = \Phi(z) < \alpha$$

, where $P(S < s)$ is a p-value of S being at least as small as s, and the value of Φ is a significance probability function for the normal distribution $\mathcal{N}(0,1)$ and can be computed from the normal distribution table. Note that, in this case, the positive value of z means the result is not significant.

As with any statistical test, failure to reject the null hypothesis is inconclusive; it is not a basis for concluding that H_0 is true. A more appropriate conclusion is that the experiment should be repeated with a larger sample size. If the sample size is already so large that the test is approaching maximal sensitivity (probability of detecting even small differences between the systems is greater than 90%), then failure to reject the null hypothesis can be regarded as suggestive that the null hypothesis might hold true.

Bibliography

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

[Aha, Kibler, & Albert, 1991] Aha, D. W.; Kibler, D.; and Albert, M. K. 1991. Instance-based learning algorithms. *Machine Learning* 6:37–66.

[Aha, 1992] Aha, D. W. 1992. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies* 36:267–287.

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

[Baker, 1974] Baker, K. R. 1974. *Introduction to Sequencing and Scheduling.* New York, NY: John Wiley & Sons.

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

[Biefeld & Cooper, 1991] Biefeld, E., and Cooper, L. 1991. Bottleneck identification using process chronologies. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 218–224. Sydney, Australia: IJCAI.

[Buchanan & Feigenbaum, 1982] Buchanan, B. G., and Feigenbaum, E. A. 1982. Forward. In Davis, R., and Lenat, D. B., eds., *Knowledge Based Systems in Artificial Intelligence.* New York, NY: McGraw-Hill.

[Carbonell, 1986] Carbonell, J. G. 1986. Derivational analogy : A theory of reconstructive problem solving and expertise acquisition. In Michalski, R. S.; Carbonell, J. G.; and Mitchell, T. M., eds., *Machine Learning, An Artificial Intelligence Approach, Vol.2.* Palo Alto, CA: Morgan Kaufmann.

[Chandrasekaran, Johnson, & Smith, 1992] Chandrasekaran, B.; Johnson, T. R.; and Smith, J. W. 1992. Task-structure analysis for knowledge modeling. *Communications of ACM* 35(9):124–137.

[Chandrasekaran, 1988] Chandrasekaran, B. 1988. Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. *The Knowledge Engineering Review* 3(3):183–210.

[Chandrasekaran, 1990] Chandrasekaran, B. 1990. Design problem solving: A task analysis. *AI Magazine* 11(4):59–71.

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

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

[Clancey, 1985] Clancey, W. J. 1985. Heuristic classification. *Artificial Intelligence* 27:289–350.

[Creecy *et al.*, 1992] Creecy, R. H.; Masand, B. M.; Smith, S. J.; and Waltz, D. L. 1992. A trading MIPS and memory for knowledge engineering. *Communications of ACM* 35(8):48–64.

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

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

[Dent *et al.*, 1992] Dent, L.; Boticario, J.; McDermott, J.; Mitchell, T.; and Zabowski, D. 1992. A personal learning apprentice. In *Proceedings of the Tenth National Conference on Artificial Intelligence.* AAAI.

[Dowsland, 1993] Dowsland, K. A. 1993. Simulated annealing. In Reeves, C. R., ed., *Modern Heuristic Techniques for Combinatorial Problems.* New York, NY: Halsted Press.

[Eshelman, Ehret, & McDermott, 1987] Eshelman, L.; Ehret, D.; and McDermott, J. 1987. MOLE: A tenacious knowledge-acquisition tool. *International Journal of Man-Machine Studies* 26(1):41–54.

[Etzioni & Etzioni, 1994] Etzioni, O., and Etzioni, R. 1994. Statistical methods for analyzing speedup learning. *Machine Learning* 14(3):333–347.

[Fikes, Hart, & Nilsson, 1972] Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4):251–288.

[Fisher, 1987] Fisher, D. H. 1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2:139–172.

[Fox, 1983] Fox, M. 1983. *Constraint-Directed Search: A Case Study in Job Shop Scheduling.* Ph.D. Dissertation, Department of Computer Science, Carnegie Mellon University.

