RL-TR-94-206
Final Technical Report
November 1994

KNOWLEDGE-BASED LOGISTICS PLANNING: ITS APPLICATION IN MANUFACTURING AND LOGISTICS PLANNING

Carnegie Mellon University

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 7726

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

APPROVED: [illegible]

NORTHRUP FOWLER III, Ph.D.
Project Engineer

FOR THE COMMANDER: [illegible]

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

Accesion For		
NTIS CRA&I	☒	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

KNOWLEDGE-BASED LOGISTICS PLANNING: ITS APPLICATION IN MANUFACTURING AND LOGISTICS PLANNING

Nicola Muscettola
Steve Roth
Norman Sadeh
Katia Sycara

Contractor: Carnegie Mellon University
Contract Number: F30602-91-C-0016
Effective Date of Contract: 31 January 1991
Contract Expiration Date: 30 June 1994
Short Title of Work: Knowledge-Based Logistics Planning
Period of Work Covered: Jan 91 - Jun 94

Principal Investigator: Norman Sadeh
Phone: (402) 268-8827
RL Project Engineer: Northrup Fowler III
Phone: (315) 330-3011

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Northrup Fowler III, RL (C3C), 525 Brooks Rd, Griffiss AFB NY 13441-4505.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Final Jan 91 - Jun 94

4. TITLE AND SUBTITLE
KNOWLEDGE-BASED LOGISTICS PLANNING: ITS APPLICATION IN MANUFACTURING AND LOGISTICS PLANNING

5. FUNDING NUMBERS
C - F30602-91-C-0016
PE - 62301E
PR - G726
TA - 00
WU - 12

6. AUTHOR(S)
Nicola Muscettola, Steve Roth, Norman Sadeh, and Katia Sycara

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh PA 15213-3891

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Rome Laboratory (C3C)
525 Brooks Rd
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-94-206

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Northrup Fowler III/C3C/(315) 330-3011

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This document summarizes research in CORTES, a project in constraint-based planning, scheduling, and control for complex large-scale domains such as military transportation and manufacturing. The CORTES approach keeps the planning/scheduling combinatories in check, using quantitative problem space metrics called texture measures to: (1) identify critical decisions that require early attention; and (2) steer search toward promising solutions. This basic approach has been applied and validated in several key contexts, including (1) micro-opportunistic search, which focuses on efficient generation and dynamic maintenance of complex large-scale Just-In-Time schedules; (2) simulated annealing search, where texture measures have been developed to focus search and learn to recognize (un)promising runs; (3) iterative constraint posting, which combines flexible schedule representation and dynamic identification of conflicts requiring further arbitration; (4) integration of predictive planning/scheduling and execution control, where texture measures and flexible schedule representations are combined to coordinate multiple planning/scheduling and control agents; and (5) interactive schedule repair, where adaptive similarity metrics direct re-use of previous repair histories and help select repair focus and actions. Visualization issues have been addressed by developing comprehensive (see reverse)

14. SUBJECT TERMS
Artificial intelligence, Planning, Scheduling, Resource allocation, Constraint satisfaction problems, Transportation

15. NUMBER OF PAGES
326

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

13. (Cont'd)

languages for characterizing: (1) domain concepts to be displayed; (2) user analysis tasks; and (3) graphical presentation techniques that can be assembled to create displays.

Table of Contents

1 Overview of the Project

Objectives of the Project The CORTES project (contract #F30602-91-C-0016) aims at developing constraint-based technologies for coordinated and distributed planning, scheduling and control in complex large-scale domains, such as military transportation and manufacturing. Sources of difficulty for planning/scheduling in these domains are many:

- *Combinatorics*: The planning/scheduling problems of interest are characterized by extremely large search spaces in which the number of satisficing solutions represents only a tiny fraction of the total number of explorable alternatives. Even under highly idealistic conditions, scheduling problems such as job shop scheduling are known to be NP-hard [Garey 79].
- *Ill-Defined Problems*: often problems are ill-defined in terms of the specific tasks that need to be performed (e.g. specific orders to be scheduled in manufacturing or specific move requirements in military transportation scheduling), the objectives and preferences to be optimized (which often are not even compatible), etc.
- *Uncertainty*: problem constraints tend to change over time (e.g. resources become unavailable, new tasks need to be performed, execution of some activities take longer or less time than anticipated, etc.)
- *Decentralization*: The complexity of large-scale planning and scheduling problems, and the distributed nature of the executing environment generally requires decomposition and decentralization of decision-making responsibility. Because component subproblems are rarely independent and subproblem solution proceeds asynchronously, interactions and conflicts in the overall solution must be effectively and efficiently managed.

Accordingly, research in the CORTES project aims at: (1) efficiently generating and maintaining high quality solutions to large scale planning/scheduling problems that adequately capture the causal dependencies of these domains; (2) flexibly integrating predictive planning/scheduling and reactive execution control; (3) interactively acquiring new user constraints and preferences and plan/schedule repair experience, and (4) visualizing and interactively manipulating large amounts of diverse information.

The CORTES Approach: In our approach, the combinatorics of the problem is kept in check through use of quantitative problem space metrics called *textures*. Texture measures are regularly computed to capture various types of constraint and preference interactions (e.g. resource contention). They are used to focus search on critical decisions and efficiently steer it towards promising areas of the search space (e.g. variable/value ordering heuristics, backtracking heuristics, repair focusing heuristics, etc.).

This basic approach has been applied and validated in several problem solving contexts:

1. *micro-opportunistic search* which focuses on the efficient generation and dynamic maintenance of Just-In-Time schedules for large-scale generalized job shop scheduling problems subject to sequence-dependent setups, resource alternatives ("parallel resources") and temporal windows;

2. *simulated annealing search* where texture measures are used to (1) dynamically focus search on critical subproblems by artificially inflating the costs associated with major sources of inefficiency in the exisiting solution and (2) learn to recognize (un)promising simulating runs and decide when to abandon a run and where to restart search;
3. *iterative constraint posting* which combines flexible schedule representation and dynamic identification of conflict areas where additional arbitration is needed;
4. *integration of predictive planning/scheduling and reactive execution control* where texture measures and flexible temporal representations are combined to coordinate multiple planning/scheduling and control agents;
5. *interactive schedule repair* where adaptive similarity metrics direct re-use of previous repair histories and help select current repair focus and actions.

The complex task of visualizing information needed for problem solving activities has been addressed by developing comprehensive languages for characterizing domain concepts that must be displayed, information analysis tasks which users must perform, and libraries of graphical presentation techniques which can be assembled to create displays. These form the representational foundation for encoding graphics presentation knowledge for creating an automatic design system.

2 Constraint-Directed Scheduling

Our work in constraint-directed scheduling has revolved around two search paradigms:

- *micro-opportunistic search*: this search paradigm emphasizes rapid development and revision of high quality schedules, using resource contention metrics to help focus solution (re)optimization on critical subproblems [Xiong 92] [Sadeh 94a] [Sadeh 94b] [Sadeh 92a] [Sadeh 91a] [Li&Sadeh 93] [Sadeh 93a] [Sadeh 94c] [Sadeh 91b] [Sadeh 91c] [Sadeh 92b] [Sadeh 93b].
- *adaptive simulated annealing search*: this search approach, which remains slower than the first one, emphasizes the development of near-optimal solutions, using texture-based heuristics to increase the efficiency of simulated annealing search. These heuristics include "focus-of-attention" heuristics to identify solution inefficiencies on which to dynamically focus the problem solving effort [Sadeh&Nakakuki 94] as well as search cutoff and restart criteria based on new metrics to identify (un)promising simulated annealing runs [Nakakuki&Sadeh 94].

Major accomplishments in each of these areas are summarized below. Further details on our work over the past 3 years can be found in [Xiong 92] [Sadeh 92c] [Sadeh 92a] [Sadeh 91a] [Sadeh 93a] [Sadeh 94c] [Sadeh 94a] [Sadeh 91b] [Sadeh 91c] [Li&Sadeh 93] [Sadeh 92b] [Chen 93] [Swaminathan 93] [Swaminathan 94] [Sadeh 93b] [Sadeh&Nakakuki 94] [Nakakuki&Sadeh 94].

2.1 Micro-Opportunistic Scheduling

2.1.1 Background and Overview of Accomplishments

In contrast to earlier bottleneck-centered scheduling approaches (e.g. [Goldratt 80, Ow 88, Adams 88]) which rely on the optimization of large resource sub-problems, micro-opportunistic scheduling aims at increasing search efficiency and solution quality through use of a more flexible/finer grain search procedures. In this approach, resource contention is continuously monitored during the construction/repair of the schedule, and the problem solving effort can be redirected at any time towards the most critical sub-problem. In our earlier research [Sadeh 91c], we showed that because of their extra flexibility ("opportunism"), micro-opportunistic search procedures are better equipped than traditional (less flexible) bottleneck-centered scheduling approaches to deal with:

- *localized bottlenecks*, namely resources that are bottlenecks only over one or several portions of the scheduling horizon (e.g. due to changes over time in the mix of orders to be scheduled);
- *multiple bottlenecks*, which traditional bottleneck scheduling techniques have problems dealing with, as they tend to focus on the optimization of one bottleneck resource at the expense of others;
- *bottleneck dynamics*, namely the fact that the very scheduling decisions that are made by the system can increase or decrease the severity of bottlenecks and that, as a result, it is crucial to closely monitor resource contention throughout the construction of the schedule.

In early 1991, when the current project started (ARPA contract #F30602-91-C-0016), micro-opportunistic scheduling techniques had been developed to solve two classes of job shop scheduling problems: (1) job shop scheduling constraint satisfaction problems where a feasible job shop schedule has to be built given a set of jobs each with one or several non-relaxable time windows (earliest/latest possible start time window) within which it has to be scheduled and (2) Just-In-Time job shop scheduling problems where the objective is to build a schedule that minimizes the sum of tardiness and inventory costs of all jobs. At the time, these initial micro-opportunistic scheduling techniques, which had already been shown to significantly outperform a variety of competing techniques proposed both in the Artificial Intelligence and Operations Research literature, (1) could only solve relatively small problems, (2) could not solve problems with setups or parallel machines, and (3) did not support any reactive or interactive functionalities.

During the course of this three-year project, we have dramatically scaled-up our micro-opportunistic scheduling heuristics, moving from prototypical procedures to a set of powerful scheduling techniques capable of efficiently generating high quality solutions to large and complex problems, as well as providing flexible interactive and reactive solution revision capabilities. We have shown that micro-opportunistic scheduling techniques are not only capable of producing high quality Just-In-Time solutions (approx. 25% improvement in schedule quality against a combination of 39 combinations of dispatch rules and release policies [Sadeh 94c]) but can also be successfully adapted to solve a wide range of realistic problems. Over the past three years, we have speeded up our micro-opportunistic scheduling techniques by two orders of magnitude, improved the quality of the solutions they produce by an average of about 20%, and extended our heuristics to solve problems with sequence dependent setups and parallel machines [Sadeh 94c, Sadeh 93b]. At the same time, variations of the Micro-Boss scheduling heuristics were also adapted in the context of the Knowledge Based Logistics Planning Shell (KBLPS) developed by Carnegie Group, Inc. (CGI) and LB&M Associates to solve U.S. army transportation scheduling problems and ammunition distribution planning problems [Saks 92]. Other efforts using variations of our micro-opportunistic techniques are described in [Torma 91, Berry 91, Linden 91, Paolucci 92] and [Winklhofer 92].

At the present time, the Micro-Boss scheduling system which is written in C++ and has an X/Motif interface, is undergoing customization for the scheduling of a Printed Wire Assembly shop at Raytheon's Andover facility. This involves scheduling over 20,000 operations per month on a total of about 150 resources subject to a variety of complex constraints, including overlapping constraints between successive manufacturing steps.

The following further outlines technical accomplishments in micro-opportunistic scheduling over the past 3 years. Additional details can be found in the papers provided in appendix.

2.1.2 Improving the Basic Micro-Opportunistic Search Procedure

Over the past three years, the average speed of our micro-opportunistic scheduling techniques has been increased by two orders of magnitude and average schedule quality has improved by about 20%. These performance improvements were obtained through modifications of several keys aspects of the basic micro-opportunistic search procedure:

- **Hierarchical Demand Profile Construction**: A key strength of our micro-opportunistic search procedure comes from the detailed demand profiles it continuously updates to identify areas of high contention and determine which operation(s) to schedule next. We have been able to significantly reduce the time required to compute these demand profiles by (1) incrementally updating rough/coarse demand profiles for each resource in the problem and (2) using these rough demand profiles to dynamically identify critical resource/time intervals over which to perform a more detailed contention analysis.
- **Variable Search Granularity**: rather than performing a detailed contention analysis at each step (i.e. each time an operation is scheduled), we have identified a set of conditions under which it is safe to schedule more than one operation at a time. One such condition occurs when one or several unscheduled operations have only one reservation left, in which case all these operations can be scheduled at once without performing any additional contention analysis. The result is a search procedure whose granularity (which is determined by the number of operations that are scheduled before a new contention analysis is performed) varies over time.
- **Additional Improvements**: Additional improvements include (1) the development

of new heuristics to dynamically update the best remaining reservations of unscheduled operations and evaluate incremental tardiness and inventory costs incurred by an operation if it is not allocated one of its best remaining reservations, and (2) the development of new heuristics to compute demand contention based on these results.

The resulting system can schedule problems with over 1,000 operations in a matter of minutes. In comparison against 39 combinations of well-regarded priority dispatch rules and release policies (taking the best schedule produced by these 39 techniques on each problem and comparing it with the schedule obtained by Micro-Boss), Micro-Boss was shown to reduce schedule cost by close to 25%. This includes significant reductions in work-in-process and finished goods inventory as well as important improvements in due date satisfaction. In comparison against the Weighted Covert dispatch rule taken in isolation, Micro-Boss improves due date performance by an even more impressive 28% while reducing average inventory costs by 40%. Comparisons against a variety of coarser bottleneck-centered scheduling procedures have produced similarly impressive results.

2.1.3 New Bottleneck Optimization Procedures

The Micro-Boss scheduling procedures have also been adapted to solve problems with parallel machines (i.e. multiple machines with either similar or dissimilar capabilities) and setups. A key element in adapting our procedures has been the development of new bottleneck optimization heuristics for the one-machine early/tardy problem with setups. Briefly, our heuristic opportunistically selects between two simpler techniques: (1) a clustering scheme that identifies clusters of early/tardy jobs and resequences them using variations of the Weighted Longest (Shortest) Processing Time dispatch rule and (2) a two-parameter technique that generalizes a dispatch rule developed by Ow and Morton for the single-machine early/tardy problem without setups [Ow 89]. Based on measures of the tightness of the bottleneck optimization problem at hand, our technique dynamically selects between these two heuristics and further optimizes the resulting solution using a neighborhood search procedure in combination with an optimal idle time insertion procedure first proposed by Garey and Tarjan [Garey 88] (additional details are provided in the Micro-Boss paper in appendix). Extensive evaluation of this bottleneck optimization technique on a total of 1,920 scheduling problems characteristic of a wide range of scheduling conditions shows that (1) it can solve large problems in a matter of seconds and (2) it produces solutions that are consistently within 5 to 10% of the optimum.

2.1.4 Intelligent Backtracking Heuristics

This work has focused on a version of the job shop scheduling problem in which some operations have to be scheduled within non-relaxable time windows (i.e. earliest/latest possible start time windows). This problem is a well-known NP-complete Constraint Satisfaction Problem (CSP). A popular method for solving this type of problems involves using depth-first backtrack search. In our earlier work [Sadeh 91c, Sadeh 91b], we focused on the development of consistency enforcing techniques and variable/value ordering heuristics that improve the efficiency of this search procedure. Under the current project, we combined these techniques with new look-back schemes that help the search procedure recover from so-called deadend search states (i.e. partial solutions that cannot be completed without violating some constraints). More specifically, we developed three new "intelligent" backtracking schemes: (1) *Dynamic Consistency Enforcement*, which dynamically identifies critical subproblems and determines how far to backtrack by selectively enforcing higher levels of consistency among variables participating in these critical subproblems, (2) *Learning Ordering From Failure*, which dynamically modifies the order in which variables are instantiated based on earlier conflicts, and (3) *Incomplete Backjumping Heuristic*, which abandons areas of the search space that appear to require excessive computational efforts. These schemes have been shown to (1) further reduce the average complexity of the backtrack search procedure, (2) enable our system to efficiently solve problems that could not be solved otherwise due to excessive computation cost, and (3) be more effective at solving job shop scheduling problems than other look-back schemes advocated in the literature.

The benchmark problems used in this research have been made available to the research community at large through an anonymous ftp account set up at CMU and have been *widely disseminated*, providing for the first time a common set of problems in this area. A *high point* in this work was reached in March 1992 at the AAAI Spring Symposium held in Stanford, when *our group and a group from NASA were the first ones to announce they could efficiently solve all 60 of the benchmark problems* (paper to appear in the Artificial Intelligence Journal [Sadeh 94b] and provided in appendix). The speed of our procedure has been further improved since then, making it possible to solve most problems in 1 to 2 CPU seconds on a DECstation 5000/200. While similar results have been achieved since then by a couple of other groups, our results remain quite competitive on this class of problems and could probably be further improved. However, rather than dwelling on this class of problems, we have concentrated our efforts on more complex Just-In-Time scheduling problems where the objective is not just to build feasible schedules but instead requires minimizing tardiness and inventory costs (both in-process costs and earliness costs), a formulation that better captures situations found in many manufacturing and transportation domains. The backtracking heuristics we developed work just as well on these more complex Just-In-Time scheduling problems.

2.1.5 Reactive Scheduling

Earlier approaches to reactive schedule repair have emphasized the use of iterative repair heuristics [Smith 90a, Minton 90, Zweben 91]. In the process of resolving schedule conflicts, these repair heuristics are allowed to introduce new conflicts, which in turn need to be repaired. This iterative behavior may sometimes lead to myopic decisions and can potentially become expensive. In contrast to these approaches, schedule repair in Micro-Boss attempts to take a more global view of the problem and capitalizes on the strengths of micro-opportunistic search. Concretely, we have developed an approach in which schedule repair is performed in two steps: (1) a set of operations that need to be rescheduled is identified using a so-called conflict propagation procedure and all the operations in this set are unscheduled, (2) the scheduling problem consisting of all these unscheduled operations and the constraints imposed on these operations by operations that have already been executed or have not been unscheduled is passed to the micro-opportunistic scheduling procedure.

In comparison with a micro-opportunistic technique that rebuilds brand new schedules from scratch, our reactive approach has been shown to produce schedules that are almost as good while only rescheduling a much smaller number of operations. Our reactive approach has also been shown to outperform several iterative repair techniques.

2.1.6 Supporting Mixed Initiative Functionalities

Because of their flexibility, micro-opportunistic scheduling heuristics also seem particularly well suited to support mixed initiative capabilities. An initial set of such capabilities has been developed in the context of the Micro-Boss system, making it possible to interleave both manual and automatic (micro-opportunistic) scheduling decisions and enabling the user to incrementally manipulate, save, analyze and compare alternative (complete or partial) schedules (e.g., "What-if" type of analysis).

2.2 Adaptive Simulated Annealing Search

Our work on Adaptive Simulated Annealing Search is complementary to our research on micro-opportunistic search procedures and aims at the development of near-optimal (though possibly slower) scheduling procedures.

Simulated Annealing (SA) procedures can potentially yield near-optimal solutions to many difficult combinatorial optimization problems (not just scheduling problems), though often at the expense of intensive computational efforts. The single most significant source of inefficiency in SA search is the inherent stochasticity of the procedure, typically requiring a large number of runs before a near-optimal solution is found. Our work in this area aims at developing mechanisms that (1) speed up the basic SA procedure while improving average solution quality and (2) reduce the number of runs required to obtain near-optimal solutions. Specifically, we have developed two sets of techniques: (1) focus of attention mechanisms that dynamically identify major inefficiencies in the solution on which to focus the optimization effort [Sadeh&Nakakuki 94] and (2) speedup learning mechanisms that learn to recognize (un)promising runs and can be used to determine when to abandon a run and where to restart search [Nakakuki&Sadeh 94].

This work, which is more recent, is described in two papers provided in appendix.

2.3 Summary of Accomplishments and Plans for the Future

During the course of this three-year project we have shown that micro-opportunistic scheduling techniques are not only capable of producing high quality Just-In-Time solutions (approx. 25% improvement in schedule quality against a combination of 39 combinations of dispatch rules and release policies [Sadeh 94c]) but can also be successfully adapted to solve a wide range of realistic problems. Over the past three years, we have speeded up our micro-opportunistic scheduling techniques by two orders of magnitude, improved the quality of the solutions they produce by an average of about 20%, extended our heuristics to solve problems with sequence dependent setups and parallel machines, and have shown that these techniques can support powerful reactive and mixed initiative capabilities [Sadeh 94c, Sadeh 93b].

Our micro-opportunistic techniques have been adapted in the context of the Knowledge Based Logistics Planning Shell (KBLPS) developed by Carnegie Group, Inc. (CGI) and LB&M Associates to solve U.S. army transportation scheduling problems and ammunition distribution planning problems [Saks 92], demonstrating the dual-use applicability of this technology. We have now embarked on a technology demonstration effort with Raytheon, which involves customizing Micro-Boss for the scheduling of the Printed Wire Assembly area at Raytheon's Andover manufacturing facility and will continue to work with ARPA to further transition this technology into practical environments (manufacturing, transportation or others).

In the longer term, we see several important areas for future development of this technology:

- **Iterative improvement techniques**: with the advent of ever more powerful computers, we believe that it is now possible to complement micro-opportunistic scheduling techniques with anytime iterative improvement techniques that could be applied to post-process schedules. Our work in Simulated Annealing suggests that the efficiency of such techniques can greatly be enhanced using simple speedup learning mechanisms.

- **Integration with higher-level planning decisions**: Traditionally, manufacturing scheduling has focused on sequencing and release decisions, ignoring higher level (MRP-level) decisions that critically constrain the lower-level scheduling problem. Examples of such decisions include batching decisions, overtime decisions, safety stock/safety leadtime decisions, subcontracting decisions, order promising, procurement and other supply chain management decisions. We believe that significant improvements in scheduling practice could be achieved by integrating some of these decisions with sequencing and release decisions. Similar integrations are also required in other scheduling domains such as transportation scheduling.

- **Integration with Process Planning**: Another area in Computer Integrated Manufacturing that has received very little attention involves integrating process planning decisions with production scheduling and control decisions. Such integration is particularly critical to support Agile Manufacturing scenarios in which customer orders require the generation of new process plans that need to be dynamically integrated into the production schedule.

- **Analysis Tools for Mixed Initiative Decision Support**: Micro-Boss has demonstrated the usefulness of texture measures to dynamically identify critical decisions and guide the scheduling process. Similar texture measures could be developed to support mixed initiative scenarios, helping the user identify sources of inefficiency in a current solution, evaluate and possibly even propose alternative options for solution improvement (e.g. where to add extra capacity, how much capacity to add, which deadlines to relax, what is the right mix of transportation modes, etc.).

3 Distributed Scheduling

3.1 Introcduction

We have developed a computational framework for *collective problem solving* by a society of reactive agents [Liu.Sycara 93a] [Liu.Sycara 93b] [Liu.Sycara 94a] [Liu.Sycara 94b]. Problem solving is viewed as an emergent functionality from the evolving process of the society of diverse, interacting, and well-coordinated reactive agents. Agents are situated in their environment and act by stimulus and response. Coordinated interactions are based on simple flows of information. The collective actions of the reactive agents potentially provide an effective tool for complex problem solving. Specifically, the development of the collective problem solving framework involves the following issues:

- *Problem decomposition*: The transformation from a problem to a society of simple agents is defined by a decomposition scheme. Each agent is assigned to a task corresponding to a small part of the problem. Situation-action rules specify how agents would act to achieve their tasks. The problem is solved when all agents achieve their tasks simultaneously.

- *Interaction analysis*: When a problem is mapped into a society of agents, intense interactions among agents ensue. In order for the society to move toward coherence, influences of agents' actions on each other need to be identified. These interactions are viewed as rich information sources that can be exploited to guide agents' behaviors toward group coherence.

- *Coordination mechanism*: Group behavior of agents is characterized by the coordination mechanism in the society. For our problem-solving purpose, we require the group of agents to reach coherence in order to provide a solution. In addition, we seek for rapid convergence to improve problem-solving efficiency. The design of a coordination mechanism includes regulation policies and communication among agents.

- *Behavior design*: An agent's behavior corresponds to various actions it performs to achieve its goal. The collective behavior of agents represents problem-solving activities that the group performs. In this framework, it is critical to analyze agent interactions, investigate useful information exchange between agents, and coordinate the highly distributed activities. All of these lead to designing agents' behaviors such that (1) they avoid harmful interactions with other agents, (2) they react appropriately towards rapid group convergence.

The problem domains of the collective problem solving framework that we have investigated are *Constraint Satisfaction Problems (CSPs).* Many problems of theoretical and practical interest (e.g., parametric design, resource allocation, scheduling) can be formulated as CSPs. A CSP is defined by a set of *variables*, each having a corresponding *domain*, and a set of *constraints*. A constraint is a subset of the Cartesian product which specifies which values of the variables are compatible with each other. The *variable set* of a constraint (or a set of constraints), is the set of non-duplicating variables restricted by the constraint (or the set of constraints). A solution to a CSP is an assignment of values (an instantiation) for all variables, such that all constraints are satisfied. Numerical CSPs (NCSPs) are a subset of CSPs, in which constraints are represented by numerical relations between quantitative variables usually with fairly large domains of possible values. Many CSPs of practical importance, such as scheduling, and parametric design, are NCSPs. Constraint satisfaction algorithms typically suffer from feasibility/efficiency problems for NCSPs due to their enormous search space.

We have developed a collective problem-solving framework, called Constraint Partition and Coordinated Reaction (CP&CR), for a subset of NCSPs. In CP&CR, a society of specialized and well-coordinated reactive agents collectively and asynchronously solve an NCSP. Agents are situated in their environment, react to others' actions, and communicate with others by leaving and perceiving particular messages on the objects they act on. A solution emerges from the evolutionary interaction process of the society of diverse agents. Specifically, CP&CR provides a framework to decompose an NCSP into a set of subproblems based on constraint type and constraint connectivity, identify their interaction characteristics and, accordingly construct effective coordination mechanisms. CP&CR assumes that an NCSP has at least two types of constraints.

3.2 Summary of Experimental Results

We evaluated the performace of CP&CR on a benchmark suite of job shop scheduling CSPs. The experiemntal results show that:

- exchange of coordination information increases the efficiency of group convergence
- CP&CR works considerably well as compared to other state-of-the-art scheduling techniques both on number of problems successfully solved and efficiency in finding a solution
- the performace of CP&CR is almost independent of its starting point of search, i.e. it can start with random assignments of values to variables
- CP&CR exhibits near-linear scaling-up characteristics

We are currently extending the methodology to Constraint Optimization Problems. Preliminary experimental results are very encouraging.

4 Interactive Schedule Repair

4.1 Introduction

Practical scheduling problems generally require allocation of resources in the presence of a large, diverse and typically conflicting set of constraints and optimization criteria. The ill-structuredness of both the solution space and the desired objectives make scheduling problems difficult to formalize. 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. Therefore, there is the need for a human operator to interact with the schedule and impart to it user preferences in terms of what is a good schedule. These preferences, then should guide schedule optimization. The value of incorporating such user preferences and constraints in operational scheduling environments is becoming increasingly recognized (e.g. [McKay 88]) but good techniques are currently lacking. Moreover, operational environments for scheduling systems (e.g. factories) are dynamic. Unpredictable events, such as machine breakdown or operator absence, often happen during schedule execution. Therefore, a schedule that is only *predictive* (i.e. it is created assuming that the world is static and predictable) will be brittle. It is clear that any effective scheduling system should be *reactive*, i.e. perform schedule revision in response to unforeseen events during schedule execution.

Our research [Miyashita.Sycara 94a, Miyashita.Sycara 94b], [Miyashita.Sycara 93, Sycara ed, Sycara 94a, Sycara 94b], [Sycara 94c, Zeng.Sycara ng]developed a case-based learning method *for acquiring context-dependent user optimization preferences and tradeoffs* and using them to incrementally improve schedule quality in generating a predictive schedule and also in reactively managing the schedule in response to unexpected execution events. The approach, implemented in the CABINS system, uses acquired user preferences to dynamically modify search control to guide schedule improvement. Unlike other systems that utilize iterative repair to find a feasible solution (e.g. [Zweben 90, Minton 90]), where executability of the schedule was not guaranteed at the end of each repair iteration, CABINS produces an executable schedule after each repair that has guaranteed monotonic increase in quality the more time it is allowed for repair, thus exhibiting *anytime executable* behavior [Dean 88]. This is a very desirable quality especially in reactive contexts since there could only be a certain limited amount of time for the system to react.

CABINS can operate in different modes that exhibit various levels of autonomy. First, *user-directed* mode, where the user selects a repair tactic and evaluates the results of its application. Second, *interactive assistance* mode, where CABINS suggests repair tactics and evaluations of repair tactic application, 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 case base. Third, *autonomous* mode where, without user intervention, CABINS uses the case base that was acquired in the training phase for repair selection and evaluation of repair results.

Our approach uses integration of Case-based Reasoning (CBR) [Kolodner 85] and fine granularity constraint-directed scheduling mechanisms. Integrating CBR with constraint-based scheduling stems from a variety of motivations. Although scheduling is an ill-structured domain, we assume that it exhibits domain regularities that could be captured, albeit only approximately, in a case. In CABINS, a case represents application of a revision action to one activity in the schedule, thus expressing dependencies among features of the schedule, the repair context and a suitable repair action. CBR allows capture and re-use of this dependency knowledge to dynamically adapt the search procedure and differentially bias scheduling decisions in future similar situations.

Since it is impossible to judge a priori the effects of a scheduling decision on the optimization objectives, a scheduling decision must be applied to a schedule and its outcome must be evaluated in terms of the resulting effects on scheduling objectives. Therefore, having a single scheduling decision as a case seemed to provide advantages in terms of focus and traceability of the problem solving process. Focus and traceability mean that we could capture a user's evaluation of the results of a single scheduling decision in a case, and, if the result was unacceptable, we could apply another scheduling decision to the same scheduling entity until either all available scheduling decisions had been exhausted or an acceptable result had been obtained. Therefore, it became clear that it was better to have a single activity/operation of a scheduling job as the "scheduling entity" on which a scheduling decision was applied. Hence in CABINS, a case describes the *application of a schedule revision decision on a single activity of a job.* Operationalization of a schedule revision decision is done by means of a *schedule repair action.* Currently, CABINS has 11 repair actions.

Since the result of a scheduling decision needed to be evaluated with regard to the optimization preferences for a schedule as a whole, it is clear that constructive methods which incrementally augment a partial schedule at every scheduling decision point would be unsuitable for our purposes. Moreover, contextual information, which can only be provided by having a complete schedule, is very useful in applying CBR. Therefore, revision-based scheduling was chosen as the underlying scheduling methodology.

Because of the tightly coupled nature of scheduling decisions, a revision in one part of the schedule may cause constraint violations in other parts. Therefore, constraint propagation techniques are necessary to determine the *ripple effects* that spread conflicts to other parts of the schedule as case-based repair actions are applied and specific schedule revisions are made. We use constraint propagation to propagate the effects of a schedule repair action to the rest of the schedule.

The evaluation criteria for judging the acceptability of the outcome of a repair action are often multiple, conflicting, context dependent and reflect user judgment of tradeoffs. Therefore, it is difficult to describe the evaluation criteria and the associated tradeoffs in a simple manner. The case base incorporates a distribution of examples that collectively and implicitly capture a user's schedule evaluation preferences and tradeoffs under diverse problem solving circumstances and enable CABINS to induce these tradeoffs from the case base. Hence, user preferences are reflected in the case base in two ways: as *preferences for selecting a repair action* depending on the features of the repair context, and as *evaluation preferences* for the repair outcome that resulted from selection and application of a specific repair action.

During iterative repair, cases are exploited for: (1) repair action selection, (2) evaluation of intermediate repair results and (3) recovery from revision failures. The method allows the system to dynamically switch between repair heuristic actions, each of which operates with respect to a particular local view of the problem and offers selective repair advantages. Application of a repair action tunes the search procedure to the characteristics of the local repair problem. This is achieved by dynamic modification of the search control bias. There is no a priori characterization of the amount of modification that may be required by repair actions. However, experimental results on job shop scheduling problem show that (1) the approach is potentially effective in capturing user preferences and optimization tradeoffs that are difficult to model, (2) it improves schedule quality irrespective of method of initial schedule generation, (3) it produces high quality schedules at much lower computational cost as compared to simulated annealing, a well-known iterative repair method, and (4) it is suitable as a reactive scheduling method because it maintains high schedule quality and minimizes disruptions in the face of execution time failures.

4.2 Case Indexing

Each application of a repair results in a new schedule. The search space of CABINS is the space of complete schedules that incorporate acceptable user optimization tradeoffs. Hence the predictive case features that are suitable for case indexing should be ones that capture good tradeoffs. Although schedule optimization is ill-structured, we make the hypothesis that there are regularities of the domain that can be captured, albeit in an approximate manner, in these features.

In CABINS, indices are divided into three categories. The first category consists of the *global features*. Since the results of schedule revision associated with a single activity pertain to the whole schedule, global features that express characteristics of a whole schedule are relevant and operate as contextual information for selection of a particular repair action. The *local features* comprise the second category. Since it is not possible to predict in general the bounds of repair necessitated by application of a repair action (due to constraint ripple effects), and since reasoning about the effects of a repair action on the whole schedule a priori would amount to unlimited lookahead analysis which is in general intractable, we confine the range of lookahead analysis to a limited *repair time horizon*. Associated with this time horizon, there are local features that allow CABINS to estimate the effects of each repair action.

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

CABINS searches the space of complete schedules. Control for this search is provided by CBR in two ways: First, search control is provided through case-based selection of the next repair action to be applied and second through case-based evaluation of the outcome for the schedule that resulted from application of a selected repair action. The global and local features are the indices that are used to retrieve a case that suggests the next repair action to be applied. The features associated with the repair history are used to retrieve cases that suggest evaluations of a repair outcome.

4.3 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 schedules by hand. At first, the user selects the repair tactic that is deemed to be appropriate and uses CABINS's tactic application procedure to apply the chosen tactic to the current schedule.

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

An outcome is judged as unacceptable, if the schedule resulting from the application of the revision heuristic does not make any improvement with respect to the user's criteria. This could happen because harmful effects outweighed, in the user's judgment, the effected improvement. For example, if reduction of job tardiness enforces increased utilization of low-quality machine, although the total cost of this repair may be low, it may be unacceptable to a user who worries that the quality of resulting products might be low. Therefore such a repair might be judged as unacceptable. The user's judgment as to balancing favorable and unfavorable effects related to a particular optimization objective constitute the explanations of the repair outcome. The user supplies an explanation in terms of rating the importance of each effect. At the end of each repair iteration, the applied repair tactic, the effects of the repair and user judgment / explanation as to the repair outcome are recorded in a case along with the current problem features. If the effects are acceptable to the user, the repair outcome is recorded as "acceptable" and the user tries to repair another activity. If the user does not like the tradeoffs that are incorporated in the repair effects, then the outcome of the current repair tactic ("unacceptable"), the effects calculated by CABINS and the salience assigned by the user are recorded in the repair history of the case. Subsequently, the user tries to utilize another repair tactic to repair the same activity.

The process continues until an acceptable outcome is reached, or failure is declared. Failure is declared when all available tactics have been used to repair an activity, but the user finds each repair outcome unacceptable. The sequence of application of successive repair actions, the effects, user's judgment and explanation in case of failed application are recorded in the repair history of the case. Two remarks are in order here with respect to case acquisition. First, a new case is acquired only when a new activity is under repair. When an activity is repeatedly repaired due to unacceptable repair tactic application results, no new case is acquired, but the repair history of the same case is augmented by each successive repair tactic application, its effects and outcome. In this way, a number of cases are accumulated in the case-base.

4.4 Case Re-Use

Once CABINS has constructed a case-base from training data, it can perform schedule repair without any interaction with its user. Cases are retrieved for three purposes: selection of a repair tactic to be applied, evaluation of the resulting schedule after application of the selected repair tactic, and, in case of failure, retrieval of a tactic that had fixed a previous similar failure. In each of these three situations, CABINS utilizes a different set of indices for case retrieval. In order to retrieve cases to select a repair tactic, global and local features of the current case (the current focal_activity) are used. For each of the three case retrieval situations described above, CABINS uses a k-Nearest Neighbor method (k-NN) [Dasarathy 90] for case retrieval.

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

If the outcome of current revision is decided as unacceptable, CABINS performs another CBR invocation using as indices the conjunction of the current outcome (unacceptable), the failed heuristic and the case global and local features to find another possibly applicable revision heuristic. Invoking CBR with these indices retrieves cases that have failed in the past in a similar manner as the current revision. This use of CBR in the space of failures is a domain-independent method of failure recovery [Sycara 88, Simpson 85], and allows the problem solver to access past solutions to the failure. If the result is acceptable, then CABINS proceeds to repair another activity.

4.5 Evaluation of the Approach

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

1. Our approach is potentially effective in capturing user preferences and optimization tradeoffs that are difficult to model.
2. Our approach improves schedule quality irrespective of method of initial schedule generation.
3. Our approach produces high quality schedules at much lower computational cost as compared to simulated annealing, a well-known iterative repair method.
4. Our approach is suitable as a reactive scheduling method because it maintains high schedule quality and minimizes disruptions in the face of execution time failures.

We evaluated the approach on a benchmark suite of job shop scheduling problems where parameters, such as number of bottlenecks, range of due dates and activity durations were varied to cover a broad range of parallel machine job shop scheduling problem instances. In particular, the benchmark problems have the following structure: each problem has 10 orders of 5 activities each. Each order has a linear process routing specifying a sequence where each order must visit bottleneck resources after a fixed number of activities, so as to increase resource contention and make the problem tighter. Two parameters were used to cover different scheduling conditions: a range parameter, RG, controlled the distribution of order due dates and release dates, and a bottleneck parameter, BK, controlled the number of bottleneck resources. To ensure that we had not unintentionally hardwired knowledge of the problem into the solution strategies, we used a problem generator function that embodied the overall problem structure described above to generate job shop scheduling instances where the problem parameters were varied in controlled ways. In particular, six groups of 10 problems each were randomly generated by considering three different values of the range parameter (static, moderate, dynamic), and two values of the bottleneck configuration (1 and 2 bottleneck problems). The slack was adjusted as a function of the range and bottleneck parameters to keep demand for bottleneck resources close to 100\% over the major part of each problem. Durations for activities in each order were also randomly generated.

The benchmark problems are variations of the problems originally reported in [Sadeh 91c] and used as a benchmark by a number of researchers (e.g. [Muscettola 92, Liu.Sycara 93a]). Our problem sets are, however, different in two respects: (a) we allow substitutable resources for non-bottleneck resources, thus solving the parallel machine rather than the simple job shop scheduling problem, and (b) the due dates of orders in our problems are tighter by 20 percent than in the original problems.

A cross-validation method was used to evaluate the capabilities of CABINS. Each problem set in each class was divided in half. The overall training sample, consisting of 30 problems, each of which has 50 activities, was repaired to gather cases. A case is acquired for each activity that is repaired. An activity (and consequently a job) may be repaired more than once during an overall repair cycle, since it could require repair after being moved as a result of repairing another activity. Allowing each activity to be repaired once for each problem would give a maximum of 30X50 = 1,500 cases for each training sample. In our experiments, some of the activities did not need repair. So, for each training sample, CABINS was trained with approximately 1,100 cases. These cases were then used for case-based repair of the validation problems (the other 30 problems). We repeated the above process by interchanging the training and the test sets. Since it is not possible to theoretically predict the bounds of repair or the global optimum, in the experiments, CABINS was allowed to run for three overall repair cycles.

4.6 Summary of Experimental Results

Extensive experimentation on the benchmark suite of problems showed that:

1. CABINS is capable of acquiring user and state dependent schedule optimization preferences. In addition, CABINS can acquire user preferences that *change over time.*
2. In predictive schedule generation, the methodology consistently improves the quality of schedules generated by a variety of scheduling methods.
3. In predictive schedule generation, CABINS generates schedules of higher quality along a variety of optimization objectives with much lower processing cost (almost an order of magnitude better) as compared to simulated annealing.
4. In recovering from execution time failures, the approach (1) attends to schedule quality both in terms of optimization objectives, and disruption, and (2) is responsive in that it allows continuation of execution without delays in response to execution failures, and (3) it exhibits anytime executable behavior.
5. Different scheduling objectives implicitly reflected in the case base differentially bias the schedule repair procedure. Experiments showed that learning of the control model for repair action selection improved schedule quality by 90% as compared with random selection of repair actions.
6. The approach scales up in that knowledge gathered in the form of cases from a smaller set of problems (e.g. 10-job problems) produces schedules of high quality when used to repair larger problems (e.g. 20-job problems). In addition the pattern of schedule quality improvement independent of method of initial schedule generation holds for the larger set of problems.
7. With respect to the question of the case size that will give the "optimal" tradeoff with respect to schedule quality vs case acquisition and retrieval cost, our results showed that:
    - The larger the number of cases, the better the schedule quality. However, the marginal payoff from the increase in case base size decreases. This can be explained partially by the fact that some number of cases (say, 1000 cases) capture well characteristics of the problem space, and additional 1000 new cases may give much redundant information. When the size of case-base is relatively small, every time new cases are acquired, we may get information about a different part of the problem space which results in higher quality improvement.

- In terms of balancing efficiency of finding the solution and solution quality, the experiments showed that the case-base with 1000 cases affords the best tradeoff.

4.6.1 Discussion

We believe the power of the approach stems from the following four reasons. First, as has been pointed out by others (e.g. [Minton 92]), revision-based approaches, by making available a complete assignment (a complete schedule) provide more information that can guide search as compared with constructive methods where only a partial assignment is available. Our CBR-based revision method captures such relevant information in global case features and exploits it as contextual information during case retrieval. Second, although job shop schedule optimization belongs to the category of "hard" NP-complete problems, the case features were able to capture some important domain regularities, such as repair flexibility. This was complemented by keeping information about failed applications of revisions in the repair case history and also keeping failed cases in the case memory. These failures were exploited by CBR to prune unpromising paths in the search space in future similar situations. Finally, we believe that some of the regularities in the structure of the experimental problems were captured in cases during the training phase and this information was transferable to solve the test problems. Moreover, this information seems to transfer also across problem size. For example, the cases acquired during training with a set of 10-job problems were effective in solving test problems with 20 jobs.

The effort expended to capture a large number of cases can be amortized by future repeated use of the case base to get high quality schedules efficiently. More importantly, CABINS can acquire the cases through user interaction during the process of solution improvement without imposing undue overhead on the user.

5 Integration of Predictive Planning/Scheduling and Reactive Control

A plan/schedule represents a predictive view of how the future should look and expresses expectations about future events. For example, a factory schedule expresses an expectation that particular operations will be assigned to particular resources at particular start times and will execute on these resources for the indicated operation durations. However, since the world is unpredictable, the expectations associated with the predictive plan/schedule might not be realized (e.g., operations might finish earlier or later than their durations indicate in the schedule, capacity might be lost due to machine breakdowns etc). These realities of execution uncertainty give rise to the following issues: (1) how execution of a plan/schedule should be controlled, (2) how a schedule can be reactively managed and (3) how the behavior of the world could be simulated (since our system does not have access to a real factory floor that can be sensed, simulation of the world must be employed).

Our research create a distributed testbed for planning and execution that is suitable to experiment with interactions and tradeoffs arising from adopting a variety of control regimes to select tasks for execution, the actual execution policies and the physics of the world.

The distributed testbed cosists of the following agents: a planner/scheduler, a controller (in the manufacturing domain this is called the dispatcher) and a simulator that simulates the behavior of the world. The overall system behavior is the result of interactions among the three agents. The goal of this effort is to make the testbed sufficiently parametrized and general so that it can operate using a variety of possible scheduling strategies, a variety of control execution regimes and a variety of assumptions in terms of the physics of the world. In addition, monitoring processes will be gathering performance statistics for particular experimental combinations of scheduling strategies, control regimes and world models.

In order to gather statistics on the performance of the distributed planner/scheduler, controller and simulator, we implemented an additional *monitoring agent*. Its role is to monitor the state of the different agents and the state of the messages exchanged among the different agents and return appropriate statistics. For example, by collecting the messages announcing the start of the first operation and the end of the last operation for each job, the monitor can return statistics on tardiness; analogously, messages on start and end of resource down-time, type of failure and start and end of failure repair activities can be gathered to obtain statistics on the effectiveness of the reaction of the dispatching policies to unpredictable events.

In general, the system performs as follows. The scheduler passes to the dispatcher a DAG of operations to be executed. The simulatior simulates the execution of operations based on different conditions of uncertian execution (e.g., every third operation has a probability x of excheeding its time bounds by y). If an operation executes successfuly (i.e. within its time bounds) then, it becomes a finished operation. If its execution causes constraint violation, then, the dispatcher tries to "fix" the situation performing local repairs. If the dispatcher repairs do not resolve the constraint conflicts, the dispatcher returne the operations to the scheduler for re-scheduling.

The dispatcher has limited time horizon within which to impose repairs. This limited time horizon has two effects (1) it limits the search space of the dispatcher thus enabling quick repairs, and (2) keeps the repairs local, i.e. without causing inconsistencies in the global scheduler's constraints. these two characteristics enable the dispatcher to respond efficiently to detected problems and opportunities.

5.0.1 Simulator

The simulator allows generation of behaviors of finite state machines equivalent to HSTS models used for planning and scheduling. The explicit simulation of the behavior of a system under consideration and of the interactions with control and prediction (planning and scheduling) agents is the basis for the study of appropriate regimes of coordination. The simulator takes as input a **model**, an **initial state**, and a start and an end time for the simulation. The model defines state transitions, agents and objects that can respond to messages. Both agents and objects communicate by sending and receiving messages. For the sake of better exposition, we call the messages handled by objects *signals* and the messages handled by agents *stimuli*. These messages are contained in temporal order in the simulator's queue. Objects have associated state variables and state transition rules; given the current state and a set of received messages, an object can change its state according to its transistion rules. Agents consist of arbitrary code that can receive messages (stimuli), and can respond with commands. Only the method of communication is described for agents; no visibility is given on the internal state of the agents. This capability will allow the incorporation of complex agents (e.g., planner, scheduler) in the simulation of complex regimes of coordination.

The simulator maintains the value of the state variables associated to the model objects. The current set of assignments of the state variables within the simulator constitute the *current state*. State transitions can be described as:

```
    old state + signal => new state + post new signals
```

An agent receives stimuli generated by the simulator and sends commands to the simulator. Objects have handler functions that select the appropriate state transition rules and apply them to generate changes in the value of its state variables.

State Transition Rules (STR) consist of:

- signal-test. It is used to determine whether this STR should be considered given the current signal.
- prior-state. It is used to determine whether this STR should be considered given the current state and the current signal.
- post-state. It is used to assign to state variables the values they will have after all variable assignments are done. There are two types of value assignment: (1) a simple assignment where the value given to the state variable is a function of prior-state, signal, and STR, and (2) a functional assignment, where the state variable assignment is a function of prior-state, signal, STR and new value of the given variable (caused by the assignmenet of some other STR).
- validity-test. It is a test to check whether the resulting state is a valid one.

5.0.2 Scheduler and Dispatcher

The scheduler agent of the predictor/controller/simulator architecture is based on the Conflict Partition Scheduling procedure. The scheduler generates an HSTS temporal data base representing the network of precedence among activities and their time bounds. The dispatcher receives a subnetwork consisting of the first operations that can be executed and the activities that follow them on a chain of job precedences (i.e., op_i before op_j because of a job imposed constraint) or on a chain of resource precedences (i.e., op_i before op_k because they use the same resource). At any point in time the dispatcher will contain up to *n* links on any chain; this is an extension of the approach in [Smith 90b] which considers chains of length 2 at most. The constraints imposed by the rest of the schedule not visible to the dispatcher are represented as absolute temporal constraints (i.e., due date constraints); the constraints are updated every time new operations are added to the dispatcher's temporal data base.

Unexpected events are represented in the dispatcher's network with the change of activity parameters (e.g., change of an activity duration constraint if the simulation determines that its duration is shorter or longer than expected) or the introduction of new activities (e.g., a resource unavailability requires the introduction of a "resource down" state token and a "resource repair" activity token). There are two categories of unexpected events: (1) Small changes, such as delays in the execution of activities, that could be "absorbed" in the current time map flexibility (because, for ex. there is downstream slack present). In this case the controller does not need to execute any adjustment to its portion of the schedule (over which it has visibility). In constrast, a controller using "crisp" schedules, such as the one described in [Smith 90b] often deals with similar situations with an active modification of the schedule through appropriate reactive scheduling rules that could incurr computational cost. (2) Large perturbations to the dispatcher's time map, due for example to a machine break down or to large execution delays, might generate inconsistencies in the current schedule. In this case it is necessary to determine where the inconsistency is and how to repair it. We concentrate on the detemination of inconsistencies that are representable as temporal constraint violations in the network. The determinations of these inconsistencies requires an extension of the HSTS temporal propagation mechanism. The HSTS temporal propagation, in fact, can detect the presence of inconsistencies but cannot localize them; this is sufficient for the backtrack-based approaced to planning and scheduling but not in repair based approaches where we need to identify the constraints that participate in the conflict and, therefore, need to be modified. We started the investigation of propagation procedures that localize temporal constraint inconsistencies.

We implemented and integrated in the HSTS temporal data base an additional temporal propagation mechanism which allows detection of the location of conflicting sets of constraints. The method applies the Floyd-Warshall (FW) algorithm for the determination of all-pair shortest paths. The algorithm has been advocated for the resolution of consistent networks of temporal constraints in [Dechter 91]. While in most cases FW computes much more information than it is needed (e.g., consistency of a temporal data base), in case of network inconsistency it can detect shortest cyclical paths of temporal constraints with negative distance. The elimination of such paths from the temporal data base is a necessary condition for the removal of the network's inconsistency. As for the ordinary time bound propagator, in HSTS the FW propagator can be called on demand, given the computational burden of the algorithm. A typical situation is one in which FW is called after the ordinary and cheaper time bound propagation detects an inconsistency. The wealth of information returned by FW can be used to guide the design of effective repair rules. For example, having detected a number of disjoint negative distance cycles, it is necessary to repair all of them before the temporal database can become consistent again. The repair rules will use different patterns of repair (e.g., repair one cycle at a time, exchange constraints among cycles, etc.) depending on the topology of the temporal data base.

5.0.3 Dispatcher Operation

The dispatcher receives an externally generated DAG of operations with nominal start and end times associated with each operation. This DAG constitutes a consistent schedule.

Execution time failures

The dispatcher's reactive behavior in response to execution time failure, such as an activity finishign late, is as follows:

- Identification of deviation from nominal plan/schedule behavior: this is done using the execution information that the managed system communicates to dispatcher.
- Localization of violated constraints: this is done via use of Floyd Warshall algorithm to identify negative cycles in the temporal network under dispatcher's control.
- Identification of cause of deviation: activities that are part of a negative cycle are potential causes.
- Local repair:
    1. Next activity of a negative cycle is unlinked from the network.
    2. Constraint propagation is performed.
    3. If propagation shows that the network is now consistent, the unlinked activity is relinked at a place where there is enough slack. Repair is complete. Otherwise, previous activity is put in its old place. Go to step 1.
    4. If all activities in a negative cycle have been tested but the network is still inconsistent, the dispatcher sends to the scheduler all activities that have not finished executing for rescheduling.

Dispatcher execution opportunities

The dispatcher also responds to execution opportunities (e.g., an activity finishing earlier than its expected finish time). The intuition behind a revision to recognize and take advantage of opportunities is to try, if possible, to locally re-optimize the part of the schedule under the control of the dispatcher, The "opportunity revision" can be initiated under the following circumstances: (1) when an activity finishes earlier than its expected finish time, (2) when an activity finishes later than its expected finish time (but the dispatcher's constraint network is still consistent). In this case, the "opportunity revision" is tried along with the 'repair revision". (3) when an activity finishes at its expected finish time, and (4) when the dispatcher gets a new set of activities to dispatch.

The opportunity revision is checked for each idle resource, and each activity, call it "current activity", on that resource (except the first activity). The opportunity revision steps are as follows:

1. Break resource links before and after the "current activity"
2. Add resource link between the old prior-activity to the "current activity", and old

next-activity to the "current activity"

3. Try placing the "current activity" in the first position on the resource
4. Check if the constraint network is consistent. If it is, then the "current activity" is a "candidate" to be moved. Keep track of its minimum legal start time in the "candidate-set". If not, place the "current activity" in its old position (taking care of maintaining the correct links) and try another activity.
5. Collect all activities in the "candidate-set" and select the one with the earliest minimum start time.

The algorithm ensures that the activity that can be dispatched at the earliest time is found.

5.0.4 Coordination protocol scheduler-dispatcher-simulator

The coordination assumption that underlie this protocol is that the dispatcher is actively asking the scheduler for activities to dispatch rather than waiting for the scheduler to send it activities. The implication of this coordination regime is that activities will be dispathed expeditiously, i.e., the next ply of activities will be dispatched as soon as the previous ply has been sent to the factory model. The dispatcher gets alerted that the scheduler has finished scheduling (initial scheduling or rescheduling) by receiving a *schedule-ready* message from the scheduler. The message that the dispatcher sends to the scheduler to ask for activities to dispatch is *get-next-ply*. The scheduler responds to this message by sending a series of *dispatch-activity* messages, one for each activity in the first n plies. n is currently equal to 2. When the dispatcher dispatches an activity, it checks to see if it needs another ply, and if it does, it sends get-next-ply message to scheduler. If, due to delays in activity execution, the dispatcher finds unrepairable inconsistencie, it send a series of *rescehdule-activity* messages to the scheduler. The argument to each of these messages is an activity to be rescheduled.

The dispatcher dispatches activities by sending *start-activity* messages to the factory. The factory responds to the dispatcher by sending *activity-started* message when it starts executing an activity, and *activity-finished* message at the end of each activity execution.

5.0.5 Operation of Overall System

The overall system operates as follows;

1. At simulation time=-2, a set of self-initialization messages is sent by the simulation infrastructure to all agents. The result of recieving these messages is that each agent initializes itself (i.e. it local variables and message types it can send and receive).
2. At simulation time=-1, a set of messages is sent to all agents for initialization of others. For example, the scheduler initializes its interaction with the dispatcher by sending a "schedule-ready" message; the dispatcher initializes its interactin with the scheduler by sending a "get-next-ply" message (to get the first 2 plies), to which the scheduler responds by the "dispatch-activity" series of messages, seding it the requested activities. When the dispatcher gets the activities, it builds a

temporal constrant network for the activities.

3. At simulation time=0, a start message is sent to all agents. At that time the dispatcher builds its temporal data base and checks it for consistency.

4. At simulation time=k, where k is the earliest release date of the activities to be dispatched, the dispatcher sends messages to the factory to start execution of each activity in the first ply, on the specified resource for each activity.

In the rest of the simulation, the following significant events occur:

- The factory sends activity-finished messages to dispatcher, as activities finish execution.
- As the dispatcher dispatches each activity, it checks to see whether it needs an additional ply from the scheduler. If it does, it sends a "get-next-ply" message, which has a papameter n equal to the number of plies to be sent. The scheduler responds by "dispatch-activity" series of messages, thus sending the dispatcher the set of activities in the next n plies.
- If the dispatcher finds that its temporal data base is inconsistent, it tries to locally repair the inconsistency. If the inconsistency is locally fixed, the dispatcher continues processing. If the inconsistency cannot be fixed locally, the dispatcher sends the scheduler a "reschedule-activity" series of messages containing the activities to be rescheduled.

References

[Adams 88]	J. Adams, E. Balas, and D. Zawack. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science* 34(3):391-401, 1988.
[Berry 91]	Pauline M. Berry. *The PCP: A Predictive Model for Satisfying Conflicting Objectives in Scheduling Problems.* Technical Report, Centre Universitaire d'Informatique, Universite de Geneve, 12, Rue du Lac, CH-1207, Geneva, Switzerland, 1991.
[Chen 93]	Chen, S., S. Talukdar and N. Sadeh. Job Shop Scheduling Using Asynchronous Teams of Optimization Agents. In *Proceedings of the IJCAI-93 Workshop on Knowledge-based Production Planning, Scheduling, and Control.* Chambery, France, August, 1993.
[Dasarathy 90]	Belur V. Dasarathy (editor). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamos, CA, 1990.
[Dean 88]	Dean, T. and Boddy, M. An Analysis of Time Dependent Planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49-54. AAAI, Saint Paul, Minnesota, 1988.
[Dechter 91]	Dechter, R. and Meiri, I and Pearl, J. Temporal Constraint Networks. *Artificial Intelligence* 49:61-95, May, 1991.
[Garey 79]	M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman and Co., 1979.
[Garey 88]	Michael R. Garey, Robert E. Tarjan, and Gordon T. Wilfgong. One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties. *Mathematics of Operations Research* 13(2):330-348, May, 1988.
[Goldratt 80]	Eliyahu M. Goldratt. Optimized Production Timetable: Beyond MRP: Something Better is finally Here. October, 1980 Speech to APICS National Conference.
[Kolodner 85]	Kolodner, J. and Simpson, R. and Sycara, K. A Process of Case-Based Reasoning in Problem Solving. In *Proceedings of the Ninth International Joint Conference on Aritificial Intelligence*, pages 284-290. IJCAI, Los Angeles, CA, 1985.

[Li&Sadeh 93] Gang Li and Norman Sadeh.
Single-Machine Early/Tardy Scheduling Problem with Setups: A Hybrid Heuristic Approach.
Technical Report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1993.
Working paper. Presented at the Joint National ORSA/TIMS meeting held in San Francisco, November 1-4, 1992.

[Linden 91] Theodore A. Linden.
Preference-Directed, Cooperative Resource Allocation and Scheduling.
Technical Report, Advanced Decision Systems, 1500 Plymouth St., Mountain View, CA 94043, September, 1991.

[Liu.Sycara 93a] Liu, J. and Sycara, K.
Distributed Constraint Satisfaction through Constraint Partition and Coordinated Reaction.
In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence.* AAAI, Hidden Valley, PA., 1993.

[Liu.Sycara 93b] Liu, J. and Sycara, K.
Constraint Satisfaction through Multi-Agent Coordinated Interaction.
In *Proceedings of the 5th European Workshop on Modeling Autonomous Agents in a Multi-Agent World.* MAAMAW, Neuchatel, Switzerland, 1993.

[Liu.Sycara 94a] Liu, J. and Sycara, K.
Problem Solving through Coordinated Reaction.
In *Proceedings of the IEEE Conference on Evolutionary Computation.* IEEE, Orlando, Fla., 1994.

[Liu.Sycara 94b] Liu, J. and Sycara, K.
Distributed Problem Solving through Coordination in a Society of Agents.
In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence.* AAAI, Seattle, WA., 1994.

[McKay 88] K. McKay, J. Buzacott, and F. Safayeni.
The Scheduler's Knowledge of Uncertainty: The Missing Link.
Technical Report, Department of Management Sciences, University of Waterloo, Waterloo, Ontario, Canada, N2K 2G4, 1988.
Also presented at IFIP Working Conference on Knowledge Based Production Management Systems, Galway, Ireland, August 1988.

[Minton 90] S. Minton, M.D. Johnston, A.B. Philips, P. Laird.
Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method.
In *Proceedings of the Eighth National Conference on Artificial Intelligence,* pages 17-24. 1990.

[Minton 92] Minton, S. and Johnston, M and Philips, A. and Laird, P.
Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems.
Artificial Intelligence 58(1-3):161-205, 1992.

[Miyashita.Sycara 93]
Miyashita, K., Sycara, K.
Adaptive Control of Schedule Revision.
In Fox, M. and Zweben, M. (editor), *Knowledge-Based Scheduling*. Morgan Kaufmann, San Mateo, CA, 1993.

[Miyashita.Sycara 94a]
Kazuo Miyashita and Katia Sycara.
A Framework for Case-Based Revision for Schedule Generation and Reactive Schedule Management.
Journal of Japanese Society for Artificail Intelligence 9(3):426-435, 1994.

[Miyashita.Sycara 94b]
Miyashita, K. and Sycara, K.
CABINS: A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair.
Technical Report, Carnegie Mellon University, CMU-RI-TR-94-34, 1994.

[Muscettola 92] N. Muscettola.
Scheduling by Iterative Partition of Bottleneck Conflicts.
Technical Report CMU-RI-TR-92-05, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1992.

[Nakakuki&Sadeh 94]
Nakakuki, Yoichiro, and Norman Sadeh.
Increasing the Efficiency of Simulated Annealing Search by Learning to Recognize (Un)Promising Runs.
In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1316-1322. 1994.

[Ow 88] Peng Si Ow and Stephen F. Smith.
Viewing Scheduling as an Opportunistic Problem-Solving Process.
Annals of Operations Research 12:85-108, 1988.

[Ow 89] Peng Si Ow and Thomas Morton.
The Single Machine Early/Tardy Problem.
Management Science 35(2):177-191, 1989.

[Paolucci 92] Paolucci, E., Patriarca, E., Sem, M., and Gini G.
Predit: A Temporal Predictive Framework for Scheduling Systems.
In *Proceedings of the AAAI Spring Symposium on Practical Approaches to Scheduling and Planning*, pages 150-154. 1992.

[Sadeh 91a] Norman Sadeh and Mark S. Fox.
Micro- vs. Macro-opportunistic Scheduling.
In G. Doumeingts, J. Browne, and M Tomljanovich (editor), *Proceedings of the Fourth IFIP Conference on Computer Applications in Production and Engineering (CAPE'91)*, pages 651-658. Elsevier Science Publishers B.V. (North Holland), 1991.

[Sadeh 91b] N. Sadeh and M.S. Fox.
Variable and Value Ordering Heuristics for Hard Constraint Satisfaction Problems: an Application to Job Shop Scheduling.
Technical Report CMU-RI-TR-91-23, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1991.
Submitted to the Artificial Intelligence Journal.

[Sadeh 91c] Norman Sadeh.
Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.
PhD thesis, School of Computer Science, Carnegie Mellon University, March, 1991.

[Sadeh 92a] Norman M. Sadeh.
The Micro-Boss Scheduling System: Current Status and Future Efforts.
In *Proceedings of the 1992 AAAI Spring Symposium on Practical Approaches to Scheduling and Planning*, pages 37-41. Stanford University, Stanford, CA, March, 1992.
Also appeared in Proceedings of the Sixth Annual Workshop on Space Operations Applications and Research (SOAR'92) held in Houston, TX, Aug. 4-6,1992.

[Sadeh 92b] Norman M. Sadeh.
Micro-Boss: A Micro-opportunistic Decision Support System for Factory Scheduling.
The 1991 Annual Research Review of the Robotics Institute.
1992

[Sadeh 92c] Norman Sadeh, Katia Sycara, and Yalin Xiong.
Backtracking Techniques for Hard Scheduling Problems.
Technical Report CMU-RI-TR-92-06, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1992.
Submitted to the Artificial Intelligence Journal.

[Sadeh 93a] Norman Sadeh.
MICRO-BOSS: A Micro-opportunistic Factory Scheduler.
Expert Systems With Applications 6(3):377-392, July-September, 1993.
Special Issue on Scheduling Expert Systems and their Performances. Also published as Carnegie Mellon University technical report CMU-RI-TR-91-22.

[Sadeh 93b] Sadeh, N.M., S. Otsuka, and R. Schnelbach.
Predictive and Reactive Scheduling with the Micro-Boss Production Scheduling and Control System.
In *Proceedings of the IJCAI-93 Workshop on Knowledge-based Production Planning, Scheduling, and Control.* Chambery, France, August, 1993.

[Sadeh 94a] Norman M. Sadeh.
Micro-Boss: Towards a New Generation of Manufacturing Scheduling Shells.
In *Proceedings of the ARPA/Rome Laboratory Knowledge-Based Planning and Scheduling Initiative*, pages 191-203. Tucson, AZ, Februrary, 1994.

[Sadeh 94b] Norman Sadeh, Katia Sycara, and Yalin Xiong.
Backtracking Techniques for the Job Shop Scheduling Constraint Satisfaction Problem.
Artificial Intelligence Journal , 1994.
To appear in Special Issue on 'Planning and Scheduling'.

[Sadeh 94c] Norman Sadeh.
Micro-Opportunistic Scheduling: The MICRO-BOSS Factory Scheduler.
Intelligent Scheduling.
In Mark Fox and Monte Zweben,
Morgan Kaufmann Publishers, 1994, Chapter 4.

[Sadeh&Nakakuki 94]
Sadeh, Norman, and Yoichiro Nakakuki.
Focused Simulated Annealing Search: An Application to Job Shop Scheduling.
Technical Report, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1994.
Submitted to Annals of Operations Research, Issue on 'Metaheuristics in Combinatorial Optimization.

[Saks 92] Victor Saks, Al Kepner, and Ivan Johnson.
Knowledge Based Distribution Planning.
Technical Report, Carnegie Group, Inc., 5 PPG Place, Pittsburgh, PA 15222, 1992.

[Simpson 85] Simpson, R.L.
A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation.
PhD thesis, School of Information and Computer Science Georgia Institute of Technology, 1985.

[Smith 90a] Stephen F. Smith, Peng Si Ow, Nicola Muscettola, Jean-Yves Potvin, Dirk Matthys.
An Integrated Framework for Generating and Revising Factory Schedules.
Journal of the Operational Research Society 41(6):539-552, 1990.

[Smith 90b] Smith, S.F., N. Keng, and K. Kempf.
Exploiting Local Flexibility During Execution of Pre-Computed Schedules.
Technical Report CMU-TR-RI-90-13, The Robotics Institute, Carnegie Mellon Univeristy, June, 1990.

[Swaminathan 93] Swaminathan, J., N.M. Sadeh, and S.F. Smith.
A Knowledge-Based Multi-Agent Simulation Testbed to Support Supply Chain Design and Management Decisions.
In *Proceedings of the IJCAI-93 Workshop on Knowledge-based Production Planning, Scheduling, and Control.* Chambery, France, August, 1993.

[Swaminathan 94] Swaminathan, J., N.M. Sadeh, and S.F. Smith.
Impact of Supplier Information on Supply Chain Performance.
Technical Report, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1994.
Submitted to the Journal of Operations Management, Special issue on 'Economics of Operations Management'.

[Sycara 88] Sycara, K.
Patching Up Old Plans.
In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.* Montreal, Canada, 1988.

[Sycara 94a] Sycara, K., and Miyashita, K.
Adaptive Schedule Repair.
In *Proceedings of the 27th Hawaii International Conference on System Sciences.* Maui, Hawaii, 1994.

[Sycara 94b] Sycara, K. and Miyashita K.
Case-Based Acquisition of User Preferences for Solution Improvement in Ill-Structured Domains.
In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94).* Seattle, WA., 1994.

[Sycara 94c] Sycara, K. and Miyashita, K.
Evaluation and Improvement of Schedules According to Interactively Acquired User-Defined Criteria.
In *Proceedings of the Planning Initiative Workshop.* Arpa, Tucson, AZ., 1994.

[Sycara ed] Katia Sycara and Kazuo Miyashita.
Learning Control Knowledge through Case-Based Acquisition of User Optimization Preferences in Ill-Structured Domain.
In Tecuci, G. and Kodratoff, Y. (editor), *Machine Learning and Knowledge Acquisition: Integrated Approaches.* Morgan Kaufmann, San Mateo, CA, To be published.

[Torma 91] Seppo Torma, Ora Lassila and Markku Syrjanen.
Adapting the Activity-Based Scheduling Method to Steel Rolling.
In G. Doumeingts, J. Browne, and M Tomljanovich (editor), *Proceedings of the Fourth IFIP Conference on Computer Applications in Production and Engineering (CAPE'91)*, pages 159-166. Elsevier Science Publishers B.V. (North Holland), 1991.

[Winklhofer 92] Andreas Winklhofer, Manfred Maierhofer, and Paul Levi.
Efficient Propagation and Computation of Problem Features for Activity-Based Scheduling.
In *Proceedings of the Seventh Symposium on Information Control Problems in Manufacturing Technology (INCOM-92).* Toronto, Canada, 1992.

[Xiong 92] Yalin Xiong, Norman Sadeh, and Katia Sycara.
Intelligent Backtracking Techniques for Job Shop Scheduling.
In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 14-23. KR'92, Cambridge, MA, October, 1992.

[Zeng.Sycara ng] Zeng, D. and Sycara, K.
Case-Based Acquisition of User Changing Preferences.
Technical Report, Carnegie Mellon University, Forthcoming.

[Zweben 90] M. Zweben and M. Deale and M. Gargan.
Anytime Rescheduling.
In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 251-259. DARPA, San Diego, CA., 1990.

[Zweben 91] Monte Zweben, Eugene Davis, and Michael Deale.
Iterative Repair for Scheduling and Rescheduling.
Technical Report, NASA Ames Reserch Center, MS 244-17, Moffett Field, CA 94035, 1991.

Micro-Opportunistic Scheduling

Norman M. Sadeh
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, Pennsylvania 15213-3891
sadeh@cs.cmu.edu

Abstract

A major challenge for research in production management is to develop new finite-capacity scheduling techniques and tools that (1) can account more precisely for actual production management constraints and objectives, (2) are better suited for handling production contingencies, and (3) allow the user to interactively manipulate the production schedule to reflect idiosyncratic constraints and preferences not easily amenable to representation in the computer model. This chapter describes Micro-Boss, a decision-support system for factory scheduling currently under development at Carnegie Mellon University. Micro-Boss aims at generating and maintaining high-quality realistic production schedules by combining powerful predictive, reactive, and interactive scheduling capabilities. Specifically, the system relies on new *micro-opportunistic* search heuristics that enable it to constantly revise its scheduling strategy during the construction or repair of a schedule. These search heuristics are shown to be more effective than less flexible scheduling techniques proposed in the Operations Research and Artificial Intelligence literature.

This chapter summarizes our work in micro-opportunistic scheduling and describes predictive, reactive and interactive capabilities developed in the context of the Micro-Boss scheduling system. It is a condensed version of three papers: [Sadeh 94], [Sadeh 93] and [Li&Sadeh 93].

This research was supported, in part, by the Advanced Research Projects Agency under contract #F30602-91-F-0016 and in part by grants from McDonnell Aircraft Company, Digital Equipment Corporation.

1 Introduction

In a global market economy, the need for cost-efficient production management techniques is becoming more critical every day. In contrast with this need, current production management practice is too often characterized by low levels of due date satisfaction, high levels of inventory and, more generally, a state of chaos in which the computer systems that are used to provide managerial guidance do not accurately reflect the current state of affairs, because they rely on oversimplified and rigid models of the production environment. A major challenge for research in this area is to develop new production management techniques and tools that (1) can account more precisely for actual production management constraints and objectives, (2) are better suited for handling production contingencies, and (3) allow the user to interactively manipulate the production schedule to reflect idiosyncratic constraints and preferences not easily amenable to representation in the computer model. This chapter describes Micro-Boss, a decision-support system for factory scheduling currently under development at Carnegie Mellon University. Micro-Boss aims at generating and maintaining high-quality realistic production schedules by combining powerful predictive, reactive, and interactive scheduling capabilities. Specifically, the system relies on new *micro-opportunistic* search heuristics that enable it to constantly revise its scheduling strategy during the construction or repair of a schedule. These search heuristics are shown to be more effective than less flexible scheduling techniques proposed in the Operations Research and Artificial Intelligence literature.

1.1 The Production Scheduling Problem

Production scheduling requires allocating resources (e.g., machines, tools, human operators) over time to a set of jobs while attending to a variety of constraints and objectives.

Typical constraints include

- *functional constraints* limiting the types of operations that a specific resource can perform

- *capacity constraints* restricting the number of jobs a resource can process at any given time

- *availability constraints* specifying when each resource is available (e.g., number of shifts available on a group of machines)

- *precedence constraints* existing between operations in a job, as specified in the job's process routing

- *processing time constraints* specifying how long it usually takes to perform each operation

- *setup constraints* requiring that each machine be in the proper configuration before performing a particular task (e.g., proper sets of fixtures and tools)

- *time-bound constraints* specifying for each job an earliest acceptable release date before which the job cannot start (e.g., because its raw materials cannot arrive earlier) and a due date by which ideally it should be delivered to a customer

Some of these constraints must be satisfied for a schedule to be valid (so-called non-relaxable or hard constraints). For instance, milling operations can only be performed on milling machines. Other groups of constraints are not always satisfiable and might need to be relaxed (so-called relaxable or soft constraints). For instance, due date constraints often need to be relaxed for a couple of jobs because of the limited capacity of the production facility. Availability constraints are another example of constraints that can be relaxed, by either working overtime or adding extra shifts. A good schedule is one that satisfies all hard constraints while selectively relaxing soft constraints to maximize performance along one or several metrics.

Two factors that critically influence the quality of a schedule are due date satisfaction and inventory levels. Missing a customer due date can result in tardiness penalties, loss of customer orders, delayed revenue receipts, etc. Inventory costs include interests on the costs of raw materials, direct inventory holding costs, interests on processing costs, etc. One often distinguishes between in-process inventory costs (also referred to as work-in-process inventory costs) and finished-goods inventory costs. Work-In-Process (WIP) inventory costs account for inventory costs resulting from orders that have not yet been completed, and finished-goods inventory costs result from completed orders that have not yet been shipped to customers.

Manufacturing contingencies such as machine breakdowns, late arrivals of raw materials, and variations in operation durations and yields further complicate production scheduling. In the face of contingencies, schedules need to be updated to reflect the new state of affairs. The sheer size of most factory scheduling problems precludes the generation of new schedules from scratch each time an unanticipated event occurs. In fact, most contingencies do not warrant such extreme actions and are best handled by repairing a portion of the existing schedule [Bean 91].

As schedules are optimized at a more detailed level, they can also become more sensitive to disruptions and require more frequent repairs. In general, there is a limit to the amount and detail of information that one can reasonably expect to represent in a computer model. For instance, a worker's preference for performing more demanding tasks in the morning might not be worth storing in the computer model and, instead, might be best accounted for by allowing the end-user to interactively manipulate the schedule.

Even under idealized conditions such as simplified objectives (e.g., minimizing total tardiness or maximizing throughput) and deterministic assumptions, scheduling has been shown to be an NP-hard problem [Garey 79, Graves 81, French 82]. Uncertainty further adds to the difficulty of the problem, and makes it even more impractical to look for optimal solutions. Instead, practical approaches to production scheduling are heuristic in nature. The next subsection briefly reviews earlier approaches to production scheduling; identifies some of their shortcomings; and introduces a new search paradigm, called micro-opportunistic search, that shows promise for addressing some of these shortcomings.

1.2 A Micro-Opportunistic Approach to Production Scheduling

To this date, the most widely used computer-based approach to production scheduling remains by far the Material Requirements Planning (MRP) or Manufacturing Resource Planning (MRP-II) approach developed in the 1970s [Orlicky 75, Wight 81, Wight 84]. In this approach, demand for end-products as specified in a Master Production Schedule is exploded into time-phased requirements for component items (subassemblies, parts, raw materials, etc.) required for the production of these end-products[1]. Because their time-phasing logic relies on standard operation leadtimes that do not account for the actual load of the production facility, MRP systems often fail to produce realistic schedules. They sometimes overload the facility, thereby causing orders to be delivered late. In an attempt to alleviate this problem, MRP systems often pad the schedule by inserting generous "safety" leadtimes. These safety leadtimes tend to be rather arbitrary and produce unnecessarily large amounts of inventory. In fact, because they are often unrealistic and are not meant to be updated in real-time[2], MRP schedules are not directly used to schedule production but rather to assign priorities to jobs [Panwalkar 77, Vollmann 88]. These priorities in turn determine the order in which jobs are actually processed at each work center.

Shortcomings of the traditional MRP approach reflect limitations of computing technologies available in the 1970s. In the 1980s with the advent of more powerful computers, several more sophisticated techniques emerged [Goldratt 80, Fox 83, Ow 85, Adams 88, Ow 88a, Morton 88]. The first and by far most publicized of these techniques is the one developed by Goldratt and his colleagues within the context of the OPT factory scheduling system [Goldratt 80, Jacobs 84, Fox 87]. OPT demonstrated the benefits of building detailed production schedules that account for the actual load of the plant and the finite capacity of its resources ("finite scheduling" approaches). This system also underscored the potential benefits of distinguishing between bottleneck and non-bottleneck resources [Jacobs 84, Fox 87]. In OPT, bottlenecks are scheduled first to optimize the throughput of the plant. Later, the production schedule is completed by compactly scheduling non-bottleneck operations to reduce inventory. The distinction between bottleneck and non-bottleneck machines was pushed one step further in the OPIS system [Smith 86, Ow 88a], as it was recognized that new bottlenecks can appear during the construction of the schedule. The OPIS scheduler combines two scheduling perspectives: a resource-centered

[1]For instance, if an end-product required by the end of week 2 is obtained by assembling two sub-components and the assembly process typically takes a week to be completed, both sub-components will be required by the end of week 1.

[2]MRP systems are generally run on a weekly, possibly even a monthly basis.

perspective for scheduling bottleneck resources, and a job-centered perspective to schedule non-bottleneck operations on a job-by-job basis. Rather than relying on its initial bottleneck analysis, OPIS typically repeats this analysis each time a resource or a job has been scheduled. This ability to detect the emergence of new bottlenecks during the construction of the schedule and revise the current scheduling strategy has been termed *opportunistic scheduling* [Ow 88a]. Nevertheless, the opportunism in this approach remains limited in the sense that it typically requires scheduling an entire bottleneck (or at least a large chunk of it) before being able to switch to another one. For this reason, we actually refer to these techniques as *macro-opportunistic*.

In fact, variations in the job mix over time often cause different machines (or groups of machines) to be bottlenecks over different time intervals. Bottlenecks are sometimes said to "wander over time". Also, as a schedule is constructed for a bottleneck machine, a new machine can become more constraining than the original bottleneck. For instance, scheduling decisions on a bottleneck machine might require that a large number of jobs be processed on a preceding machine over a short period of time. At some point during the construction of the schedule, contention for the preceding machine might become higher than that for the original bottleneck. A scheduling technique that can only schedule large resource/job subproblems will not be able to take such considerations into account. It will overconstrain its set of alternatives before having worked on the subproblems that will most critically affect the quality of the entire schedule. This, in turn, will often result in poorer solutions. A more flexible approach would stop scheduling operations on a resource as soon as another resource is identified as more constraining. In the presence of multiple bottlenecks, such a technique would be able to shift attention from one bottleneck to another during the construction of the schedule rather than focus on the optimization of a single bottleneck at the expense of others. This chapter presents such a flexible approach to scheduling. We call it *micro-opportunistic* scheduling. In this approach, resource contention is continuously monitored during the construction of the schedule, and the problem solving effort is constantly redirected toward the most serious bottleneck resource. In its simplest form, this micro-opportunistic approach results in an *operation-centered* view of scheduling, in which each operation is considered an independent decision point and can be scheduled without requiring that other operations using the same resource or belonging to the same job be scheduled at the same time[3].

Experimental results presented at the end of this chapter indicate that micro-opportunistic scheduling procedures often yield better schedules than less flexible bottleneck-centered approaches. Because of their flexibility, micro-opportunistic scheduling heuristics also seem

[3]An alternative approach in which resources can be resequenced to adjust for resource schedules built further down the road is described in [Adams 88] and [Dauzere-Peres 90]. This approach has been very successful at minimizing *makespan*, namely, the total duration of the schedule. This measure is closely related to the throughput of the plant but does not account for individual job due dates, tardiness costs or inventory costs. Attempts to generalize the procedure to account for due dates seem to have been less successful so far [Serafini 88]. It should be pointed out that the idea of continuously reoptimizing the current partial schedule is compatible with a micro-opportunistic approach.

particularly well suited to solving problems in which some operations have to be performed within non-relaxable time windows [Sadeh 91a, Sadeh 92] as well as repairing schedules in the face of contingencies. Finally, we find that they can easily be integrated in interactive systems in which manual and automatic scheduling decisions can be interleaved, thereby allowing the user to incrementally manipulate and compare alternative schedules (e.g., "What-if" type of analysis).

1.3 Paper Outline

The remainder of this chapter successively reviews the predictive, reactive, and interactive capabilities of the Micro-Boss scheduling system.

Section 2 describes the micro-opportunistic search procedure implemented in Micro-Boss, focusing on look-ahead techniques used to measure contention, and heuristics to identify and schedule critical operations. A small example illustrating the use of these techniques is provided in Section 3. Section 4 describes the reactive and interactive components of the system. Section 5 reports the results of an experimental study comparing Micro-Boss with several popular scheduling approaches, including coarser opportunistic schedulers, under a wide range of simulated situations. Finally, Section 6 briefly reviews current research efforts and summarizes the impact of this work.

2 A Micro-opportunistic Search Procedure

In this section, a deterministic scheduling model is assumed, in which all jobs to be scheduled are known in advance. Issues pertaining to reactive scheduling and control in the face of manufacturing contingencies such as machine breakdowns are addressed in a later section.

2.1 A Deterministic Scheduling Model

For the time being, we consider a deterministic scheduling problem in which a set of jobs $J=\{j_1,...,j_n\}$ has to be scheduled on a set of physical resources $RES=\{R_1,...,R_m\}$. Each job j_l consists of a set of operations $O^l=\{O^l_1,...,O^l_{n_l}\}$ to be scheduled according to a process routing that specifies a partial ordering among these operations (e.g., O^l_i BEFORE O^l_j). We further assume scheduling problems with in-tree process routings, namely process routings in which operations can have one or several direct predecessors but at most one direct successor (e.g., assembly process routings). This is by far the most common type of process routing encountered in manufacturing.

Additionally, each job j_l has an earliest acceptable release date, erd_l, a due-date, dd_l, and a latest acceptable completion date, lcd_l, where $lcd_l \geq dd_l \geq erd_l$. All jobs need to be scheduled between their earliest acceptable release date and latest acceptable completion date[4]. The earliest acceptable release date might correspond to the earliest possible arrival date of raw materials. It is assumed that the actual release date (or job start date) will be determined by the schedule that is constructed. The latest acceptable completion date might correspond to a date after which the customer will refuse delivery. If such a date does not actually exist, it can always be chosen far enough in the future so that it is no longer a constraint.

Each operation O^l_i has an expected duration, du^l_i, and a start time, st^l_i (to be determined), whose domain of possible values is delimited by an earliest start time, est^l_i, and a latest start time, lst^l_i (initially derived from the job's earliest acceptable release date erd_l and latest acceptable completion date lcd_l). We assume that each operation O^l_i requires a single resource R^l_i for which there might be several alternatives in *RES*. The model further allows for resource availability constraints that specify the times when each resource is normally available (e.g., what the number of shifts is and whether the resource is available over the week-end). Finally, setup operations might be required before an operation can start on a machine. Examples of setup operations include changing the fixtures holding a part, loading a new part, cleaning a painting station when switching from one color to another, etc.

The objective of the scheduling system, under deterministic assumptions, is to build a schedule that satisfies the above constraints and minimizes (as much as possible) the costs incurred for missing due dates or carrying overhead inventories. These costs are briefly described below.

[4]Notice that this formulation does not exclude infeasible problems.

COSTS

Each job j_l has

• A **marginal tardiness cost**, *$tard_l$*: This is the cost incurred for each unit of time that the job is tardy (i.e., finishes past its due date). Marginal tardiness costs generally include tardiness penalties, interest on delayed profits, loss of customer goodwill, etc[5]. The tardiness cost of job j_l in a given schedule is

$$TARD_l = tard_l \times Max(0, C_l - dd_l) \tag{1}$$

where $C_l = st^l_{n_l} + du^l_{n_l}$ is the completion date of job j_l in that schedule, assuming that $O^l_{n_l}$ is the last operation in job j_l.

• **Marginal in-process and finished-goods inventory costs**: In our model, each operation O^l_i can incrementally introduce its own non-negative marginal inventory cost, inv^l_i. Typically, the first operation in a job introduces marginal inventory costs that correspond to interest on the costs of raw materials, interest on processing costs (for that first operation), and marginal holding costs. Downstream operations[6] introduce additional marginal inventory costs such as interest on processing costs or interest on the costs of additional raw materials required by these operations. The total inventory cost for a job j_l, in a given schedule, is:

$$INV_l = \sum_{i=1}^{n_l} inv^l_i \times [Max(C_l, dd_l) - st^l_i] \tag{2}$$

This cost accounts for both work-in-process and finished-goods inventory costs[7]

The total cost of a schedule is obtained by summing the cost of each job schedule:

$$\boxed{Schedule\ Cost = \sum_{l=1}^{n} (TARD_l + INV_l)} \tag{3}$$

A SMALL EXAMPLE

Figure 1 depicts a small scheduling problem with four jobs that will be used in this section to

[5]In this model, inventory costs incurred after the due date are not included in the tardiness costs but, rather, in the inventory costs described below.

[6]An operation O^k_i is said to be downstream (upstream) of another operation O^k_j within its job if O^k_i is a direct or indirect successor (predecessor) of O^k_j in that job, as defined by the job's process routing.

[7]Note that, in this deterministic model, minimizing work-in-process inventory costs is equivalent to minimizing job leadtimes or flowtimes.

illustrate the behavior of the micro-opportunistic scheduling heuristics implemented in Micro-Boss. Each square box represents an operation and is labeled by the name of this operation (e.g., O_1^1), its (expected) duration (e.g., $du_1^1 = 2$), and the resource it requires (e.g., $R_1^1 = R_1$). In this simple example, each operation is assumed to require a single resource, for which there are no substitutes. The arrows represent precedence constraints. For instance, job j_1 has 5 operations O_1^1, O_2^1,..., O_5^1. O_1^1 has to be performed before O_2^1, O_2^1 before O_4^1, etc. The other arcs in the graph represent capacity constraints that require that each resource be allocated to only one operation at a time. There is a capacity constraint between each pair of operations that require the same resource. Notice that R_2 is the only resource required by four operations (one from each job). Notice also that in three out of four jobs (namely, j_1 , j_3 , and j_4), the operation requiring R_2 is one of the job's longest operations. Consequently, resource R_2 can be expected to be the main bottleneck of the problem. We will see that, to some extent, resource R_1 constitutes a secondary bottleneck.

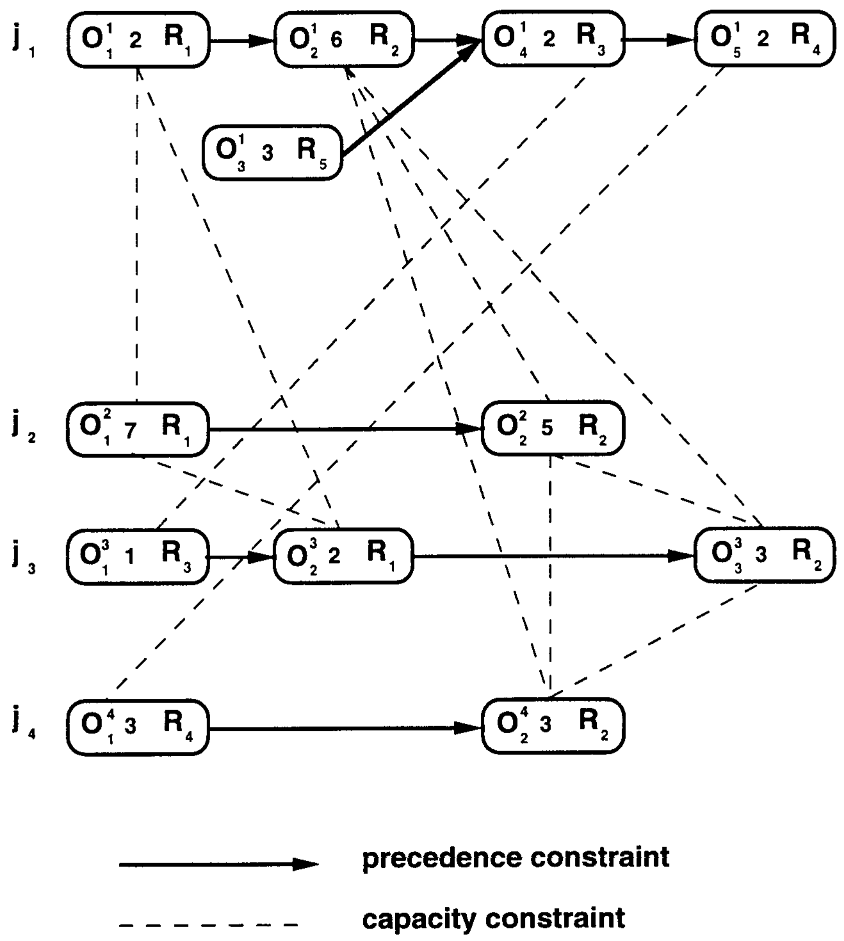


Figure 1: A simple job shop problem with four jobs. Each node is labeled by the operation that it represents, its duration, and the resource that it requires.

The earliest acceptable release dates, due dates, and latest acceptable completion dates of the jobs are provided in Table 1 along with the marginal tardiness and inventory costs of these jobs.

Earliest acceptable release dates, due dates, latest acceptable completion dates, and costs									
Job j_l	erd_l	dd_l	lcd_l	$tard_l$	inv_1^l	inv_2^l	inv_3^l	inv_4^l	inv_5^l
j_1	0	12	20	20	2	1	2	0	0
j_2	0	14	20	20	5	0	-	-	-
j_3	0	9	20	5	1	0	0	-	-
j_4	0	18	20	10	1	0	-	-	-

Table 1: Earliest acceptable release dates, due dates, latest acceptable completion dates and marginal costs.

2.2 Overview of the Search Procedure

In Micro-Boss, each operation is considered an independent decision point. Any operation can be scheduled at any time, if deemed appropriate by the system. There is no obligation to simultaneously schedule other operations upstream or downstream within the same job, nor is there any obligation to schedule other operations competing for the same resource.

Micro-Boss proceeds by iteratively selecting an operation to be scheduled and a reservation (i.e., a resource/time interval) to be assigned to that operation. Every time an operation is scheduled, a new *search state* is created, where new constraints are added to account for the reservation assigned to that operation. A consistency enforcing procedure then takes care of updating the set of remaining possible reservations of each unscheduled operation. If an unscheduled operation is found to have no possible reservations left, a *deadend state* has been reached, in which case the system needs to *backtrack* (i.e., it needs to undo some earlier reservation assignments to be able to complete the schedule). If the search state does not appear to be a deadend, the system moves on and looks for a new operation to schedule and a reservation to assign to that operation.

To enhance search efficiency[8] and produce high quality schedules, Micro-Boss interleaves search with the application of consistency enforcing mechanisms and a set of look-ahead techniques that help decide which operation to schedule next (*operation ordering heuristic*) and which reservation to assign to that operation (*reservation ordering heuristic*).

1. **Consistency Enforcing/Checking:** Consistency enforcing techniques prune the

[8]We define search efficiency as the ratio of the number of operations to be scheduled over the number of search states generated. If the number of search states generated to build the schedule is equal to the number of operations, search efficiency is equal to 1.

search space by inferring new constraints resulting from earlier reservation assignments [Mackworth 85, Sadeh 91b]. By constantly accounting for earlier scheduling decisions, these techniques reduce the chances of reaching a deadend (i.e., a partial schedule that cannot be completed without backtracking). Simultaneously, by allowing for the early detection of deadend states, these techniques limit the amount of work wasted in the exploration of fruitless alternatives.

2. **Look-Ahead Analysis**: A two-step look-ahead procedure is applied in each search state, which first optimizes reservation assignments within each job and then, for each resource, computes contention between jobs over time. Resource/time intervals where contention is the highest help identify the critical operation to be scheduled next (*operation ordering heuristic*). Reservations for that operation are then ranked according to their ability to minimize the costs incurred by the jobs contending for the critical resource (*reservation ordering heuristic*). By constantly redirecting its effort toward the most serious conflicts, the system is able to build schedules that are closer to the global optimum. Simultaneously, because the scheduling strategy is aimed at reducing job contention as rapidly as possible, chances of reaching deadend states tend to quickly subside too.

The opportunism in Micro-Boss results from the ability of the system to constantly *revise its search strategy and redirect its effort toward the scheduling of the operation that appears to be the most critical in the current search state.* This degree of opportunism differs from the one displayed by earlier approaches where scheduling entities were large resource/job subproblems [Ow 88a, Collinot 88], i.e., where large resource/job subproblems had to be scheduled before the system could revise its scheduling strategy.

Concretely, given a scheduling problem such as the one described in Figure 1, Micro-Boss starts in a search state in which no operation has been scheduled yet[9], and proceeds according to the following steps:

1. If all operations have been scheduled, then stop; else go on to 2.

2. Apply the **consistency enforcing** procedure.

[9]Alternatively, Micro-Boss can also complete a partial schedule, in which case the initial search state corresponds to the initial partial schedule. A description of reactive and interactive capabilities of the system is provided in Section 4.

3. If a deadend is detected then **backtrack**; else go on to 4.

4. If one or more operations were found to have only one possible reservation left, then schedule these operations (creating a new search state for each one). If all operations have been scheduled, then stop; else go on to 5.

5. Perform a **look-ahead** analysis: Rank the possible reservations of each unscheduled operation according to how well they minimize the costs of the job to which the operation belongs (step 1), and evaluate resource contention over time (step 2).

6. Select the next operation to be scheduled (i.e., **operation ordering** heuristic).

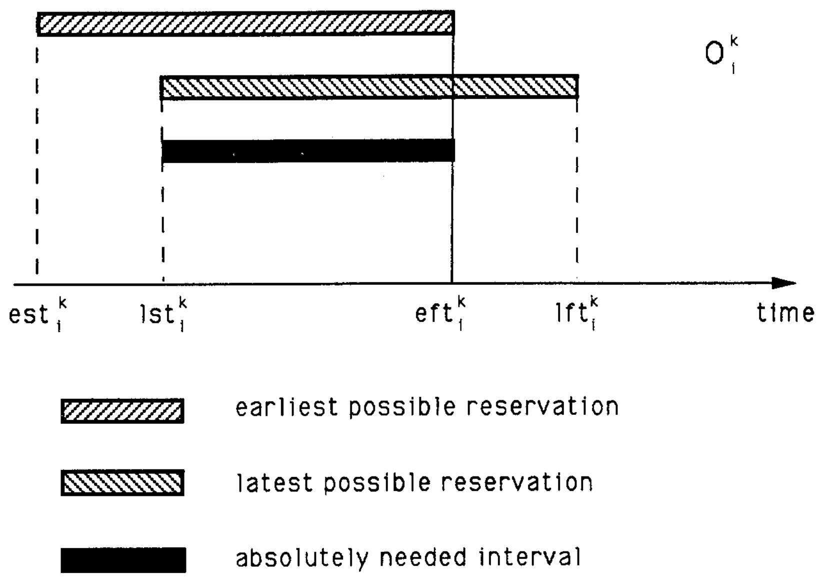
7. Select a reservation for that operation (i.e., **reservation ordering** heuristic).