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

[Garey & Johnson, 1979] Garey, M. R., and Johnson, D. S. 1979. *Computers And Intractability: A Guide to the Theory of NP-Completeness.* New York, NY: W. H. Freeman and Company.

[Golding & Rosenbloom, 1991] Golding, A. R., and Rosenbloom, P. S. 1991. Improving rule-based systems through case-based reasoning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 22–27. AAAI.

[Gratch, Chien, & DeJong, 1993] Gratch, J.; Chien, S.; and DeJong, G. 1993. Learning search control knowledge to improve schedule quality. In *Proceedings of IJCAI workshop on knowledge-based production planning, scheduling and control*, 159–168. IJCAI.

[Green & Margerison, 1977] Green, J. R., and Margerison, D. 1977. *Statistical Treatment of Experimental Data.* Amsterdam, The Netherlands: Elsevier.

[Gruber, 1989] Gruber, T. R. 1989. *The Acquisition of Strategic Knowledge.* New York, NY: Academic Press.

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

[Howe, 1992] Howe, A. E. 1992. Analyzing failure recovery to improve planner design. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 387–392. San Jose, CA: AAAI.

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

[Jones, 1993] Jones, R. 1993. Problem solving via analogical retrieval and analogical search control. In Meyrowitz, A. L., and Chipman, S., eds., *Foundations of Knowledge Acquisition: Machine Learning.* Boston, MA: Kluwer Academic.

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

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

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

[Kira & Rendell, 1992] Kira, K., and Rendell, L. A. 1992. The feature selection problem : Traditional methods and a new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 129–134. San Jose, CA: AAAI.

[Kirkpatrick, Gelatt, & Vecchi, 1983] Kirkpatrick, S.; Gelatt, Jr., C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220:671–680.

[Klinker *et al.*, 1991] Klinker, G.; Bhola, C.; Dallemagne, G.; Marques, D.; and McDermott, J. 1991. Usable and reusable programming constructs. *Knowledge Acquisition* 3(2):117–135.

[Klinker, 1988] Klinker, G. 1988. KNACK: Sample-driven knowledge acquisition tool for reporting system. In Marcus, S., ed., *Automating Knowledge Acquisition for Expert Systems*. Boston, MA: Kluwer Academic.

[Kolodner, Simpson, & Sycara, 1985] Kolodner, J.; Simpson, R.; and Sycara, K. 1985. A process of case-based reasoning in problem solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 284–290. Los Angeles, CA: IJCAI.

[Kolodner, 1993] Kolodner, J. 1993. *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann.

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

[Koton, 1989] Koton, P. 1989. SMARTplan: A case-based resource allocation and scheduling system. In *Proceedings of the Case-Based Reasoning Workshop*, 285–294. Pensacola, FL: DARPA.

[Laarhoven, Aarts, & Lenstra, 1992] Laarhoven, P. J. M. V.; Aarts, E. H. L.; and Lenstra, J. K. 1992. Job shop scheduling by simulated annealing. *Operations Research* 40(1):113–125.

[Laird, Rosenbloom, & Newell, 1986] Laird, J. E.; Rosenbloom, P. S.; and Newell, A. 1986. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning* 1(1):11–46.

[Langley & Allen, 1993] Langley, P., and Allen, J. A. 1993. A unified framework for planning and learning. In Minton, S., ed., *Machine Learning Methods for Planning.* San Mateo, CA: Morgan Kaufmann.

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

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

[Marcus & McDermott, 1987] Marcus, S., and McDermott, J. 1987. SALT: A knowledge acquisition tool for propose-and-revise systems. *Artificial Intelligence* 39(1):1–37.

[Mark, 1989] Mark, W. S. 1989. Case-based reasoning for autoclave management. In *Proceedings of the Case-Based Reasoning Workshop,* 176–180. Pensacola, FL: DARPA.

[Markovitch & Scott, 1993] Markovitch, S., and Scott, P. D. 1993. Information filtering: Selection mechanisms in learning systems. *Machine Learning* 10:113–151.

[McDermott, 1988] McDermott, J. 1988. Using problem-solving methods to impose structure on knowledge. In *Proceedings of International Workshop on Artificial Intelligence for Industrial Applications,* 7–11.