8. Create a **new search state** by adding the new reservation assignment to the current partial schedule. Go back to 1.

As in other constraint-directed scheduling systems [LePape 87], the consistency enforcing procedure used in Micro-Boss (1) maintains for each unscheduled operation a pair of earliest/latest possible start times and (2) marks as unavailable those resource/time intervals allocated to already scheduled operations. Additionally, reservation pruning performed by the Micro-Boss consistency procedure also accounts for resource/time intervals that are absolutely needed by unscheduled operations. Figure 2 displays an example of an unscheduled operation O_i^k whose earliest and latest possible reservations overlap. Whichever reservation this operation is ultimately assigned, it will always need time interval $[lst_i^k, eft_i^k]$. Accordingly, the Micro-Boss consistency procedure prunes the set of remaining possible reservations of other unscheduled operations requiring that resource by removing all those reservations that overlap with time interval $[lst_i^k, eft_i^k]$[10].

Results presented in this chapter were obtained using a simple chronological backtracking scheme. Experimentation with more sophisticated backtracking schemes is described in [Sadeh

[10]This differs from an earlier version of the system [Sadeh 91b], in which resource/time intervals needed by unscheduled operations were only used to detect conflicts. In this earlier version, a conflict would be detected when two or more unscheduled operations needed overlapping resource/time intervals. Rather than waiting for such conflicts to arise, our new consistency procedure efficiently prevents such conflicts from occurring, thereby further reducing backtracking. A generalized version of this procedure is used for parallel machines.



O_i^k
est_i^k
lst_i^k
eft_i^k
lft_i^k
time
earliest possible reservation
latest possible reservation
absolutely needed interval

Figure 2: An example of an unscheduled operation that absolutely needs a resource/time interval.

92].

The remainder of this section gives a more detailed description of the look-ahead analysis and the operation/reservation ordering heuristics used in Micro-Boss. Further details on these techniques, as well as other aspects of the system, can be found in [Sadeh 91b].

2.3 Look-Ahead Analysis in Micro-Boss

2.3.1 Optimizing Critical Conflicts First

If all jobs could be scheduled optimally (i.e., just-in-time), there would be no scheduling problem. Generally, this is not the case. Jobs typically have conflicting resource requirements. The look-ahead analysis carried out by Micro-Boss in each search state aims at helping the scheduling system focus its effort on those conflicts that currently appear most critical. A critical conflict is one that will require an important trade-off, i.e., a trade-off that will significantly impact the quality of the *entire* schedule. By first focusing on critical conflicts, Micro-Boss ensures that it has as many options as possible to optimize these conflicts. As illustrated by a trace provided in the next section, once critical trade-offs have been worked out, the remaining unscheduled operations tend to become more decoupled and, hence, easier to optimize[11]. As contention subsides, so does the chance of needing to backtrack. In other words, by constantly redirecting search towards those trade-offs that appear most critical, Micro-Boss is

[11]This is similar to the way bottleneck schedules drive other scheduling decisions in OPT.

expected to produce better schedules and simultaneously keep backtracking at a low level.

More specifically, a two-step look-ahead procedure is applied to each search state. This procedure first optimizes reservation assignments within each job and then, for each resource, computes contention between jobs over time. The so-called *demand profiles* produced by these computations help identify operations whose good reservations (as identified in the first step) conflict with the good reservations of other operations. These operations define the critical conflicts on which Micro-Boss works first.

This two-step look-ahead analysis is further detailed below.

2.3.2 Step 1: Reservation Optimization within a Job

In order to measure contention between the resource requirements of unscheduled operations, Micro-Boss keeps track of the best start times that remain available to each unscheduled operation within its job. Additionally, the system implicitly maintains, for each remaining possible start time τ of each unscheduled operation O_i^k, a function $mincost_i^k(\tau)$ that indicates the minimum additional costs that would be incurred by job j_k (the job to which O_i^k belongs), if O_i^k were to start at $st_i^k = \tau$ rather than at one of its best possible start times. By definition, if $st_i^k = \tau$ is one of the best start times that remain available to O_i^k within its job, then $mincost_i^k(\tau) = 0$. Rather than explicitly maintaining *mincost* functions, Micro-Boss simply maintains for each unscheduled operation O_i^k (1) an *apparent marginal tardiness cost*, $app\text{–}tard_i^k$, that approximates the cost incurred by job j_k for each unit of time that O_i^k starts past its latest best start time and (2) an *apparent marginal inventory cost*, $app\text{–}inv_i^k$, that approximates the cost incurred by job j_k for each unit of time that O_i^k starts before its earliest best start time. These costs are updated in each search state to account for earlier scheduling decisions, using a set of efficient propagation procedures described in [Sadeh 91b].

2.3.3 Step 2: Building Demand Profiles to Identify Critical Resource/Time Intervals

In Micro-Boss, critical conflicts are identified as groups of operations whose good reservations (within their jobs) conflict with each other. The importance of a conflict depends on the number of operations that are competing for the same resource, the amount of temporal overlap between the requirements of these operations, the number of alternative reservations still available to each of these conflicting operations and their costs, as determined by the *mincost* functions computed in step 1.

To identify critical conflicts, Micro-Boss uses a probabilistic framework in which each remaining possible start time τ of an unscheduled operation O_i^l is assigned a *subjective probability* $\sigma_i^l(\tau)$ to be selected for that operation in the final schedule. Possible start times with lower *mincost* values are assigned a larger probability, thereby reflecting our expectation that they will yield better schedules. Given these start time probability distributions, the probability that an unscheduled operation O_i^l uses its resource[12] at time t, which is referred to as the *individual demand* of O_i^l for R_i^l, is:

$$D_i^l(t) = \sum_{t - du_i^l < \tau \leq t} \sigma_i^l(\tau) \tag{4}$$

where du_i^l is the duration of O_i^l. $D_i^l(t)$ is also a (subjective) measure of the reliance of operation O_i^l on the availability of its resource at time t. By adding the individual demands of all unscheduled operations requiring a given resource, say R_k, the system obtains an *aggregate demand profile*, $D_{R_k}^{aggr}(t)$, that indicates contention between (all) unscheduled operations for that resource R_k as a function of time:

$$D_{R_k}^{aggr}(t) = \sum D_i^l(t) \tag{5}$$

where the summation is carried over all unscheduled operations that need resource R_k.

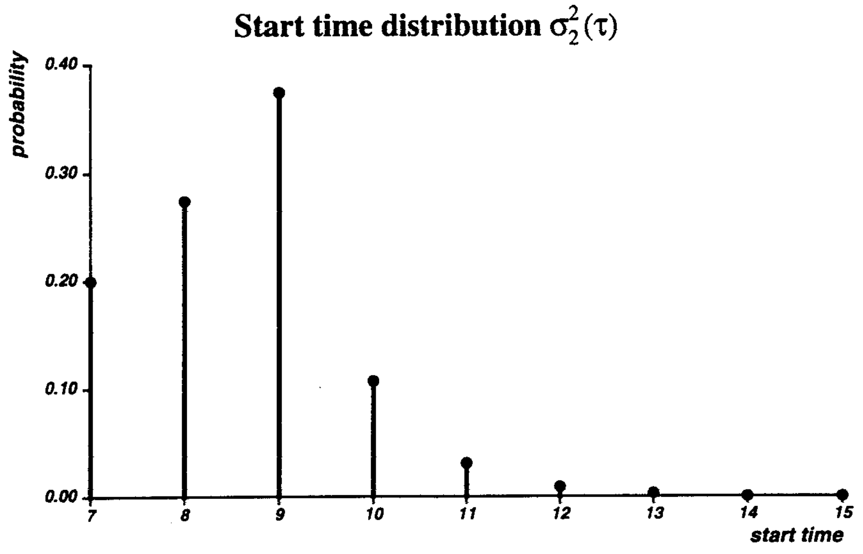


Figure 3: Start time distribution $\sigma_2^2(\tau)$ for operation O_2^2 in the initial search state for the problem defined in Figure 1.

[12]For the sake of simplicity, we assume here that each operation requires a single resource for which there are no alternatives. The construction of demand profiles can easily be generalized to deal with parallel machines by building profiles for entire groups of machines and normalizing them based on their remaining available capacities over time.

Figure 3 displays $\sigma_2^2(\tau)$, the start time distribution of operation O_2^2 in the problem defined in Figure 1. This start time distribution is depicted in the initial search state, where all operations still have to be scheduled. In this search state, start time $st_2^2=9$ is the best possible start time for O_2^2: it corresponds to a just-in-time schedule of job j_2. Later start times have a lower subjective probability because they would force the job to finish after its due date. Earlier start times are also suboptimal because they would produce additional inventory. In this example, the marginal tardiness cost of job j_2, $tard_2=20$, is four times larger than the marginal inventory cost introduced by operation O_1^2, $inv_1^2=5$. Accordingly, $\sigma_2^2(\tau)$ decreases faster for $\tau > 9$ than for $\tau < 9$.

Figure 4 displays the individual demand profiles of the four operations requiring resource R_2. These demand profiles represent the subjective probability that each one of these operations uses resource R_2 as a function of time. The aggregate demand for resource R_2 is obtained by summing these four individual demands over time. The individual demands of operations O_3^3 and O_2^4 are quite uniform because these two operations have relatively low apparent marginal costs (see the marginal tardiness and inventory costs of job j_3 and job j_4 in Table 1). In contrast, operations O_2^1 and O_2^2, which have larger apparent marginal costs, have individual demands that are concentrated around their best reservations.

Similar computations can be performed for each of the five resources in the problem. The resulting aggregate demands (in the initial search state) are displayed in Figure 5. As expected, resource R_2 appears to be the most contended for. The aggregate demand for that resource is well above 1.0 over a large time interval, with a peak at 1.79. Resource R_1 appears to be a potential bottleneck at the beginning of the problem, with a demand peaking at 1.52. Whether R_1 will actually be an auxiliary bottleneck or not cannot be determined directly from the curves displayed in Figure 5. Instead, the system needs to update these curves in each search state to account for earlier decisions. It could be the case that as operations requiring R_2 are scheduled, the aggregate demand for R_1 becomes smoother. In this example, this is not the case. On the contrary, as operations are scheduled on resource R_2, some operations on resource R_1 end up with only one possible reservation and need to be immediately scheduled, as indicated by the trace provided in Section 4.

2.4 Operation Selection

Critical operations are identified as operations whose good reservations (as identified in the first step of the look-ahead analysis) conflict with the good reservations of other operations. The largest peak in the aggregate demand profiles determines the next conflict (or micro-bottleneck) to be optimized; the operation with the largest reliance on the availability of the corresponding resource/time interval (i.e., the operation with the largest individual contribution to the peak) is selected to be scheduled next. Indeed, this operation is the one whose good reservations are the most likely to become unavailable if other operations contending for the current micro-bottleneck were scheduled first.

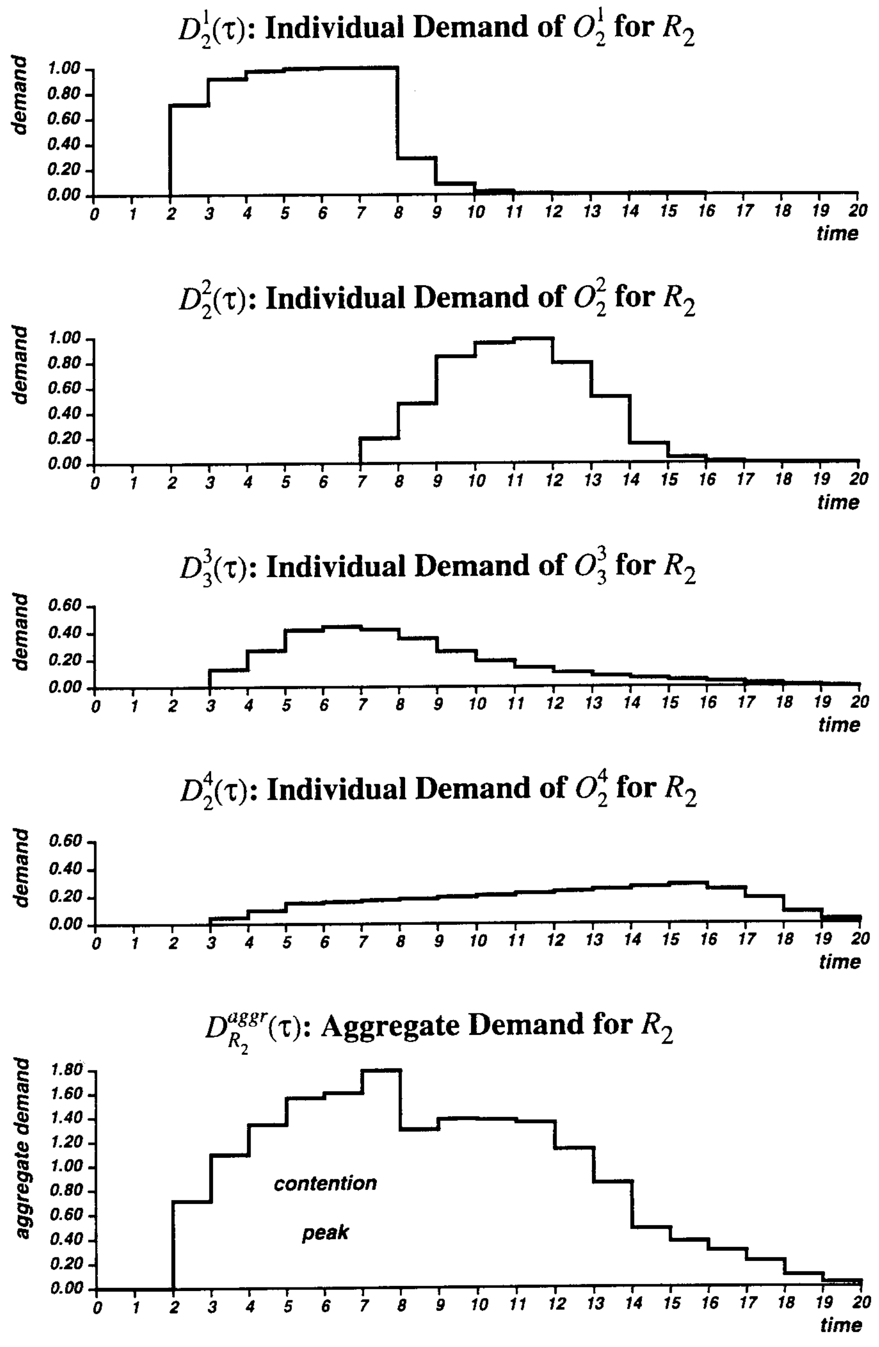


Figure 4: Building R_2's aggregate demand profile in the initial search state.

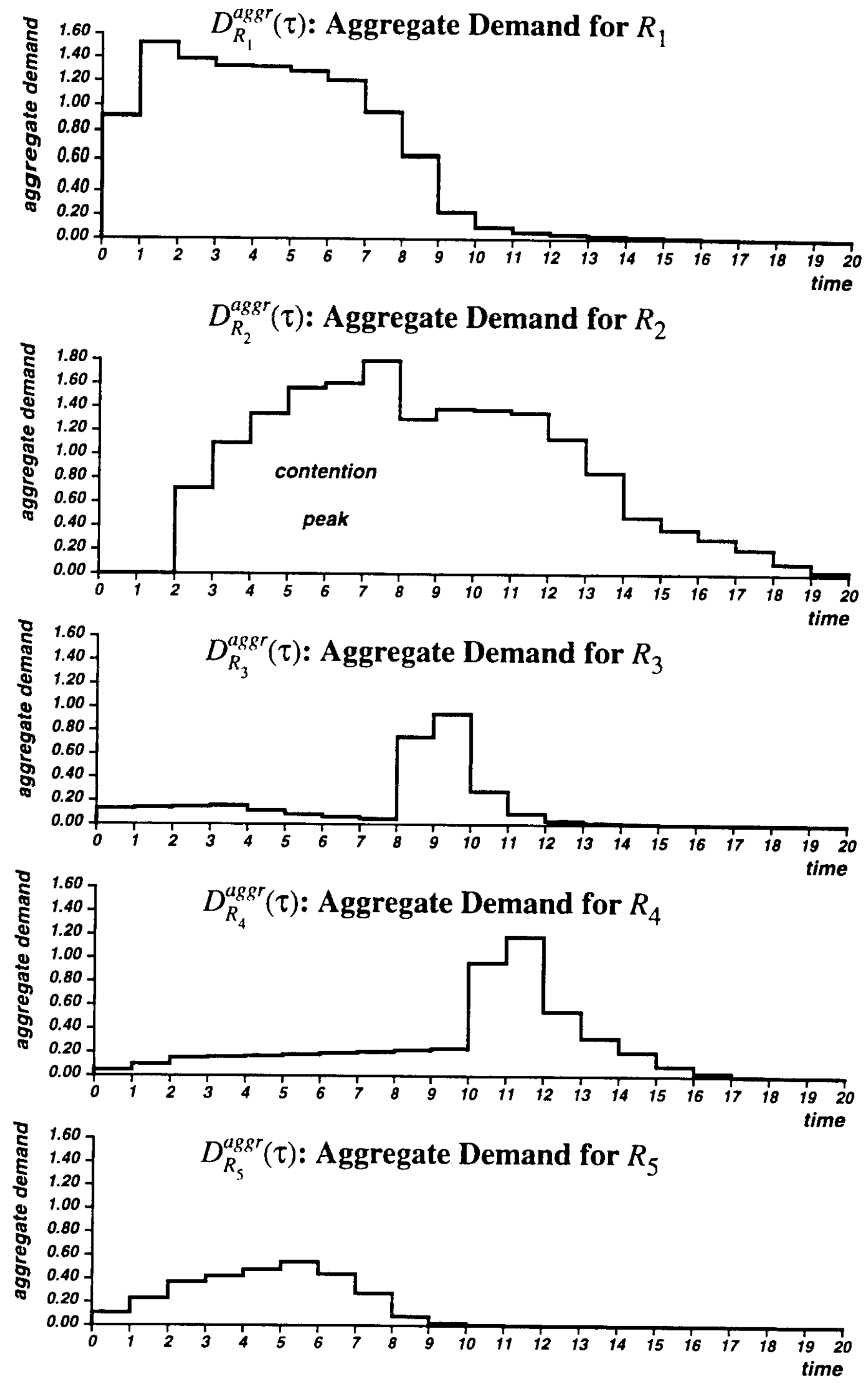


Figure 5: Aggregate demands in the initial search state for each of the five resources.

In the example introduced earlier, the largest demand peak is the one for resource R_2 over

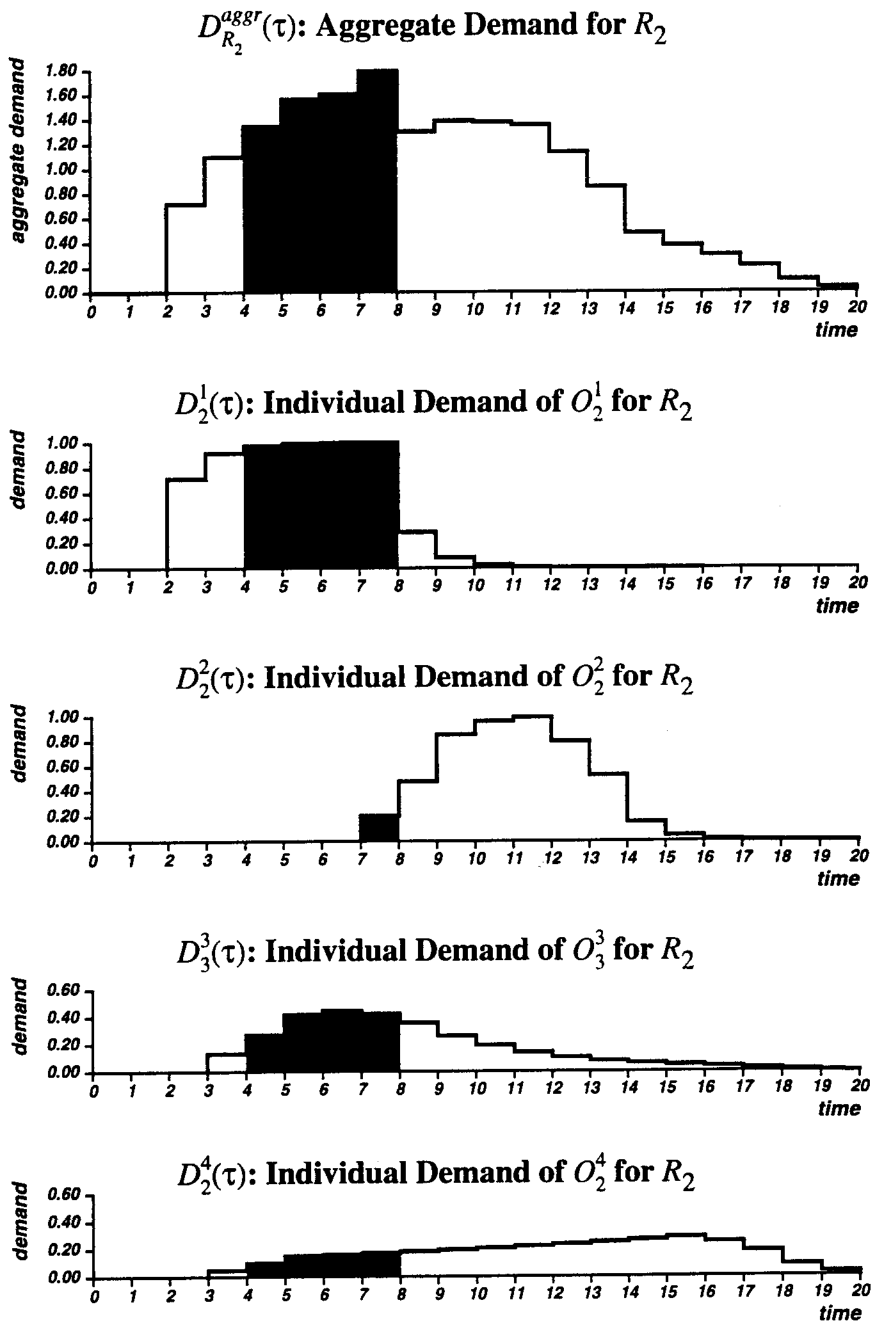


Figure 6: Operation selection in the initial search state.

interval [4,8[. Figure 6 displays the aggregate demand for resource R_2 together with the individual demands of the four operations requiring this resource. The operation with the largest contribution to the demand peak is O_2^1. Therefore this operation is selected to be scheduled next. This is no real surprise: O_2^1 belongs to one of the two jobs in the problem that have a high marginal tardiness cost ($tard_1 = 20$). While any delay in starting job j_1 will result in large tardiness costs, job j_3 (i.e., the job with the next highest contribution) can tolerate a small delay and is subject to lower tardiness penalties.

The computation of demand profiles, as described in 2.3.3 can be quite expensive when performed for each resource in each search state over the entire scheduling horizon. Micro-Boss can avoid this problem by incrementally maintaining a set of *rough* demand profiles for each resource (or group of identical resources). These rough demand profiles use a much coarser time granularity and are obtained by splitting the demand of each unscheduled operation into two components. One component (50% of the operation's total demand in the current implementation [13]) is evenly spread between the start and end times of the latest best reservation of the operation while the second component (the remaining 50% of the operation's demand) is evenly spread between the earliest start time and latest finish time of the operation. Rough demand profiles can be quickly updated as the system moves from one search state to the next and are used in each search state to identify a small number of critical resource/time intervals over which the more detailed demand profiles described in 2.3.3 are then constructed.

2.5 Reservation Selection

To schedule the critical operation identified in 2.4, the system attempts to identify a reservation (for the critical operation) that will reduce as much as possible the costs incurred by the job to which that operation belongs and the other jobs with which that operation competes. This is approximated as a single-machine or parallel-machine early/tardy scheduling problem in which operations scheduled past their best start times incur penalties determined by their apparent marginal tardiness costs, while operations scheduled before their best start times incur earliness penalties, as determined by their apparent marginal inventory costs [Baker 90, Sadeh 91b]. For problems without setups, several variations of a single-machine early/tardy procedure developed by Ow and Morton [Ow 89, Sadeh 91b] are successively run and the single-machine schedule with the lowest cost is used to determine the reservation assigned to the critical operation. More recently, a new scheduling heuristic has also been developed to solve problems with setups [Li&Sadeh 93]. This heuristic is further described below along with experimental results comparing it with Ow and Morton's heuristic.

Briefly, our heuristic for problems with setups opportunistically selects between two simpler techniques:

[13]The total demand of an operation is equal to its duration.

- A clustering technique that identifies clusters of early(tardy) jobs and resequences them using variations of the Weighted Longest (Shortest) Processing Time dispatch rule;
- A two-parameter technique ("ET-2") that generalizes the priority dispatch rule developed by Ow&Morton for the problem without setups [Ow 89].

Our heuristic opportunistically selects between ET-2 and the clustering heuristic (based on the tightness of the problem at hand) to generate an initial schedule, which is then refined using a neighbordhood search procedure and an optimal idle time insertion procedure [Li&Sadeh 93].

To evaluate the performance of this heuristic, a set of 1920 problems was generated by adjusting the following 6 parameters:

- Two different product mixes (with 3 product families each)
- Early/tardy cost ratios: 0.05, 0.1, 0.25, 0.5, 1.0 and 5.0
- Tardiness Factor (tight vs. loose due dates)
- Due date ranges (wide vs. narrow)
- Setup severity (average value of setup time divided by processing time): low and high
- Problem sizes: 9-job problems and 50-job problems

By combining the different values of these 6 parameters, a total of 192 problem sets was generated. Ten problems were randomly generated within each set.

The smaller 9-job problems were used to see how far our heuristic is from the optimum: on these smaller problems, it is possible to find an optimal solution, using a simple branch-and-bound procedure (this is not possible on larger problems, as the procedure would take for ever). Figure 7 and 8 depict the average deviation of the solutions produced by our heuristic from the optimum. We see that on low tardy factor problems (loose due dates), our heuristic is within 11.62% of the optimum. On high tardy factor problems, the results are even better: our heuristic is consistently within 10% of the optimum and, on average,it is within 4.77% of the optimum. Figure 9 further indicates that, compared to the one-parameter developed by Ow and Morton, our heuristic reduces schedule cost by an average of 30.61%.

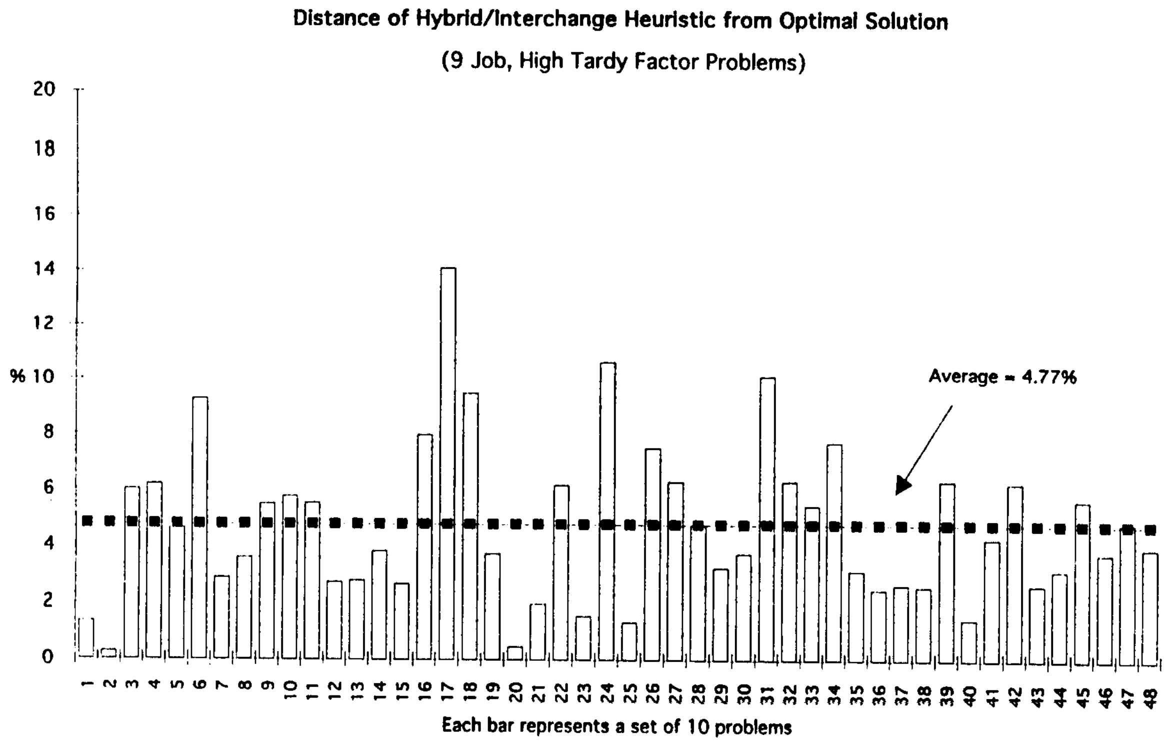


Figure 7: Distance from Optimum - High Tardy Factor

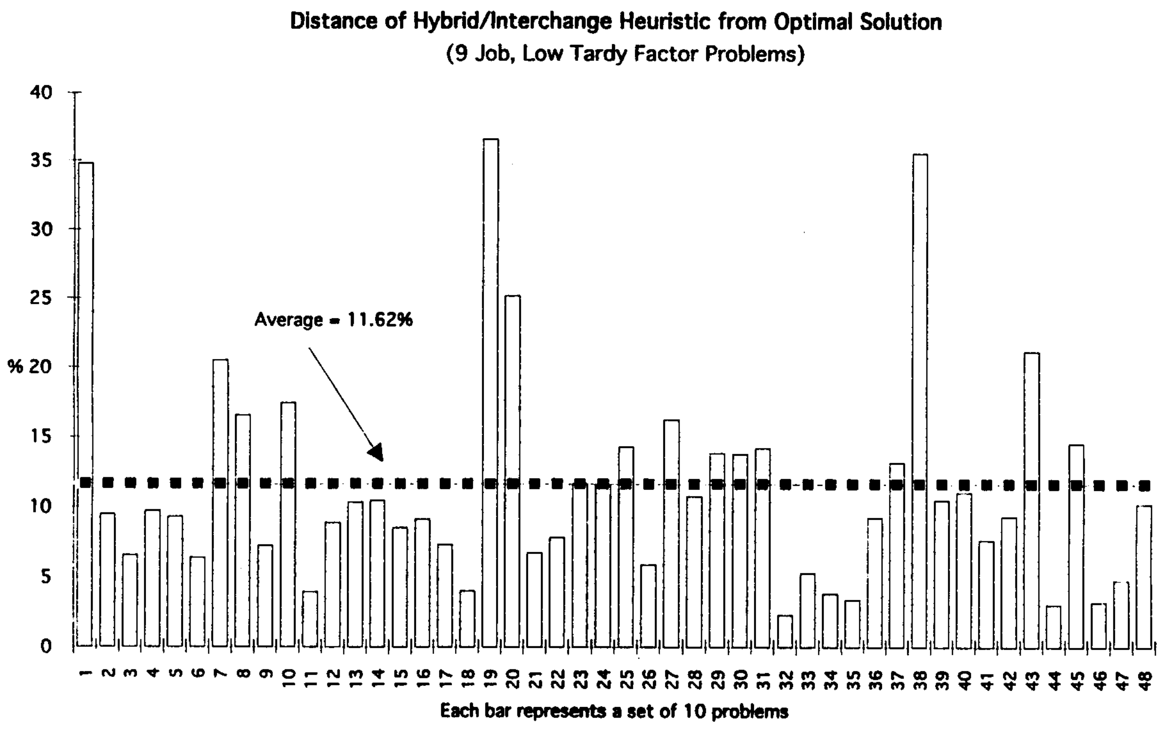


Figure 8: Distance from Optimum - Low Tardy Factor

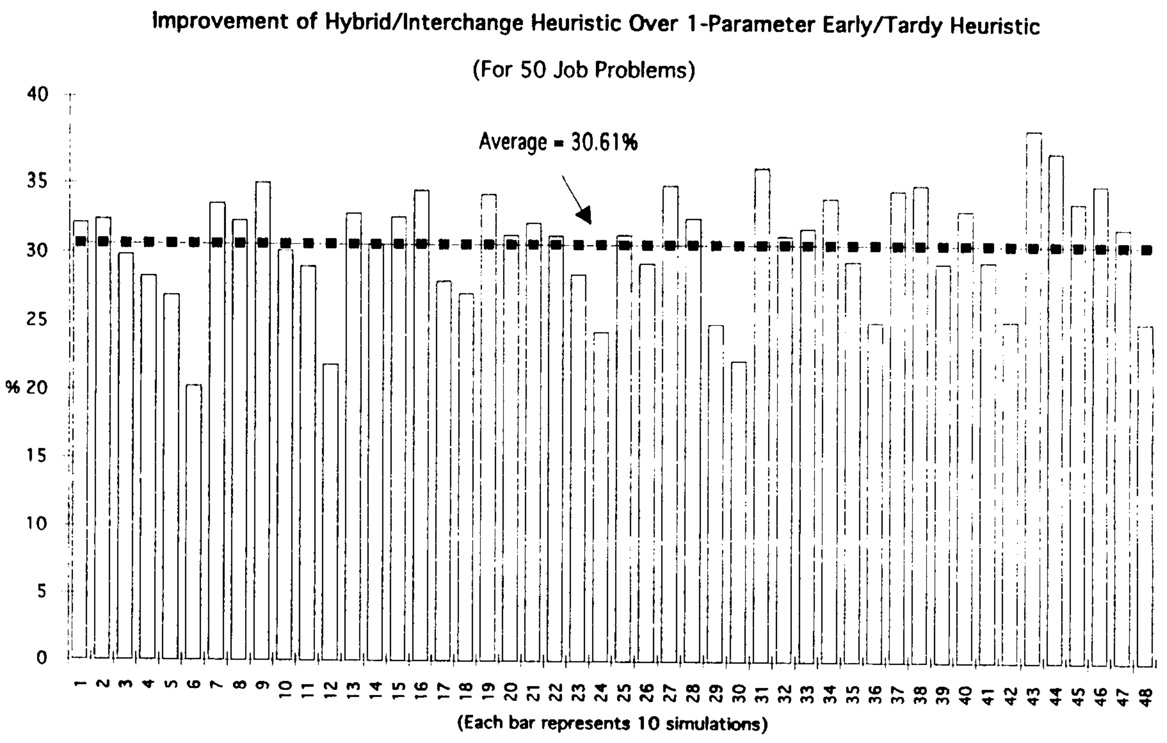


Figure 9: Improvement over one-parameter early/tardy heuristic.

3 A Small Example

Micro-Boss is implemented in C++ with an X™/Motif™ interface. The small example used throughout this chapter requires less than 0.1 CPU seconds on a DECstation™ 5000/200 running under UNIX™[14]. An edited trace of this example is given in Figure 10.

In this example, the scheduling procedure first focuses on the scheduling of the main bottleneck resource, R_2. However, as it schedules operations on this resource, the system can also jump to other resources and consolidate the schedule by allocating reservations to critical operations requiring these other resources. In this small example where operations have a small number of possible reservations, this is mainly accomplished through the identification of operations that have only one possible reservation left (e.g. the scheduling of O_1^1 or O_1^2). In

[14]X Window System is a registered trademark of the Massachusetts Institute of Technology. Motif is a registered trademark of the Open Software Foundation, Inc. UNIX is a registered trademark of UNIX Systems Laboratories, Inc. DECstation is a registered trademark of the Digital Equipment Corporation.

general, this can be done based on the contention analysis performed by Micro-Boss (e.g., the identification of a critical conflict on resource R_4 at depth 6). As a result, the system jumps back and forth between several resources, always trying to focus on what appears to be the most critical decision.

The *average expected demand* displayed in each search state is the average demand for the critical demand peak, and the *average contribution* is the percentage of the total demand for the peak that comes from the critical operation. When search starts, contention is relatively high, as illustrated by the average expected demand for the critical peak (1.58 at depth 0, 1.73 at depth 2 and 1.50 at depth 4) and the relatively low contribution of the critical operation to the demand for the peak (e.g., O_2^1 contributes only 63% of the total demand for the peak in the initial search state, O_2^2 57% at depth 2, etc.) indicating that the resource requirements of the critical operation compete with those of several other operations. During construction of the schedule, the average demand for the critical peak progressively decreases[15] and the critical operation progressively contributes a larger percentage of the demand for the critical peak. This indicates that contention between unscheduled operations decreases. After half of the operations have been scheduled (depth 7), contention has totally disappeared: the critical operation is the only one to contribute to the demand for the peak. The resource requirements of the operations that still need to be scheduled no longer compete with each other. This is not particular to this example: the same has been observed on all the problems we have run and suggests that the operation ordering heuristic implemented in Micro-Boss is indeed very effective at redirecting search towards the most serious conflicts.

Notice also that no backtracking was necessary to schedule this problem. The resulting schedule is displayed in Figure 11.

[15]Remember that the demand peak corresponds to the interval of highest contention in the current search state.

```
>> Depth: 0, Number of states visited: 0
   Critical demand peak:
   $R_2$ between 4 and 8, Avg. expected demand: 1.58
   Critical Operation: $O_2^1$, Avg. contrib.: 63%
   $O_2^1$ is scheduled between 2 and 8 on $R_2$

>> Depth: 1, Number of states visited: 1
   $O_1^1$ has only one possible reservation left
   and is scheduled between 0 and 2 on $R_1$

>> Depth: 2, Number of states visited: 2
   Critical demand peak:
   $R_2$ between 10 and 14, Avg. expected demand: 1.73
   Critical Operation: $O_2^2$, Avg. contrib.: 57%
   $O_2^2$ is scheduled between 9 and 14 on $R_2$

>> Depth: 3, Number of states visited: 3
   $O_1^2$ has only one possible reservation left
   and is scheduled between 2 and 9 on $R_1$

>> Depth: 4, Number of states visited: 4
   Critical demand peak:
   $R_2$ between 14 and 18, Avg. expected demand: 1.50
   Critical Operation: $O_2^4$, Avg. contrib.: 50%
   $O_2^4$ is scheduled between 14 and 17 on $R_2$

>> Depth: 5, Number of states visited: 5
   $O_3^3$ has only one possible reservation left
   and is scheduled between 17 and 20 on $R_2$

>> Depth: 6, Number of states visited: 6
   Critical demand peak:
   $R_4$ between 10 and 12, Avg. expected demand: 1.12
   Critical Operation: $O_5^1$, Avg. contrib.: 73%
   $O_5^1$ is scheduled between 10 and 12 on $R_4$

>> Depth: 7, Number of states visited: 7
   $O_4^1$ has only one possible reservation left
   and is scheduled between 8 and 10 on $R_3$
```

Figure 10: An edited trace

```
>> Depth: 8, Number of states visited: 8
   Critical demand peak:
   $R_5$ between 5 and 8, Avg. expected demand: 0.95
   Critical Operation: $O_3^1$, Avg. contrib.: 100%
   $O_3^1$ is scheduled between 5 and 8 on $R_5$

>> Depth: 9, Number of states visited: 9
   Critical demand peak:
   $R_4$ between 7 and 9, Avg. expected demand: 0.96
   Critical Operation: $O_1^4$, Avg. contrib.: 100%
   $O_1^4$ is scheduled between 7 and 10 on $R_4$

>> Depth: 10, Number of states visited: 10
   Critical demand peak:
   $R_1$ between 14 and 17, Avg. expected demand: 0.65
   Critical Operation: $O_2^3$, Avg. contrib.: 100%
   $O_2^3$ is scheduled between 15 and 17 on $R_1$

>> Depth: 11, Number of states visited: 11
   Critical demand peak:
   $R_3$ between 13 and 15, Avg. expected demand: 0.52
   Critical Operation: $O_1^3$, Avg. contrib.: 100%
   $O_1^3$ scheduled between 14 and 15 on $R_3$

>> Depth: 12, Number of states visited: 12
   Schedule Completed
   Total Cost: 180
   Total Tardiness Cost: 55
   Total Inventory Cost: 125
   Avg. Weighted Tardiness: 1.0
   Avg. Weighted Flowtime (WIP): 10.33
   Avg. Weighted Inventory (Flowtime + Earliness): 10.42
   CPU time: 0.067 seconds
```

Figure 10, concluded

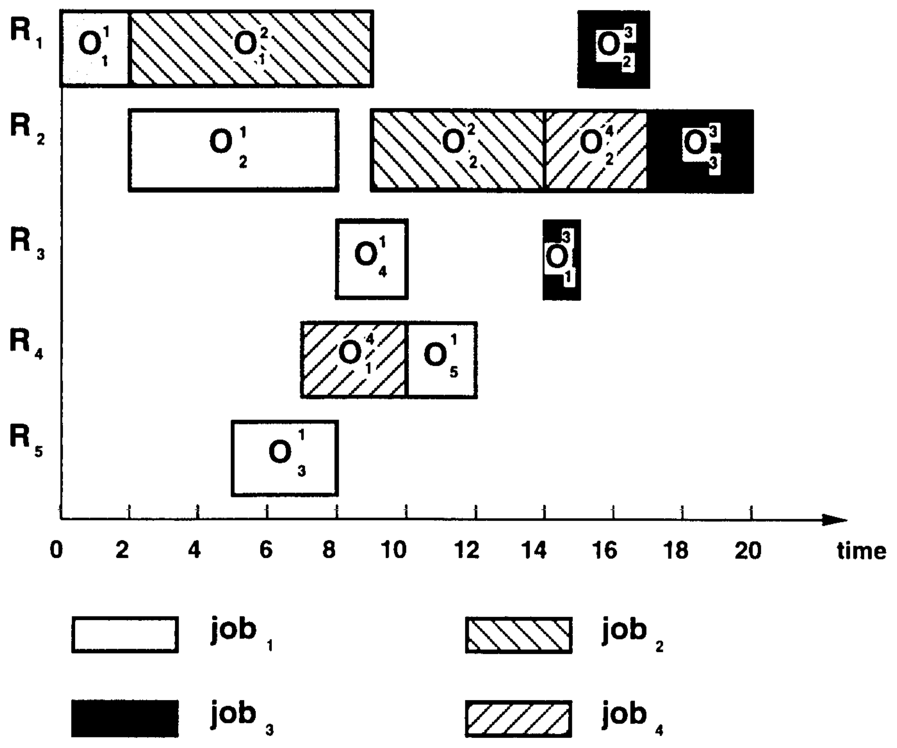


Figure 11: Gantt chart of the final schedule produced by Micro-Boss.

4 Reactive and Interactive Scheduling in Micro-Boss

Manufacturing is a process often fraught with contingencies and subject to a multitude of constraints and preferences that are not always easily amenable to representation in a computer model.

Operation durations tend to vary, machines break down, raw materials fail to arrive on time, new customer orders arrive, others get cancelled, etc. Many ad hoc constraints and preferences that vary over time, such as the preference of a worker on a specific day to perform more demanding tasks in the morning, might be best accounted for via interactive manipulation of the schedule. This section briefly outlines reactive and interactive scheduling capabilities currently under development in the Micro-Boss decision support system.

4.1 Reactive Scheduling and Control Issues

Small disruptions such as minor deviations in operation durations often do not warrant major modifications of the schedule. However, as the impact of small disruptions accumulate or as more severe disruptions occur, such as long machine breakdowns, it is sometimes desirable to reoptimize the schedule from a more global perspective. Accordingly, in Micro-Boss, schedule disruptions can be handled at two levels based on their severity and the required response time:

1. **Control level**: Small disruptions that require fast responses are handled by simple control heuristics such as *"process the operation with the earliest scheduled start time first"*, or, *"when a machine is down, reroute critical jobs to equivalent machines, if any"*.
2. **Scheduling level**: In the face of more severe deviations from the schedule, the control level calls upon the Micro-Boss scheduling module to repair/reoptimize the schedule from a more global perspective, while possibly continuing to attend to immediate decisions.

Determining when disruptions should be handed over to the scheduling level can be tricky. Decisions at the control level tend to be rather fast as they are based on local heuristics with a very restricted view of the problem. Decisions at the scheduling level tend to produce better repairs but take longer as they are based on more global considerations. There is generally a tradeoff between the responsiveness of the overall system and the amount of reoptimization that can be performed. In manufacturing environments where disruptions are very frequent, a large number of disruptions may need to be handled at the control level, whereas, in less chaotic environments, a larger proportion of disruptions may be processed at the scheduling level. A similar two-tier approach to handling schedule disruptions was first proposed by Smith et al. [Smith 90a]. Within this approach, the scheduling level restricts the set of alternatives to be considered at the control level by imposing a legal temporal window of execution on each operation. If the controller cannot respect an operation's window of execution, it has to request a new schedule (and a new set of execution windows) from the scheduler. One objective of ongoing research in reactive scheduling and control within Micro-Boss aims at assessing the merits of different coordination regimes between the scheduling and control levels.

Schedule repair in Micro-Boss differs from recent approaches that emphasized the use of iterative repair heuristics [Smith 90b, Minton 90, Zweben 91]. In the process of resolving schedule conflicts, iterative repair heuristics are allowed to introduce new conflicts, which in turn need to be repaired. This iterative behavior may sometimes lead to myopic decisions and can potentially become expensive. In contrast to these approaches, schedule repair in Micro-Boss attempts to take a more global view of the problem and capitalize on the strengths of the micro-opportunistic search procedures in the system. Concretely, schedule repair in Micro-Boss is performed in two steps: (1) a set of operations that need to be rescheduled is identified using a so-called conflict propagation procedure and all the operations in this set are unscheduled, (2) the scheduling problem consisting of all these unscheduled operations and the constraints imposed on these operations by operations that have already been executed or have not been unscheduled is passed to the micro-opportunistic scheduling module described in the previous sections. The set of operations unscheduled in the first phase is selected in such a way that the resulting scheduling problem (i.e. the one solved in phase (2)) generally admits a solution. In the event that a feasible schedule cannot be built in phase (2), the system needs to return to phase (1) and undo a larger number of operations. In practice, this situation can generally be avoided, as explaine in the next subsection. For particularly severe schedule disruptions such as the breakdown of a bottleneck machine over a long time period, we are also considering rescheduling techniques that subdivide the scheduling horizon and only reschedule those operations that are expected to fall within the near future while overlooking conflicts with operations whose execution is expected to take place later.

4.2 An Initial Set of Conflict Propagation Techniques

In this subsection, we summarize initial experiments with different conflict propagation heuristics used to determine which operations to reschedule in order to recover from constraint violations introduced in the schedule by contingencies such as machine breakdowns or variations in operation processing times.

Ideally, a good conflict propagation procedure should (1) minimize the number of operations to be rescheduled (to maximize system responsiveness and minimize schedule disruptions) and (2) identify a set of operations which, when rescheduled, will (a) be sufficient to *restore schedule integrity* (i.e. eliminate all constraint violations) and (b) maximize schedule quality. Clearly these objectives are not always compatible. Procedures that reschedule more operations are likely to yield better schedules but may also lead to less responsive and more disruptive behaviors.

Figure 12 displays a simple reactive heuristic, which, given a conflict, simply bumps operations forward in time until schedule integrity is restored. This reactive procedure is sometimes refered to as a Right Shifting (*RSh*) heuristic [Ow 88b] and is always sufficient to restore schedule integrity. Clearly, this procedure can be quite inefficient, as it does not resequence operations but simply bumps them. Rather than actually bumping operations, a simple conflict propagation heuristic involves unscheduling all the operations that would be bumped by *RSh*. These unscheduled operations can then be rescheduled using the micro-

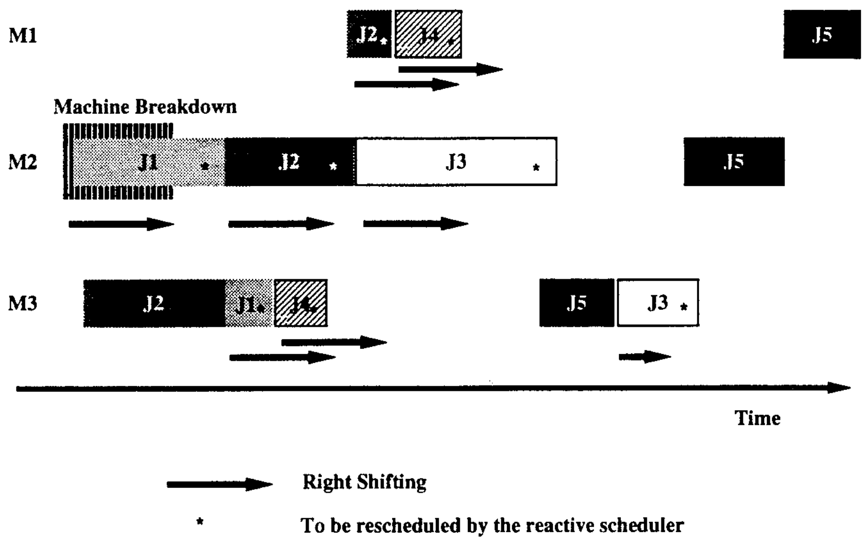


Figure 12: Simple "Right Shifting" Conflict Propagation Procedure

opportunistic procedures described in Section 3, thereby taking advantage of possible resequencing opportunities. Below we refer to this elementary conflict propagation technique as a right shifting conflict propagation procedure and denote by *React(RSh)* the reactive procedure that micro-opportunistically reschedules the operations identified by this conflict propagation procedure.

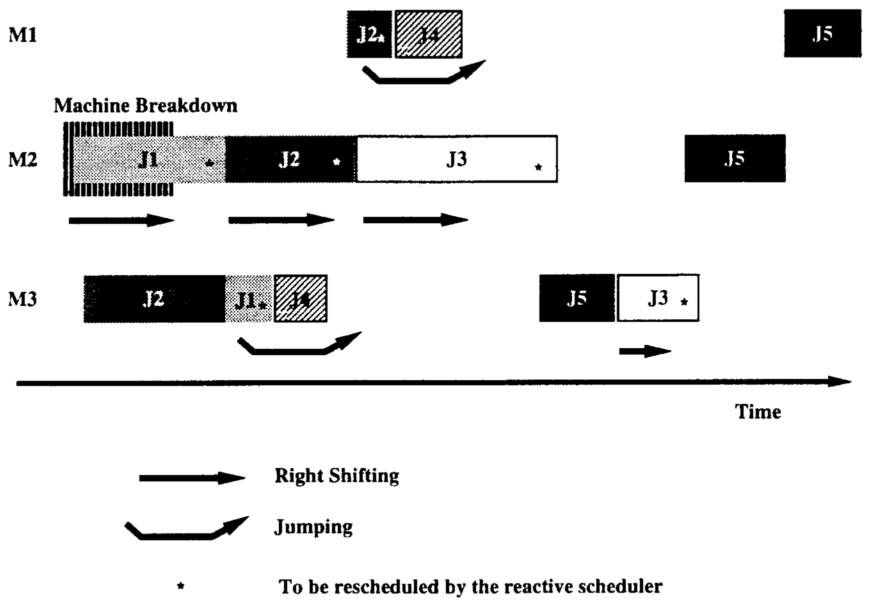


Figure 13: "Right Shifting and Jumping" Conflict Propagation Procedure

A slightly more sophisticated reactive heuristic first described in [Ow 88b] involves bumping operations forward in time, while jumping over some operations that do not need to be moved, as illustrated in Figure 13. We refer to the immediate implementation of this reactive procedure as

"Right Shifting and Jumping" (*RShJ*). Additionally, denote by *React(RShJ)* the procedure that involves (1) using *RShJ* as a conflict propagation procedure and (2) micro-opportunistically rescheduling the operations identified by this conflict propagation procedure. Like *RSh*, *RShJ* is sufficient to guarantee restoration of schedule integrity and is less disruptive than *RSh*. Similarly *React(RShJ)* is less disruptive than *React(RSh)*. However, due to the smaller number of operations it micro-opportunistically reschedules, *React(RShJ)* can be expected to sometimes produce schedules that are not as good as those obtained with *React(RSh)* (See also experiments reported below). Several procedures can be devised to improve the quality of schedules produced by *React(RShJ)*. Below we report initial experiments with two such procedures:

1. *React(RShJ+Job)* first unschedules the operations that would normally be

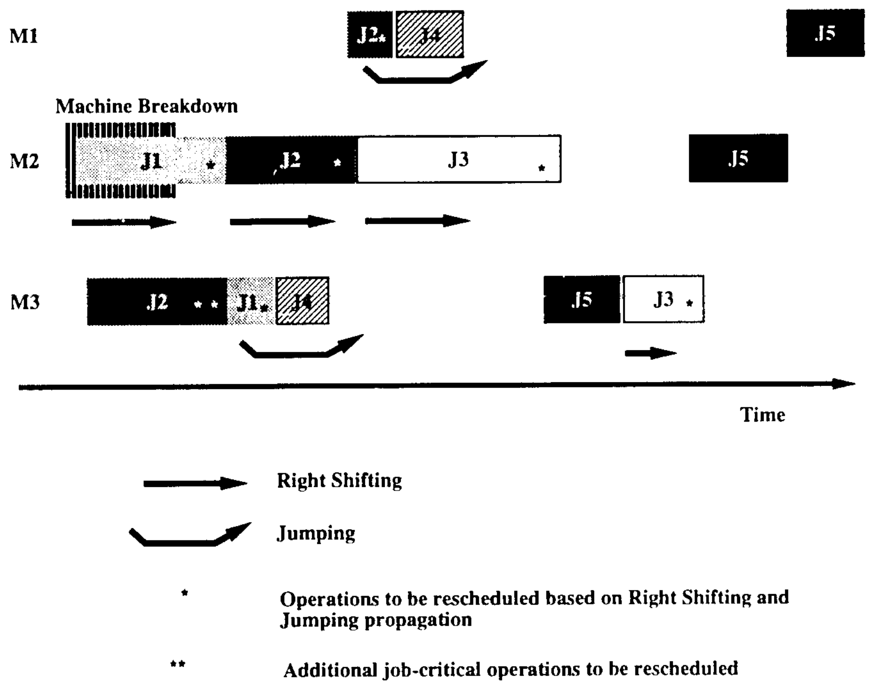

M1
J2*
J4
J5
Machine Breakdown
M2
J1 *
J2 *
J3 *
J5
M3
J2 **
J1*
J4
J5
J3 *
Time
Right Shifting
Jumping
* Operations to be rescheduled based on Right Shifting and Jumping propagation
** Additional job-critical operations to be rescheduled

Figure 14: Unscheduling additional job-critical operations.

unscheduled using *React(RShJ)* then selectively unschedules additional job-critical operations using the following two heuristics (See also Figure 14):

a. *Heuristic 1*: If the next to last operation in a job has been unscheduled, then also unschedule the last operation in that job. The intuition here is that the last operation in a job is particularly critical as it determines the completion date of the job (and hence its tardiness or earliness). By also unscheduling the job's last operation, this heuristic gives more room to the micro-opportunistic procedure to find a good schedule.

b. *Heuristic 2*: If the second operation in a job has been unscheduled, then also unschedule the first operation. Again the idea here is to give more room to the micro-opportunistic procedure, in this case by unscheduling the operation that determines the job's release date (i.e. an operation that critically impacts inventory costs).

2. *React(RShJ+Job+Int)*: In addition to rescheduling the operations unscheduled by *React(RShJ+Job)*, this procedure selectively unschedules additional operations in jobs and on resources. In the experiments reported below, the following two heuristics were used:
   a. *Heuristic 1*: If two operations surrounding an operation O on a resource R have both been unscheduled, then also unschedule operation O.
   b. *Heuristic 2*: If the two operations surrounding an operation O in a job J have both been unscheduled, then also unschedule operation O.

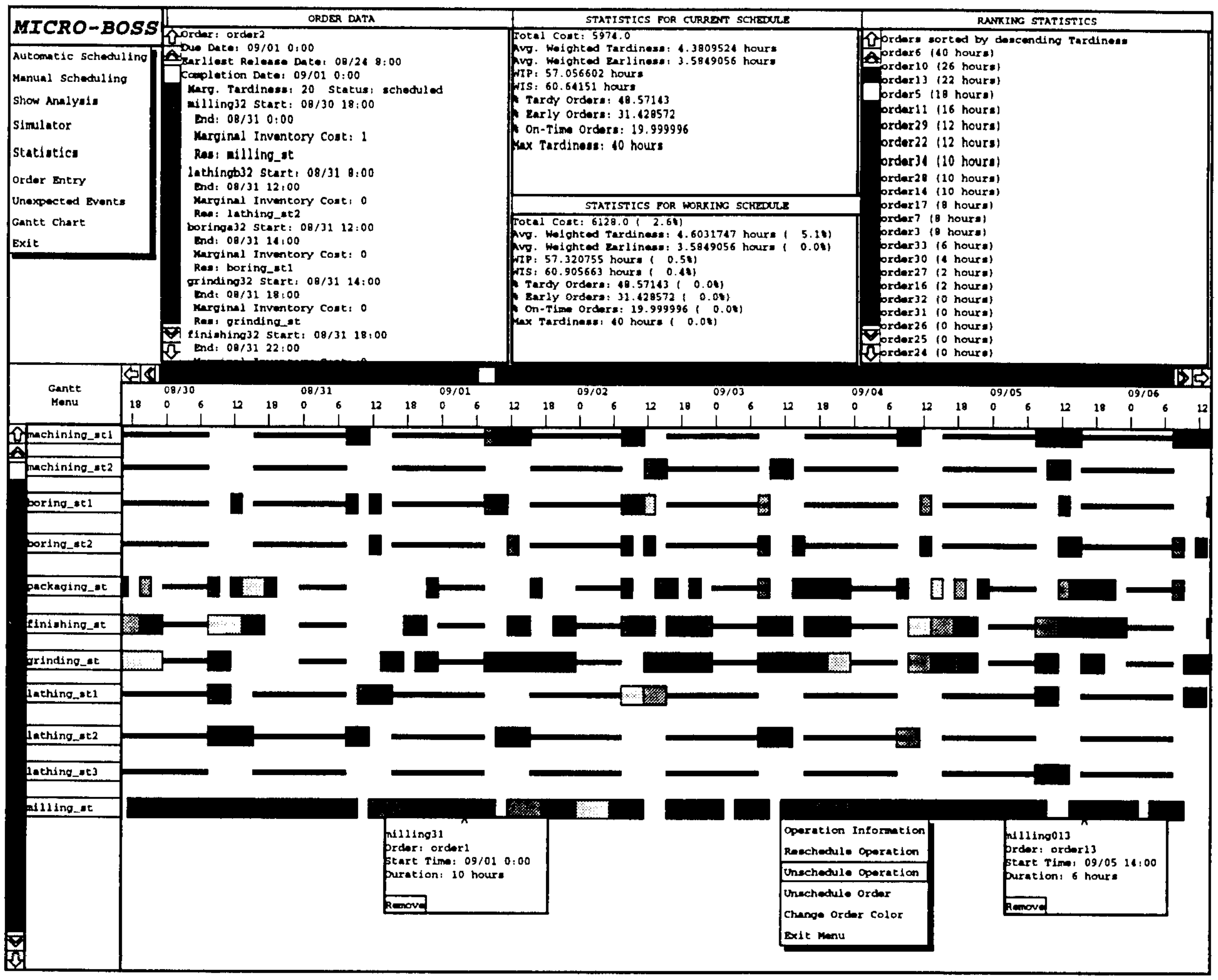
4.3 Interactive Scheduling with Micro-Boss

Although the combinatorial complexity of factory scheduling problems is best handled by automatic scheduling procedures such as the ones described earlier in this chapter, ad hoc scheduling constraints and preferences that occur very infrequently or change over time are often best accounted for through interactive manipulation of the schedule. Interactive user support should also include mechanisms that help the user identify sources of inefficiency in the schedule (e.g., tardy orders, overloaded resources, etc.) and ways of correcting these inefficiencies (e.g., adding overtime on a set of resources, rerouting some orders, etc.). Through interaction with the system, the user should be able to explore "what-if" scenarios and weigh different alternatives (e.g., decide whether to complete some jobs past their due dates or work overtime).

The Micro-Boss decision support system enables the end-user to interleave both manual and automatic (micro-opportunistic) scheduling decisions, analyze, edit, save, and compare complete and partial schedules.

Interactive schedule manipulation is performed using an interactive Gantt chart that displays each resource along with the operations to which that resource has been allocated over time (Figure 15). Schedule manipulation is performed under the supervision of the Micro-Boss consistency enforcing module, which enforces consistency with earlier scheduling decisions (manual and automatic). Partial or complete schedules can be saved and compared against each other along different metrics, including total schedule cost, average weighted tardiness, average weighted earliness, Work-In-Process and Work-In-System (which accounts for both Work-In-Process inventory and finished goods inventory). Optimistic estimates are used for partial schedules for which these metrics cannot be computed exactly. By interleaving both manual and automatic scheduling decisions and saving/restoring partial and complete schedules, the user can compare the impact of alternative scheduling decisions and perform "what-if" analyses.

Figure 15 shows a typical view of the Micro-Boss user interface. In this example, the user is getting ready to modify the working schedule displayed in the Gantt chart, by manually unscheduling an operation on which he/she just clicked. Statistics for the working schedule are compared with statistics for the "current" schedule, namely, the schedule currently in force in the system. These statistics are continuously updated as the user edits the schedule. In another



```
ORDER DATA
Order: order2
Due Date: 09/01 0:00
Earliest Release Date: 08/24 8:00
Completion Date: 09/01 0:00
 Marg. Tardiness: 20  Status: scheduled
 milling32 Start: 08/30 18:00
  End: 08/31 0:00
  Marginal Inventory Cost: 1
  Res: milling_st
 lathingb32 Start: 08/31 8:00
  End: 08/31 12:00
  Marginal Inventory Cost: 0
  Res: lathing_st2
 boringa32 Start: 08/31 12:00
  End: 08/31 14:00
  Marginal Inventory Cost: 0
  Res: boring_st1
 grinding32 Start: 08/31 14:00
  End: 08/31 18:00
  Marginal Inventory Cost: 0
  Res: grinding_st
 finishing32 Start: 08/31 18:00
  End: 08/31 22:00
```

```
STATISTICS FOR CURRENT SCHEDULE
Total Cost: 5974.0
Avg. Weighted Tardiness: 4.3809524 hours
Avg. Weighted Earliness: 3.5849056 hours
WIP: 57.056602 hours
WIS: 60.64151 hours
% Tardy Orders: 48.57143
% Early Orders: 31.428572
% On-Time Orders: 19.999996
Max Tardiness: 40 hours
```

```
STATISTICS FOR WORKING SCHEDULE
Total Cost: 6128.0 (  2.6%)
Avg. Weighted Tardiness: 4.6031747 hours (  5.1%)
Avg. Weighted Earliness: 3.5849056 hours (  0.0%)
WIP: 57.320755 hours (  0.5%)
WIS: 60.905663 hours (  0.4%)
% Tardy Orders: 48.57143 (  0.0%)
% Early Orders: 31.428572 (  0.0%)
% On-Time Orders: 19.999996 (  0.0%)
Max Tardiness: 40 hours (  0.0%)
```

```
RANKING STATISTICS
Orders sorted by descending Tardiness
order6 (40 hours)
order10 (26 hours)
order13 (22 hours)
order5 (18 hours)
order11 (16 hours)
order29 (12 hours)
order22 (12 hours)
order34 (10 hours)
order28 (10 hours)
order14 (10 hours)
order17 (8 hours)
order7 (8 hours)
order3 (8 hours)
order33 (6 hours)
order30 (4 hours)
order27 (2 hours)
order16 (2 hours)
order32 (0 hours)
order31 (0 hours)
order26 (0 hours)
order25 (0 hours)
order24 (0 hours)
```

```
milling31
Order: order1
Start Time: 09/01 0:00
Duration: 10 hours
```

```
milling013
Order: order13
Start Time: 09/05 14:00
Duration: 6 hours
```

Figure 15: The Micro-Boss user interface allows for interactive manipulation of schedules. By interleaving both manual and automatic scheduling decisions, saving and comparing alternative schedules, the user can easily assess different trade-offs and locally impose ad hoc constraints or preferences that are not easily amenable to representation in the computer model.

window, the user can check information about specific orders (*order2* in this example). In yet another window, he/she has elected to rank orders based on their tardiness in the working schedule. Alternative metrics to rank jobs or resources can be selected in the statistics menu (e.g., cost, tardiness, flowtime, resource utilization, etc.). By clicking on boxes displayed in the Gantt chart, the user can directly obtain information on specific operations (e.g., information on operation *milling31*), manually unschedule and reschedule operations (by moving the corresponding box in the Gantt chart), unschedule jobs, or highlight a job by changing its color. The Gantt menu also allows for zooming in and out of the Gantt chart, unscheduling specific resource/time intervals, displaying contention measures over time for different resources, etc.

5 Performance Evaluation

Experimental studies performed with an initial version of Micro-Boss implemented using Knowledge Craft^{TM} were reported in [Sadeh 91b]. These experiments studied the performance of the system under a variety of scheduling conditions and different cost assumptions[16]. They included comparisons with combinations of popular priority dispatch rules and release policies advocated in the Operations Research literature, comparisons with coarser bottleneck-centered approaches to scheduling described in the Artificial Intelligence literature and a comparison with a variation of Micro-Boss in which resource contention was measured using unbiased demand profiles.

In this chapter, we first report the results of a similar study performed on the same set of scheduling problems with a more recent version of the system written in C++. Additionally, we also report experiments evaluating the effectiveness of new bottleneck optimization heuristics developed for problems with setups and report experiments in reactive scheduling

At the present time (January 1994), the new version of Micro-Boss is two orders of magnitude faster than the version described in [Sadeh 91b] on this set of problems, mainly because of the C++ reimplementation and the use of rough demand profiles to identify small areas of high contention over which more detailed profiles are then constructed (see section 2.4). The new system also uses a more powerful consistency enforcing procedure (See Subsection 2.2) than the original version, which almost eliminates the need for backtracking on the experiments reported in this chapter. Finally, the new system also produces schedules that are significantly better than those obtained with the earlier version. This improvement in schedule quality is mainly attributed to the use of a more accurate set of propagation heuristics to update the best remaining start time(s) of unscheduled operations during construction of the schedule and the use of a stronger bias in the construction of demand profiles.

The results reported below were obtained on a suite of 80 scheduling problems. The suite, which is described in detail in [Sadeh 91b], consists of eight sets of scheduling problems obtained by adjusting three parameters to cover a wide range of scheduling conditions. The three parameters are the following: an average due date parameter (tight versus loose average due date), a due date range parameter (narrow versus wide range of due dates), and a parameter controlling the number of major bottlenecks (in this case one or two). For each parameter combination, a set of 10 scheduling problems was randomly generated (see Table 2), thereby resulting in a total of 80 scheduling problems (10 problems x 2 average due date values x 2 due date ranges x 2 bottleneck configurations). Each problem requires scheduling 20 jobs on 5 resources for a total of 100 operations. Marginal tardiness costs in these problems were set to be, on the average, five times larger than marginal inventory costs to model a situation where

[16]Knowledge Craft is a registered trademark of Carnegie Group.

Problem Sets			
Problem Set	Number of Bottlenecks	Avg. Due Date	Due Date Range
1	1	loose	wide
2	1	loose	narrow
3	1	tight	wide
4	1	tight	narrow
5	2	loose	wide
6	2	loose	narrow
7	2	tight	wide
8	2	tight	narrow

Table 2: Characteristics of the eight problem sets.

tardiness costs dominate but inventory costs are non-negligible[17].

Micro-Boss required between 10 and 15 CPU seconds to schedule each problem on a DECstation™ 5000/200. Nearly all problems were solved without any backtracking.

5.1 Comparison Against Combinations of Priority Dispatch Rules and Release Policies.

In a first set of experiments, Micro-Boss was compared with the best of a set of 39 combinations of popular priority dispatch rules and release policies. The priority dispatch rules used in these experiments were of two types:

1. a set of five priority dispatch rules that have been reported to be particularly good at reducing tardiness under various scheduling conditions [Vepsalainen 87]: the Weighted Shortest Processing Time (WSPT) rule; the Earliest Due Date (EDD) rule; the Slack per Remaining Processing Time (S/RPT) rule; and two parametric rules, the Weighted Cost OVER Time (WCOVERT) rule and the Apparent Tardiness Cost (ATC) rule.

2. an exponential version of the parametric early/tardy dispatch rule recently developed by Ow and Morton [Ow 89, Morton 88] and referred to below as EXP-ET. This rule differs from the other five in that it can explicitly account for both

[17]Experiments under different cost assumptions were also reported in [Sadeh 91b].

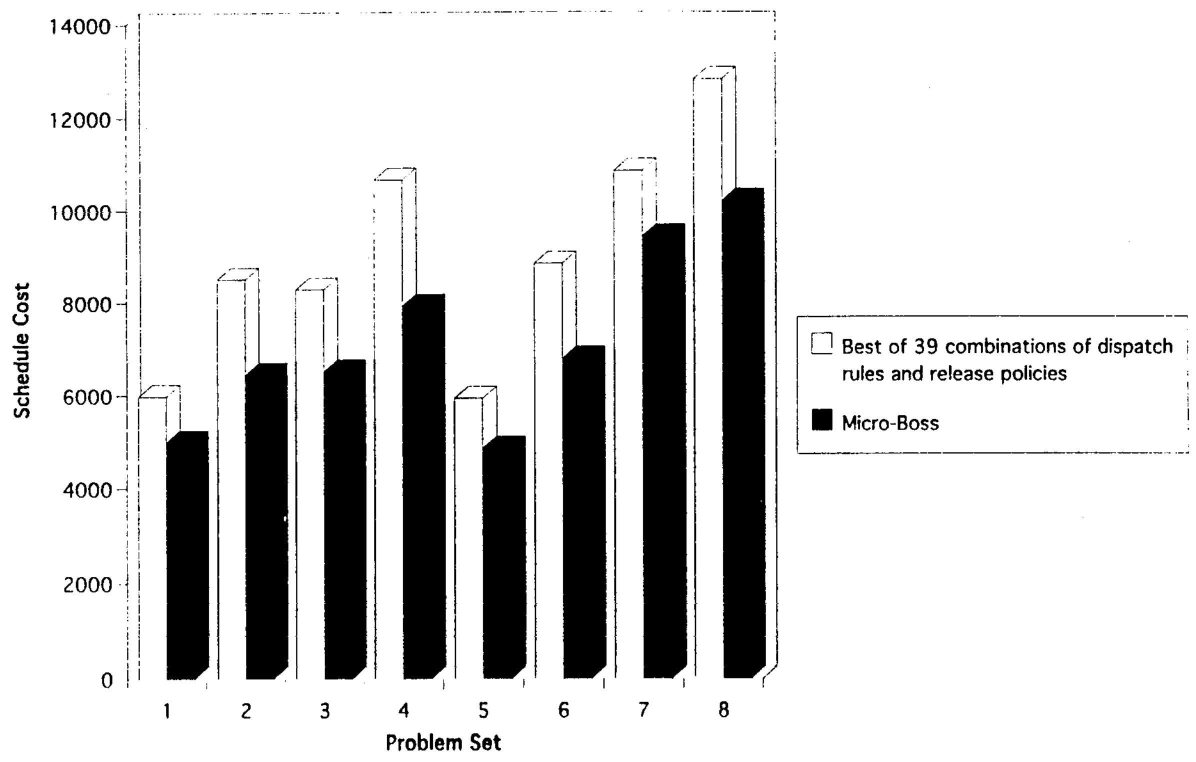


Figure 16: Comparison of Micro-Boss and the best of 39 combinations of priority dispatch rules and release policies under 8 different scheduling conditions (10 problems were generated under each condition).

tardiness and inventory costs.

EXP-ET was successively run in combination with an immediate release policy (IM-REL) that allows each job to be released immediately and with an intrinsic release policy that only releases jobs when their priorities become positive, as suggested in [Morton 88]. The other five dispatch rules were successively run in combination with two release policies: an immediate release policy (IM-REL) and the Average Queue Time release policy (AQT) described in [Morton 88]. AQT is a parametric release policy that estimates queuing time as a multiple of the average job duration (the look-ahead parameter serving as the multiple). A job's release date is determined by offsetting the job's due date by the sum of its total duration and its estimated queuing time. Combinations of release policies and dispatch rules with a look-ahead parameter were successively run with four different parameter values that generally appeared to produce the best schedules. By combining these different dispatch rules, release policies, and parameter settings a total of 39 heuristics[18] was obtained. On each problem, the best of the 39 schedules produced by these heuristics was compared with the schedule obtained by Micro-Boss. Among the 39 scheduling heuristics (i.e., excluding Micro-Boss), each of the 6 dispatch rules (WSPT, EDD, S/RPT, WCOVERT, ATC and EXP-ET) and each of the 3 release policies (IM-REL, AQT and EXP-ET's intrinsic release policy) performed best on at least one problem out of the 80 and 12 combinations out of the 39 performed best on at least 1 problem.

Figure 16 compares the average cost of the schedules produced by Micro-Boss with the average cost obtained by the best of the 39 combinations of dispatch rules and release policies on each problem set. Schedule cost was computed as the sum of tardiness and inventory costs, as specified in Equation (3). The results indicate that Micro-Boss consistently outperformed the combination of 39 heuristics under all eight conditions of the study. Overall Micro-Boss yielded reductions of 20% in schedule cost over the 39 heuristics. A more detailed analysis indicates that this reduction in schedule cost corresponds to a reduction of about 20% in tardiness costs and about 23% in inventory costs (combined work-in-process and finished goods inventory costs).

5.2 Comparison Against Coarser Opportunistic Scheduling Procedures

Micro-Boss was also compared with several coarser opportunistic schedulers that dynamically combine a resource-centered perspective and a job-centered perspective, such as in the OPIS scheduling system [Ow 88a]. Although OPIS relies on a set of repair heuristics to recover from inconsistencies [Ow 88b], the macro-opportunistic schedulers of this study were built to use the same consistency enforcing techniques and the same backtracking scheme as Micro-Boss[19]. The macro-opportunistic schedulers also used the same demand profiles as Micro-Boss. When average demand for the most critical resource/time interval was above some threshold level (a parameter of the system that was empirically adjusted), the macro-opportunistic scheduler focused on scheduling the operations requiring that resource/time interval; otherwise, it used a job-centered perspective to identify a critical job and schedule some or all of the operations in that job. Each time a resource/time interval or a portion of a job was scheduled, new demand profiles were computed to decide which scheduling perspective to use next.

[18]The 39 combinations were as follows: EXP-ET and its intrinsic release policy (times four parameter settings), EXP-ET/IM-REL (times four parameter settings), EDD/AQT (times four parameter settings), EDD/IM-REL, WSPT/AQT (times four parameter settings), WSPT/IM-REL, S/RPT/AQT (times four parameter settings), S/RPT/IM-REL, WCOVERT/IM-REL (times four parameter settings), WCOVERT/AQT (times four parameter settings), ATC/IM-REL (times four parameter settings), and ATC/AQT (times four parameter settings).

[19]An alternative would have been to implement a variation of Micro-Boss using the same repair heuristics as OPIS. Besides being time-consuming to implement, such a comparison would have been affected by the quality of the specific repair heuristics currently implemented in the OPIS scheduler.

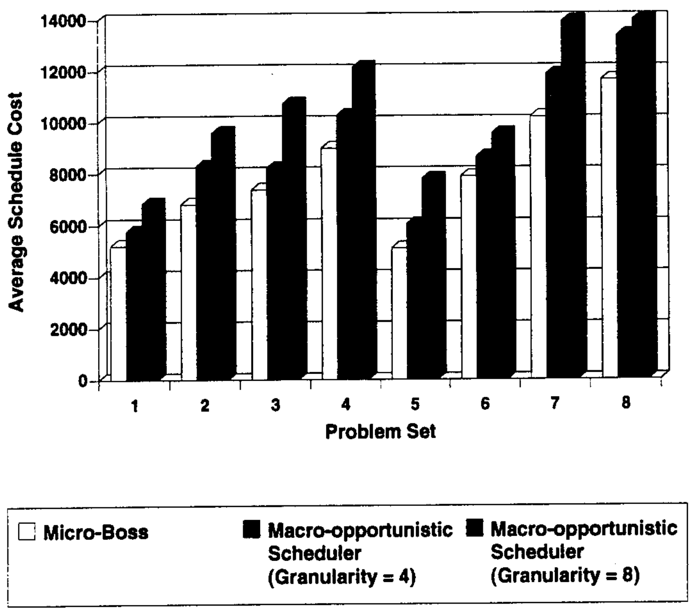


Figure 17: Comparison of Micro-Boss and two coarser opportunistic schedulers.

Figure 17 summarizes the results of a comparison between Micro-Boss[20] and two macro-opportunistic schedulers that differed in the number of operations that they were allowed to schedule at once in their resource-centered perspective (referred to below as the granularity of the scheduler). The macro-opportunistic scheduler with granularity 4 was allowed to schedule as many as 4 operations in its resource-centered perspective, after which it had to compute new demand profiles and decide which subproblem (job-centered or resource-centered) to focus on next. The macro-opportunistic scheduler with granularity 8 was allowed to schedule at once as many as 8 operations in its resource-centered perspective. The results in Figure 17 indicate not only that Micro-Boss consistently produced better schedules than the two macro-opportunistic schedulers but also that schedule performance degraded as the granularity of the macro-opportunistic scheduler was increased, namely, as the search procedure became less flexible. More detailed performance measures not presented here indicate that the reductions in schedule cost achieved by Micro-Boss correspond to reductions in both tardiness and inventory costs.

Overall, these results strongly suggest that the additional flexibility of a micro-opportunistic scheduling procedure over coarser opportunistic procedures generally yields important improvements in schedule quality.

[20]These experiments as well as the ones presented in the next subsection were performed in 1993 with an earlier version of Micro-Boss than the one used in the comparison with dispatch rules.

5.3 Evaluating the Impact of Using Biased Demand Profiles

A third set of experiments was carried out to test the effect of using biased demand profiles to guide the micro-opportunistic scheduler. A variation of Micro-Boss using unbiased demand profiles was run on the same set of 80 scheduling problems.

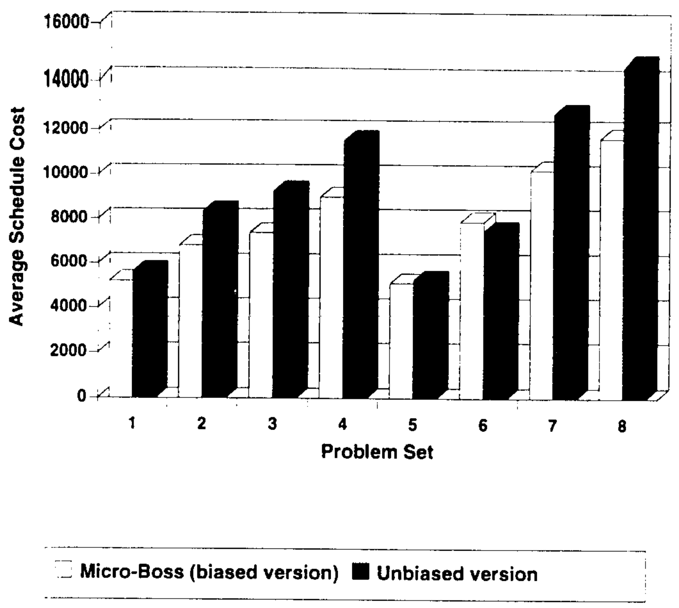


Figure 18: Comparison of the cost of the schedules produced by Micro-Boss and a variation of the system that used unbiased demand profiles.

Figure 18 compares the average schedule costs obtained by both variations of Micro-Boss. In 7 out of the 8 scheduling situations of the study, biasing the demand profiles produced reductions in schedule cost ranging from 3 to 22 percent, including an impressive 20 percent in the most difficult scheduling situation (Problem Set 8 with two bottlenecks, a tight average due date and a narrow range of due dates). In the one case (out of eight) where the unbiased version produced better schedules, the biased version was only 5% worse. A more detailed analysis of the results indicates that overall, the biased version of Micro-Boss performed 30% better with respect to tardiness while incurring a slight increase of 0.6% in inventory costs. Altogether, biasing the demand profiles reduced schedule costs by more than 15%. These results validate both the idea of building biased demand profiles to guide the micro-opportunistic search procedure and the particular technique used in Micro-Boss to operationalize this idea (namely, the use of the *mincost* functions). In general, it should be possible to obtain even better results by varying the bias according to specific problem characteristics. One could also consider fine-tuning the bias during the construction of the schedule.

5.4 Evalutation of the Micro-Boss Bottleneck Optimization Heuristics for Problems with Setups

5.5 Reactive Scheduling Experiments

Table 3 summarizes the results of experiments conducted with the reactive heuristics described earlier in this chapter and a a procedure that systematically reschedules all available operations (i.e. all operations that had not yet been started at the time of the contingency considered in each problem).

A total of 40 reactive problems were generated similar to the problems used for the experiments reported in Section 4. For each problem, a single machine breakdown was randomly generated in such a way that it conflicted with the schedule of at least one operation. Breakdowns were generated both on bottleneck and non-bottleneck machines. Performance is reported along the following dimensions:

- *Normalized Cost*: a normalized measure of the average schedule cost obtained with each technique;
- *Tardiness*: the average weighted tardiness of the schedules produced by each technique;
- *WIS*: the average work-in-system inventory of the schedules produced by each technique, as defined in [Sadeh 91b]. This is a measure that accounts for both work-in-process and finished-goods inventories.
- *Nb. op. resched.*: the average number of operations rescheduled by each technique.

Method	Normalized Cost	Tardiness	WIS	Nb. op. resched.
Total Rescheduling	1.06	12.4	61.2	52.3
RSh	1.23	15.3	62.8	34.4
RShJ	1.20	14.6	63.0	32.5
React(RSh)	1.13	13.1	63.5	34.4
React(RShJ)	1.16	13.5	64.3	32.5
React(RShJ + Job)	1.12	13.0	62.8	34.3
React(RShJ + Job + Int)	1.09	12.8	62.2	35.7

Table 3: Comparison of Seven Reactive Scheduling Procedures.

As expected, the results indicate that the worst schedules (highest cost) were produced by *RSh* and the best ones by the total rescheduling procedure. The total rescheduling procedure was also the most disruptive one (over 52 operations rescheduled on the average) and took the most time. The right shifting procedure on the other hand was generally the fastest one, requiring an average of 3 CPU seconds in these experiments. More interestingly, the results indicate that *React(RShJ+Job+Int)* produced schedules almost as good as those obtained with the the total rescheduling procedure and did so while rescheduling significantly fewer operations. In these experiments, *React(RShJ+Job+Int)* required about 9 CPU seconds on the average, only 3 times as much as the simplest procedure, *RSh*.

6 Concluding Remarks

Current computer solutions to production management such as the one implemented in MRP/MRP-II systems are of limited help, because they rely on oversimplified models of the plant and only provide weak feedback loops to update the production schedule during execution (typically, complete updates of the schedule are only performed on a weekly basis). A major challenge for researchers in production scheduling is to come up with new techniques that can account more precisely for actual manufacturing objectives and constraints, including execution contingencies such as machine breakdowns, new job arrivals, variations in processing times, yields, etc. New production scheduling tools should also enable the user to interactively perform "what-if" analysis and account for ad hoc constraints and/or preferences that are not easily amenable to representation in the computer model.

In this chapter, we presented Micro-Boss, a decision support system for factory scheduling. Micro-Boss aims at combining powerful predictive, reactive, and interactive scheduling capabilities. To this end, the system relies on a new micro-opportunistic search procedure that enables it to continuously track the evolution of micro-bottlenecks (or conflicts) during the construction or repair of the schedule and to refocus its optimization effort on those micro-bottlenecks that appear most critical. This approach differs from earlier opportunistic approaches [Ow 88a, Collinot 88], because it does not require scheduling large resource subproblems or large job subproblems before revising the current scheduling strategy. The results of an experimental study comparing Micro-Boss with combinations of popular priority dispatch rules and release policies advocated in the Operations Research literature as well as coarser opportunistic scheduling approaches proposed in the Artificial Intelligence literature, suggest that the flexibility of this new search procedure can often yield important improvements in schedule quality. We find that because of their flexibility, micro-opportunistic scheduling procedures are also particularly well suited for repairing schedules in the face of execution contingencies and can easily be integrated in interactive decision support systems that enable the user to incrementally manipulate and compare alternative schedules.

Although our work on Micro-Boss has focused on generalized versions of the job shop scheduling problem, micro-opportunistic scheduling techniques have been applied to other manufacturing problems and other classes of problems such as transportation scheduling. Rautaruukki Oy, a large Finnish steel manufacturer, and researchers at the Helsinki University of Technology have reported adapting an earlier version of our micro-opportunistic scheduling heuristics to schedule a steel rolling mill [Torma 91]. Variations of the Micro-Boss scheduling heuristics are also used in the Knowledge Based Logistics Planning Shell (KBLPS) developed by Carnegie Group, Inc. (CGI) and LB&M Associates to solve U.S. army transportation scheduling problems and ammunition distribution planning problems [Dunmire 90, Camden 90, Saks 92]. Other efforts using variations of the micro-opportunistic techniques developed in the context of Micro-Boss are described in [Berry 91, Linden 91, Paolucci 92] and [Winklhofer 92].

Current research efforts within our project aim at applying and extending the existing approach to solve both manufacturing and transportation scheduling problems.

[Adams 88] J. Adams, E. Balas, and D. Zawack.
The Shifting Bottleneck Procedure for Job Shop Scheduling.
Management Science 34(3):391-401, 1988.

[Baker 90] Kenneth R. Baker and Gary D. Scudder.
Sequencing with Earliness and Tardiness Penalties: A Review.
Operations Research 38(1):22-36, January-February, 1990.

[Bean 91] Bean, J.C.,J.R. Birge, J. Mittenthal, C.E. Noon.
Matchup Scheduling with Multiple Resources, Release Dates and Disruptions.
Operations Research 39(3):470-483, May-June, 1991.

[Berry 91] Pauline M. Berry.
The PCP: A Predictive Model for Satisfying Conflicting Objectives in Scheduling Problems.
Technical Report, Centre Universitaire d'Informatique, Universite de Geneve, 12, Rue du Lac, CH-1207, Geneva, Switzerland, 1991.

[Camden 90] Camden, R., Dunmire, C., Goyal, R., Sathi,N., Elm, B., and Fox, M.
Distribution Planning: An Integration of Constraint Satisfaction and Heuristic Search Techniques.
In *Proceedings of the Conference on AI Applications in Military Logistics.* 1990.

[Collinot 88] A. Collinot, C. Le Pape and G. Pinoteau.
SONIA: a Knowledge-based Scheduling System.
International Journal of Artificial Intelligence in Engineering 2(4):86-94, 1988.

[Dauzere-Peres 90]
S. Dauzere-Peres and J.B. Lasserre.
A Modified Shifting Bottleneck Procedure for Job Shop Scheduling.
Technical Report LAAS 90106, Laboratoire d'Automatique et d'Analyse des Systemes, 7, Av. du Colonel Roche, 31077 Toulouse Cedex, France, 1990.

[Dunmire 90] Dunmire, C., Sathi,N., Goyal, R., Fox, M., and Kott, A.
Ammunition Inventory Planning: An Integration of Configuration and Resource Allocation Techniques.
In *Proceedings of the Conference on AI Applications in Military Logistics.* 1990.

[Fox 83] Mark S. Fox.
Constraint-Directed Search: A Case Study of Job-Shop Scheduling.
PhD thesis, Department of Computer Science, Carnegie-Mellon University, 1983.

[Fox 87] R.E. Fox.
OPT: Leapfrogging the Japanese.
Just-in-time Manufacture.
In C.A. Voss,
IFS Ltd, Springer Verlag, 1987.

[French 82] S. French.
Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.
Wiley, 1982.

[Garey 79] M.R. Garey and D.S. Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
Freeman and Co., 1979.

[Goldratt 80] Eliyahu M. Goldratt.
Optimized Production Timetable: Beyond MRP: Something Better is finally Here.
October, 1980
Speech to APICS National Conference.

[Graves 81] Graves, S.C.
A Review of Production Scheduling.
Operations Research 29(4):646-675, July-August, 1981.

[Jacobs 84] F. Robert Jacobs.
OPT Uncovered: Many Production Planning And Scheduling Concepts Can Be Applied With Or Without The Software.
Industrial Engineering 16(10):32-41, October, 1984.

[LePape 87] Claude Le Pape and Stephen F. Smith.
Management of Temporal Constraints for Factory Scheduling.
Technical Report, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1987.
Also appeared in Proc. Working Conference on Temporal Aspects in Information Systems, Sponsored by AFCET and IFIP Technical Committee TC8, North Holland Publishers, Paris, France, May 1987.

[Li&Sadeh 93] Gang Li and Norman Sadeh.
Single-Machine Early/Tardy Scheduling Problem with Setups: A Hybrid Heuristic Approach.
Technical Report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1993.
Working paper. Presented at the Joint National ORSA/TIMS meeting held in San Francisco, November 1-4, 1992.

[Linden 91] Theodore A. Linden.
Preference-Directed, Cooperative Resource Allocation and Scheduling.
Technical Report, Advanced Decision Systems, 1500 Plymouth St., Mountain View, CA 94043, September, 1991.

[Mackworth 85] A.K. Mackworth and E.C. Freuder.
The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems.
Artificial Intelligence 25(1):65-74, 1985.

[Minton 90] S. Minton, M.D. Johnston, A.B. Philips, P. Laird.
Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method.
In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 17-24. 1990.

[Morton 88] T.E. Morton, S.R. Lawrence, S. Rajagopolan, S Kekre.
SCHED-STAR: A Price-Based Shop Scheduling Module.
Journal of Manufacturing and Operations Management :131-181, 1988.

[Orlicky 75] Joseph Orlicky.
Material Requirements Planning.
McGraw Hill, New York, 1975.

[Ow 85] Peng Si Ow.
Focused Scheduling in Proportionate Flowshops.
Management Science 31(7):852-869, 1985.

[Ow 88a] Peng Si Ow and Stephen F. Smith.
Viewing Scheduling as an Opportunistic Problem-Solving Process.
Annals of Operations Research 12:85-108, 1988.

[Ow 88b] P.S. Ow, S.F. Smith, and A. Thiriez.
Reactive Plan Revision.
In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 77-82. 1988.

[Ow 89] Peng Si Ow and Thomas Morton.
The Single Machine Early/Tardy Problem.
Management Science 35(2):177-191, 1989.

[Panwalkar 77] S.S. Panwalkar and Wafik Iskander.
A Survey of Scheduling Rules.
Operations Research 25(1):45-61, January-February, 1977.

[Paolucci 92] Paolucci, E., Patriarca, E., Sem, M., and Gini G.
Predit: A Temporal Predictive Framework for Scheduling Systems.
In *Proceedings of the AAAI Spring Symposium on Practical Approaches to Scheduling and Planning*, pages 150-154. 1992.

[Sadeh 91a] N. Sadeh and M.S. Fox.
Variable and Value Ordering Heuristics for Hard Constraint Satisfaction Problems: an Application to Job Shop Scheduling.
Technical Report CMU-RI-TR-91-23, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1991.
Submitted to the Artificial Intelligence Journal.

[Sadeh 91b] Norman Sadeh.
Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.
PhD thesis, School of Computer Science, Carnegie Mellon University, March, 1991.

[Sadeh 92] Norman Sadeh, Katia Sycara, and Yalin Xiong.
Backtracking Techniques for Hard Scheduling Problems.
Technical Report CMU-RI-TR-92-06, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1992.
Submitted to the Artificial Intelligence Journal.

[Sadeh 93] Sadeh, N.M., S. Otsuka, and R. Schnelbach.
Predictive and Reactive Scheduling with the Micro-Boss Production Scheduling and Control System.
In *Proceedings of the IJCAI-93 Workshop on Knowledge-based Production Planning, Scheduling, and Control.* Chambery, France, August, 1993.

[Sadeh 94] Norman Sadeh.
Micro-Opportunistic Scheduling: The MICRO-BOSS Factory Scheduler.
Intelligent Scheduling.
In Mark Fox and Monte Zweben,
Morgan Kaufmann Publishers, 1994, Chapter 4.

[Saks 92] Victor Saks, Al Kepner, and Ivan Johnson.
Knowledge Based Distribution Planning.
Technical Report, Carnegie Group, Inc., 5 PPG Place, Pittsburgh, PA 15222, 1992.

[Serafini 88] P. Serafini, W. Ukovich, H. Kirchner, F. Giardina, and F. Tiozzo.
Job-shop scheduling: a case study.
Operations Research Models in FMS.
In F. Archetti, M. Lucertini, and P. Serafini,
Springer, Vienna, 1988.

[Smith 86] S. Smith, M. Fox, and P.S. Ow.
Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems.
AI Magazine 7(4):45-61, Fall, 1986.

[Smith 90a] Smith, S.F., N. Keng, and K. Kempf.
Exploiting Local Flexibility During Execution of Pre-Computed Schedules.
Technical Report CMU-TR-RI-90-13, The Robotics Institute, Carnegie Mellon Univeristy, June, 1990.

[Smith 90b] Stephen F. Smith, Peng Si Ow, Nicola Muscettola, Jean-Yves Potvin, Dirk Matthys.
An Integrated Framework for Generating and Revising Factory Schedules.
Journal of the Operational Research Society 41(6):539-552, 1990.

[Torma 91] Seppo Torma, Ora Lassila and Markku Syrjanen.
Adapting the Activity-Based Scheduling Method to Steel Rolling.
In G. Doumeingts, J. Browne, and M Tomljanovich (editor), *Proceedings of the Fourth IFIP Conference on Computer Applications in Production and Engineering (CAPE'91)*, pages 159-166. Elsevier Science Publishers B.V. (North Holland), 1991.

[Vepsalainen 87] Ari P.J. Vepsalainen and Thomas E. Morton.
Priority Rules for Job Shops with Weighted Tardiness Costs.
Management Science 33(8):1035-1047, 1987.

[Vollmann 88] Thomas Vollmann, William Berry, and Clay Whybark.
Manufacturing Planning and Control.
Dow Jones-Irwin, Homewood, IL, 1988.
Second Edition.

[Wight 81] Oliver Wight.
MRP II: Unlocking America's Productivity Potential.
Oliver Wight Limited Publications, Williston, VT, 1981.

[Wight 84] Oliver Wight.
Manufacturing Resource Planning: MRPII.
Oliver Wight Limited Publications, Essex Junction, VT, 1984.

[Winklhofer 92] Andreas Winklhofer, Manfred Maierhofer, and Paul Levi.
Efficient Propagation and Computation of Problem Features for Activity-Based Scheduling.
In *Proceedings of the Seventh Symposium on Information Control Problems in Manufacturing Technology (INCOM-92).* Toronto, Canada, 1992.