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

[Minton *et al.*, 1989] Minton, S.; Knoblock, C. A.; Kuokka, D. R.; Gil, Y.; Joseph, R. L.; and Carbonell, J. G. 1989. PRODIGY 2.0 : The manual and tutorial. Technical Report CMU-RI-TR-89-146, The Robotics Institute, Carnegie Mellon University.

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

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

[Minton, 1988] Minton, S. 1988. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* Boston, MA: Kluwer Academic Publishers.

[Mitchell, Keller, & Kedar-Cabelli, 1986] Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1(1):47–80.

[Mitchell, Mahadevan, & Steinberg, 1985] Mitchell, T. M.; Mahadevan, S.; and Steinberg, L. 1985. LEAP: A learning apprentice for VLSI design. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence.* Los Angeles, CA: IJCAI.

[Mitchell, Ugoff, & Banerji, 1983] Mitchell, T. M.; Ugoff, P. E.; and Banerji, R. 1983. Learning by experimentation: Acquiring and refining problem-solving heuristics. In Michalski, R.; Carbonell, J.; and Mitchell, T., eds., *Machine Learning: An Artificial Intelligence Approach.* San Mateo, CA: Morgan Kaufmann.

[Miyashita & Sycara, 1993] Miyashita, K., and Sycara, K. 1993. Improving schedule quality through case-based reasoning. In *Proceedings of AAAI-93 Workshop on Case-Based Reasoning*, 101–110. Washington, DC: AAAI.

[Miyashita & Sycara, 1994a] Miyashita, K., and Sycara, K. 1994. Adaptive case-based control of schedule revision. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling.* San Mateo, CA: Morgan Kaufmann.

[Miyashita & Sycara, 1994b] Miyashita, K., and Sycara, K. 1994. CABINS: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. *Artificial Intelligence.* Submitted.

[Miyashita & Sycara, 1994c] Miyashita, K., and Sycara, K. 1994. Exploitation of cases for schedule quality improvement. *Journal of Japanese Society for Artificial Intelligence* 9(4):559–568.

[Miyashita & Sycara, 1994d] Miyashita, K., and Sycara, K. 1994. A framework for case-based revision for schedule generation and reactive schedule management. *Journal of Japanese Society for Artificial Intelligence* 9(3):426–435.

[Mizoguchi, Tijerino, & Ikeda, 1992] Mizoguchi, R.; Tijerino, Y.; and Ikeda, M. 1992. Task ontology and its use in a task analysis — two-level mediating representation in MULTIS —. In *Proceedings of the Second Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, 185–198.

[Mizoguchi, 1992] Mizoguchi, R. 1992. Consideration on design process from a knowledge engineering point of view. *Journal of Japanese Society for Artificial Intelligence* 7(2):45–52. In Japanese.

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

[Muscettola, 1993] Muscettola, N. 1993. Scheduling by iterative partition of bottleneck conflicts. In *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*, 49–55. Orlando, FL: IEEE.

[Musen *et al.*, 1987] Musen, M. A.; Fagan, L. M.; Combs, D. M.; and Shortlife, E. H. 1987. Use of a domain model to drive an interactive knowledge-editing tool. *International Journal of Man-Machine Studies* 12:63–87.

[Musen, 1989] Musen, M. A. 1989. Automated support for building and extending expert models. *Machine Learning* 4(3/4):347–375.

[Musick & Russell, 1992] Musick, R., and Russell, S. 1992. How long will it take? In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 466–471. San Jose, CA: AAAI.

[Navinchandra, 1991] Navinchandra, D. 1991. *Exploration and Innovation in Design Towards a Computational Model.* New York, NY: Springer-Verlag.

[Newell & Simon, 1972] Newell, A., and Simon, H. 1972. *A Human Problem Solving.* Englewood Cliffs, NJ: Prentice-Hall.

[Newell, 1982] Newell, A. 1982. The knowledge level. *Artificial Intelligence* 18(1):87–127.

[Offutt, 1988] Offutt, D. 1988. SIZZLE: A knowledge-acquisition tool specialized for the sizing task. In Marcus, S., ed., *Automating Knowledge Acquisition for Expert Systems.* Norwell, MA: Kluwer Academic Publishers.