[Zweben 91] Monte Zweben, Eugene Davis, and Michael Deale.
Iterative Repair for Scheduling and Rescheduling.
Technical Report, NASA Ames Reserch Center, MS 244-17, Moffett Field, CA 94035, 1991.

Backtracking Techniques for Hard Job Shop Scheduling Problems

Norman Sadeh, Katia Sycara and Yalin Xiong

The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, Pennsylvania 15213-3891

Abstract

This paper studies a version of the job shop scheduling problem in which some operations have to be scheduled within non-relaxable time windows (i.e. earliest/latest possible start time windows). This problem is a well-known NP-complete Constraint Satisfaction Problem (CSP). A popular method for solving this type of problems involves using depth-first backtrack search. In our earlier work, we focused on the development of consistency enforcing techniques and variable/value ordering heuristics that improve the efficiency of this search procedure. In this paper, we combine these techniques with new look-back schemes that help the search procedure recover from so-called deadend search states (i.e. partial solutions that cannot be completed without violating some constraints). More specifically, we successively describe three "intelligent" backtracking schemes: (1) *Dynamic Consistency Enforcement* dynamically identifies critical subproblems and determines how far to backtrack by selectively enforcing higher levels of consistency among variables participating in these critical subproblems, (2) *Learning Ordering From Failure* dynamically modifies the order in which variables are instantiated based on earlier conflicts, and (3) *Incomplete Backjumping Heuristic* abandons areas of the search space that appear to require excessive computational efforts. These schemes are shown to (1) further reduce the average complexity of the backtrack search procedure, (2) enable our system to efficiently solve problems that could not be solved otherwise due to excessive computation cost, and (3) be more effective at solving job shop scheduling problems than other look-back schemes advocated in the literature.

This paper is a short version of a paper to appear in the *Artificial Intelligence* Journal [Sadeh94a].

This research was supported, in part, by the Defense Advanced Research Projects Agency under contract #F30602-91-F-0016, and in part by grants from McDonnell Aircraft Company and Digital Equipment Corporation.

1. Introduction

This paper is concerned with the design of recovery schemes for incremental scheduling approaches that sometimes require undoing earlier scheduling decisions in order to complete the construction of a feasible schedule.

Job shop scheduling deals with the allocation of resources over time to perform a collection of tasks. The job shop scheduling model studied in this paper further allows for operations that have to be scheduled within non-relaxable time windows (e.g. earliest possible start time/latest possible finish time windows). This problem is a well-known NP-complete Constraint Satisfaction Problem (CSP) [Garey 79]. Instances of this problem include factory scheduling problems, in which some operations have to be performed within one or several shifts, spacecraft mission scheduling problems, in which time windows are determined by astronomical events over which we have no control, factory rescheduling problems, in which a small set of operations need to be rescheduled without revising the schedule of other operations, etc.

One approach to solving CSPs is to use depth-first backtrack search [Walker 60, Golomb 65, Bitner 75]. Using this approach, scheduling problems can be solved through the iterative selection of an operation to be scheduled next (i.e. variable selection) and the tentative assignment of a reservation (i.e. value) to that operation. If in the process of constructing a schedule, a partial solution is reached that cannot be completed without violating some of the problem constraints, one or several earlier assignments need to be undone. This process of undoing earlier assignments is referred to as *backtracking*. It deteriorates the efficiency of the search procedure and increases the time required to come up with a solution. While the worst-case complexity of backtrack search is exponential, several techniques to reduce its average-case complexity have been proposed in the literature [Dechter 88]:

- *Consistency Enforcing Schemes*: These techniques prune the search space from alternatives that cannot participate in a global solution [Mackworth 85]. There is generally a tradeoff between the amount of consistency enforced in each search state[1] and the savings achieved in search time.
- *Variable/Value Ordering Heuristics*: These heuristics help judiciously decide which variable to instantiate next and which value to assign to that variable [Bitner 75, Haralick 80, Purdom 83, Dechter 88, Fox 89, Sadeh 91]. By first instantiating difficult variables, the system increases its chances of completing the current partial solution without backtracking [Haralick 80, Fox 89, Sadeh 91]. Good value ordering heuristics reduce backtracking by selecting values that are expected to participate in a large number of solutions [Dechter 88, Sadeh 91].
- *Look-back Schemes*: [Stallman 77, Doyle 79, Gaschnig 79, Dechter 89a] While it is possible to design consistency enforcing schemes and variable/value ordering

[1]A search state is associated with each partial solution. Each search state defines a new CSP whose variables are the variables that have not yet been instantiated and whose constraints are the initial problem constraints along with constraints reflecting current assignments.

heuristics that are, on average, very effective at reducing backtracking, it is generally impossible to efficiently guarantee backtrack-free search. Look-back schemes are designed to help the system recover from deadend states and, if possible, learn from past mistakes .

In our earlier work, we focused on the development of efficient consistency enforcing techniques and variable/value ordering heuristics for job shop scheduling CSPs [Sadeh 88, Sadeh 89, Fox 89, Sycara 91, Sadeh 90, Sadeh 91, Sadeh 92]. In this paper, we combine these techniques with new look-back schemes. These schemes are shown to further reduce the average complexity of the search procedure. They also enable our system to efficiently solve problems that could not be efficiently solved otherwise. Finally, experimental results indicate that these techniques are more effective at solving job shop scheduling problems than other look-back schemes advocated in the literature.

The simplest deadend recovery strategy goes back to the most recently instantiated variable with at least one alternative value left, and assigns ones of the remaining values to the variable. This strategy is known as *chronological backtracking*. Often the source of the current deadend is not the most recent assignment but an earlier one. Because it typically modifies assignments that have no impact on the conflict at hand, chronological backtracking often returns to similar deadend states. When this happens, search is said to be *thrashing*. Thrashing can be reduced using backjumping schemes that attempt to backtrack all the way to one of the variables at the source of the conflict [Gaschnig 79]. Search efficiency can be further improved by learning from past mistakes. For instance, a system can record earlier conflicts in the form of new constraints that will prevent it from repeating earlier mistakes [Stallman 77, Doyle 79]. Dependency-directed backtracking is a technique incorporating both backjumping and constraint recording [Stallman 77]. Although dependency-directed backtracking can greatly reduce the number of search states that need to be explored, this scheme is often impractical due to the exponential worst-case complexity of its constraint recording component (both in time and space). Simpler techniques have also been developed that approximate dependency-directed backtracking. *Graph-based backjumping* reduces the amount of book-keeping required by full-blown backjumping by assuming that any two variables directly connected by a constraint may have been assigned conflicting values [Dechter 89a][2]. *N-th order deep and shallow learning* reduce the constraint recording complexity of dependency-directed backtracking by only recording conflicts involving N or fewer variables [Dechter 89a].

Graph-based backjumping works best on CSPs with sparse constraint graphs [Dechter 89a]. Instead, job shop scheduling problems have highly interconnected constraint graphs. Furthermore graph-based backjumping does not increase search efficiency when used in combination with forward checking [Haralick 80] mechanisms or stronger consistency enforcing mechanisms such as those entailed by job shop scheduling problems [Sadeh 91]. Our

[2]Two variables are said to be "*connected*" by a constraint if they both participate in that constraint.

experiments suggest that N-th order deep and shallow learning techniques often fail to improve search efficiency when applied to job shop scheduling problems. This is because these techniques use constraint size as the only criterion to decide whether or not to record earlier failures. When they limit themselves to small-size conflicts, they fail to record some important constraints. When they do not, their complexities become prohibitive.

Instead, this paper presents three look-back techniques that have yielded good results on job shop scheduling problems:

1. *Dynamic Consistency Enforcement* (DCE): a selective dependency-directed scheme that dynamically focuses its effort on critical resource subproblems,
2. *Learning Ordering From Failure* (LOFF): an adaptive scheme that suggests new variable orderings based on earlier conflicts,
3. *Incomplete Backjumping Heuristic* (IBH) a scheme that gives up searching areas of the search space that require too much work.

Related work in scheduling includes that of Prosser and Burke who use N-th order shallow learning to solve one-machine scheduling problems [Burke 89], and that of Badie et al. whose system implements a variation of deep learning in which a minimum set is heuristically selected as the source of the conflict [Badie et al 90].

The remainder of this paper is organized as follows. Section 2 provides a more formal definition of the job shop CSP. Section 3 describes the backtrack search procedure considered in this study. Sections 4, 5 and 6 successively describe each of the three backtracking schemes developed in this study. Experimental results are presented in section 7. Section 8 summarizes the contributions of this paper.

2. The Job Shop Constraint Satisfaction Problem

The job shop scheduling problem requires scheduling a set of jobs $J = \{j_1, \dots, j_n\}$ on a set of resources $RES = \{R_1, \dots, R_m\}$. Each job j_l consists of a set of operations $O^l = \{O^l_1, \dots, O^l_{q_l}\}$ to be scheduled according to a process routing that specifies a partial ordering among these operations (e.g. O^l_i BEFORE O^l_j).

In the job shop CSP studied in this paper, each job j_l has a release date rd_l and a due-date dd_l between which all its operations have to be performed. Each operation O^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 operation is initially constrained by the release and due dates of the job to which the operation belongs. If necessary, the model allows for additional unary constraints that further restrict the set of admissible start times of each operation, thereby defining one or several time windows within which an operation has to be carried out (e.g. one or several shifts in factory scheduling). In order to be successfully executed, each operation O^l_i requires p^l_i different resources (e.g. a milling machine and a machinist) R^l_{ij} $(1 \leq j \leq p^l_i)$, for each of which there may be a pool of physical resources from which to choose, $\Omega^l_{ij} \subseteq RES$ (e.g. one or several milling machines).

More formally, the problem can be defined as follows:

VARIABLES:

A vector of variables is associated with each operation, O^l_i ($1 \leq l \leq n,\ 1 \leq i \leq q_l$]), which consists of:

1. the *start time*, st^l_i of the operation, and

2. its *resource requirements*, R^l_{ij}, ($1 \leq j \leq p^l_i$).

CONSTRAINTS:

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

1. *Precedence constraints* defined by the process routings translate into linear inequalities of the type: $st^l_i + du^l_i \leq st^l_j$ (i.e. O^l_i BEFORE O^l_j);

2. *Capacity constraints* that restrict the use of each resource to only one operation at a time translate into disjunctive constraints of the form: $(\forall p\, \forall q\ R^k_{ip} \neq R^l_{jq}) \vee st^k_i + du^k_i \leq st^l_j \vee st^l_j + du^l_j \leq st^k_i$. These constraints simply express that, unless they use different resources, two operations O^k_i and O^l_j cannot overlap[3].

Additionally, our model can accommodate unary constraints that restrict the set of possible values of individual variables. These constraints include non-relaxable due dates and release dates, between which all operations in a job need to be performed. More generally, the model can accommodate any type of unary constraint that further restricts the set of possible start times of an operation.

Time is assumed discrete, i.e. operation start times and end times can only take integer values and each resource requirement R^l_{ij} has to be selected from a set of resource alternatives, $\Omega^l_{ij} \subseteq RES$.

OBJECTIVE:

In the job shop CSP studied in this paper, the objective is to come up with a feasible solution as fast as possible. Notice that this objective is different from simply minimizing the number of search states visited. It also accounts for the time spent by the system deciding which search state to explore next.

EXAMPLE:

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

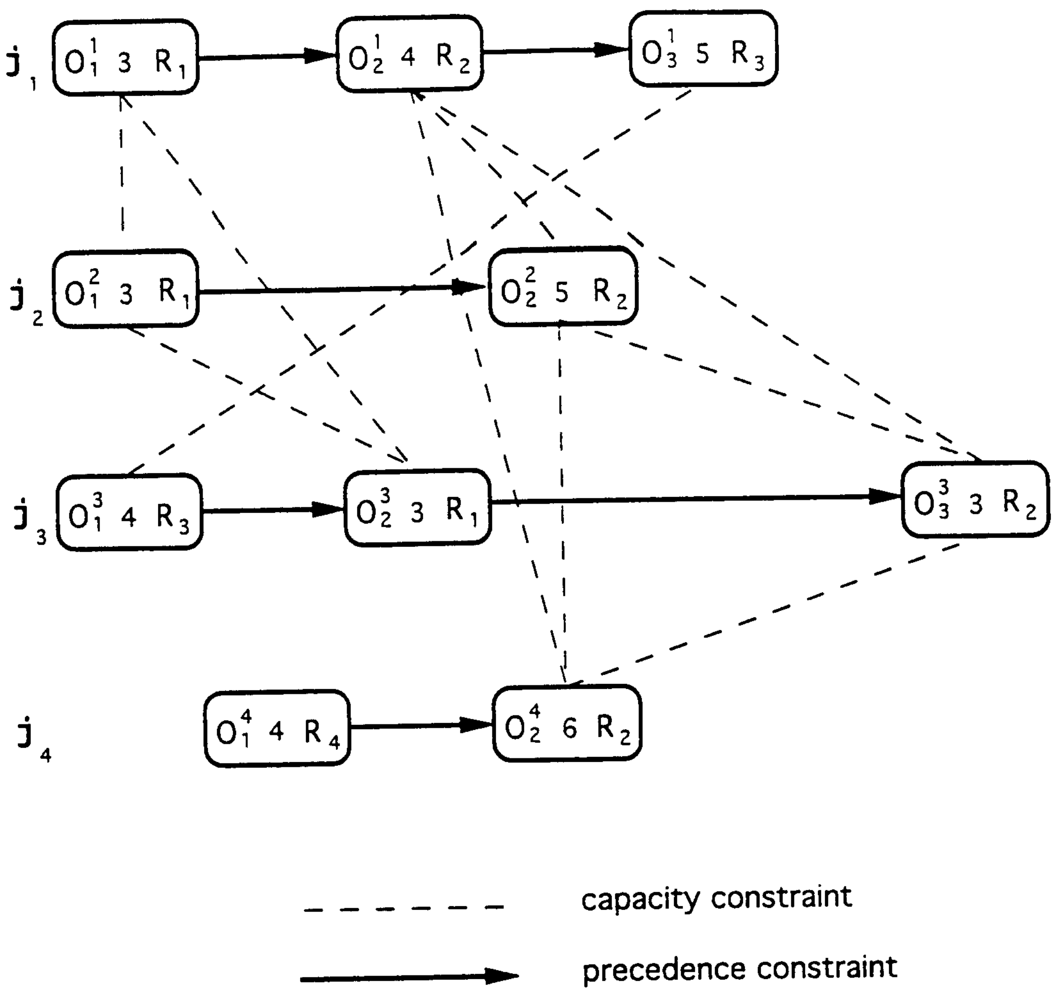


Figure 1: A simple problem with 4 jobs. Each node is labeled by the operation that it represents, its duration and the resource it requires.

Figure 1 depicts a simple job shop scheduling problem with four jobs $J=\{j_1,j_2,j_3,j_4\}$ and four resources $RES=\{R_1,R_2,R_3,R_4\}$. In this example, each operation has a single resource requirement with a single possible value. It is further assumed that all jobs are released at time 0 and have to be completed by time 20. Please note that none of these simplifying assumptions is required by the techniques to be discussed: jobs can have different release and due dates, operations can have several resource requirements, and several alternatives for each of these requirements. Note also that the problem we have just defined is infeasible. None of the operations on resource R_2 can start before time 3 and the sum of durations of these operations is 18. Hence, it is impossible to complete these operations before time 21. As we will see, this observation can easily be operationalized in the form of a simple consistency checking rule. However, as the number of operations to schedule grows, the exponential complexity of applying this simple rule to all possible subsets of operations on a given resource quickly becomes prohibitive, hence the need to be more selective in applying such checks. Additionally, passing such a check is no guarantee that a problem is feasible, hence the need to also rely on more complex mechanisms, as described below.

3. The Search Procedure

A depth-first backtrack search procedure is considered, in which search is interleaved with the application of consistency enforcing mechanisms and variable/value ordering heuristics that attempt to steer clear of deadend states[4] Specifically, search starts in a state where all operations still have to be scheduled. The BASIC-DEPTH-FIRST procedure proceeds by incrementally scheduling operations one by one. Each time an operation is scheduled, a new search state is created in which a consistency enforcing procedure (or constraint propagation procedure) is first applied to update the set of possible reservations of unscheduled operations. Next, an operation is selected to be scheduled and a reservation is selected for that operation. The procedure goes on, recursively calling itself, until either all operations are successfully scheduled or an inconsistency (or conflict) is detected. In the latter case, the procedure needs to undo earlier decisions or backtrack. The simplest possible backtracking mechanism for such a procedure is a "chronological" procedure that systematically goes back to the most recently scheduled operation and tries alternative reservations for that operation. If no alternative reservation is left, the procedure goes back to the next most recently scheduled operation and so on. If the procedure returns to the initial search state (i.e. the state with an empty schedule), the problem is infeasible.

The default consistency enforcing mechanisms and variable/value ordering heuristics used in our study are the ones described in [Sadeh 91]. These mechanisms, which have been favorably compared against a number of other heuristics [Sadeh 91, Sadeh 92], are briefly described below.

Consistency Enforcing Procedure: The consistency enforcing procedure we use combines three consistency mechanisms:

1. *Consistency with respect to precedence constraints*: Consistency with respect to precedence constraints is maintained using a longest path procedure that incrementally updates, in each search state, a pair of earliest/latest possible start times for each unscheduled operation. Essentially, as in PERT/CPM [Johnson 74], earliest start time constraints are propagated downstream within the job whereas latest start time constraints are propagated upstream (Figure 2). The complexity of this simple propagation mechanism is linear in the number of precedence constraints. In the absence of capacity constraints, the procedure can be shown to guarantee decomposability [Dechter 89b], i.e. it is sufficient to guarantee backtrack-free search [Sadeh 91].
2. *Forward consistency checks with respect to capacity constraints*: Enforcing consistency with respect to capacity constraints is more difficult due to the

[4]Additional details on this procedure, including pseudo-code, can be found in [Sadeh 94a].

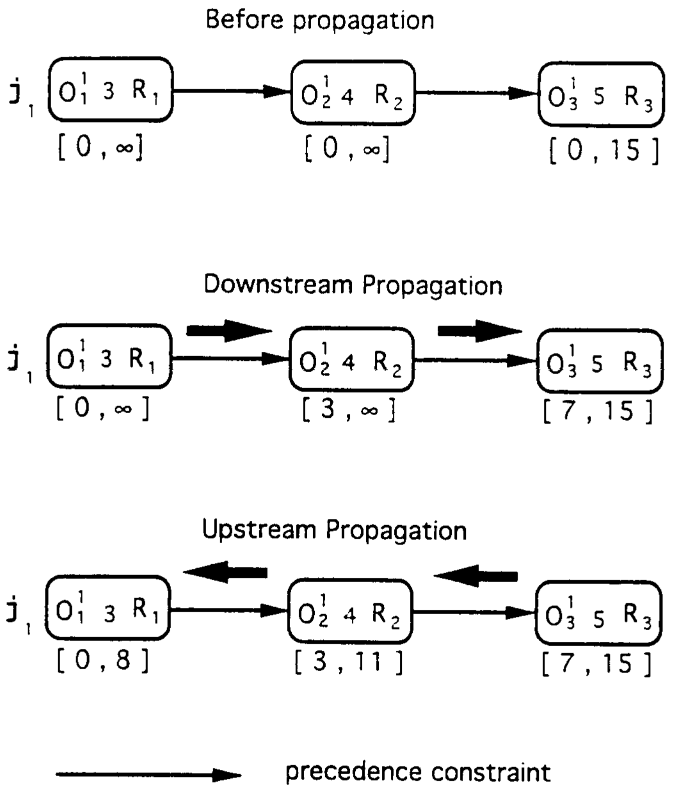


Figure 2: Consistency with respect to precedence constraints.

disjunctive nature of these constraints. Whenever a resource is allocated to an operation over some time interval, a "forward checking" mechanism [Haralick 80]checks the set of remaining possible reservations of other operations requiring that same resource, and removes those reservations that would conflict with the new assignment, as first proposed in [LePape 87] (See Figure 3).

3. *Additional consistency checks with respect to capacity constraints*: Additionally, our default consistency enforcing mechanism checks that no two unscheduled operations require overlapping resource/time intervals. An example of such a situation is illustrated in Figure 4, where two operations requiring the same resource, O_i^k and O_j^l, rely on the availability of overlapping time intervals, namely the intervals between their respective latest start times and earliest finish times ($[lst_i^k, eft_i^k]$ and $[lst_j^l, eft_j^l]$). This additional consistency mechanism has been shown to often increase search efficiency, while only resulting in minor computational overheads [Sadeh 91].

Variable/Value Ordering Heuristics: The default variable/value ordering heuristics used by our search procedure are the *Operation Resource Reliance* (ORR) variable ordering heuristic

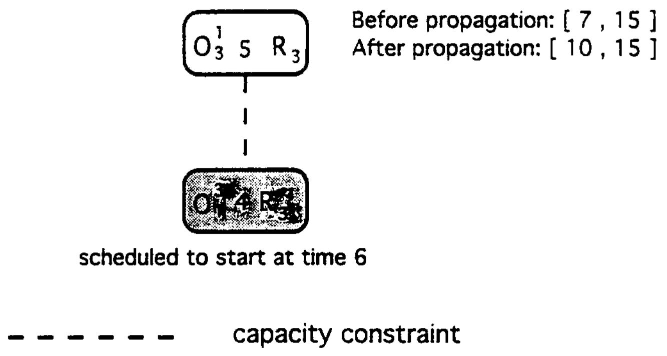
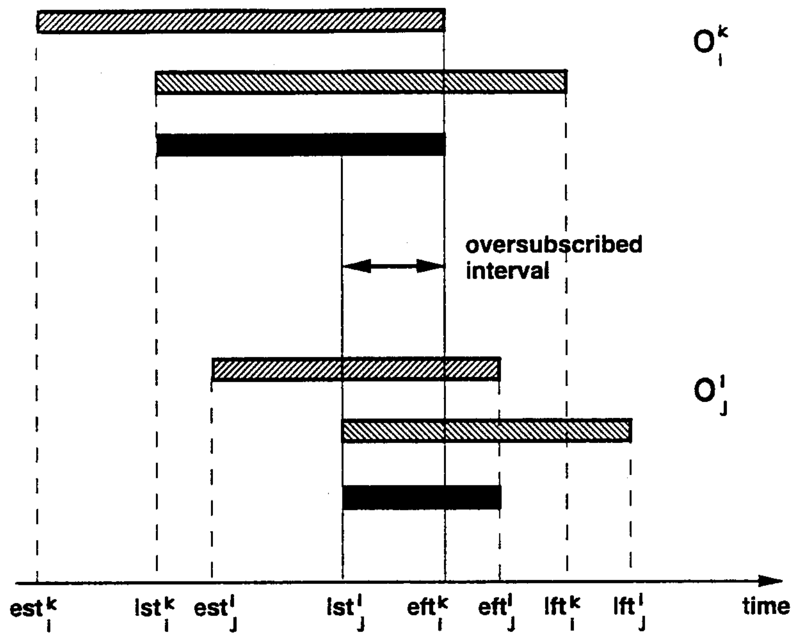

O_3^1 5 R_3
Before propagation: [7 , 15]
After propagation: [10 , 15]
scheduled to start at time 6
capacity constraint

Figure 3: Forward consistency checks with respect to capacity constraints.



▨	earliest possible reservation
▧	latest possible reservation
■	time interval absolutely required by the operation, whatever its start time

Figure 4: Detecting situations where two unscheduled operations requiring the same resource are in conflict.

and *Filtered Survivable Schedules* value ordering heuristic described in [Sadeh 91]. The ORR variable ordering heuristic aims at reducing backtracking by first scheduling difficult operations, namely operations whose resource requirements are expected to conflict with those of other operations. The FSS value ordering heuristic is a least constraining value ordering heuristic. It attempts to further reduce backtracking by selecting reservations that are expected to be compatible with a large number of schedules.

These default heuristics have been reported to outperform several other schemes described in the literature, both generic CSP heuristics and specialized heuristics designed for similar scheduling problems [Sadeh 91, Sadeh 92]. They seem to provide a good compromise between the efforts spent enforcing consistency, ordering variables, or ranking assignments for a variable and the actual savings obtained in search time. Nevertheless, the job shop CSP is NP-complete and, hence, these efficient procedures are not sufficient to guarantee backtrack-free search.

The remainder of this paper describes new backtracking schemes that help the system recover from deadend states. We show that, when the default consistency enforcing mechanisms and/or variable ordering heuristics are not sufficient to steer clear of deadends, look-back mechanisms can be devised that modify these schemes so as to avoid repeating past mistakes (i.e.so as to avoid reaching similar deadend states).

4. Dynamic Consistency Enforcement (DCE)

Backtracking is generally an indication that the default consistency enforcing scheme and/or variable/value ordering heuristics used by the search procedure are insufficient to deal with the subproblems at hand. Consequently, if search keeps on relying on the same default mechanisms after reaching a deadend state, it is likely to start thrashing. Experiments reported in [Sadeh 91, Sadeh 92], in which search always used the same set of consistency enforcing procedures and variable/value ordering heuristics, clearly illustrated this phenomenon. Search in these experiments exhibited a dual behavior. The vast majority of the problems fell in either of two categories: a category of problems that were solved with no backtracking whatsoever (by far the largest category) and a category of problems that caused the search procedure to thrash.

Theoretically, thrashing could be eliminated by enforcing full consistency in each search state. Clearly, such an approach is impractical as it would amount to performing a complete search. Instead, our approach involves (1) heuristically identifying one or a few small subproblems that are likely to be at the source of the conflict, (2) determining how far to backtrack by enforcing full consistency among the variables in these small subproblems, and (3) recording conflict information for possible reuse in future backtracking episodes. This approach is operationalized in the context of a backtracking scheme called *Dynamic Consistency Enforcement* (DCE). Given a deadend state and a history of earlier backtracking episodes within the same search space (i.e. while working on the same problem), this technique dynamically identifies small critical resource subproblems expected to be at the source of the current deadend. DCE then backtracks, undoing assignments in a chronological order, until a search state is reached, within which consistency has been fully restored in each critical resource subproblem (i.e. consistency with respect to capacity constraints in these subproblems). Experimental results reported in Section 7 suggest that often, by *selectively* checking for consistency in small resource subproblems, DCE can quickly recover from deadends. The remainder of this section further describes the mechanics of this heuristic.

4.1. Identifying Critical Resource Subproblems

The critical resource subproblems used by DCE consist of groups of operations participating in the current conflict along with groups of critical operations identified during earlier backtracking episodes involving the same resources. Below, we refer to the group of (unscheduled) operations identified by the default consistency enforcing mechanism as having no possible reservations left as the *Partial Conflicting Set* of operations (PCS). In order to restore consistency, the search procedure needs to at least go back to a search state in which each PCS operation has one or more possible reservations[5]. DCE attempts to identify such additional operations by maintaining a group of critical resource subproblems identified during earlier backtracking episodes. Below, we refer to this data structure as the *Former Dangerous Groups* of operations (FDG). Details on how this data structure is created and maintained are provided in Subsection 4.3.

For each capacity constraint violation among operations in the PCS, DCE checks the FDG data structure and retrieves all related resource subproblems. A resource subproblem in the FDG is considered related to a capacity constraint violation in the PCS if, in an earlier backtracking episode, operations in that resource subproblem were involved in a capacity constraint violation on the same resource and over a "close" time interval. A system parameter is used to determine if two resource conflicts are "close". In the experiments reported at the end of this paper, two conflicts were considered close if the distance separating them was not greater than twice the average operation duration. Related critical subproblems identified by inspecting the FDG data structure are then merged with corresponding operations in the PCS to form a new set of one or more critical resource subproblems, which we refer to as the as the *Dangerous Group* of operations (DG) for the conflict at hand. Like the FDG, the DG is organized in subgroups of resource subproblems consisting of operations contending for the same resource over close or overlapping time intervals. While backtracking, operations that are unscheduled are inserted in the DG, either by being added to existing resource subproblems or by creating new resource subproblems.

[5]Clearly, this is not guaranteed to be sufficient, as other operations may also contribute to the conflict.

4.2. Backtracking While Selectively Enforcing Consistency

Once the initial DG has been identified, DCE backtracks, undoing assignments in a chronological order, until it reaches a search state in which consistency is restored within each of the resource subproblems defined by operations in the DG[6]. This is done by enforcing full consistency with respect to capacity constraints in each of the resource subproblems in the DG. As long as conflicts are detected, the procedure continues to backtrack and unscheduled operations are inserted into existing or new resource subproblems in the DG. While restoring consistency within each of these resource subproblems is a necessary condition to backtrack to a consistent search state, it is not always a sufficient one. In other words, the effectiveness of DCE critically depends on its ability to heuristically focus on the right resource subproblems[7].

Because full consistency checking can be expensive on large subproblems, if a resource subproblem in the DG becomes too large, k-consistency is enforced instead of full-consistency, where k is a parameter of the system [Freuder 82]. In the experiments reported at the end of this paper, k was set to 4. At the end of a backtracking episode, the DG has maximum size, call it DG_{max}. Assuming that the procedure was able to backtrack to a consistent search state, DG_{max} is expected to contain all the operations at the origin of the deadend[8] and often more. DG_{max} is then saved for later use in the FDG data structure. Additional details regarding the management of this data structure are provided in the next subsection. If a related backtracking episode is later encountered by the system, DG_{max} can be retrieved and combined with the PCS of this new episode.

4.3. Storing Information About Past Backtracking Episodes

The purpose of the *Former Dangerous Groups* of operations (FDG) maintained by the system is to help determine more efficiently and more precisely the scope of each deadend by focusing on critical resource subproblems. Each group of operations in the FDG consists of operations that are in high contention for the allocation of a same resource. Accordingly, whenever, a conflict is detected that involves some of the operations in one group, the backtracking procedure checks for consistency among *all* operations in that group.

The groups of operations in the FDG are built from the *Dangerous Groups* (DGs) obtained at the end of previous backtracking episodes (DG_{max}). Indeed, whenever a backtracking episode is completed, DG_{max} is expected to contain all the conflicting operations at the origin of this episode. Generally, DG_{max} may involve one or several resource subproblems (i.e. groups of

[6]Additional details on this procedure, including pseud-code can be found in [Sadeh 94a]

[7]Note that DCE is not expected to be very effective at recovering from complex conflicts involving interactions between multiple resource subproblems. A heuristic which is often more effective for these complex conflicts is described in Section 6.

[8]Clearly, while this is not guaranteed, experimental results suggest that this is often the case.

operations requiring the same resource). Each one of these subproblems is merged with *related* subproblems currently stored in the FDG. If there is no related group in FDG, the new group is separately added to the data structure. Finally, as operations are scheduled, they are removed from the FDG.

4.4. An Example

Figure 5 illustrates the behavior of DCE on the small scheduling problem introduced in Figure 1. After scheduling operations O_2^4 and O_2^2 on resource R_2, the procedure detects that operation O_3^3 has no possible reservations left. Given that the FDG data structure is initially empty (no prior backtracking episode), we have $PCS = DG = \{ O_3^3 \}$. The procedure unschedules the most recently scheduled operation, namely O_2^2, and inserts it in DG together with operation O_3^3, as both of these operations require the same resource. At this point, DCE enforces full consistency with respect to capacity constraints between these two operations[9] and finds that, after consistency checking, the operations still admit some possible reservations. This marks the end of the first backtracking episode. The procedure saves the current DG in FDG, for possible reuse, then schedules operation O_2^2 at its next best available start time[10], namely start time 6. In the process, O_2^2 is removed from the FDG. Another conflict is detected in this new search state, which marks the beginning of a second backtracking episode. This time the consistency enforcing procedure finds that operation O_1^2 has no possible reservations left (i.e. $PCS = \{ O_2^1 \}$). Using the FDG, the system adds operation O_3^3 to the group of dangerous operations, $DG = \{ O_2^1, O_3^3 \}$. Accordingly, this time, when it unschedules operation O_2^2, DCE enforces full consistency[11] with respect to capacity constraints in $DG = \{ O_2^1, O_2^2, O_3^3 \}$. When it finds that the current search state is still inconsistent, DCE proceeds and unschedules operation O_2^4, thereby returning to the root search state with $DG = \{ O_2^1, O_2^2, O_3^3, O_2^4 \}$. In this search state, full consistency with respect to capacity constraints between operations in DG indicates that the problem is infeasible. In total, the system only generates three search states to find that the problem is infeasible. In contrast, a total of 50 search states is required for the same small problem, when relying on a simple chronological backtracking procedure. The example also shows how the use of the Formerly Dangerous Groups (FDG) of operations helps the system identify critical resource subproblems. If it was not for this mechanism, the procedure would not detect an inconsistency when it comes back to Depth 1 in the second backtracking episode, as it would only check for consistency between O_2^1 and O_2^2. More generally, experimental results presented in Section 7 show that DCE often results in important increases in search efficiency and important reductions in computation time.

---

[9]This is equivalent to 2-consistency or arc-consistency, given that there are only 2 operations [Freuder 82].

[10]Actually, start time 6 is not the start time picked by our reservation ordering heuristic. The system was manually forced to pick this value to make the example more interesting.

[11]This time the system enforces 3-consistency, given that there are 3 operations in DG.

>> Depth: 0, Number of states visited: 0, $FDG=\emptyset$
O_2^4 is scheduled between 14 and 20 on R_2

>> Depth: 1, Number of states visited: 1, $FDG=\emptyset$
O_2^2 is scheduled between 9 and 14 on R_2

>> Depth: 2, Number of states visited: 2, $FDG=\emptyset$
Conflict detected: O_3^3 has no possible reservations left:
$PCS=DG=\{[O_3^3]\}$ [Beginning of first backtracking episode]
O_2^2 is unscheduled

>> Depth: 1, Number of states visited: 2, $FDG=\emptyset$
$DG=\{[O_3^3,O_2^2]\}$
Full consistency checking with respect to capacity constraints in DG:
Remaining possible start times:
O_2^2: $\{3,4,5,6\}$
O_3^3: $\{8,9,10,11\}$
$FDG=\{[O_2^2,O_3^3]\}$ [End of first backtracking episode]
O_2^2 is scheduled between 6 and 11 on R_2

>> Depth: 2, Number of states visited: 3, $FDG=\{[O_3^3]\}$
Conflict detected: O_2^1 has no possible reservations left:
$PCS=\{[O_2^1]\}$, $DG=\{[O_2^1,O_3^3]\}$ [Beginning of second backtracking episode]
O_2^2 is unscheduled

>> Depth: 1, Number of states visited: 3, $FDG=\{[O_3^3]\}$
$DG=\{[O_2^1,O_2^2,O_3^3]\}$
Full consistency checking with respect to capacity constraints in DG:
Conflict detected
O_2^4 is unscheduled

>> Depth: 0, Number of states visited: 3, $FDG=\{[O_3^3]\}$
$DG=\{[O_2^1,O_2^2,O_3^3,O_2^4]\}$
Full consistency checking with respect to capacity constraints in DG:
Conflict detected
Infeasible Problem [End of second backtracking episode]

Figure 5: An edited trace illustrating the DCE procedure.

4.5. Additional "Watch Dog" Consistency Checks

Because groups of operations in the FDG are likely deadend candidates, our system further performs simple "watch dog" checks on these dynamic groups of operations.

More specifically, for each group G of operations in FDG, the system performs a rough check to see if the resource can still accommodate all the operations in the group. This is done using redundant constraints of the form:

$$Max(lst_i^l + du_i^l, O_i^l \in G) - Min(est_i^l, O_i^l \in G) \geq \sum_{O_i^l \in G} du_i^l$$

where est_i^l and lst_i^l are respectively the earliest and latest possible start times of O_i^l in the current search state.

Whenever such a constraint is violated, an inconsistency has been detected. Though very simple and inexpensive, these checks enable to catch inconsistencies involving large groups of operations that would not be immediately detected by the default consistency mechanisms. Clearly, some inconsistencies can still escape these rough checks.

While backtracking, the same "watch dog" checks can be used prior to enforcing full consistency with respect to capacity constraints in the critical resource subproblems in DG. This can significantly reduce computation time. For instance, in the second backtracking episode in Figure 5, these simple checks are sufficient to detect inconsistencies at depth 1 and 0. For example, at depth 1, where $DG = \{ [O_2^1, O_2^2, O_3^3] \}$,

$$Max(lst_i^l + du_i^l, O_i^l \in DG) - Min(est_i^l, O_i^l \in DG) = 14 - 3 = 11,$$

$$\text{while} \sum_{O_i^l \in DG} du_i^l = 12 .$$

5. Learning Ordering From Failures (LOFF)

Often, reaching a deadend state is also an indication that the default variable ordering was not adequate for dealing with the subproblem at hand. Typically, the operations participating in the deadend turn out to be more difficult to schedule than the ones selected by the default variable ordering heuristic. In other words, it is often a good idea to first schedule the operations participating in the conflict that was just resolved. *Learning Ordering From Failure* (LOFF) is an adaptive procedure that overrides the default variable ordering in the presence of conflicts.

After recovering from a deadend, namely after backtracking all the way to an apparently consistent search state, LOFF uses the Partial Conflicting Set (PCS) of the deadend to reorganize the order in which operations will be rescheduled and make sure that operations in the PCS are scheduled first. This is done using a quasi-stack, QS, on which operations in the PCS are pushed in descending order of domain size, i.e. PCS operations with a large number of remaining reservations are pushed first on the quasi-stack. When the quasi-stack is empty, the procedure uses its default variable ordering heuristic, as described in Section 3. However, when QS contains some operations, the procedure first schedules these operations, starting with the ones on top of the quasi-stack, namely those QS operations with the smallest number of remaining reservations.

If a candidate operation is already in QS, i.e. if it is encountered for a second time, it is pushed again on QS as if it had a smaller domain. This orders operations based on the recency of the conflict in which they were last involved and based on their number of remaining reservations.

6. An Incomplete Backjumping Heuristic

Traditional backtrack search procedures only undo decisions that have been proven to be inconsistent. Proving that an assignment is inconsistent with others can be very expensive, especially when dealing with large conflicts. Graph-based backjumping and N-th order shallow/deep learning attempt to reduce the complexity of full-blown dependency-directed backtracking by either simplifying the process of identifying inconsistent decisions (e.g. based on the topology of the constraint graph) or restricting the size of the conflicts that can be detected. The Dynamic Consistency Enforcement (DCE) procedure described in Section 6 also aims at reducing the complexity of identifying the source of a conflict by dynamically focusing its effort on small critical subproblems. Because these techniques focus on smaller conflicts, they all have problems dealing with more complex conflicts involving a large number of variables[12]. It might in fact turn out that the only effective way to deal with more complex conflicts is by using heuristics that undo decisions not because they have been proven inconsistent but simply because they appear overly restrictive. This is the approach taken in the heuristic described in this section. Clearly, the resulting search procedure is no longer complete and may fail to find solutions to feasible problems, hence the name of *Incomplete Backjumping Heuristic* (IBH).

Texture measures such as the ones described in [Fox 89] could be used to estimate the tightness of different search states, for instance, by estimating the number of global solutions compatible with each search state. Clearly, a search state whose partial solution is compatible with a large number of global solutions is loosely constrained, whereas one compatible with a small number of global solutions is tightly constrained. Assignments leading to much tighter search states would be prime candidates to be undone when a complex conflict is suspected. The *Incomplete Backjumping Heuristic* (IBH) used in this study is simpler and, yet, often seems to be sufficient. Whenever the system starts thrashing, this heuristic backjumps all the way to the first search state and simply tries the next best value (i.e. reservation) for the critical operation in that state (i.e. the first operation selected by the variable ordering heuristic). IBH considers that the search procedure is thrashing, and hence that it is facing a complex conflict, when more than θ assignments had to be undone since the last time the system was thrashing or since the procedure began, if no thrashing occurred earlier. θ is a parameter of the procedure.

[12]Clearly, there are some conflicts involving large numbers of variables that are easy to catch, as illustrated by the watch dog checks described in Section 4.

7. Empirical Evaluation

This section reports the results of empirical studies conducted to assess the performance of the look-back schemes presented in this paper. The first study reports performance on a suite of 60 benchmark problems introduced in [Sadeh 91]. This is followed by a more detailed study comparing the performance of the first two look-back schemes introduced in this paper (DCE&LOFF) against that of second-order deep learning [Dechter 89a] and chronological backtracking. Finally, we compare the performance of the complete search procedure relying on DCE&LOFF with that of an incomplete procedure combining all three of the look-back schemes presented in this paper (DCE&LOFF&IBH).

7.1. Performance Evaluation On a First Suite of Problems

A first set of experiments was run on a testsuite of 60 job shop scheduling problems first introduced in [Sadeh 91]. In the experiments reported in [Sadeh 91], the default variable and value ordering heuristics used in our study (i.e. the ORR and FSS heuristics described in Section 3) were shown to outperform a variety of other variable/value ordering combinations, though they still failed to solve 8 out of the 60 problems. In contrast, the results presented below indicate that the combination of our three look-back techniques (DCE&LOFF&IBH) can efficiently solve all 60 problems in the testsuite.

Specifically, the testsuite consists of 6 groups of 10 problems each. Each problem requires scheduling 10 jobs on 5 resources and involves a total of 50 operations (5 operations per job). Each job has a linear process routing specifying a sequence in which it has to visit each one of the five resources. This sequence varies from one job to another, except for a predetermined number of bottleneck resources (one or two in these experiments) which are always visited after the same number of steps. The six groups of problems were obtained by varying two parameters:

1. the number of apriori bottlenecks (BTNK): one (BTNK=1) or two (BTNK=2), and
2. the spread (SP) of the release and due dates between which each job has to be scheduled: wide (SP=W), narrow (SP=N), or null (SP = 0).

The SP parameter and the operation durations have been adjusted so that bottleneck utilization remains close to 100% over most of the span of each problem. In these problems, each operation had slightly over 100 possible start times (i.e. values) after application of the consistency enforcing techniques in the initial search state. Additional details on these problems can be found in [Sadeh 91][13].

Table 1 compares the performance of the following two procedures:

1. a basic depth-first procedure relying on chronological backtracking and on the

[13]The problems are also accessible via anonymous ftp to cimds3.cimds.ri.cmu.edu, where they can be found in /usr/sadeh/public/csp_test_suite. A README file details the content of the various files in the directory.

default consistency enforcing techniques and variable/value ordering heuristics described in Section 3. This is also the procedure reported to perform best in [Sadeh 91].

2. the same procedure enhanced with the DCE, LOFF and IBH look-back schemes presented in this paper.

For each of the 60 problems, search was stopped if it required more than 500 search states. Performance in each problem category is reported along three dimensions:

1. *Search efficiency*: the average ratio of the number of operations to be scheduled over the total number of search states that were explored. In the absence of backtracking, only one search state is generated for each operation, and hence search efficiency is equal to 1.

2. *Number of experiments* solved in less than 500 search states.

3. *CPU seconds*: this is the average CPU time required to solve a problem. When a solution could not be found, this time was approximated as the CPU time taken to explore 500 search states (this approximation was only used for Chronological Backtracking, since DCE&LOFF&IBH solved all problems). All CPU times were obtained on a DECstation 5000 running Knowledge Craft on top of Allegro Common Lisp. Experimentation with a variation of the system written in C indicates that the search procedure would run about 30 times faster if reimplemented in this language [Sadeh 94b].

The results indicate that DCE&LOFF&IBH consistently outperformed the chronological backtracking scheme in terms of CPU time, search efficiency and number of problems solved. On the easier problems (SP=W), both techniques solved all 20 problems in approximately the same amount of time. On the more difficult problems (SP=N and SP=0), DCE&LOFF&IBH clearly dominated chronological backtracking. In particular, on problems with SP=0 and BK=1, DCE&LOFF&IBH solved 40% more problems than the chronological backtracking scheme and, on average, proved to be 3.5 times faster. Overall, while chronological backtracking failed to solve 8 problems out of 60, DCE&LOFF&IBH efficiently solved all 60 problems, and, on average, was almost twice as fast as the procedure with chronological backtracking. Had we not stopped the chronological backtracking procedure after 500 search states, the speedup achieved by DCE&LOFF&IBH would be even more significant. In fact, based on a couple of problems for which the chronological procedure was allowed to expand a larger number of search states, it appears that problems that are not solved in 500 states often require thousands more to be solved (with chronological backtracking).

7.2. Further Evaluation

To further evaluate our look-back schemes, we picked the most difficult problem category in the testsuite, namely the category for which the default consistency enforcing procedure and variable/value ordering heuristics are least effective (SP=0) and generated an additional 80 scheduling problems, 40 with BTNK=1 and 40 with BTNK=2. The SP=0 problem category was also the most difficult one for all the other combinations of variable and value ordering heuristics

		Chronological	DCE&LOFF&IBH
SP=W BTNK=1	Search Efficiency	0.96	0.96
	Nb. exp. solved (out of 10)	10	10
	CPU seconds	88.5	90.5
SP=W BTNK=2	Search Efficiency	0.99	0.99
	Nb. exp. solved (out of 10)	10	10
	CPU seconds	93	95
SP=N BTNK=1	Search Efficiency	0.78	0.91
	Nb. exp. solved (out of 10)	8	10
	CPU seconds	331.5	106
SP=N BTNK=2	Search Efficiency	0.87	0.93
	Nb. exp. solved (out of 10)	9	10
	CPU seconds	184	119.5
SP=0 BTNK=1	Search Efficiency	0.73	0.88
	Nb. exp. solved (out of 10)	7	10
	CPU seconds	475	134.5
SP=0 BTNK=2	Search Efficiency	0.82	0.84
	Nb. exp. solved (out of 10)	8	10
	CPU seconds	300.5	226.5
Overall Performance	Search Efficiency	0.86	0.92
	Nb. exp. solved (out of 60)	52	60
	CPU seconds	245.5	128.7

Table 1: Comparison of Chronological Backtracking and DCE&LOFF&IBH on 6 sets of 10 job shop problems.

tested in the study reported in [Sadeh 91]. It corresponds to problems in which all jobs are released at a common date and need to be completed by a common due date. Among the resulting 80 problems, we only report performance on those problems for which the default schemes were not sufficient to guarantee backtrack-free search[14]. This leaves 16 scheduling problems with one bottleneck (SP=0 and BTNK=1), and 15 with two bottlenecks (SP=0 and

[14]Clearly, performance on problems that do not require backtracking is of no interest, since our backtracking schemes never get invoked, and hence CPU time remains unchanged.

BTNK=2).

Below, we successively report the results of two studies. The first one compares the performance of three complete backtracking schemes: chronological backtracking, 2nd-order deep learning, and the procedure combining the DCE and LOFF backtracking heuristics[15]. The second study compares the complete search procedure using DCE and LOFF with the incomplete search procedure combining DCE, LOFF and IBH.

Table 2: Results of One-Bottleneck Experiments.

Exp. No.	Chronological			DCE & LOFF			Deep Learning		
	No. of Nodes	CPU (sec)	Result	No. of Nodes	CPU (sec)	Result	No. of Nodes	CPU (sec	Result
1	500	1427	F	122	1232	S*	500	5756	F
2	500	1587	F	500	1272	F	500	5834	F
3	74	148	S	63	117	S	25	36000	F
4	69	152	S	52	120	S	69	391	S
5	500	1407	F	65	134	S	500	11762	F
6	500	1469	F	500	1486	F	500	8789	F
7	500	1555	F	59	130	S	500	9681	F
8	500	1705	F	41	145	S*	500	9560	F
9	53	108	S	53	102	S	53	122	S
10	500	1529	F	500	1536	F	500	9114	F
11	500	1460	F	85	1800	F	500	14611	F
12	500	1694	F	500	1131	F	500	21283	F
13	51	109	S	51	81	S	51	88	S
14	500	1762	F	63	138	S	500	18934	F
15	500	1798	F	69	142	S	500	9600	F
16	500	1584	F	500	1183	F	65	36000	F

S: Solved ; F: Failure; S*: Proved infeasible
Time Limit: 1800 sec (Except Deep Learning)
Node Limit: 500

The results of the first study comparing chronological backtracking, 2nd-order deep learning [Dechter 89a] and the DCE & LOFF procedures advocated in Section 4 and 5 are summarized in Table 2 and 3. The results reported here were obtained using a search limit of 500 nodes and a time limit of 1800 seconds (except for deep learning, for which the time limit was increased to 36,000 seconds[16]). All CPU times reported below were obtained on a DECstation 5000 running Knowledge Craft on top of Allegro Common Lisp. As already indicated above, comparison between C and Knowledge Craft implementations of similar variable and value ordering heuristics indicates that the code would run about 30 times faster in C [Sadeh 94b].

On the one-bottleneck problems. chronological backtracking solved only 4 problems out of 16 (See Table 2). Interestingly enough, deep learning showed no improvement over chronological backtracking either in the number of problems solved or in CPU time. As a matter of fact, deep

[15]Besides the experiments reported below, additional experiments were performed to assess the benefits of using DCE and LOFF separately. These experiments show that both techniques contribute to the improvements reported in this section.

[16]This was motivated by the fact that our implementation of deep learning may not be optimal.

learning was even too slow to find solutions to some of the problems solved by chronological backtracking. This is attributed to the fact that the constraints in job shop scheduling are more tightly interacting than those in the zebra problem, where the improvement of deep learning over chronological backtracking was originally ascertained [Dechter 89a]. On the other hand, DCE & LOFF solved 10 problems out of 16 (2 out of these 10 problems were successfully proven infeasible). As expected, by focusing on a small number of critical subproblems, DCE & LOFF is able to discover larger more useful conflicts than 2nd-order deep learning, while requiring only a fraction of the time. Another observation is that DCE & LOFF expanded fewer search states than chronological backtracking for the problems that chronological backtracking solved. However, each of the DCE & LOFF expansions took slightly more CPU time, due to the higher level of consistency enforcement.

Table 3: Results of Two-bottleneck Experiments

Exp. No.	Chronological			DCE & LOFF			Deep Learning		
	No. of Nodes	CPU (sec)	Result	No. of Nodes	CPU (sec)	Result	No. of Nodes	CPU (sec)	Result
1	500	1139	F	113	1800	F	18	36000	F
2	500	1444	F	425	1800	F	115	36000	F
3	84	175	S	109	202	S	84	811	S
4	56	123	S	56	112	S	56	213	S
5	51	101	S	51	113	S	13	36000	F
6	500	1531	F	321	1800	F	328	36000	F
7	500	1775	F	500	1357	F	500	2793	F
8	52	102	S	52	115	S	33	36000	F
9	500	1634	F	247	974	S	500	1519	F
10	500	1676	F	91	1800	F	26	36000	F
11	66	163	S	59	104	S	66	2240	S
12	56	139	S	58	104	S	58	281	S
13	54	129	S	52	91	S	54	28900	S
14	500	1676	F	346	1800	F	500	9031	F
15	500	1522	F	324	1800	F	296	36000	F

S: Solved ; F: Failure; S*: Proved infeasible
Time Limit : 1800 sec. (36000 sec. for Deep Learning)
Node Limit : 500

Results for the set of two-bottleneck problems are reported in Table 3. Similar results are observed here again: deep learning shows no improvement over chronological backtracking and seems significantly slower. The difference between chronological backtracking and DCE&LOFF is not as impressive as in the first set of experiments. As can be seen in Table 3, chronological backtracking solved 7 out of 15 problems, whereas DCE & LOFF solved 8. On the problems solved by both chronological backtracking and DCE & LOFF, DCE & LOFF turned out to be slightly faster overall. These less impressive results suggest that the presence of multiple bottlenecks often introduces more complex conflicts. Results presented in the following subsection suggest that in this case incomplete backtracking procedures such as the one entailed by the IBH heuristic are often much more effective.

7.3. Complete vs. Incomplete Search Procedures

Table 4: Results of One-bottleneck Experiments.

Exp. No.	DCE & LOFF			DCE & LOFF & IBH		
	No. of Nodes	CPU (sec)	Result	No. of Nodes	CPU (sec)	Result
1	122	1232	S*	350	1800	F
2	500	1272	F	203	1124	S
3	63	117	S	63	123	S
4	52	120	S	52	116	S
5	65	134	S	65	144	S
6	500	1486	F	127	424	S
7	59	130	S	59	125	S
8	41	145	S*	457	1800	F
9	53	108	S	53	100	S
10	500	1536	F	67	170	S
11	85	1800	F	74	170	S
12	500	1131	F	164	616	S
13	51	81	S	51	92	S
14	63	138	S	63	149	S
15	69	142	S	69	158	S
16	500	1183	F	156	524	S

S: Solved ; F: Failure; S*: Proved infeasible
Time Limit: 1800 sec. Node Limit: 500

Table 5: Results of Two-bottleneck Experiments

Exp. No.	DCE & LOFF			DCE & LOFF & IBH		
	No. of Nodes	CPU (sec)	Result	No. of Nodes	CPU (sec)	Result
1	113	1800	F	151	456	S
2	425	1800	F	371	1780	S
3	109	202	S	95	210	S
4	56	112	S	56	108	S
5	51	113	S	51	97	S
6	321	1800	F	420	1800	F
7	500	1357	F	159	534	S
8	52	115	S	52	96	S
9	247	974	S	423	1705	S
10	91	1800	F	440	1800	F
11	59	104	S	59	113	S
12	58	104	S	58	112	S
13	52	91	S	52	102	S
14	346	1800	F	239	512	S
15	324	1800	F	73	195	S

S: Solved ; F: Failure; S*: Proved infeasible
Time Limit: 1800 sec. Node Limit: 500

Table 4 and 5 compare the performance of the complete search procedure based on DCE & LOFF against that of an incomplete search procedure using DCE & LOFF in combination with the IBH heuristic described in Section 6. While DCE & LOFF could only solve 10 out of 16 one-bottleneck problems and 8 out 15 two-bottleneck problems, DCE & LOFF combined with IBH solved 14 one-bottleneck problems and 13 two-bottleneck problems. The only one-bottleneck problems that were not solved by DCE & LOFF & IBH are the two problems identified as infeasible by the complete procedure with DCE & LOFF (see Table 2). This is hardly a surprise. While the addition of IBH to DCE & LOFF enables the search procedure to solve a larger number of problems, it also makes the procedure incomplete (i.e. infeasible problems can no longer be identified). Additional experiments combining IBH with a simple chronological backtracking scheme produced results that were not as good as those obtained by DCE & LOFF & IBH, indicating that both IBH and DCE & LOFF contribute to the performance improvement observed in Table 4 and 5.

Results on two-bottleneck problems (See Table 5) also suggest that the impact of IBH is particularly effective on these problems. This is attributed to the fact that two-bottleneck problems give rise to more complex conflicts. Identifying the assignments participating in these more complex conflicts might simply be too difficult for any exact backtracking scheme. Instead, because it can undo assignments that are not provably wrong but simply appear overly restrictive, IBH seems more effective at solving these problems.

8. Concluding Remarks

We have presented three look-back techniques for the job shop scheduling CSP:

1. *Dynamic Consistency Enforcement* (DCE), a heuristic that dynamically focuses on restoring consistency within small critical subproblems,
2. *Learning Ordering From Failure* (LOFF), a technique that modifies the order in which variables are instantiated based on earlier conflicts, and
3. *Incomplete Backjumping Heuristic* (IBH) which, when thrashing occurs, can undo assignments that are not provably inconsistent but appear overly restrictive.

The significance of this research is twofold:

1. Job shop scheduling problems with non-relaxable time windows have multiple applications (e.g. manufacturing, space, transportation, health care, etc.). We have shown that our look-back heuristics combined with powerful techniques that we had previously developed (1) further reduce the average complexity of backtrack search, and (2) enable this search procedure to efficiently solve problems that could not be solved otherwise due to excessive computational requirements. While the results reported in this study were obtained on problems that require finding a feasible schedule, the backtracking schemes presented in this paper can also be used on optimization versions of the scheduling problem, such as the Just-In-Time job shop scheduling problems described in [Sadeh 94b].
2. This research also points to shortcomings of dependency-directed backtracking schemes advocated earlier in the literature. In particular, comparison with 2nd-order deep learning indicates that this technique failed to improve performance on our set of job shop scheduling problems. More generally, N-th order deep and shallow learning techniques often appear inadequate when applied to job shop scheduling problems because they rely solely on constraint size to decide whether or not to record earlier failures. When these techniques limit themselves to small-size conflicts, they often fail to record some important constraints; when they consider larger conflicts, their computational complexity becomes prohibitive. A more general weakness of traditional backtracking schemes has to do with the fact that they never undo assignments unless they can be proven to be at the source of the conflict. When dealing with large complex conflicts, proving that a particular

assignment should be undone can be very expensive. Instead, our experiments suggest that, when thrashing cannot easily be avoided, it is often a better idea to use incomplete backjumping heuristics that undo decisions simply because they *appear* overly restrictive.

References

[Badie et al 90] C. Badie and G. Bel and E. Bensana and G. Verfaillie.
Operations Research and Artificial Intelligence Cooperation to solve Scheduling Problems.
In *First International Conference on Expert Planning Systems.* 1990.

[Bitner 75] J.R. Bitner and E.M. Reingold.
Backtrack Programming Techniques.
Communications of the ACM 18(11):651-655, 1975.

[Burke 89] Peter Burke and Patrick Prosser.
A Distributed Asynchronous System for Predictive and Reactive Scheduling.
Technical Report AISL-42, Department of Computer Science, University of Strathclyde, 26 Richmond Street, Glasgow, GI IXH, United Kingdom, October, 1989.

[Dechter 88] Rina Dechter and Judea Pearl.
Network-Based Heuristics for Constraint Satisfaction Problems.
Artificial Intelligence 34(1):1-38, 1988.

[Dechter 89a] Rina Dechter.
Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition.
Artificial Intelligence 41:273-312, 1989.

[Dechter 89b] Rina Dechter and Itay Meiri.
Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 271-277. 1989.

[Doyle 79] John Doyle.
A Truth Maintenance System.
Artificial Intelligence 12(3):231-272, 1979.

[Fox 89] Mark S. Fox and Norman Sadeh and Can Baykan.
Constrained Heuristic Search.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 309-315. 1989.

[Freuder 82] E.C. Freuder.
A Sufficient Condition for Backtrack-free Search.
Journal of the ACM 29(1):24-32, 1982.

[Garey 79] M.R. Garey and D.S. Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
Freeman and Co., 1979.

[Gaschnig 79] John Gaschnig.
Performance Measurement and Analysis of Certain Search Algorithms.
Technical Report CMU-CS-79-124, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, 1979.

[Golomb 65] Solomon W. Golomb and Leonard D. Baumert.
Backtrack Programming.
Journal of the Association for Computing Machinery 12(4):516-524, 1965.

[Haralick 80] Robert M. Haralick and Gordon L. Elliott.
Increasing Tree Search Efficiency for Constraint Satisfaction Problems.
Artificial Intelligence 14(3):263-313, 1980.

[Johnson 74] L.A. Johnson and D.C. Montgomery.
Operations Research in Production Planning, Scheduling, and Inventory Control.
Wiley, 1974.

[LePape 87] Claude Le Pape and Stephen F. Smith.
Management of Temporal Constraints for Factory Scheduling.
Technical Report, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1987.
also appeared in Proc. Working Conference on Temporal Aspects in Information Systems, Sponsored by AFCET and IFIP Technical Committee TC8, North Holland Publishers, Paris, France, May 1987.

[Mackworth 85] A.K. Mackworth and E.C. Freuder.
The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems.
Artificial Intelligence 25(1):65-74, 1985.

[Purdom 83] Paul W. Purdom, Jr.
Search Rearrangement Backtracking and Polynomial Average Time.
Artificial Intelligence 21:117-133, 1983.

[Sadeh 88] N. Sadeh and M.S. Fox.
Preference Propagation in Temporal/Capacity Constraint Graphs.
Technical Report CMU-CS-88-193, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, 1988.
Also appears as Robotics Institute technical report CMU-RI-TR-89-2.

[Sadeh 89] N. Sadeh and M.S. Fox.
Focus of Attention in an Activity-based Scheduler.
In *Proceedings of the NASA Conference on Space Telerobotics.* January, 1989.

[Sadeh 90] Norman Sadeh, and Mark S. Fox.
Variable and Value Ordering Heuristics for Activity-based Job-shop Scheduling.
In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management, Hilton Head Island, S.C.*, pages 134-144. 1990.

[Sadeh 91] Norman Sadeh.
Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.
PhD thesis, School of Computer Science, Carnegie Mellon University, March, 1991.

[Sadeh 92] N. Sadeh and M.S. Fox.
Variable and Value Ordering Heuristics for Hard Constraint Satisfaction Problems: an Application to Job Shop Scheduling.
Technical Report CMU-RI-TR-91-23, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1992.

[Sadeh 94a] Norman Sadeh, Katia Sycara, and Yalin Xiong.
Backtracking Techniques for the Job Shop Scheduling Constraint Satisfaction Problem.
Artificial Intelligence Journal , 1994.
To appear in Special Issue on 'Planning and Scheduling'.

[Sadeh 94b] Norman Sadeh.
Micro-Opportunistic Scheduling: The MICRO-BOSS Factory Scheduler.
Intelligent Scheduling.
In Mark Fox and Monte Zweben,
Morgan Kaufmann Publishers, 1994, Chapter 4.

[Stallman 77] R. Stallman and G. Sussman.
Forward Reasoning and Dependency-directed Backtracking in a Sysem for Computer-aided Circuit Analysis.
Artificial Intelligence 9:135-196, 1977.

[Sycara 91] Sycara, K. and Roth, S. and Sadeh, N. and Fox, M.
Distributed Constrained Heuristic Search.
IEEE Transactions on System, Man and Cybernetics 21(6), 1991.

[Walker 60] R.J. Walker.
An Enumerative Technique for a Class of Combinatorial Problems.
Combinatorial Analysis, Proc. Sympos. Appl. Math.
In R. Bellman and M. Hall,
American Mathematical Society, Rhode Island, 1960, pages 91-94, Chapter 7.

Focused Simulated Annealing Search: An Application to Job Shop Scheduling

Norman M. Sadeh and Yoichiro Nakakuki
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper presents a simulated annealing search procedure developed to solve job shop scheduling problems simultaneously subject to tardiness and inventory costs. The procedure is shown to significantly increase schedule quality compared to multiple combinations of dispatch rules and release policies, though at the expense of intense computational efforts. A meta-heuristic procedure is developed that aims at increasing the efficiency of simulated annealing by dynamically inflating the costs associated with major inefficiencies in the current solution. Three different variations of this procedure are considered. One of these variations is shown to yield significant reductions in computation time, especially on problems where search is more likely to get trapped in local minima. We analyze why this variation of the meta-heuristic is more effective than the others.

This paper has been submitted for publication in the "Annals of Operations Research" (special issue on Metaheuristics in Combinatorial Optimization) [25].

This research was supported, in part, by the Defense Advanced Research Projects Agency under contract F30602-91-C-0016 and, in part, by an industrial grant from NEC.

1 Introduction

Over the past several years, with the advent of ever more powerful computers, stochastic procedures such as Simulated Annealing (SA) [14, 2] (and improved variations exploiting Tabu Search principles [9, 10]) or Genetic Algorithms (GAs) [11] have attracted the attention of a growing number of researchers. This interest has been fueled by both experimental and theoretical results indicating that, if properly designed and if given enough time, these procedures are often capable of finding near-optimal solutions to complex optimization problems.

This paper presents results obtained using SA to find solutions to job shop scheduling problems where the objective is to minimize the sum of weighted tardiness and inventory costs (both work-in-process inventory and finished-goods inventory costs). The model is particularly attractive as it is compatible with the Just-In-Time objective of meeting customer demand in a timely yet cost-effective manner. In the scheduling literature, this objective function is known to be irregular, as its value may sometimes be decreased by delaying the execution of some operations [3]. As will be shown, this property needs to be taken into account in the design of SA procedures for this class of problems.

By reference to simpler problems (e.g. the one machine version of this problem), this problem can easily be shown to be NP-hard [5, 6, 22, 23]. Surprisingly enough, despite the "attractiveness" of its modeling assumptions, this problem has been given very little attention in the literature. Two notable exceptions are the work of Tom Morton on resource-pricing heuristics in the context of the Sched-Star system [17] and our earlier work on micro-opportunistic bottleneck-centered techniques in the context of the Micro-Boss factory scheduling system [24].

The first part of this paper presents a SA procedure developed to solve job shop scheduling problems subject to both tardiness and inventory costs. Experimental results are presented comparing the performance of our procedure with that of several other scheduling heuristics. The results corroborate earlier studies performed on other combinatorial optimization problems. They indicate that SA consistently produces high quality solutions, often significantly outperforming other scheduling heuristics, though at the expense of intensive computational efforts. In the second part of this paper, we introduce "Focused Simulated Annealing" (FSA), a meta-heuristic procedure that aims at improving the efficiency of SA search. The idea behind FSA is that by dynamically inflating the costs associated with major inefficiencies in the existing solution, it is possible to focus the procedure and force it to get rid of these inefficiencies. By iteratively inflating costs in different subproblems, FSA can reduce the chances that the procedure gets trapped in local minima. Three variations of this meta-heuristic are considered that differ in the type of subproblems they rely on: job subproblems, resource subproblems, or operation subproblems. Experimental results comparing these three variations of the meta-heuristic against the original SA procedure show that the job-based meta-heuristic significantly improves performance, especially on problems where search is particularly likely to get caught in local minima. We further analyze why this variation of the meta-heuristic is more effective than the others, trying to shed some light on why, in general. some decompositions are likely to work better than others for a given SA procedure.

The balance of this paper is organized as follows. Section 2 provides a formal definition of the job shop scheduling problem considered in this study. Section 3 presents a SA search procedure developed for this problem. Section 4 reports experimental results comparing the performance of the procedure against that of other scheduling heuristics. The concept of Focused Simulated Annealing is introduced in Section 5 and three variations of this meta-heuristic procedure are developed for the job shop scheduling problem with tardiness and inventory costs. Performance of these meta-heuristics is reported in Section 6. These results are further discussed and analyzed in Section 7. Section 8 presents some concluding remarks.

2 The Job Shop Scheduling Problem with Tardiness and Inventory Costs

We consider a factory, in which a finite set of jobs, $J = \{j_1, j_2, \cdots, j_n\}$, has to be scheduled on a finite set of resources, $RES = \{R_1, R_2, \cdots, R_m\}$. The jobs are assumed to be known ahead of time and all resources are assumed to be available over the entire scheduling horizon. Each job j_l requires performing a set of manufacturing operations $O^l = \{O^l_1, O^l_2, \cdots O^l_{n_l}\}$ and, ideally, should be completed by a specific due date, dd_l, for delivery to a customer. Precedence constraints specify a complete order in which operations in each job have to be performed. By convention, we assume that operation O^l_i has to be completed before operation O^l_{i+1} can start $(i = 1, 2, \cdots, n_l - 1)$.

Each job j_l has an earliest acceptable release date, erd_l, before which it cannot start, e.g. because the raw materials or components required by this job cannot be delivered before that date. Each job also has a latest acceptable completion date (or deadline), lcd_l, by which it should absolutely be completed, e.g. because the customer would otherwise refuse delivery of the order. For each job, we assume that $erd_l \leq dd_l \leq lcd_l$. Furthermore, we assume that these constraints are loose enough to always allow for the construction of a feasible schedule (i.e. we are not concerned with the detection of infeasible problems).

This paper considers problems in which each operation O^l_i requires a single resource $R^l_i \in RES$ and the order in which a job visits different resources varies from one job to another. Each resource can only process one operation at a time and is non-preemptable. The duration du^l_i of each operation O^l_i is assumed to be known.

The problem requires finding a schedule (i.e. a set of start times, st^l_i, for all operations, O^l_i) that satisfies all these constraints while minimizing the sum of tardiness costs and inventory costs of all the jobs.

Specifically, each job j_l incurs a positive marginal tardiness cost $tard_l$ for each unit of time that it finishes past its due date dd_l. Marginal tardiness costs generally correspond to tardiness penalties, interests on lost profits, loss of customer goodwill, etc. The total tardiness cost of job j_l, in a given schedule, is measured as: $TARD^l = tard_l \cdot MAX(0, C_l - dd_l)$ where C_l is the completion date of job j_l. That is $C_l = st^l_{n_l} + du^l_{n_l}$, where $O^l_{n_l}$ is the last operation of j_l.

Inventory costs on the other hand can be introduced at the level of any operation in a job. In our model, each operation O^l_i can have its own non-negative marginal inventory cost, in^l_i. This is the marginal cost that is incurred for each unit of time that spans between the start time of this operation and either the completion date of the job or its due date, whichever is larger. In other words, the total inventory cost introduced by an operation O^l_i in a given schedule is:

$$INV^l_i = in^l_i \cdot (MAX(C_l, dd_l) - st^l_i)$$

Typically, the first operation in a job introduces marginal inventory costs that correspond to interests on the costs of raw materials, interests on processing costs (for that first operation), and marginal holding costs. Following operations introduce additional inventory costs such as interests on processing costs, interests on the costs of additional raw materials or components required by these operations, etc. Additional details on this model can be found in [23, 24].

The total cost of a schedule is:

$$\sum_{l \in J} TARD^l + \sum_{l \in J} \sum_{i=1}^{n_l} INV^l_i$$

For reasons that will become clearer in Section 5, it is often useful to look at the total tardiness and inventory costs of a job as sums of tardiness and inventory costs introduced by each of the operations in the job. For each operation O^l_i, we can define a best start time (or "just-in-time" start time), bst^l_i, where:

$$bst^l_i = \begin{cases} dd_l - du^l_i & (i = n_l) \\ bst_{i+1} - du^l_i & (1 \leq i < n_l) \end{cases}$$

Accordingly, the tardiness cost $TARD^l$ of job j_l in a given schedule, can be rewritten as:

$$TARD^l = \sum_{i=1}^{n_l} tcost^l_i$$

where,

$$tcost^l_i = \begin{cases} tard_l \cdot MAX(0, st^l_i - bst^l_i) & (i = 1) \\ tard_l \cdot \{MAX(0, st^l_i - bst^l_i) - MAX(0, st^l_{i-1} - bst^l_{i-1})\} & (1 < i < n_l) \\ tard_l \cdot \{MAX(0, C_l - dd_l) - MAX(0, st^l_{i-1} - bst^l_{i-1})\} & (i = n_l) \end{cases}$$

$tcost_i^l$ can be seen as the contribution of operation O_i^l to the total tardiness cost of job j_l. Similarly, the total inventory cost of a job j_l can be rewritten as:

$$INV^l = \sum_{i=1}^{n_l} icost_i^l$$

where:

$$INV_i^l = icost_i^l = \begin{cases} inv_i^l \cdot (MAX(st_i^l + du_i^l, dd_l) - st_i^l) & (i = n_l) \\ inv_i^l \cdot (st_{i+1}^l - st_i^l) & (1 \leq i < n_l) \end{cases}$$

and $inv_i^l = \sum_{k=1}^{i} in_k^l$. Accordingly, the total cost of a schedule can also be expressed as:

$$\sum_{l \in J} TARD^l + \sum_{l \in J} INV^l = \sum_{l \in J} \sum_{i=1}^{n_l} (tcost_i^l + icost_i^l)$$

For the sake of simplicity, the remainder of this paper further assumes that time is discrete, i.e. that job due dates/earliest acceptable release dates/latest acceptable completion dates and operation durations can only can only take integer values.

The following section introduces a SA procedure developed for this problem.

3 A Simulated Annealing Procedure

Simulated Annealing (SA) is a general-purpose search procedure that generalizes iterative improvement approaches to combinatorial optimization by sometimes accepting transitions to lower quality solutions so as to avoid getting trapped in local minima [14, 2]. SA procedures have been successfully applied to a variety of combinatorial optimization problems, including Traveling Salesman Problems [2], Graph Partitioning Problems [12], Graph Coloring Problems [13], Vehicle Routing Problems [21], Design of Integrated Circuits, Minimum Makespan Flow-Shop Scheduling Problems [20], Minimum Makespan Job Shop Scheduling Problems [16, 27], etc.

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

Fig. 1 Pseudo-code for a Basic SA Search Procedure.

Figure 1 outlines the main steps of a SA search procedure designed to find a solution $x \in S$ that minimizes a real-valued cost function, $cost(x)$. The procedure starts from an initial solution x_0 and iteratively moves to other neighboring solutions, while remembering the best solution found so far ($BestSol$). Typically, the procedure only moves to neighboring solutions that are better than the current one. However, the probability of moving from a solution x to an inferior solution x' is greater than zero, thereby allowing the procedure to escape from local minima. $rand()$ is a function that randomly draws a number from a uniform distribution on the interval $[0, 1]$. The so-called temperature, T, of the procedure is a parameter controlling the probability of accepting a transition to a lower quality solution. It is initially set to a high value, T_0, thereby frequently allowing such transitions. If, after N iterations, the best solution found by the procedure has not improved, the temperature parameter T is decremented by a factor α $(0 \leq \alpha \leq 1)$. When the temperature drops below a preset level, T_1, the procedure stops and the best solution it found ($BestSol$) is returned (not shown in the pseudo-code in Figure 1).

As indicated earlier, procedures similar to the one outlined above have been successfully applied to other scheduling problems such as the minimum makespan job-shop scheduling problem [1]. When dealing with regular scheduling objectives such as minimum makespan, it is possible to limit search to permutations of operations on the same machine. For instance, in their SA procedure, Van Laarhoven et al. exploit this observation and restrict the neighborhood structure to permutations of consecutive operations on a same machine [27]. In the case of scheduling problems with irregular objectives, such a neighborhood structure would not be sufficient, as it does not allow for the insertion of idle-time in the schedule, which sometimes improves the quality of a solution [2]. Here, two main approaches can be considered. A first approach would be to combine a SA procedure relying on permutation-based neighborhoods with a procedure that inserts idle-time optimally. As it turns out, the problem of inserting idle time optimally in a schedule, given completely specified sequences of operations on each machine, can be formulated as a Linear Programming (LP) problem and, hence, can be solved in polynomial time (See Appendix A for details). When considering the permutation of two operations, the SA procedure would first invoke an idle-time insertion procedure to compute the cost of the best schedule compatible with the new set of sequencing decisions. Based on this cost and the cost of the current solution, the search procedure would probabilistically determine whether or not to accept the transition. Nevertheless, at the present time, the idle time insertion procedures that the authors are aware of for the job shop scheduling problem remain too slow and would significantly limit the number of solutions that SA could explore in a reasonable amount of time [3].

Instead, an alternative neighborhood structure was adopted that directly allows

[1]The makespan of a schedule is the length of the time interval that spans between the start time of the first released job and the end time of the last completed job.

[2]Scheduling problems with regular objective functions have been shown to be reducible to sequencing problems [1]. Given fixed operation sequences on each machine, the schedule obtained by starting each operation as early as possible is undominated. With irregular objectives, this is no longer the case and it is sometimes better to delay the start of some operations. Here, we generically refer to the problem of deciding by how much to delay operations as the problem of "inserting idle time" in the schedule.

[3]For instance, using the CPLEX Linear Programming package on a DECstation 5000/200, inserting idle time optimally in a 100 operation job shop schedule takes about 1 CPU second. Taking into account similarities between the current schedule and the schedule obtained after permuting the order of two operations on the same resource, it is generally possible to reduce the time required to re-optimize the schedule to about 0.1 to 0.2 CPU seconds. Even under these conditions, a SA run of about 10 minutes would only be able to explore a few thousand solutions.

for idle time insertion. This structure, which is described below, lends itself to quick updates of the cost function. A possibly more subjective advantage has to do with the fact that the resulting procedure relies solely on SA and hence is not affected by the performance of a separate idle time insertion procedure. Specifically, the neighborhood function used in our implementation randomly selects among three types of modification operators, respectively referred to below as "RIGHT-SHIFT", "LEFT-SHIFT" and "EXCHANGE":

RIGHT-SHIFT This operator randomly chooses a "right-shiftable" operation and increases its start time by one time unit (Figure 2-(a)). An operation is assumed to be "right-shiftable", if it can be shifted by one time unit without bumping into another operation on the same resource or violating the latest acceptable completion date of the job to which it belongs (Figure 2-(b)). Precedence constraints within a job are ignored when determining whether or not an operation can be right-shifted. Instead, as will be seen later, these constraint violations are taken care of by inserting artificial costs in the objective function.

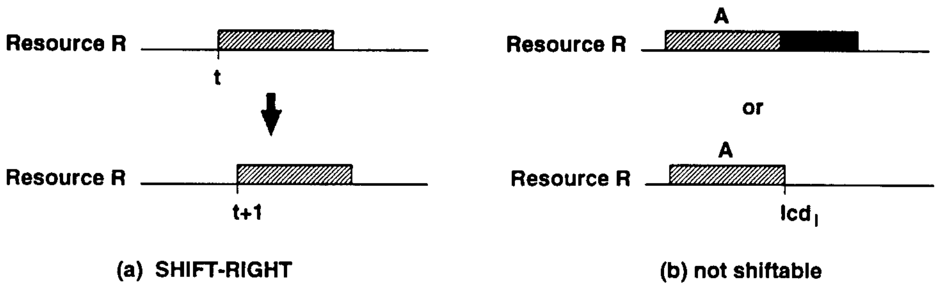


Fig. 2 RIGHT-SHIFT operator

LEFT-SHIFT This operator is the mirror image of RIGHT-SHIFT. It randomly picks a "left-shiftable" operation and decreases its start time by one time unit (Figure 3-(a)). It is assumed that an operation cannot be shifted left, if it would either bump into an adjacent operation on the same resource (top case in Figure 3-(b)) or violate the earliest acceptable release date of the job to which it belongs (bottom case in Figure

3-(b)).

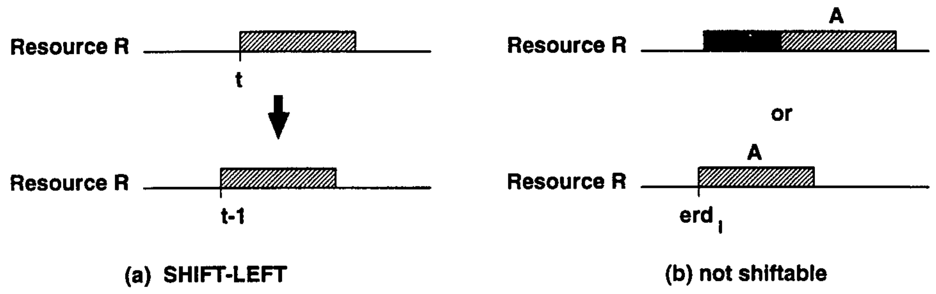


Fig. 3 LEFT-SHIFT operator

EXCHANGE This operator selects a pair of consecutive operations on a resource and exchanges the order in which the operations are scheduled to be processed on that resource. Specifically, given two consecutive operations, A and B on a resource R, with A preceding B in the current solution, the exchange operator sets the new start time of B to the old start time of A and the new end time of A to the old end time of B, as depicted in Figure 4.

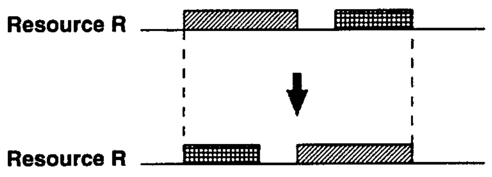


Fig. 4 EXCHANGE operator

In the experiments presented in this paper, the probability of picking the EXCHANGE operator was empirically set to 3/7 while the probabilities of picking a RIGHT- or LEFT-SHIFT operator were both set to 2/7. The initial solution x_0 used by the SA procedure is randomly generated in such a way that no two operations use the same resource at the same time. As with the RIGHT- and LEFT-SHIFT operator, precedence constraints between consecutive operations within a same job are not enforced in the process. Instead these constraints are enforced using artificial costs. If an operation O_i^l overlaps with a preceding operation O_{i-1}^l (within the same job j_l), an artificial cost $fcost_i^l$ is introduced in the objective function:

$$cost(x) = \sum_{l \in J} \sum_{i=1}^{n_l} (fcost_i^l + tcost_i^l + icost_i^l)$$

where $fcost_i$ is proportional to the amount of overlap between O_i^l and its predecessor O_{i-1}^l. Specifically:

$$\begin{aligned} &fcost_1^l = 0 \\ &fcost_i^l = \beta \cdot max(0, st_{i-1}^l + du_{i-1}^l - st_i^l) \ \ (i \geq 2) \end{aligned}$$

where β is a large positive constant.

The next section summarizes the results of experiments comparing this basic SA procedure with several other scheduling heuristics.

4 A First Set of Empirical Results

Performance of this first SA procedure was assessed through comparative studies against a number of other scheduling heuristics. This section summarizes the results of experiments comparing the SA procedure against 39 combinations of well-regarded dispatch rules and release policies (including those combinations that were reported to perform best in the evaluation of the Sched-Star scheduling system [17]) both with and without idle-time optimization, using the LP formulation provided in Appendix A.

Specifically, two types of dispatch rules were considered:

1. A set of five priority dispatch rules that have been reported to be particularly good at reducing tardiness under various scheduling conditions [28]: the Weighted Shortest Processing Time (WSPT) rule, the Earliest Due Date (EDD) rule, the Slack per Remaining Processing Time (SRPT) rule, and two parametric rules, the Weighted Cost OVER Time (WCOVERT) rule and the Apparent Tardiness Cost (ATC) rule (also referred to sometimes as the Rachamadugu&Morton rule [18]).

2. An exponential version of the parametric early/tardy dispatch rule recently developed by Ow and Morton [22, 17] and referred to below as EXP-ET. This rule differs from the other 5 in that it can explicitly account for both tardiness and inventory costs.

EXP-ET was successively run in combination with two release policies: an intrinsic release policy that only releases jobs when their priorities become positive, as suggested in [17], and an immediate release policy (IM-REL) that allowed each job to be relased immediately. The other five dispatch rules were also successively run in combination with two release policies: an immediate release policy and the Average Queue Time release policy (AQT) described in [17]. AQT is a parametric release policy that estimates queuing time as a multiple of the average job duration (the look-ahead parameter serving as the multiple). A job's release date is determined by offsetting the due date of the job by the sum of its total duration and its estimated queuing time. In their evaluation of the SCHED-STAR scheduling system, Morton et al. report that the combination of WCOVERT and AQT performed best after their SCHED-STAR system and was within 0.1% of the best schedule in 42% of the problems they studied and within 4% in 70% of their problems [17]. They also report that the next best scheduling heuristic is EXP-ET in combination with its intrinsic release policy.

Combinations of release policies and dispatch rules with a look-ahead parameter were successively run with four different parameter values that had been identified as producing the best results. By combining these different dispatch rules, release policies and parameter settings a total of 39 heuristics[4] was obtained.

These 39 combinations of priority dispatch rules and release policies were run in two different ways:

1. On each problem, the best of the 39 schedules produced by these combinations was recorded. In this case, out of the 39 combinations, 13 performed best on at least one of the 40 problems considered in the study. These 13 combinations

[4]The 39 combinations were as follows: EXP-ET and its intrinsic policy (times four parameter settings), EXP-ET/IM-REL (times four parameter settings), EDD/AQT (times four parameter settings), EDD/IM-REL, WSPT/AQT (times four parameter settings), WSPT/IM-REL, SRPT/AQT (times four parameter settings), SRPT/IM-REL, WCOVERT/IM-REL (times four parameter settings), WCOVERT/AQT (times four parameter settings), ATC/IM-REL (times four parameter settings), ATC/AQT (times four parameter settings).

included 5 of the 6 dispatch rules (SRPT was never best on this set of problems) and all 3 release policies.

2. On each problem, each of the 39 schedules obtained by these combinations was post-processed using an LP program to insert idle time optimally. Again, on each problem, the best of the 39 post-processed schedules was recorded for comparison against the SA procedure. In this case, out of the 39 combinations, 11 performed best (after post-processing) on at least one of the 40 problems considered in the study. These 11 combinations included 5 of the 6 dispatch rules (here, WSPT was never best) and all 3 release policies.

Table 1 Characteristics of the eight problem sets

Problem Set	Number of Bottlenecks	Avg. Due Date	Due Date Range
1	1	loose	wide
2	1	loose	narrow
3	1	tight	wide
4	1	tight	narrow
5	2	loose	wide
6	2	loose	narrow
7	2	tight	wide
8	2	tight	narrow

The results reported below were obtained on a suite of 40 scheduling problems similar to the ones described in [24]. The series consisted of eight sets of scheduling problems obtained by adjusting three parameters to cover a wide range of scheduling conditions (See Table 1): an average due date parameter (tight versus loose average due date), a due date range parameter (narrow versus wide range of due dates), and a parameter controlling the number of major bottlenecks (in this case one or two). For each parameter combination, a set of 5 scheduling problems was randomly generated, thereby resulting in a total of 40 problems (5 problems x 2 average due date values x 2 due date ranges x 2 bottleneck configurations). Each problem involved 20 jobs and 5 resources for a total of 100 operations. Marginal tardiness costs in these problems were set to be, on average, ten times larger than marginal inventory costs to model a situation where tardiness costs dominate but inventory costs are non-negligible[5].

[5]Similar results have also been obtained on a set of problems where marginal tardiness costs were on average five times larger than marginal inventory costs.

The SA procedure was run 10 times on each problem. For each problem, we recorded both the average performance of the procedure (referred to below as SA-AVG) as well as the best solution it found for each problem over 10 runs (SA-BEST) . In each run, the initial temperature, T_0, was set to 700, temperature T_1 was 6.25 and the cooling rate α was 0.85. The value of β was 1000 [6]. The number N of iterations in the

[6]The problems that were run typically had optimal solutions with a value ranging between 3000 and 15000. Setting β to 1000 was sufficient to guarantee that all precedence constraints were satisfied at the end of each run.

inner-loop of the procedure (See Figure 1) was set to 300,000.

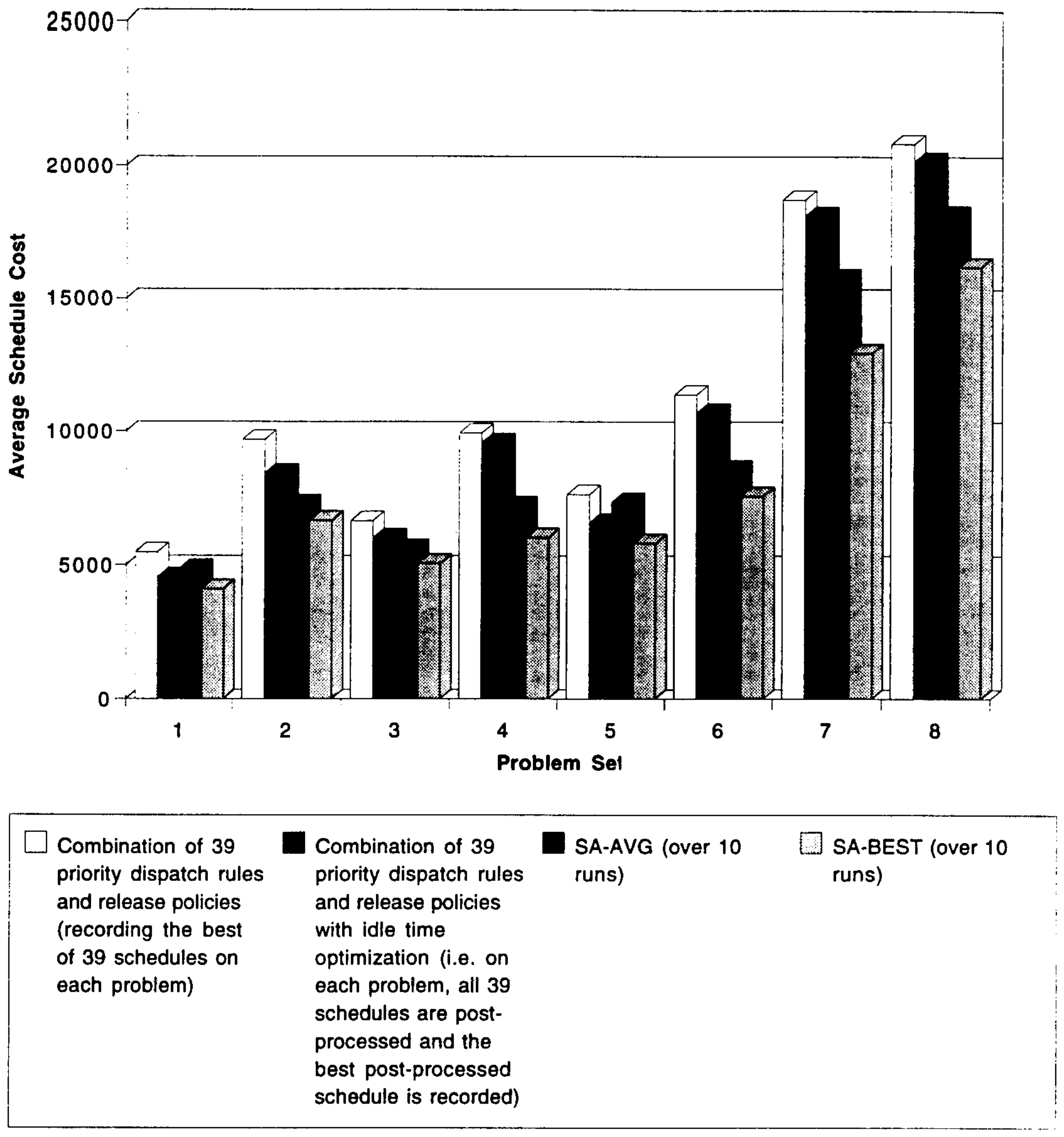


Fig. 5: Comparison of SA and a combination of 39 dispatch rules and release policies with and without optimal idle time insertion.

Figure 5 compares the schedules produced by the SA procedure with the best schedules obtained on each problem by the 39 combinations of dispatch rules and release policies both with and without idle time optimization. For instance, on Problem Set 6 (problems with two bottleneck resources, loose average due dates and narrow due date ranges), (1) SA-BEST reduced schedule cost by almost 11% compared to SA-AVG, (2) SA-AVG reduced schedule cost by about 18% compared to the 39 combinations of dispatch rules and release policies with optimal idle time insertion and (3) performance of the 39 combinations of dispatch rules and release policies (taking the best of 39 schedules on each problem) improves by more than 6% with optimal idle time insertion.

Overall, Figure 5 indicates that SA-BEST consistently outperforms the combinations of dispatch rules and release policies with and without idle time insertion on all 8 problem sets. The comparison also holds for SA-AVG with the exception of the two easier problem sets (Problem Set 1 and 5, i.e. problems with loose and widely spread due dates), where SA-AVG does slightly worse than the 39 combinations with idle time optimization. Notice that SA-AVG still outperforms the 39 combinations without idle time optimization on these two problem sets. Overall, compared against the 39 combinations of dispatch rules and release policies without idle time optimization, SA-AVG reduced schedule cost by close to 16% and SA-BEST by close to 28%. Even when, on each problem, idle time was optimally inserted in each of the 39 schedules obtained by the combinations of dispatch rules and release policies, SA-AVG still reduced schedule cost by an average of about 7% and SA-BEST by over 20%. A more detailed analysis indicates that these reductions in schedule cost reflect reductions in both tardiness and inventory costs. However, while running all 39 combinations of dispatch rules and release policies requires only a few CPU seconds on each problem and about 45 to 50 CPU seconds when idle time is optimally inserted in each of the 39 schedules, a SA run takes 3 to 5 minutes on a DECstation 5000/200 running C.

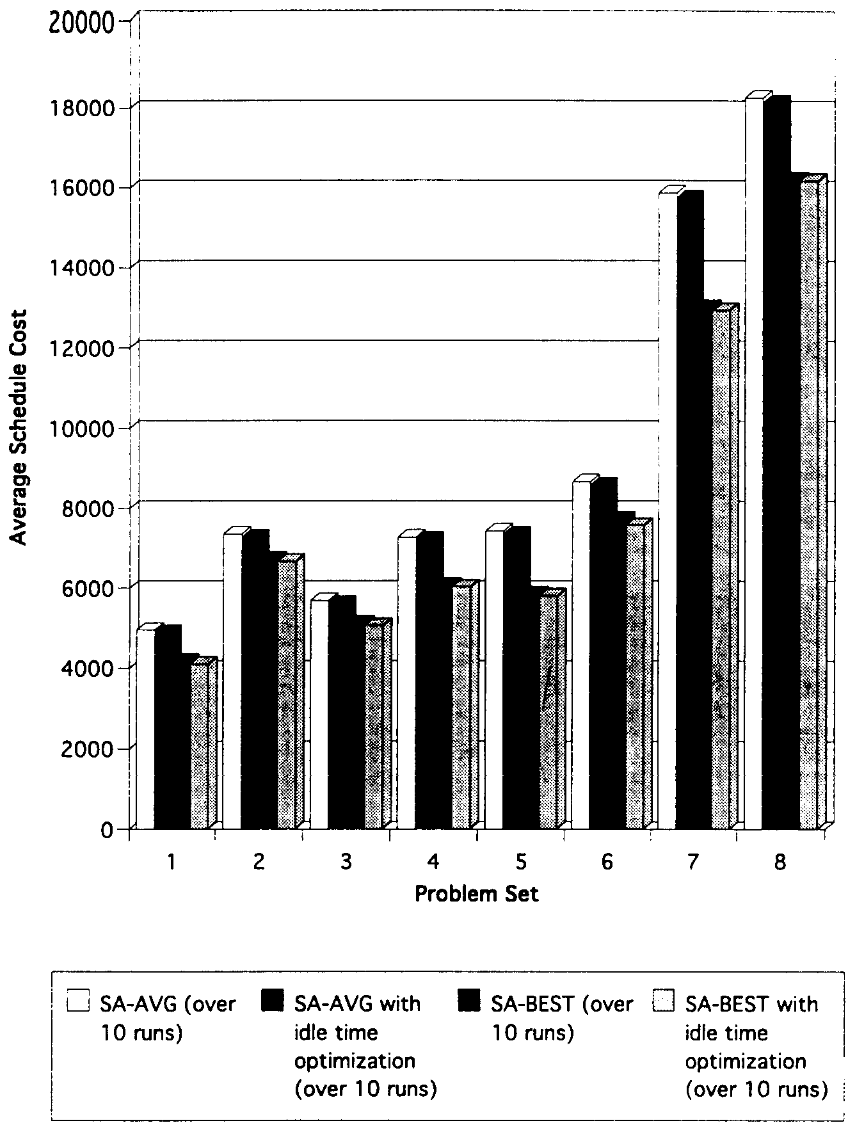


Fig. 6 Performance of the SA procedure with and without idle time optimization.

Additional experiments were also conducted to evaluate the performance of the SA procedure with respect to idle time optimization. Figure 6 summarizes these experiments, reporting both the average and best performance of the SA procedure over 10 runs with and without post-processing for optimal idle time insertion. The results clearly indicate that the schedules produced by the SA procedure are nearly optimal with respect to idle time insertion, thereby validating the choice of the LEFT- and RIGHT-SHIFT operators used to define the neighborhood of the procedure. On average, idle time insertion improved performance of SA-BEST by a meager 0.94% (with a standard deviation of 0.8%) and that of SA-AVG by 1.02% (with a standard deviation of 0.5%).