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

[Pérez & Carbonell, 1994] Pérez, M. A., and Carbonell, J. 1994. Control knowledge to improve plan quality. In *Proceedings of the Second International Conference on AI Planning Systems.*

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

[Puerta *et al.*, 1992] Puerta, A. R.; Egar, J. W.; Tu, S. W.; and Musen, M. A. 1992. A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. *Knowledge Acquisition* 4(2):171–196.

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

[Rissland & Ashley, 1988] Rissland, E., and Ashley, K. 1988. Credit assignment and the problem of competing factors in case-based reasoning. In *Proceedings of the Case-Based Reasoning Workshop*, 327–344. Clearwater, FL: DARPA.

[Ruby & Kibler, 1992] Ruby, D., and Kibler, D. 1992. Learning episodes for optimization. In *Machine Learning : Proceedings of the Ninth International Workshop (ML92)*, 379–384.

[Sadeh & Fox, 1990] Sadeh, N., and Fox, M. S. 1990. 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*, 134–144.

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

[Schreiber & Wielinga, 1993] Schreiber, G., and Wielinga, B. 1993. Model construction. In Schreiber, G.; Wiekinga, B.; and Breuker, J., eds., *KADS: A Principled Approach to Knowledge-Based System Development.* San Diego, CA: Academic Press.

[Schreiber, 1993] Schreiber, G. 1993. Operationalizing models of expertise. In Schreiber, G.; Wiekinga, B.; and Breuker, J., eds., *KADS: A Principled Approach to Knowledge-Based System Development.* San Diego, CA: Academic Press.

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

[Simon, 1973] Simon, H. A. 1973. The structure of ill structured problems. *Artificial Intelligence* 4:181–201.

[Simon, 1981] Simon, H. A. 1981. *The Sciences of the Artificial.* Cambridge, MA: MIT Press.

[Simoudis & Miller, 1990] Simoudis, E., and Miller, J. 1990. Validated retrieval in case-based reasoning. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 310–315. AAAI.

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

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

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

[Smith *et al.*, 1986] Smith, S. F.; Ow, P. S.; LePape, C.; McLaren, B.; and Muscettola, N. 1986. Integrating multiple scheduling perspectives to generate detailed production plans. In *Proceedings SME Conference on AI in Manufacturing.*

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

[Stout *et al.*, 1988] Stout, J.; Caplain, G.; Marcus, S.; and McDermott, J. 1988. Toward automating recognition of differing problem-solving demands. *International Journal of Man-Machine Studies* 29(5):599–611.

[Sussman, 1975] Sussman, G. J. 1975. *A Computer Model of Skill Acquisition.* New York, NY: American Elsevier.

[Sycara & Miyashita, To be published] Sycara, K., and Miyashita, K. To be published. Learning control knowledge through case-based acquisition of user optimization preferences in ill-structured domain. In Tecuci, G., and Kodratoff, Y., eds., *Machine Learning and Knowledge Acquisition: Integrated Approaches.* New York, NY: Academic Press.

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

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

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

[Tate, 1977] Tate, A. 1977. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 888–893. Cambridge, MA: IJCAI.

[Veloso & Carbonell, 1992] Veloso, M. M., and Carbonell, J. G. 1992. Derivational analogy in PRODIGY : Automating case acquisition, storage, and utilization. *Machine Learning.*

[Veloso, 1992] Veloso, M. M. 1992. *Learning by Analogical Reasoning in General Problem Solving.* Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University.

[Weiss & Kulikowski, 1990] Weiss, S. M., and Kulikowski, C. A. 1990. *Computer Systems That Learn : Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems.* San Mateo, CA: Morgan Kaufmann Publisher, Inc.

[Wielinga, Schreiber, & Breuker, 1993] Wielinga, B.; Schreiber, G.; and Breuker, J. 1993. Modelling expertise. In Schreiber, G.; Wiekinga, B.; and Breuker, J., eds., *KADS: A Principled Approach to Knowledge-Based System Development.* San Diego, CA: Academic Press.

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

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

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

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