The results in Figure 5 and 6 generally attest to the ability of the SA procedure to produce high quality solutions, often significantly reducing schedule cost compared to other well-regarded scheduling heuristics. They also indicate that the computational requirements of the procedure are quite large compared to these other heuristics, though experiments with larger problems suggest that the average complexity of our SA procedure only grows linearly with the size of the problem.

Finally, we observe that the performance of the SA procedure can significantly vary from one run to another, as illustrated by the results in Figure 5. In our experiments, an average run of SA produced schedules with costs 14% higher than those of the best schedule obtained over 10 runs (SA-AVG vs. SA-BEST). This suggests that important speedups could possibly be obtained if the procedure was more consistent in producing high quality solutions. In the following section, a meta-heuristic procedure is presented that aims at reducing performance variability using artificial costs to dynamically focus the SA procedure on critical subproblems.

5 Focused Simulated Annealing Search

Figure 7 depicts 5 typical runs of the SA procedure introduced in the previous sections, plotting the cost of the best schedule found in each run, as the the temperature is slowly lowered over time.

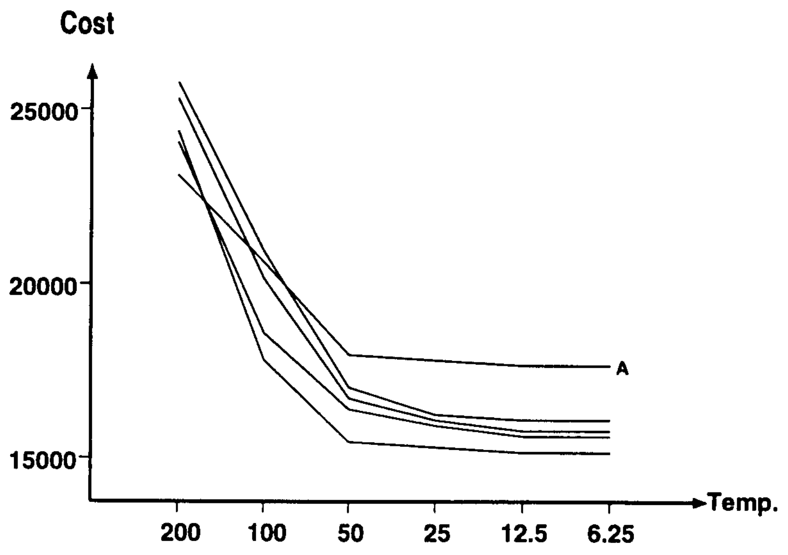


Fig. 7 Solution improvement in 5 runs of SA.

The behavior exhibited in Figure 7 is characteristic of SA search procedures: the largest improvements are observed at relatively high temperatures. In the case of our SA procedure, we observed that below $T = 50$ the quality of the solution never improved by more than a few percent. In other words, the early stage of the procedure is the one that determines whether or not the procedure will get trapped in a local minimum (e.g. See run A in Figure 7). The remainder of this section describes a meta-heuristic that relies on the dynamic introduction of artificial costs in the objective function to focus SA on critical subproblems and attempt to steer clear of local minima during the high temperature phase of the procedure. Below we refer to the resulting procedure as "Focused Simulated Annealing" (FSA) search.

To improve the quality of an existing solution, FSA iteratively identifies major inefficiencies in the current solution and attempts to make these inefficiencies more obvious to the search procedure by artificially inflating their costs. As a result, the search procedure works harder on getting rid of these inefficiencies, possibly introducing new inefficiencies in the process. By regularly tracking sources of inefficiency in the existing solution and reconfiguring the cost function to eliminate these inefficiencies, FSA can increase the chances that the procedure finds a high quality solution.

```
$T = T_0$;
$x = x_0\ (\in S)$;
$BestSol = x_0$; $M = cost(BestSol)$;
while $(T > T_1)$ {
    if $(T > T_2)$
        $CritSubp$ = identify-high-cost-subp(x);
    else
        $CritSubp = \emptyset$;
    for $i = 1, N$ {
        $x' = neighbor(x)$;
        if $(cost1(x') < cost1(x))$   $x = x'$;
        else if $(rand() < exp\{(cost1(x) - cost1(x'))/T\})$   $x = x'$;
        if $(cost(x') < M)$   $\{BestSol = x'; M = cost(BestSol);\}$
    }
    if ($M$ was not modified in the above loop)   $T = T * \alpha$;
}
```

Fig. 8 The FSA Procedure: A meta-heuristic that continuously attempts to reduce major inefficiencies in the solution.

Pseudo-code for the FSA procedure is given in Figure 8. T_2 is a threshold temperature between T_0 and T_1. Below T_2, FSA behaves exactly as the SA search procedure described in Figure 1. Before reaching this temperature, the procedure uses a different cost function to decide whether or not to accept transitions to neighboring solutions, namely:

$$cost1(x) = cost(x) + ArtifCost(x)$$

where:

$$ArtifCost(x) = \sum_{O_i^l \in CritSubp} \{k(tcost_i^l + icost_i^l)\}$$

or, equivalently:

$$cost1(x) = \sum_{l \in J} \sum_{1 \leq i \leq n_l} (fcost_i^l + tcost_i^l + icost_i^l) + \sum_{O_i^l \in CritSubp} \{k(tcost_i^l + icost_i^l)\}$$

k is a parameter that controls the amount by which the costs associated with operations in critical subproblems are inflated. In our experiments, we found that setting k to 2 and T_2 to 50 generally yielded good results. Results obtained with other values for these parameters are provided in Appendix B.

Notice that inflating the costs associated with one or several subproblems is equivalent to reducing the temperature associated with the corresponding components of the objective function (or raising the temperature in the remainder of the problem). Accordingly, FSA can be viewed as a SA procedure in which transition probabilities are subject to different temperatures in different parts of the problem. Temperatures in different subproblems are regularly modified (lowered or raised) to get rid of major inefficiencies in one part or another of the working solution. In this regard, FSA is reminiscent of the Strategic Oscillation idea of developing non-monotonic cooling schedules [7, 21, 10]. However, while Strategic Oscillation cooling schedules proposed in the literature vary temperature in the entire problem, FSA emphasizes selective temperature variations in dynamically identified subproblems, as detailed below.

The specific parts of the solution in which FSA attempts to eliminate inefficiencies are determined by the *identify-high-cost-subp()* function. Here several variations of the procedure are considered that differ in the way they decompose the problem: a job-based variation, a resource-based variation and an operation-based variation.

"Critical Job" (CJ) variation This variation of FSA dynamically inflates the costs associated with critical jobs. Here, *identify-high-cost-subp()* computes the the cost of each job j_l in the current schedule, namely,

$$\sum_{1 \leq i \leq n_l} (tcost_i^l + icost_i^l)$$

. The function then returns the set of all jobs whose costs are above $p \cdot av_R$, where av_R is the average cost of a job in the current schedule and p is a constant. In the experiments reported below, p was empirically set to 3. Below we refer to this variation of the procedure as $FSA(CJ)$. Results obtained with other values of p are also reported in Appendix B.

"Bottleneck Resource" (BR) variation This variation of FSA inflates the costs associated with critical ("bottleneck") resources in the existing schedule. This is done by computing a cost for each resource R_k:

$$\sum_{R_i^l = R_k} (tcost_i^l + icost_i^l)$$

In this case, the highest cost resource is selected and all the operations requiring this resource are returned by *identify-high-cost-subp()*. This procedure will be referred to as $FSA(BR)$.

"Critical Operation" (CO) variation Here, FSA focuses on critical operations rather than critical jobs or critical resources. The average cost of an operation in the current schedule, av_O, is computed:

$$av_O = \sum_{l \in J} \sum_{1 \leq i \leq n_l} (tcost_i^l + icost_i^l) / \sum_{l \in J} n_l$$

All the operations with a cost above $q \cdot av_O$ are considered critical. q is a constant. In the experiments reported below, q was equal to 3. We will denote this procedure *FSA(CO)*. Results obtained with other values of q are also reported in Appendix B.

6 Performance Evaluation

To evaluate the effectiveness of FSA, all three variations of the procedure were run on a set of 40 scheduling problems similar to the ones described in Section 4. Each variation was run 10 times on each problem. Table 2 compares each of the three variations of the FSA procedure against the SA procedure described in Section 3. Both the average and best performance over 10 runs are reported.

Table 2 Cost Reduction (%) obtained by FSA over SA

Problem Set	FSA(CJ)		FSA(BR)		FSA(CO)	
	Avg	Best	Avg	Best	Avg	Best
1	4.5	2.0	-0.8	0.9	-0.6	-0.6
2	6.5	5.1	4.3	2.3	3.5	4.3
3	7.3	5.2	-1.2	-2.6	-0.9	-2.3
4	0.4	0.2	-2.4	-2.2	-2.1	-2.2
5	8.6	4.5	-9.4	-1.7	1.9	3.2
6	8.0	2.4	-2.8	-4.3	2.3	4.2
7	6.7	6.2	-25.2	-6.3	-2.4	0.4
8	0.2	3.5	-2.7	-0.5	-2.1	0.7
Overall	5.2	3.6	-5.0	-2.0	-0.5	0.9

The results in Table 2 show that the dynamic introduction of artificial costs, as implemented in the FSA procedure, can potentially lead to significant improvements both in the average and best performance of the SA procedure. The results also show that the effectiveness of this approach depends on the type of subproblems considered by the procedure. While FSA(CJ) reduced schedule cost by an average of 5.2% and improved the quality of the best schedule found in 10 runs by an average of 3.6%, the other two variations of the procedure, FSA(BR) and FSA(CO), did not fare as well. FSA(CO) performed approximately like the original SA procedure and FSA(BR) actually did worse. Below, we further analyze the performance improvement obtained with FSA(CJ). In the following section, we attempt to explain why FSA(CO) and FSA(BR) did not perform as well. For now, we further analyze the performance improvements observed with FSA(CJ).

Figure 9 and 10 show the cost distributions of the schedules obtained by successively running SA and FSA(CJ) 300 times on two typical scheduling problems.

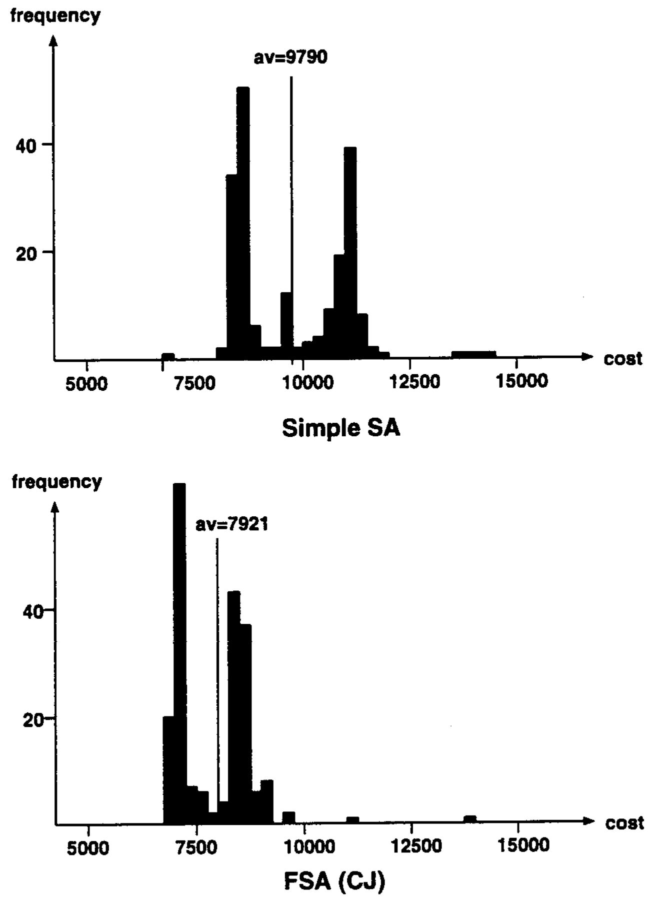


Fig. 9 Improvement of FSA(CJ) over the original SA procedure (Problem 1)

On the problem in Figure 9, the improvement obtained with FSA(CJ) is quite obvious: both the average and standard deviation of the cost distribution produced by FSA(CJ) are lower than those of the original SA procedure. Accordingly, it appears that for this problem the probability of getting trapped in a local optimum has been greatly reduced. This in turn can translate in significant reductions in computation time. For instance, while the original SA procedure would require an average of 2.5 runs to find a schedule with cost below 9000, FSA(CJ) would only require an average of 1.1 run, a saving of more than 50%. To find a schedule of cost below 8000, FSA(CJ) reduces computation time by more than 90%.

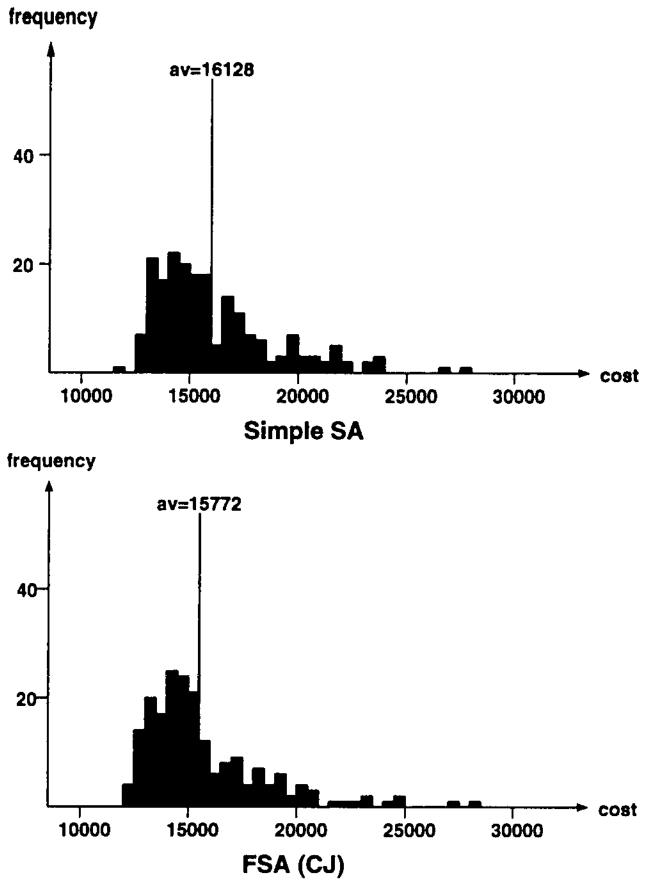


Fig. 10 Improvement of FSA(CJ) over the original SA procedure (Problem 2)

On the other hand, for the problem in Figure 10, the performance improvement yielded by FSA(CJ) is rather modest: no significant reduction in average schedule cost or even in the standard deviation of the distribution.

Looking more carefully at these two problems, we observe that, in the case of the problem in Figure 9, SA yields a cost distribution with two clearly separated peaks, thereby suggesting that the procedure is often caught in local minima. In contrast, in the case of the problem analyzed in Figure 10, the cost distribution obtained using the original SA procedure is generally more compact. This would suggest that FSA(CJ) is more effective in those situations where the original procedure is more likely to get trapped in local optima.

To verify this hypothesis, we measure for each problem the average reduction in schedule cost yielded by FSA(CJ) (compared to the original SA procedure) and the spread of the cost distribution obtained with the original SA procedure. This spread is simply measured as the standard deviation of the cost distribution obtained by SA divided by the mean of this distribution. The results for all 40 problems of the study are summarized in Figure 11.

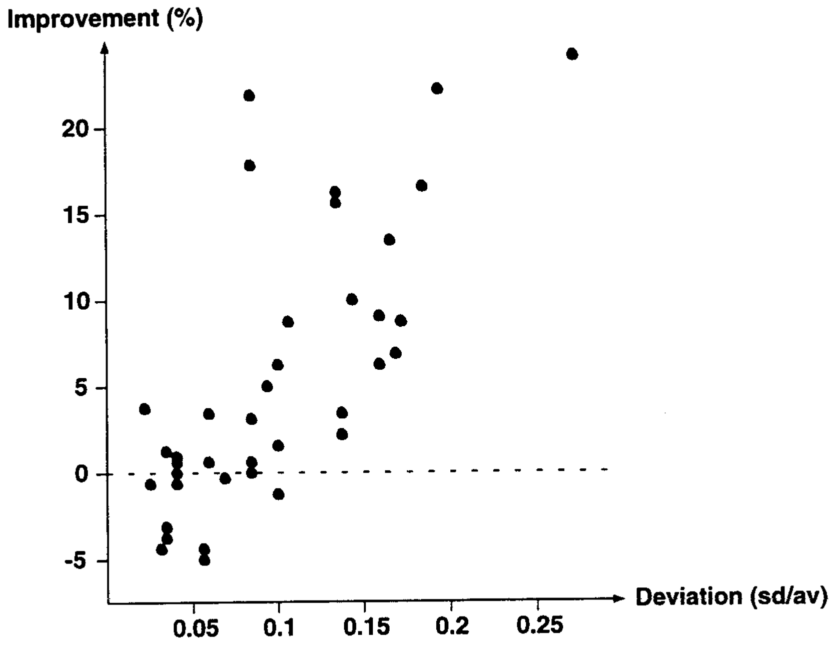


Fig. 11 Improvements obtained with FSA(CJ) as a function of the relative variation in schedule cost observed when using the original SA procedure.

The graph clearly confirms our intuition. The most important improvements are observed on problems where the original SA procedure showed the least consistency, namely those problems where it had the highest chance of getting trapped in local minima. The Figure also indicates that FSA(CJ) rarely performs worse than the original SA procedure, and, when it does, the degradation in schedule quality is marginal.

7 Further Analysis

If we are to apply FSA to other problems, we need to understand why some variations of the procedure perform better than others. There are at least two ways of approaching this question. One approach is to attempt to analyze the search procedure and the neighborhood structure it relies on and try to understand how the choice of a given type of subproblems influences the effectiveness of FSA on this specific class of scheduling problems. This approach is probably the one a scheduling expert would be tempted to follow. It could potentially lead to very insightful conclusions for the class of scheduling problems of interest in this study. However, our purpose here is different, as we are looking for insight that can possibly carry over to other domains. For this reason, we take a different approach and limit our analysis to the external behavior of the search procedure.

As pointed out at the beginning of Section 5, the early phase of a SA run, where temperature is still high, generally determines whether the procedure gets caught in a local minimum or not. Different neighborhood structures for a same class of problem can possibly lead to different types of local minima. The nature of these local minima can in turn affect the effectiveness of different problem decompositions in the FSA procedure. In Figure 12, we analyze cost reductions in different types of subproblems during the lower temperature phase of the *original* SA procedure. Specifically, Figure 12 considers improvements in three different types of subproblems:

1. CJ: the set of critical jobs that would be identified by FSA(CJ) at $T = 100$
2. BR: the critical ("bottleneck") resource that would be used by FSA(BR) at $T = 100$
3. CO: the set of critical operations that would be considered by FSA(CO) at $T = 100$

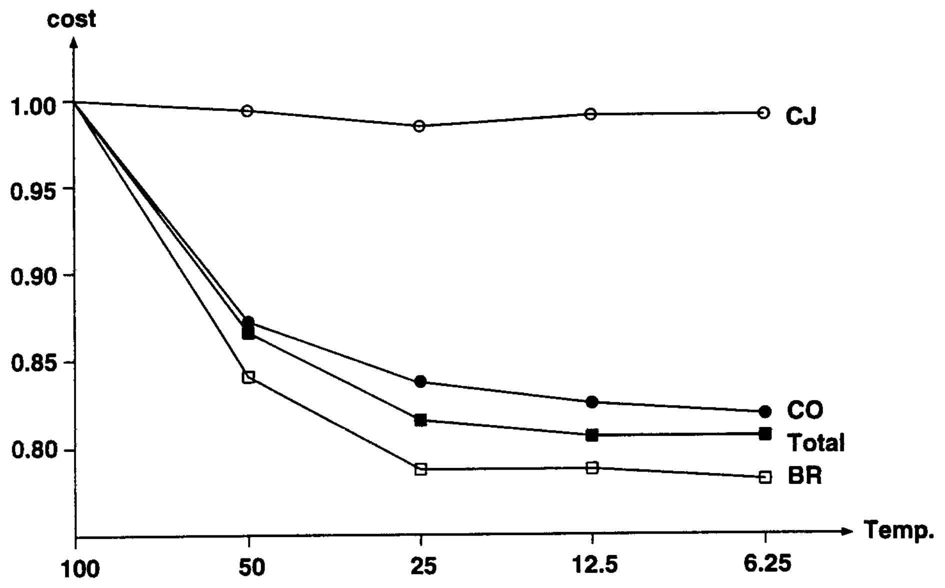


Fig. 12 Changes of Cost in the Later Stage of SA

For each of these subproblems, Figure 12 plots the average variation in cost associated with these 3 subproblems as the temperature in the original SA procedure is progressively lowered. The curve labeled "*Total*" plots the cost variations of the overall schedule as temperature decreases. The points in Figure 12 represent averages taken over the set of 40 problems studied in Section 6 and over 10 runs of SA on each problem.

Figure 12 indicates that, when using the original SA procedure, major inefficiences in job schedules do not get corrected below temperature $T = 100$, while major inefficiencies at the level of critical resources or critical operations are still easy to eliminate. This explains why FSA(CJ) is the variation that performs best: it is the one that best matches the weaknesses of the original SA procedure. By working hard on eliminating inefficiencies at the level of critical jobs, FSA(CJ) reduces the chances that such inefficiencies remain when the procedure reaches its lower temperature phase, a phase when it is no longer effective at getting rid of these inefficiencies. For the same reason, the BR curve suggests that FSA(BR) wastes its time getting rid of inefficiencies that are still easy to eliminate in the lower temperature phase of the procedure, and hence can be expected to perform poorly, as observed in the results presented in Section 6.

8 Summary and Concluding Remarks

In summary, the contribution of this work is twofold:

1. On the scheduling front, a SA procedure has been developed to solve job shop scheduling problems with both tardiness and inventory costs. The procedure has been shown to produce high quality solutions, reducing schedule cost by 28% over a combination of 39 well-regarded dispatch rules and release policies (and by 20% when the dispatch schedules are post-processed for optimal idle time insertion), though at the expense of significant computational efforts.

2. To reduce the computational requirements of this procedure, a meta-heuristic search procedure called Focused Simulated Annealing (FSA) search has been developed. This procedure aims at reducing variability in the performance of SA by dynamically focusing on the elimination of major inefficiencies in the solution. The procedure works by dynamically inflating the costs associated with critical subproblems and requires a decomposable objective function.

   Three variations of FSA have been developed for the job shop scheduling problem with tardiness and inventory costs. These variations of the procedure differ in the type of subproblems they rely on: job subproblems, resource subproblems, or operation subproblems. Experiments show that, with the right decomposition, FSA can significantly improve solution quality especially on problems where search is likely to get caught in local minima. Equivalently, for the same solution quality, FSA can greatly reduce computation time over a regular SA search.

   Our experiments also indicate that the performance of FSA critically depends on the selection of a good decomposition of the objective function. An analysis suggests that the most effective decompositions are those corresponding to subproblems whose solutions are particularly difficult to improve during the low temperature phase of the SA procedure. By focusing on inefficiencies at the level of these subproblems, FSA can greatly reduce the chance of getting trapped in local minima.

As is often the case in this type of study, many design alternatives remain to be explored. Further work will also be required to assess the effectiveness of FSA or FSA-like meta-heuristics in combination with more sophisticated SA procedures, e.g. procedures incorporating some aspects of Tabu Search [9, 26, 19]. Like Strategic Oscillation [7, 21, 10], FSA can be viewed as implementing a non-monotonic cooling schedule, though selectively, by focusing on dynamically identified subproblems. Strategic Oscillation could possibly also be exploited to control the value of β, the parameter used in our procedure to penalize precedence constraint violations within a job. Other aspects of Tabu Search such as Target Analysis [8, 15], which, like FSA, adds a term to the objective function to drive the procedure towards high quality solutions, would also be worth comparing with and possibly incorporating in the existing procedure.

References

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

[2] Cerny, V., "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," *J. Opt. Theory Appl.,* Vol. 45, pp. 41–51, 1985.

[3] French, S. "Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop," *Wiley,* 1982.

[4] Fry,T., R. Amstrong and J. Blackstone "Minimizing Weighted Absolute Deviation in Single Machine Scheduling," *IIE Transactions,* Vol. 19, pp. 445–450, 1987.

[5] Garey, M. R. and Johnson, D. S. "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman and Co., 1979

[6] Garey M.R., R.E. Tarjan, and G. T. Wilfgong, "One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties," *Mathematics of Operations Research,* Vol. 13, No. 2, pp. 330–348, 1988.

[7] Glover, F. "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences,* Vol. 8, No. 1, pp. 156–166, January 1977.

[8] Glover, F. "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computer and Operations Research,* Vol. 13, No. 5, pp. 533–549, 1986.

[9] Glover, F. "Tabu Thresholding: Improved Search by Non-Monotonic Trajectories," Technical Report, Graduate School of Business, University of Colorado, Boulder, Colorado 80309-0419, 1993.

[10] Glover, F. and M. Laguna "Tabu Search," Chapter in *Modern Heuristic Techniques for Combinatorial Problems.* pp. 70–150, Colin Reeves (Ed.), Blackwell Scientific Publications, Oxford, 1992

[11] Holland J. "Adaptation in Natural and Artificial Systems" University of Michigan Press, Ann Arbor, MI. 1975.

[12] Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. "Optimization by Simulated Annealing: Experimental Evaluation; Part I, Graph Partitioning" *Operations Research* Vol. 37 no. 6, pp. 865–892, 1989

[13] Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. "Optimization by Simulated Annealing: Experimental Evaluation; Part II, Graph Coloring and Number Partitioning" *Operations Research* Vol. 39 no. 3, pp. 378–406, 1991

[14] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science,* Vol. 220, pp. 671–680, 1983.

[15] Laguna M. and F. Glover "Integrating Target Analysis and Tabu Search for Improved Scheduling Systems," *Expert Systems with Applications,* Vol. 6, pp. 287–297, 1993.

[16] Matsuo H., C.J. Suh, and R.S. Sullivan "A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem" Tech. Report, "Dept. of Management, The Univ. of Texas at Austin Austin, TX, 1988.

[17] Morton, T.E. "SCHED-STAR: A Price-Based Shop Scheduling Module," *Journal of Manufacturing and Operations Management,* pp. 131–181, 1988.

[18] Morton, T.E. and Pentico, D.W. "Heuristic Scheduling Systems," *Wiley Series in Engineering and Technology Management,* 1993

[19] Nowicki E. and C. Smutnicki "A Fast Taboo Search Algorithm for the Job Shop Problem," *Technical University of Wroclaw, Institute of Engineering Cybernetics,* ul. Janiszewskiego 1/17, 50-372 Wrocklaw, Poland, 1993.

[20] Osman,I.H., and Potts, C.N., "Simulated Annealing for Permutation Flow-Shop Scheduling," *OMEGA Int. J. of Mgmt Sci.,* Vol. 17, pp. 551–557, 1989.

[21] Osman, I.H. "Meta-Strategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem" *Annals of Operations Research*, Vol. 41, pp. 421–451, 1993.

[22] Ow, P.S. and T. Morton, "The Single Machine Early/Tardy Problem" *Management Science*, Vol. 35, pp. 177–191, 1989.

[23] Sadeh, N. "Look-ahead Techniques for Micro-Opportunistic Job Shop Scheduling" Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, March 1991.

[24] Sadeh, N. "Micro-Opportunistic Scheduling: The Micro-Boss Factory scheduler" Ch. 4, *Intelligent Scheduling,* Zweben and Fox (Ed.), Morgan Kaufmann Publishers, 1994.

[25] Sadeh, N. and Y. Nakakuki "Focused Simulated Annealing Search: An Application to Job Shop Scheduling" Submitted to *Annals of Operations Research*, Issue on 'Metaheuristics in Combinatorial Optimization., 1994.

[26] Taillard, E. "Parallel Taboo Search Technique for the Job Shop Scheduling Problem," *ORSA Journal on Computing,* Vol. 6, pp. 108–117, 1994.

[27] Van Laarhoven, P.J., Aarts, E.H.L., and J.K. Lenstra "Job Shop Scheduling by Simulated Annealing," *Operations Research,* Vol. 40, No. 1, pp. 113–125, 1992.

[28] Vepsalainen,A.P.J. and Morton, T.E. "Priority Rules for Job Shops with Weighted Tardiness Costs," *Management Science*, Vol. 33, No. 8, pp. 1035–1047, 1987.

Appendix A: Idle Time Insertion as a Linear Program

The problem of optimally inserting idle in an existing job shop schedule (i.e. given completely defined operation sequences on each resource) can be formulated as a linear program, as detailed below:

$$MIN \sum_{l \in J} \{tard_l \cdot \tau_l + \sum_{i=1}^{n_l - 1} [inv_i^l \cdot (st_{i+1}^l - st_i^l)] + inv_{n_l}^l \cdot (du_{n_l}^l + \epsilon_l)\} \qquad (1)$$

such that:

$$\tau_l - \epsilon_l - st_{n_l}^l = du_{n_l}^l - dd_l \qquad l = 1, ..., n \qquad (2)$$

$$st_i^l - st_{i+1}^l \leq -du_i^l \qquad l = 1, ..., n \quad i = 1, ..., n_l - 1 \qquad (3)$$

$$st_{low(k,j)}^{up(k,j)} - st_{low(k,j+1)}^{up(k,j+1)} \leq -du_{low(k,j)}^{up(R_k,j)} \qquad k = 1, ..., m \quad j = 1, ..., p_k - 1 \qquad (4)$$

$$\tau_l \ , \epsilon_l \geq 0 \qquad l = 1, ..., n \qquad (5)$$

$$st_1^l \ \geq erd_l \qquad l = 1, ..., n \qquad (6)$$

$$st_{n_l}^l \ \leq lcd_l - du_{n_l}^l \qquad l = 1, ..., n \qquad (7)$$

where:

- τ_l is the tardiness of job j_l
- ϵ_l is the earliness of job j_l
- $O_{low(k,j)}^{up(k,j)}$ is the j-th operation scheduled on resource R_k (in the given schedule). In other words, $up(k,j)$ is the index of the job to which this opeation belongs and $low(k,j)$ the index of this operation within its job
- p_k is the number of operations requiring resource R_k
- The other notations are as defined in Section 2

Note that, in Equation (2), when job j_l is tardy, $\tau_l = st^l_{n_l} + du^l_{n_l} - dd_l \geq 0$ and $\epsilon_l = 0$, and, when it is early, $\epsilon_l = dd_l - (st^l_{n_l} + du^l_{n_l}) \geq 0$ and $\tau_l = 0$. A similar formulation was first proposed by Fry et al. for the one-machine early/tardy problem [4]. While more efficient procedures are described in the literature for the one-machine early/tardy problem, including an $O(NlogN)$ procedure developed by Garey et al. [6], it is not clear at this time how these procedures could be efficiently generalized to the job shop case.

Appendix B: Results Obtained Under Different Parameter Settings

This appendix summarizes results obtained with FSA for different values of the following four parameters:

- T_2: temperature above which FSA artificially inflates the costs of critical subproblems. The results in Table 3 were obtained using $FSA(CJ)$, the variation of FSA that performed best in our experiments. In these experiments, $k = 2$ and $p = 3$.

- k: the parameter by which FSA inflates the costs of critical subproblems. The results in Table 4 were obtained using $FSA(CJ)$, the variation of FSA that performed best in our experiments. In these experiments, $T_2 = 50$ and $p = 3$.

- p: the parameter used by $FSA(CJ)$ to identify critical jobs. Results obtained with different values of this parameter are summarized in Table 5. In these experiments, $T_2 = 50$ and $k = 2$.

- q: the parameter used by $FSA(CO)$ to identify critical operations. Results obtained with different values of this parameter are summarized in Table 6. In these experiments, $T_2 = 50$ and $k = 2$.

More detailed definitions of these parameters are provided in Section 5. The tables below report both average and best performance of FSA over 10 runs.

The best results are generally obtained for $T_2 = 50$, $k = 2$, $p = 3$ and $q = 3$, the values used in the experiments reported in Section 6.

Table 3 Percentage performance improvement(+)/degradation(-) observed when running FSA(CJ) with different values of T_2. Performance with $T_2 = 50$ is used as the reference.

Problem Set	Best Performance				Average Performance			
	$T_2 = 25$	$T_2 = 50$	$T_2 = 100$	$T_2 = 200$	$T_2 = 25$	$T_2 = 50$	$T_2 = 100$	$T_2 = 200$
1	-2.3	0.0	-7.7	-3.5	-2.4	0.0	-1.2	-0.2
2	-0.3	0.0	3.3	1.6	0.8	0.0	0.2	-4.1
3	-2.3	0.0	0.5	0.0	-2.4	0.0	-0.5	1.3
4	-3.7	0.0	-1.6	-0.9	-2.5	0.0	-0.7	0.7
5	-1.2	0.0	0.0	0.8	-0.1	0.0	-2.6	-1.2
6	-1.8	0.0	2.5	-0.5	-1.1	0.0	1.4	1.7
7	-0.6	0.0	-1.2	-4.0	-3.6	0.0	-0.1	-0.4
8	-2.1	0.0	-5.6	-6.8	-0.4	0.0	0.7	2.8
Overall	-1.8	0.0	-1.2	-1.7	-1.5	0.0	-0.4	0.1

Table 4 Percentage performance improvement(+)/degradation(-) observed when running FSA(CJ) with different values of k. Performance with $k = 2$ is used as the reference.

Problem Set	Best Performance				Average Performance			
	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 1$	$k = 2$	$k = 3$	$k = 4$
1	-3.8	0.0	-4.4	-0.3	1.0	0.0	0.8	0.1
2	-0.2	0.0	-1.2	-2.0	1.3	0.0	0.0	-1.3
3	-1.9	0.0	-2.6	3.2	-2.2	0.0	1.6	1.5
4	-2.2	0.0	-3.4	-3.6	0.4	0.0	-2.4	-1.1
5	-3.6	0.0	-2.8	-0.5	-1.1	0.0	-4.2	-12.1
6	-0.3	0.0	2.5	-0.9	-1.1	0.0	-2.4	-1.8
7	2.9	0.0	-5.2	2.1	3.2	0.0	0.1	-3.1
8	-3.6	0.0	-2.1	-5.0	2.9	0.0	3.0	2.5
Overall	-1.6	0.0	-2.4	-0.9	0.6	0.0	-0.4	-1.9

Table 5 Percentage performance improvement(+)/degradation(-) observed when running FSA(CJ) with different values of p. Performance with $p = 3$ is used as the reference.

Problem Set	Best Performance				Average Performance			
	$p = 1$	$p = 2$	$p = 3$	$p = 4$	$p = 1$	$p = 2$	$p = 3$	$p = 4$
1	-2.4	-1.6	0.0	0.3	0.6	1.8	0.0	-2.7
2	1.1	1.6	0.0	-1.7	2.2	1.2	0.0	-4.7
3	1.3	-1.0	0.0	-1.8	0.0	-0.6	0.0	0.9
4	1.2	1.2	0.0	3.7	-1.7	-1.3	0.0	2.5
5	-1.0	-4.1	0.0	-1.8	-32.1	-14.9	0.0	-0.2
6	2.3	1.0	0.0	-2.6	2.0	1.6	0.0	-0.4
7	-1.4	1.3	0.0	-3.1	-7.6	-2.7	0.0	0.4
8	-3.7	-2.2	0.0	-4.0	-5.2	2.1	0.0	1.6
Overall	-0.3	-0.5	0.0	-1.4	-5.2	-1.6	0.0	-0.3

Table 6 Percentage performance improvement(+)/degradation(-) observed when running FSA(CO) with different values of q. Performance with $q = 3$ is used as the reference.

Problem Set	Best Performance				Average Performance			
	$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 1$	$q = 2$	$q = 3$	$q = 4$
1	0.7	0.5	0.0	1.0	-0.8	0.8	0.0	-1.6
2	-1.7	-1.2	0.0	-2.6	0.4	0.8	0.0	-0.5
3	2.7	3.9	0.0	2.3	2.4	1.1	0.0	1.1
4	-3.6	-3.7	0.0	-8.4	-0.2	-0.4	0.0	-3.5
5	2.7	0.2	0.0	1.3	-17.3	-6.9	0.0	1.3
6	-4.7	1.2	0.0	1.5	-2.4	0.9	0.0	1.8
7	1.6	1.7	0.0	0.5	-2.8	0.1	0.0	0.8
8	-1.0	1.5	0.0	1.8	-5.2	-1.8	0.0	1.6
Overall	-0.4	0.5	0.0	-0.3	-3.2	-0.7	0.0	0.1

Increasing the Efficiency of Simulated Annealing Search by Learning to Recognize (Un)Promising Runs

Yoichiro Nakakuki and Norman M. Sadeh
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Simulated Annealing (SA) procedures can potentially yield near-optimal solutions to many difficult combinatorial optimization problems, though often at the expense of intensive computational efforts. The single most significant source of inefficiency in SA search is the inherent stochasticity of the procedure, typically requiring that the procedure be rerun a large number of times before a near-optimal solution is found. This paper describes a mechanism that attempts to learn the structure of the search space over multiple SA runs on a given problem. Specifically, probability distributions are dynamically updated over multiple runs to estimate at different checkpoints how promising a SA run appears to be. Based on this mechanism, two types of criteria are developed that aim at increasing search efficiency: (1) a *cutoff criterion* used to determine when to abandon unpromising runs and (2) *restart criteria* used to determine whether to start a fresh SA run or restart search in the middle of an earlier run. Experimental results obtained on a class of complex job shop scheduling problems show (1) that SA can produce high quality solutions for this class of problems, if run a large number of times, and (2) that our learning mechanism can significantly reduce the computation time required to find high quality solutions to these problems. The results further indicate that, the closer one wants to be to the optimum, the larger the speedups.

This paper appeared in the proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)[12].

This research was supported, in part, by the Defense Advanced Research Projects Agency under contract F30602-91-C-0016 and, in part, by an industrial grant from NEC.

1 Introduction

Simulated Annealing (SA) is a general-purpose search procedure that generalizes iterative improvement approaches to combinatorial optimization by sometimes accepting transitions to lower quality solutions to avoid getting trapped in local minima [8, 1]. SA procedures have been successfully applied to a variety of combinatorial optimization problems, including Traveling Salesman Problems [1], Graph Partitioning Problems [6], Graph Coloring Problems [7], Vehicle Routing Problems [14], Design of Integrated Circuits, Minimum Makespan Scheduling Problems [9, 13, 19] as well as other complex scheduling problems [23], often producing near-optimal solutions, though at the expense of intensive computational efforts.

The single most significant source of inefficiency in SA search is the inherent stochasticity of the procedure, typically requiring that the procedure be rerun a large number of times before a near-optimal solution is found. Glover et al. developed a set of "Tabu" mechanisms that attempt to increase the efficiency of SA and other neighborhood search procedures by maintaining a selective history of search states encountered earlier during the *same run* [4]. This history is then used to dynamically derive "tabu restrictions" or "aspirations", that guide search, preventing it, for instance, from revisiting areas of the search space it just explored. This paper describes a complementary mechanism that attempts to learn the structure of the search space *over multiple runs* of SA on a given problem. Specifically, we introduce a mechanism that attempts to predict how (un)promising a SA run is likely to be, based on probability distributions that are refined ("learned") over multiple runs. The distributions, which are built at different checkpoints, each corresponding to a different value of the temperature parameter used in the procedure, approximate the cost reductions that one can expect if the SA run is continued below these temperatures. Two types of criteria are developed that aim at increasing search efficiency by exploiting these distributions:

- *A Cutoff Criterion*: This criterion is used to detect runs that are unlikely to result in an improvement of the best solution found so far and, hence, should be abandoned;
- *Restart Criteria*: When completing a run or abandoning an unpromising one, these criteria help determine whether to start a fresh SA run or restart search in the middle of an earlier promising run.

The techniques presented in this paper have been applied to a class of complex job shop scheduling problems first described in [18]. Problems in this class require scheduling a set of jobs that each need to be completed by a possibly different due date. The objective is to minimize the sum of tardiness and inventory costs incurred by all the jobs. This class of problems is known to be NP-complete and is representative of a large number of actual scheduling problems, including Just-In-Time factory scheduling problems [18, 17]. Experimental results indicate (1) that SA can produce high quality solutions for this class of problems, if run a large number of times, and (2) that our learning mechanism can yield significant reductions in computation time. The results further indicate that, the closer one wants to be to the optimum, the larger the speedups.

The balance of this paper is organized as follows. Section 2 quickly reviews fundamentals of SA search. Section 3 analyzes the behavior of typical SA runs and introduces a mechanism that aims at learning to recognize (un)promising runs on a given problem, using the concept of *Expected Cost Improvement Distributions* (*ECID*). In Section 4, we use *ECID* distributions to develop a *cutoff criterion* to determine when to abandon unpromising runs. Section 5 presents three *restart criteria* based *ECID* distributions. Experiments obtained on a set of benchmark job shop scheduling problems with tardiness and inventory costs are reported in Section 6. A summary is provided in Section 7.

2 Simulated Annealing Search

Figure 1 outlines the main steps of a SA procedure designed to find a solution $x \in S$ that minimizes a real-valued function, $cost(x)$. The procedure starts from an initial solution x_0 (randomly drawn from S) and iteratively moves to other neighboring solutions, as determined by a neighborhood function, $neighbor(x)$, while remembering the best solution found so far (denoted by s). Typically, the procedure only moves to neighboring solutions that are better than the current one. However, the probability of moving from a solution x to an inferior solution x' is greater than zero, thereby allowing the procedure to escape from local minima. $rand()$ is a function that randomly draws a number from a uniform distribution on the interval $[0,1]$. The so-called temperature, T, of the procedure is a parameter controlling the probability of accepting a transition to a lower quality solution. It is initially set at a high value, T_0, thereby frequently allowing such transitions. If. after N iterations, the best solution found by the procedure has not improved, the temperature parameter T is decremented by a factor α $(0 < \alpha < 1)$. One motivation for progressively lowering the temperature is to obtain convergence. Additionally, as the procedure slowly moves towards globally better solutions, accepting transitions to lower quality solutions becomes increasingly less attractive. When the temperature drops below a preset level T_1, the procedure stops and s is returned (not shown in Figure 1).

```
$T = T_0$; $x = x_0\ (\in S)$; $min = \infty$;
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. 1 Basic Simulated Annealing Procedure.

Fig. 2 depicts the cost distribution of the best solutions returned by 300 SA runs on a typical combinatorial optimization problem – a job shop scheduling problem from a set of benchmarks to be described in Section 6.

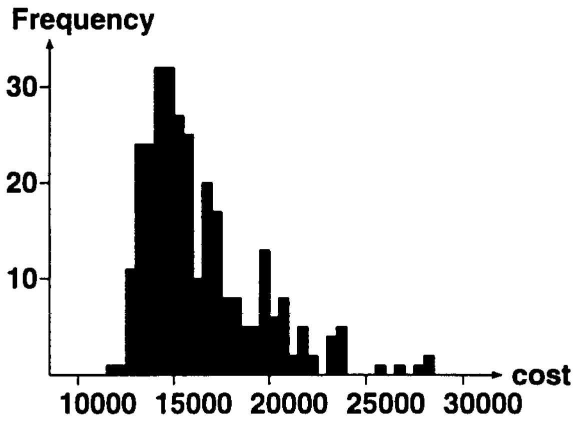


Fig. 2 Cost Distribution of the Best Solutions Found by 300 SA Runs.

The optimal solution for this problem is believed to have a cost around 11,500 – the value in itself is of no importance here. Figure 2 indicates that, if run a large number of times, SA is likely to eventually find an optimal solution to this problem. It also shows that, in many runs, SA gets trapped in local minima with costs much higher than the global minimum. For instance, 60% of the runs produce solutions with a cost at least 30% above the global minimum. This suggests that, if rather than completing all these unsuccessful runs, one could somehow predict when a run is likely to lead to a highly sub-optimal solution and abandon it, the efficiency of SA could be greatly enhanced. The following section further analyzes the behavior of typical SA runs and proposes a mechanism which, given a problem, aims at learning to recognize (un)promising SA runs.

3 Learning To Recognize (Un)promising SA Runs

Figure 3 depicts the behavior of a SA procedure on two different scheduling problems (from the set of benchmarks used in Section 6). For each problem, the figure depicts five SA runs, plotting the cost of the best solution, s, as the temperature of the procedure is progressively lowered – temperatures are shown in log scale, which is almost equivalent to computation time in linear scale. SA behaves very differently on these two problems. For instance, in Problem #1, the range of final solutions is relatively narrow, while in Problem #2 it is much wider. Another differentiating factor is the behavior of the procedure at low temperatures. It seems that for Problem #1, the quality of a run can already be estimated quite accurately at $T = 50$ (e.g. the best run at $T = 50$ remains best at lower temperatures), while this is less so for Problem #2.

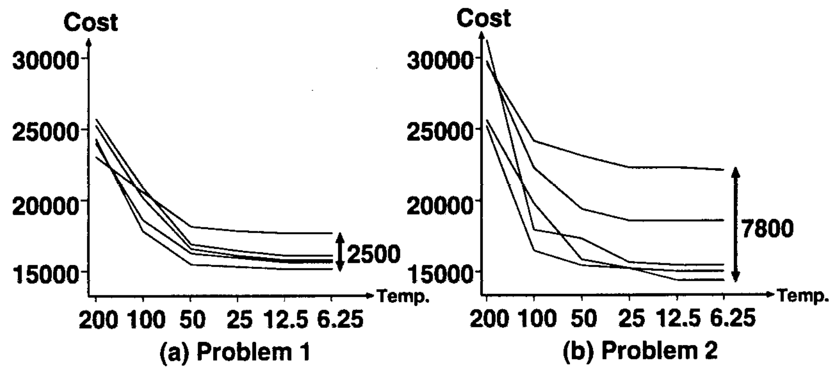


Fig. 3 Cost reductions in five SA runs on two different problems.

Clearly, such properties are not intrinsic to a problem itself. They could change if a different neighborhood structure or a different cooling profile was selected, as these parameters can affect the types of local optima encountered by the procedure and the chance that the procedure extricates itself from these local optima below a given temperature. While, in general, it may be impossible to find a SA procedure that reliably converges to near-optimal solutions on a wide class of problems, we can try to design adaptive SA procedures which, given a problem, can learn to recognize (un)promising runs and improve their performance over time. Below, we present a mechanism, which, given a problem, attempts to "learn" at different checkpoint temperatures the distribution of cost improvements that one can hope to achieve by continuing search below these temperatures.

Specifically, we postulate that, given a problem and a checkpoint temperature $T = t$, the distribution of the cost improvement that is likely to be achieved by continuing a run below t can be approximated by a normal distribution. Using performance data gathered over earlier runs on a same problem, it is possible to approximate these *Expected Cost Improvement Distributions* (*ECID*) for a set C of checkpoint temperatures and use these distributions to identify (un)promising runs.

Formally, given a combinatorial optimization problem and a SA procedure for that problem, we define c_i^t as the cost of the best solution, s, at check point t in the i-th run and c_i^0 as the cost of the best solution obtained at temperature $T = T_1$ in the i-th execution. When the $(n+1)$-st run reaches a checkpoint temperature t, the *ECID* below t is approximated as a normal distribution $N[\mu_n^t, \sigma_n^t]$, whose average, μ_n^t, and standard deviation, σ_n^t, are given by:

$$\mu_n^t = \frac{\sum_{i=1}^n (c_i^t - c_i^0)}{n}, \quad \sigma_n^t = \sqrt{\frac{\sum_{i=1}^n \{(c_i^t - c_i^0) - \mu_n^t\}^2}{n-1}}$$

By incrementally refining these estimators over multiple runs, this mechanism can in essence "learn" to recognize (un)promising SA runs. The following sections successively describe a cutoff criterion and three restart criteria based on *ECID* distributions.

4 A Cutoff Criterion

Suppose that, in a sixth run on Problem #1, the best solution obtained at checkpoint $T = 100$ is solution **A** – Figure 4(a). At this checkpoint, the distribution of c_6^0 – the cost of the best solution that will have been found if the run is completed – can be approximated by the normal distribution $N[c_6^{100} - \mu_5^{100}, \sigma_5^{100}]$. This distribution, represented in Fig. 4(a), suggests that, if continued, the current run has a good chance of improving the current best solution, **x**. Suppose that based on this analysis, the run continues until the next checkpoint, $T = 50$, and that the best solution found by the run when it reaches that temperature is **A'**. At this point, a new distribution of c_6^0 can be computed to check how the run is doing. This distribution, $N[c_6^{50} - \mu_5^{50}, \sigma_5^{50}]$ is shown in Figure 4(b). It appears much less promising than the one at $T = 100$. Now, the chances of improving the current best solution, **x**, appear remote: it probably does not make sense to continue this run.

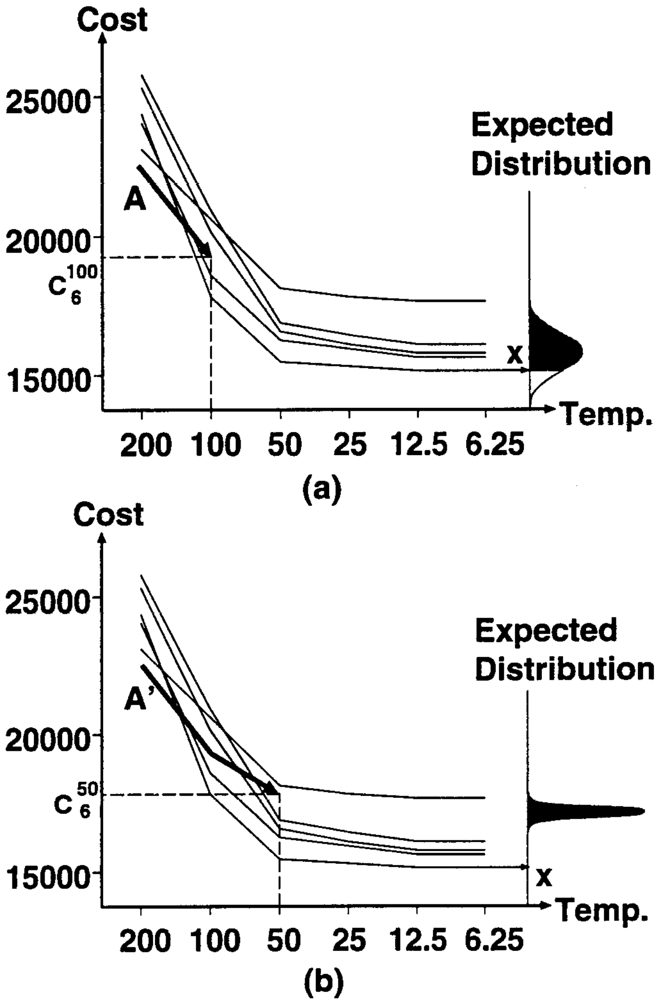


Fig. 4 Expected Cost Improvement Distributions at T=100 and T=50.

Formally, when the $(n + 1)$-st run reaches a checkpoint temperature t, a cutoff criterion is used to determine whether or not to continue this run. In the study reported in Section 6, we use a cutoff criterion of the form:

$$\frac{(c_{n+1}^t - \mu_n^t) - x_n}{\sigma_n^t} > threshold$$

where x_n is the cost of the best solution found during the previous n runs and *threshold* is a threshold value. If the inequality holds, the current run is abandoned. For example, if $threshold = 3$ (the value used in our experiments) and the cutoff inequality holds at a given checkpoint temperature t, the probability of improving x_n by continuing the run below t is expected to be less than 1% [2].

5 Three Restart Criteria

Whenever a run is completed or abandoned, two options are available: either start a fresh new annealing run *or*, instead, restart an earlier (promising) run, using a different sequence of random numbers ("reannealing"). In total, if reannealing is constrained to start from one of the checkpoint temperatures, there are up to $n \cdot |C| + 1$ possible options, where n is the number of earlier runs and $|C|$ the number of checkpoints in set C. Below, we describe three "restart criteria" that aim at selecting among these options so as to maximize the chances of quickly converging to a near-optimal solution.

5.1 Maximum Cost Reduction Rate Criterion

When considering several points from which to restart search, two factors need to be taken into account: (1) the likelihood that restarting search from a given point will lead to an improvement of the current best solution and (2) the time that it will take to complete a run from that point. Restarting from a low temperature will generally bring about moderate solution improvements, if any, while requiring little CPU time. Starting fresh new runs or restarting from higher temperatures can lead to more significant improvements, though less consistently and at the cost of more CPU time. In general, the cost improvements that can be expected from different temperatures will vary from one problem to another, as illustrated in Figure 3 (and as formalized by $ECID$ distributions).

A natural restart criterion is one that picks the restart point expected to maximize the rate at which the cost of the current best solution will improve. For each restart candidate O_k (fresh annealing or reannealing), this can be approximated as the expected cost reduction (in the best solution), if search is restarted from O_k, divided by the expected CPU time required to complete a run from that restart point. Below, we use $R(O_k)$ to denote this approximation of the expected cost reduction rate, if search is restarted from O_k:

$$R(O_k) = \frac{\textit{expected-reduction}(O_k)}{\textit{expected-CPU}(O_k)}$$

where *expected-reduction*(O_k) is the expected cost reduction at the end of a run starting from O_k and *expected-CPU*(O_k) is the CPU time that this run is expected to require. *expected-CPU*(O_k) can be approximated as the average time required to complete earier runs from O_k's temperature. *expected-reduction*(O_k) can be evaluated using $ECID$ distributions, as detailed below.

Given a reannealing point O_k at checkpoint temperature t and n earlier SA runs completed from t or above, *expected-reduction*(O_k) can be approximated as:

$$\textit{expected-reduction}(O_k) = \int_{LB}^{x_n} \{P_{nk}^t(x) \cdot (x_n - x)\} dx$$

where $P_{nk}^t(x)$ is the density function of the normal distribution $N[c_k - \mu_n^t, \sigma_n^t]$, c_k is the cost of O_k's best solution[1], x_n is the cost of the best solution obtained over the first n runs, and LB is a lower-bound on the optimal solution[2]

Similarly, if O_k is a fresh SA run, *expected-reduction*(O_k) can be approximated as:

$$\textit{expected-reduction}(O_k) = \int_{LB}^{x_n} \{P_n(x) \cdot (x_n - x)\} dx$$

where $P_n(x)$ is the density function of the normal distribution $N[\mu_n^0, \sigma_n^0]$, with

$$\mu_n^0 = \frac{\sum_{i=1}^n c_i^0}{n}, \quad \sigma_n^0 = \sqrt{\frac{\sum_{i=1}^n \{c_i^0 - \mu_n^0\}^2}{n-1}}.$$

5.2 Randomized Criterion

One possible problem with the above criterion is its high sensitivity to possible inaccuracies in approximations of $ECID$ distributions. This can be a problem when the number of earlier runs is still small. When inaccurate $ECID$ distributions lead the criterion to choose a poor restart point, the procedure may take a long time before it improves the quality of the current best solution. In the meantime, it may keep on coming back to the same poor restart point. For this reason, it is tempting to use a randomized version of the criterion. One such variation involves randomly picking from a set of promising restart points, $H = \{O_l | R(O_l) > \beta \cdot Max\{R(O_k)\}\}$, while assuming that each element in H has the same probability, $1/|H|$, of being selected. β is a constant whose value is between 0 and 1.

5.3 Hybrid Criterion

A third alternative involves keeping some level of stochasticity in the restart criterion, while ensuring that more promising restart points have a higher chance of being selected. This is done by selecting restart points in H according to a Boltzmann distribution that assigns to each element $O_l \in H$ a probability

$$p(O_l) = \frac{exp(R(O_l)/\tau)}{\sum_{O_k \in H} exp(R(O_k)/\tau))}$$

[1]To be consistent, if O_k correponds to the i-th SA run, $c_k = c_i^t$, as defined in Section 3.

[2]In the experiments reported in this paper, LB was simply set to 0.

Here, τ is a positive constant. If τ is very large, this method becomes equivalent to the randomized criterion described in subsection 5.2. If$\tau \approx 0$, this criterion becomes similar to the criterion of subsection 5.1. A similar distribution is used in the Q-learning algorithm described in [21].

6 Performance Evaluation

6.1 The Job Shop Scheduling Problem with Tardiness and Inventory Costs

To evaluate performance of our cutoff and restart criteria, we consider a set of complex job shop scheduling problems first introduced in [18]. The problems assume a factory, in which a set of jobs, $J = \{j_1, j_2, \cdots, j_n\}$, has to be scheduled on a set of resources, $RES = \{R_1, R_2, \cdots, R_m\}$. Each job requires performing a set of operations $O^l = \{O_1^l, O_2^l, \cdots O_{n_l}^l\}$ and, ideally, should be completed by a given due date, dd_l, for delivery to a customer. Precedence constraints specify a complete order in which operations in each job have to be performed. By convention, it is assumed that operation O_i^l has to be completed before operation O_{i+1}^l can start ($i = 1, 2, \cdots, n_l - 1$). Each operation O_i^l has a deterministic duration du_i^l and requires a resource $R_i^l \in RES$. Resources cannot be assigned to more than one operation at a time. The problem is to find a feasible schedule that minimizes the sum of tardiness and inventory costs of all the jobs ("Just-In-Time" objective). This problem is known to be NP-complete [18] and is representative of a large number of actual factory scheduling problems where the objective is to meet customer demand in a timely yet cost effective manner. Additional details on this model can be found in [18].

Experimental results reported below suggest that a good neighborhood function for this problem can be obtained by randomly applying one of the following three operators to the current schedule[3]:

- SHIFT-RIGHT: randomly select a "right-shiftable" operation and increase its start time by one time unit[4].
- SHIFT-LEFT (mirror image of SHIFT-RIGHT): randomly select a "left-shiftable" operation and decrease its start time by one time unit.
- EXCHANGE: randomly select a pair of adjacent operations on a given resource and permute the order in which they are processed by that resource. Specifically,

[3]In the scheduling jargon, the Just-In-Time objective considered in this study is known to be irregular[10]. Prior applications of SA to job shop scheduling have only considered regular objectives such as Minimum Makespan. It can be shown that the neighborhoods used in these earlier studies are not adequate to deal with irregular objectives such as the one considered here [16].

[4]An operation is said to be "right(left)-shiftable" if its start time can be increased (decreased) by one time unit without overlapping with another operation.

given two consecutive operations. A and B on a resource R, with A preceding B in the current solution, the exchange operator sets the new start time of B to the old start time of A and the new end time of A to the old end time of B [5].

In our experiments, the probability of picking the EXCHANGE operator was empirically set to 3/7 while the probabilities of picking SHIFT-RIGHT or SHIFT-LEFT were each set to 2/7. Additionally, the values of parameters in the SA procedure (see Figure 1) were set as follows: $T_0 = 700$, $T_1 = 6.25$, $N = 200,000$ and $\alpha = 0.85$.

The performance of this SA procedure has been evaluated in a comparison against 39 combinations of well-regarded dispatch rules and release policies previously used to assess the performance of the Sched-Star [11] and Micro-Boss [18, 17] systems on a set of 40 benchmark problems similar to the ones described in [18]. The 40 benchmarks consisted of 8 problem sets obtained by adjusting three parameters to cover a wide range of scheduling condition: an average due date parameter (tight versus loose average due date), a due date range parameter (narrow versus wide range of due dates), and a parameter controlling the number of major bottlenecks (in this case one or two). For each parameter combination, a set of 5 scheduling problems was randomly generated, thereby resulting in a total of 40 problems. Each problem involved 20 jobs and 5 resources for a total of 100 operations. On average, when compared against the best solution found on each problem by the 39 combinations of dispatch rules and release policies, SA reduced schedule cost by 15% (average over 10 SA runs). When comparing the best solution obtained in 10 SA runs against the best solution obtained on each problem by the 39 combinations of dispatch rules and release policies, SA produced schedules that were 34% better. However, while running all 39 combinations of dispatch rules and release policies takes a few CPU seconds on a problem, a single SA run takes about 3 minutes on a DECstation 5000/200 running C. Additional details on these experiments can be found in [16].

6.2 Empirical Evaluation of Cutoff and Restart Criteria

We now turn to the evaluation of the cutoff and restart criteria presented in this paper and compare the performance of five variations of the SA procedure presented in 6.1:

- **N-SA**: regular SA, as described in 6.1 (no learning).
- **P-SA**: SA with cutoff criterion.
- **B-SA**: SA with cutoff and Maximum Cost Reduction Rate restart criteria.

[5]In our implementation, exchanging two operations is allowed even if a precedence constraint is violated in the process. Precedence constraint violations are handled using large artificial costs that force the SA procedure to quickly get rid of them [16].

- **R-SA**: SA with cutoff and randomized restart criteria ($\beta = 0.5$).
- **H-SA**: SA with cutoff and hybrid restart criteria ($\beta = 0.5$ and $\tau = 1$).

When running P-SA, B-SA, R-SA, and H-SA, the cutoff and/or restart criteria were only activated after 5 complete SA runs to allow for the construction of meaningful *ECID* distributions. All four of these procedures used the same set of checkpoints, $C = \{200, 100, 50, 25, 12.5\}$.

The five procedures were compared on the same 40 benchmark problems described in subsection 6.1[6]. Each SA procedure was run for 2 hours on each benchmark problem. Furthermore, to eliminate possible noise effects, each two-hour experiment was repeated a total of 15 times. The results presented here were obtained by averaging performance of these 15 runs of each procedure on each problem.

Fig. 5 depicts the performance of the five SA procedures on a typical benchmark problem. The first 15 minutes are not represented, as they correspond to the first 5 runs when the cutoff and restart criteria have not yet been activated.

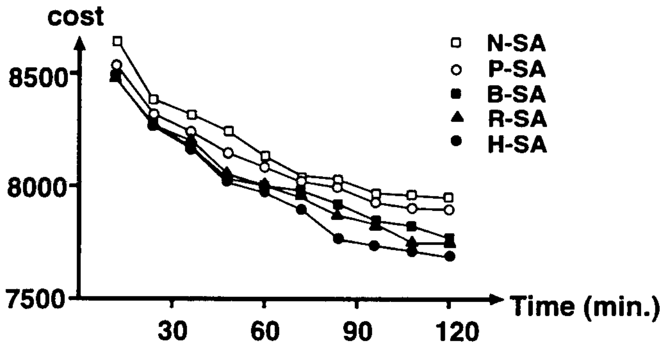


Fig. 5 Improvement of the best solution over time.

The figure shows that throughout its run, N-SA was dominated by the other four procedures. It also indicates that both the cutoff criterion and the restart criteria contributed to this performance improvement. Among the three restart criteria, H-SA appears to perform best. Figure 5 further suggests that the restart criterion in H-SA improves performance through the entire run, as the gap between H-SA and N-SA widens over time. These observations are confirmed by results obtained on the 8 problem sets of the study, as depicted in Figure 6. Fig. 6(a) shows the average cost reductions yielded by P-SA, B-SA, R-SA and H-SA over N-SA at the end of the two-hour runs. Figure 6(b) gives the average reduction in the CPU time required by each of these four procedures to find a solution of equal or better quality than the best

[6]At the present time, only a subset of the problems in each of the 8 problem sets have been completed. Complete results will be presented in the final version of the paper.

solution found by N-SA in two hours. It can be seen that H-SA requires between 30% and 70% less CPU time than N-SA.

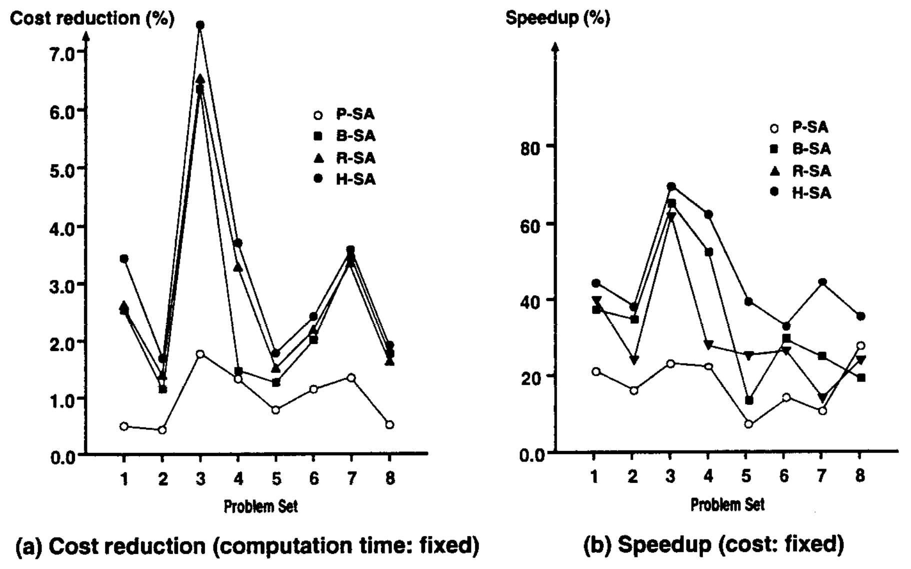


Fig. 6 Empirical comparison.

A finer analysis indicates that performance improvements produced by our cutoff and restart criteria increase as one requires higher quality solutions. Figure 7, compares the average CPU time of each of the five procedures as the required quality of solutions is increased. While all five procedures take about as long to find a solution with cost below 9000 or 8800, the time required to find a solution below 8500 varies significantly (e.g. H-SA can find such a solution in 3500 seconds while N-SA requires close to 10,000 seconds).

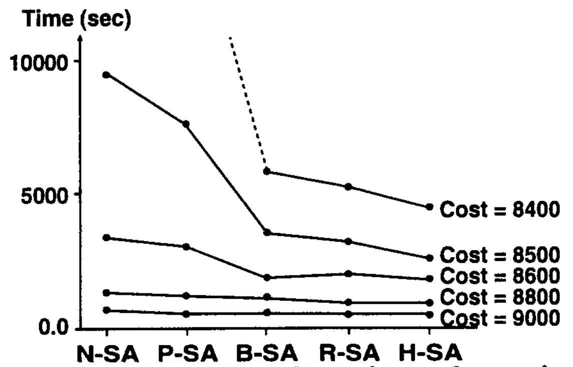


Fig. 7 Speedups as a function of required solution quality.

As already indicated in Section 5, the difference in performance between B-SA, R-SA and H-SA suggests that a deterministic use of *ECID* distributions to decide where to restart search can be tricky, as these distributions may not be accurate, especially when only a small number of runs has been completed. By injecting non-determinism in the restart criterion, R-SA and H-SA ensure that the procedure will not always restart from the same point. The procedure is forced to sample a wider area and in the process gets a chance to refine *ECID* distributions. From this point of view, B-SA is a procedure that places more emphasis on using existing knowledge of the search space than acquiring new one, while R-SA places more emphasis on learning and less on exploiting already acquired information. H-SA appears to provide the best balance between these two requirements.

Finally, it should be obvious that the CPU time and memory overheads of our cutoff and restart criteria are very moderate. All in all, in our experiments, the CPU time required to learn *ECID* distributions and apply the cutoff and restart criteria was well under 1% of total CPU time.

7 Summary

In summary, we have developed a mechanism that learns to recognize (un)promising SA runs by refining "Expected Cost Improvement Distributions" (*ECID*s) over multiple SA runs, and have developed search cutoff and restart criteria that exploit these distributions. These mechanisms can be applied to any SA procedure and have been validated on complex job shop scheduling problems with tardiness and inventory costs, where they have been shown to dramatically reduce the computational requirements of a competitive SA procedure. Experiments presented in this paper further indicate that the closer one seeks to be to the optimum, the larger the speedups.

References

[1] Cerny, V., "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," *J. Opt. Theory Appl.,* Vol. 45, pp. 41–51, 1985.

[2] Beyer, W.H. "CRC Standard Mathematical Tables, 28th Edition," CRC Press, Inc., Boca Raton, Florida, 1987.

[3] Garey, M. R. and Johnson, D. S. "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman and Co., 1979

[4] Glover, F. and Laguna M. "Tabu Search," To appear as a Chapter in *Modern Heuristic Techniques for Combinatorial Problems*, June 1992

[5] Graves, S. C. "A Review of Production Scheduling," *Operations Research* Vol. 29 no. 4, pp. 646–675, 1981

[6] Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. "Optimization by Simulated Annealing: Experimental Evaluation; Part I, Graph Partitioning" *Operations Research* Vol. 37 no. 6, pp. 865–892, 1989

[7] Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. "Optimization by Simulated Annealing: Experimental Evaluation; Part II, Graph Coloring and Number Partitioning" *Operations Research* Vol. 39 no. 3, pp. 378–406, 1991

[8] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science,* Vol. 220, pp. 671–680, 1983.

[9] Matsuo H., C.J. Suh, and R.S. Sullivan "A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem" Technical Report, "Department of Management, The University of Texas at Austin Austin, TX, Working Paper, 1988. 1992.

[10] Morton, T.E. and Pentico, D.W. "Heuristic Scheduling Systems," *Wiley Series in Engineering and Technology Management,* 1993

[11] Morton, T.E. "SCHED-STAR: A Price-Based Shop Scheduling Module," *Journal of Manufacturing and Operations Management,* pp. 131–181, 1988.

[12] Nakakuki,Y. and Sadeh,N. "Increasing the Efficiency of Simulated Annealing Search by Learning to Recognize (Un)Promising Runs," *Proceedings of the Twelfth National Conference on Artificial Intelligence.* To appear, 1994.

[13] Osman.I.H., and Potts, C.N., "Simulated Annealing for Permutation Flow-Shop Scheduling," *OMEGA Int. J. of Mgmt Sci.*, Vol. 17, pp. 551–557, 1989.

[14] Osman. I.H. "Meta-Strategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem" Technical Report, Institute of Mathematics and Statistics, University of Kent, Canterbury, Kent CT2 7NF, UK 1992.

[15] Palay, A. "Searching With Probabilities" PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, July 1983.

[16] Sadeh, N. and Nakakuki Y. "Focused Simulated Annealing Search: An Application to Job Shop Scheduling" Technical Report, The Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213 1994.

[17] Sadeh. N. "Micro-Opportunistic Scheduling: The Micro-Boss Factory scheduler" Morgan Kaufmann Publishers, 1993.

[18] Sadeh. N. "Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling" PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, March 1991.

[19] Van Laarhoven, P.J., Aarts, E.H.L., and J.K. Lenstra "Job Shop Scheduling by Simulated Annealing," *Operations Research,* Vol. 40, No. 1, pp. 113–125, 1992.

[20] Vepsalainen,A.P.J. and Morton, T.E. "Priority Rules for Job Shops with Weighted Tardiness Costs," *Management Science.* Vol. 33, No. 8, pp. 1035–1047, 1987.

[21] Watkins, C. J. C. M., "Learning with Delayed Rewards" PhD thesis, Cambridge University, Psychology Department, 1989.

[22] Wefald. E.H. and Russel, S.J. "Adaptive Learning of Decision-Theoretic Search Control Knowledge" *Sixth International Workshop on Machine Learning*, Ithaca, NY, 1989.

[23] Zweben, M., Davis E., Daun B., Deale M., "Rescheduling with Iterative Repair" Technical Report FIA-92-15, NASA Ames Research Center, Artificial Intelligence Research Branch, Moffett Field, CA 94025 April, 1992.

Case-based Acquisition of User Preferences for Solution Improvement in Ill-Structured Domains

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

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

Abstract

[1] [2] We have developed an approach to acquire complicated user optimization criteria and use them to guide iterative solution improvement. The effectiveness of the approach was tested on job shop scheduling problems. The ill-structuredness of the domain and the desired optimization objectives in real-life problems, such as factory scheduling, makes the problems difficult to formalize and costly to solve. Current optimization technology requires explicit global optimization criteria in order to control its search for the optimal solution. But often, a user's optimization preferences are state-dependent and cannot be expressed in terms of a single global optimization criterion. In our approach, the optimization preferences are represented implicitly and extensionally in a case base. Experimental results in job shop scheduling problems support the hypotheses that our approach (1) is capable of capturing diverse user optimization preferences and re-using them to guide solution quality improvement, (2) is robust in the sense that it improves solution quality independent of the method

[1]This research was partially supported by the Defense Advance Research Projects Agency under contract #F30602-91-C-0016. Most of the work was performed when the second author was a visiting scientist at the Robotics Institute at Carnegie Mellon University under the support of Matsushita Electric Industrial Co.

[2]This paper appeared in the Proceedings of AAAI-94, Seattle, Washington, August 1994.

of initial solution generation, and (3) produces high quality solutions, which are comparable with solutions generated by traditional iterative optimization techniques, such as simulated annealing, at much lower computational cost.

1 Introduction

We present an approach, implemented in the CABINS system, to demonstrate the capability of acquiring user context-dependent optimization preferences and reusing them to guide iterative solution optimization in ill-structured domains. This capability is very important for two main reasons. First, traditional search methods, both Operations Research-based and AI-based, that are used in combinatorial optimization, need explicit representation of objectives in terms of a cost function to be optimized [15]. In many practical problems, such as scheduling and design, optimization criteria often involve context- and user-dependent tradeoffs which are impossible to realistically consolidate in a cost function. Second, expert system approaches, while having the potential to capture context-dependent tradeoffs in rules, require considerable knowledge acquisition effort [14]. Our approach uses case-based reasoning (CBR) which has been successful in dealing with exceptional data [5, 16], acquiring user knowledge in complex domains [3, 10], and expending less effort in knowledge acquisition compared with knowledge acquisition for rule-based systems [9]. CABINS acquires, stores and reuses two categories of concepts that reflect user preferences (1) what heuristic local optimization action to choose in a particular context. and (2) what combinations of effects of application of a particular local optimization action constitutes an acceptable or unacceptable outcome. These are recorded in the case base and are used by CABINS to guide iterative optimization and induce optimization tradeoffs to evaluate the current solution. The optimization criteria are not explicitly represented as case features or in terms of a cost function but are implicitly and extensionally represented in the case base.

Previous case-based systems for incremental solution revision (e.g. [6, 18]) 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. Case-based knowledge acquisition systems, (e.g. [1]) require causal explanations from an expert teacher to acquire domain knowledge. In our approach neither the user nor the program are assumed to possess causal domain knowledge. The user's expertise lies in his/her ability to perform consistent evaluation of the results of problem solving and impart to the program cases of problem solving experiences and histories of evaluation tradeoffs.

In this paper, we present initial experimental results to test three hypotheses. First, our CBR-based incremental revision methodology shows good potential for capturing user optimization preferences in ill-structured domains, such as job shop scheduling, and re-using them to guide optimization. Second, the method is robust in the sense that it improves solution quality independent of the method of initial solution generation. Third, CABINS produces high quality solutions. To test this, we compared the solutions produced by CABINS with explicit optimization criteria, with solutions produced by simulated annealing (a well known iterative optimization technique [7, 19, 8]) for the same criteria. Our investigation was conducted in the domain of job shop schedule optimization and the experimental results, shown in section 5.1 confirmed these hypotheses.

2 Job Shop Schedule Optimization

The job shop scheduling problem is one of the most difficult NP-hard combinatorial optimization problems [4]. Job shop scheduling deals with allocation of a limited set of resources to a number of activities (operations) associated with a set of jobs so as to respect given temporal relations (e.g. precedence relations among activities), temporal constraints (e.g. job release and due dates) and resource capacity restrictions in order to optimize a set of objectives, such as minimize tardiness, minimize work in process inventory (WIP), maximize resource utilization etc. Due to the tight interactions among scheduling constraints and the often conflicting nature of optimization criteria, it is impossible to assess with any precision the extent of schedule revision or the impact of a scheduling decision on the global satisfaction of optimization criteria. For example, in figure 1 moving forward the last activity of ORDER3 creates downstream cascading constraint violations. Therefore, a repair action must be applied and its repair outcome must be evaluated in terms of the resulting effects on scheduling objectives. In addition, the evaluation itself of what is a "high quality" schedule is difficult because of the need to balance conflicting objectives and trade-off among them. Such tradeoffs typically reflect user preferences, which are difficult to express as a cost function. For example, WIP and weighted tardiness are not always compatible with each other. As shown in figure 2, there are situations where a repair action can reduce weighted tardiness, but WIP increases. Which is a better schedule depends on user preferences.

CABINS *incrementally revises a complete but sub-optimal schedule* to improve its quality, based on flexible optimization tradeoffs. Revision-based approaches to scheduling have also been investigated by [11, 19, 2, 8]. In those systems, the initial schedule is repaired by several techniques, such as the min-conflict heuristic or simulated annealing, to minimize the number of constraint violations or optimize a simple cost function (e.g. make-span) of the schedule. The value of incorporating context-dependent user preferences in operational scheduling environments is becoming increasingly recognized (e.g. [10]) but adequate techniques are lacking.

3 CABINS Overview

CABINS is composed of three modules: (1) an initial schedule builder, (2) an interactive schedule repair (case acquisition) module and (3) an automated

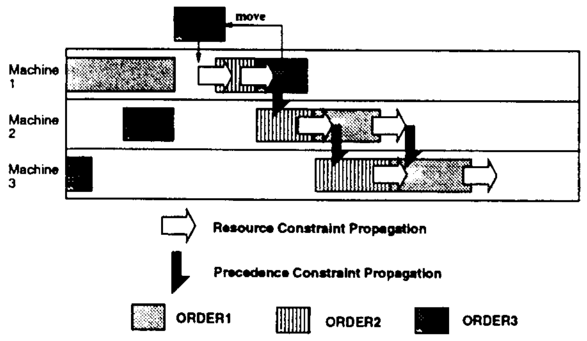

move
Machine 1
Machine 2
Machine 3
Resource Constraint Propagation
Precedence Constraint Propagation
ORDER1
ORDER2
ORDER3

Figure 1: **Example of Tight Constraint Interactions**

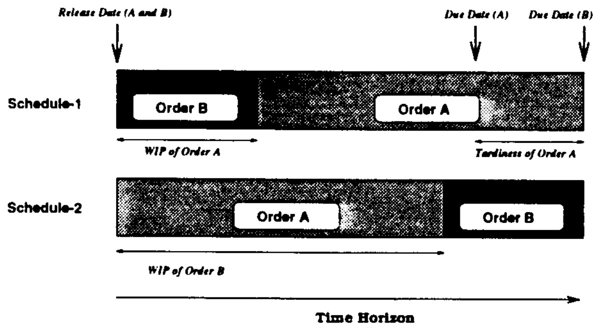


Figure 2: **Example of Conflicting Objectives**

schedule repair (case re-use) module. To generate an initial schedule, CABINS can use any of several scheduling methods (e.g. traditional dispatching rules or a constraint-based scheduler).

3.1 Case representation

In each repair iteration. CABINS focuses on one activity at a time, the *focal_activity*, and tries to repair it. A case in CABINS describes the application of a particular modification to a focal_activity. Figure 3 shows the information content of a case. Our assumption, borne out by the experimental results, is that despite the ill-structuredness of the domain, the global, local and repair history features express (in an approximate manner) domain regularities. The global features reflect an abstract characterization of potential repair flexibility for the whole schedule. High 'Resource Utilization Average', for example, often indicates a tight schedule without much repair flexibility. Associated with a focal_activity are local features that we have identified, based on those reported in [13], and which potentially are predictive of estimating the effects of applying a particular repair tactic to the schedule. For example, 'Predictive Shift Gain' predicts how much overall gain will be achieved by moving the current focal_activity earlier in its time horizon. In particular, it predicts the likely reduction of the focal_activity's waiting time when moved to the left within the repair time horizon.

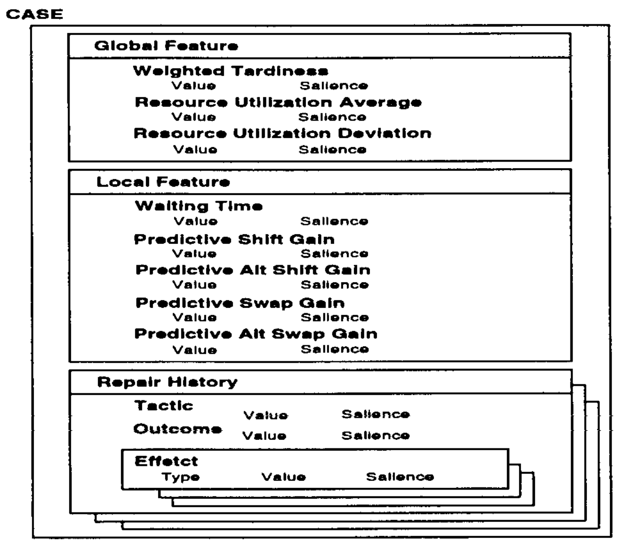


Figure 3: **CABINS Case Representation**

The repair history records the sequence of applications of successive repair tactics, the repair outcome and the effects. Repair effect values describe the impact of the application of a repair action on scheduling objectives (e.g. weighted tardiness, WIP). A repair outcome is the evaluation assigned to the set of effects of a repair action and takes values in the set ['acceptable', 'unacceptable']. Typically the outcome reflects tradeoffs among different objectives. If the application of a repair tactic results in a feasible schedule, the result is judged as either acceptable or unacceptable with respect to the repair objectives. An outcome is 'acceptable' if the user accepts the tradeoffs involved in the set of effects for the current application of a repair action. Otherwise, it is 'unacceptable'. The effect salience is assigned when the outcome is 'unacceptable', and it indicates the significance of the effect to the repair outcome. This value is decided subjectively and interactively. The user's judgment as to balancing favorable and unfavorable effects related to a particular objective constitutes the explanation of the repair outcome.

3.2 Case acquisition

To gather cases, sample scheduling problems are solved by a scheduler. CABINS identifies jobs that must be repaired in the initial sub-optimal schedule. Those jobs are sorted according to the significance of defect, and repaired manually by a user according to this sorting. For example, if the user's optimization criterion is to minimize order tardiness, the most tardy order is repaired first. The user selects a repair tactic to be applied. Tactic application consists of two parts: (a) identify the activities, resources and time intervals that will be involved in the repair, and (b) execute the repair by applying constraint-based scheduling to reschedule the activities identified in (a). Currently CABINS has 11 tactics and a flexible interface through which the user can define more.

After tactic selection and application, the repair effects are calculated and shown to the user who is asked to evaluate the outcome of the repair. If the user evaluates the repair outcome as 'acceptable', CABINS proceeds to repair another focal_activity and the process is repeated. If the user evaluates the repair outcome as 'unacceptable', s/he is asked to supply an explanation in terms of rating the salience/importance of each of the effects. The repair is undone and the user is asked to select another repair tactic for the same focal_activity. The process continues until an acceptable outcome for the current focal_activity is reached, or the repair is given up. Repair is given up when there are no more tactics to be applied to the current focal_activity; in this situation, CABINS carries on repair of another activity. The sequence of applications of successive repair actions, the effects, the repair outcome, and the user's explanation for failed application of a repair tactic are recorded in the repair history of the case. In this way, a number of cases are accumulated in the case base.

3.3 Case re-use

Once cases have been gathered, CABINS repairs sub-optimal schedules without user interaction. CABINS repairs the schedules by (1) recognizing schedule sub-optimalities, (2) focusing on a focal_activity to be repaired in each repair cycle, (3) invoking CBR with the set of global and local features as indices to decide the most appropriate repair tactic to be used for each focal_activity, (4) invoking CBR using the repair effect features (type, value and salience) as indices to evaluate the repair result, and (5) when the repair result is unacceptable, deciding which repair tactic to use next. Note that in contrast to traditional local iterative optimization approaches, (e.g. tabu search, simulated annealing) where the schedule generated in the current iteration as a result of local revision is directly compared (in terms of its associated cost function) with the current schedule, in CABINS, evaluation of the revision is provided by the case base, thus obviating the need for the presence of an explicit cost function.

The similarity between i-th case and the current problem is calculated as follows :

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

where SL_j^i is the salience of j-th feature of i-th case in the case-base, and its value has been heuristically defined by the user. CF_j^i is the value of j-th feature of i-th case, PF_j is the value of j-th feature in the current problem, E_D_j is the standard deviation of j-th feature value of all cases in the case-base. Feature values are normalized by division by a standard deviation of the feature value so that features of equal salience have equal weight in the similarity function.

4 An Example

We briefly illustrate the repair process with a very simple example schedule to be repaired shown in figure 4. The example has ten jobs $(J_1, \ldots, J_{10})$ and each job has five activities with linear precedence constraints. (e.g. O_1^n BEFORE O_2^n, ... , O_4^n BEFORE O_5^n). Resources R_1 and R_2, R_3 and R_5 are substitutable; resource R_4 is a bottleneck. Suppose that the job under repair is J_8. This job has a weight of 2, a due date of 1250 and the scheduled end-time of its last activity is 1390. Hence it has a weighted tardiness of $2 \times (1390 - 1250) = 280$. Suppose the current focal_activity is O_4^8. CBR is invoked with global features (weighted tardiness= 280, resource utilization average=0.544, resource utilization deviation=0.032) plus the set of local features as indices and selects swap as a repair tactic. One can see from the figure that this is a good choice since the focal_activity is scheduled on machine R_4, which doesn't have any substitutable machine and any idle time in the repair time horizon (time between the end of O_3^8 and the end of O_4^8).

To apply swap, CABINS calculates the activity with which O_4^8 will be swapped. To do this, CABINS selects the activity which, if swapped with O_4^8, will result in least amount of precedence constraint violations. In the example, activity O_4^4 is selected as the activity to be swapped with the current focal_activity O_4^8. Job J_4 has weight 3 and weighted tardiness $3 \times (1370 - 1320) = 150$. The effect of applying the swap tactic is that O_4^8 and O_4^4 are unscheduled on R_4 and O_4^8 is re-scheduled to start at time 1090 (the start time of activity O_4^4 prior to the swap). The repair process resolves occurring constraint violations. The repaired schedule is shown in figure 5.

The effects of repairing O_4^8 are calculated. CABINS calculates the effects on J_8 and J_4, the jobs affected by the application of the swap on O_4^8. Machine utilization did not change but J_8 had an estimated decrease in weighted-tardiness of 180 time units and an estimated decrease in WIP of 200 units, J_4 had an increase in weighted-tardiness of 150 units and an increase in WIP

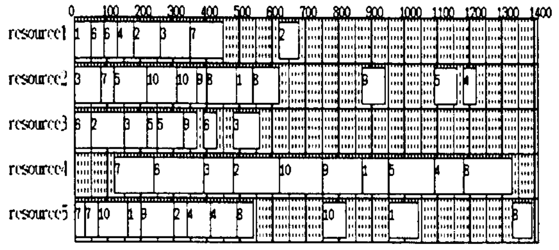


Figure 4: Original Schedule Results

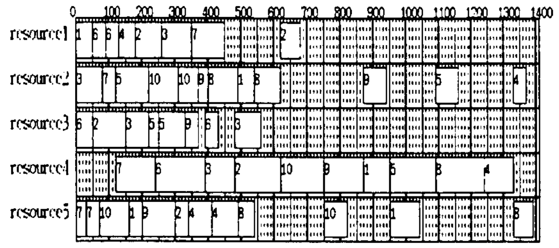


Figure 5: Schedule Results after Repair on O_4^8

of 750 units. CBR is invoked using these effect values, weighted tardiness, WIP, as indices to determine whether this repair outcome is acceptable. The acceptability or unacceptability of the repair will depend on the biases reflected in the case base.

5 Evaluation of the Approach

We conducted a set of experiments to test the hypothesis that (1) our CBR-based incremental modification and re-use methodology could be effective in capturing user schedule optimization preferences and re-using them to control schedule optimization, (2) the approach is robust in that the schedules produced by CABINS consistently improve a schedule independent of the method used for initial schedule generation and (3) as an iterative optimization method, the approach produces schedules of high quality. These hypotheses are difficult to test since, due to the subjective and ill-defined nature of user preferences, it is not obvious how to correlate scheduling results with the captured preferences or how to define quality of a schedule whose evaluation is subjective.

To address these issues, we had to devise a method to test the hypotheses in a consistent manner. To do that, it is necessary to know the optimization criterion that would be implicit in the case base, so that the experimental results can be evaluated. In the experiments reported here, we used two different explicit criteria (weighted tardiness; WIP+weighted tardiness) to reflect the user's optimization criterion and built a rule-based reasoner (RBR) that goes through a trial-and-error repair process to optimize a schedule. For each repair, the repair effects were calculated and, on this basis, since RBR had a predefined evaluation objective, it could evaluate the repair outcome in a consistent manner. Thus, we used RBR with different rules each time to generate different case bases (each with 1,000 cases) [3] for different explicit optimization objectives. Naturally, an objective, though known to us, is not known to CABINS and is only implicitly and indirectly reflected in an extensional way in each case base. By designing an objective into the RBR so it could be reflected in the corresponding case base we got an experimental baseline against which to evaluate the schedules generated by CABINS.

We evaluated the approach on a benchmark suite of 60 job shop scheduling problems where parameters, such as number of bottlenecks, range of due dates and activity durations were varied to cover a range of job shop scheduling problem instances with the following structure. Each problem class has 10 jobs of 5 operations each and 5 machines. Two parameters were used to cover different scheduling conditions: a range parameter controlled the distribution of job due dates and release dates, and a bottleneck parameter

[3]Since a case represents the application of one repair tactic to an activity, if, for example, 5 repair tactics are utilized in an attempt to successfully repair an activity, then 5 cases would be created.

controlled the number of bottleneck resources. Six groups of 10 problems each were randomly generated by considering three different values of the range parameter, and two values of the bottleneck configuration (1 and 2 bottleneck problems). These problems are variations of the problems originally reported in [17]. Our problem sets are, however, different in two respects: (a) we allow substitutable resources for non-bottleneck resources whereas the original problems did not, and (b) the due dates of jobs in our problems are tighter by 20 percents than in the original problems. We also tested the approach on another set of 60 problems of 20 orders and 5 resources with similar results.

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

5.1 Experimental Results

Figures 6 show the performance of CABINS using "weighted tardiness" case base (labeled in the figures as CABINS(WT)) vs performance of CABINS using the "weighted tardiness and WIP" case base (labeled in the figures as CABINS(WT+WIP)). The cases constituted the only source of knowledge for CABINS. In other words, there was no objective given to CABINS explicitly. The case-bases were used both as a source of suitable repairs, and also as a source of advice regarding repair evaluation. From the results we observe that CABINS(WT) generated higher quality schedules with respect to minimizing weighted tardiness than CABINS(WT+WIP). Conversely, CABINS(WT+WIP) generated higher quality schedules with respect to WIP, and weighted tardiness plus WIP than CABINS(WT). These results indicate that CABINS can acquire different and subjective user preferences.

In order to test the hypothesis that CABINS consistently improves schedule quality independent of the method of initial schedule generation, we generated initial schedules for the benchmark suite of problems using three different state-of-the-art dispatch scheduling heuristics (EDD, WSPT, R&M) [12] and a constraint-based scheduler (CBS). The optimization objective was

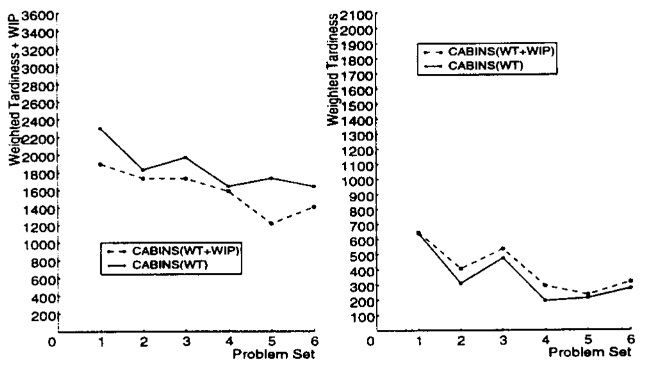


Figure 6: Scheduling Results with Different Case Bases

	Wei.Tar.	WIP	Total	CPU Sec.
EDD	956.0	1284.6	2240.6	0.1
CABINS	349.5	1311.2	1660.7	73.5
SA	340.5	1333.4	1673.9	388.2
WSPT	584.0	1241.0	1825.0	0.1
CABINS	321.0	1254.9	1575.9	72.1
SA	328.5	1320.4	1684.9	398.3
R&M	556.0	1242.0	1798.0	0.1
CABINS	305.3	1264.9	1570.2	84.9
SA	330.1	1290.8	1620.9	450.5
CBS	1173.0	1481.0	2654.0	17.4
CABINS	405.3	1195.0	1600.3	296.5
SA	395.5	1220.0	1615.5	1380.0

Table 1: Repair by CABINS and SA based on Different Methods of Initial Schedule Generation

WT+WIP. Table 1 presents the average of all 60 problems in the benchmark and shows that CABINS improved schedule quality independent of method to create the initial schedule. To test the hypothesis that CABINS generates schedules of high quality, we compared the schedules generated by CABINS against schedules generated by simulated annealing with the explicit objective of WT+WIP. Table 1 shows that CABINS generated schedules of comparable quality but was on the average 4-5 times more efficient than simulated annealing.

6 Conclusions

We have presented a case-based approach to acquire user optimization preferences and reuse them to guide iterative solution optimization in ill-structured domains. We demonstrated the effectiveness of the approach in capturing user preferences and creating efficiently high quality solutions on job shop scheduling problems. One crucial issue is how much effort should be spent to capture "enough" number of cases for "sufficient" solution quality improvement. This is an issue we are currently pursuing. Initial experiments to determine case base size versus quality improvement have shown that a case base of 800 cases gives on the average 20% higher quality improvement at 15% lower computational cost than a case base of 400 cases. It seems that the effort expended to capture a big number of cases can be amortized by future repeated use of the case base to get high quality schedules efficiently. More importantly, CABINS can acquire those cases from user's interaction during the process of solution improvement, thus imposing low additional effort on the user but enhancing solution improvement. We believe that CABINS has the potential for accommodating acquisition of user preferences that change over time. Future work will investigate this issue.

References

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

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

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

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

[5] Andrew R. Golding and Paul S. Rosenbloom. Improving Rule-Based Systems Through Case-Based Reasoning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 22–27. AAAI, 1991.

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

[7] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization By Simulated Annealing: An Experimental Evaluation, Part II (Graph Coloring and Number Partitioning). *Operations Research*, 1991.

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

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

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

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

[12] Thomas E. Morton. *HEURISTIC SCHEDULING SYSTEMS: With Application to Production Systems and Product Management.* GSIA, Carnegie Mellon University, Pittsburgh, PA., 1992. Course Textbook.

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

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

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

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

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

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

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

Improving Schedule Quality through Case-Based Reasoning

Kazuo Miyashita

Matsushita Electric Industrial Co. Ltd.,

2-7 Matsuba-cho, Kadoma, Osaka 571, JAPAN

miyasita@mcec.ped.mei.co.jp

Katia Sycara

The Robotics Institute

Carnegie Mellon University

Pittsburgh, PA 15213, U.S.A.

katia@cs.cmu.edu

Abstract

[1] [2]We describe a framework, implemented in CABINS, for iterative schedule revision based on acquisition and reuse of user optimization preferences to improve schedule quality. Practical scheduling problems generally require allocation of resources in the presence of a large, diverse and typically conflicting set of constraints and optimization criteria. The ill-structuredness of both the solution space and the desired objectives make scheduling problems difficult to formalize. CABINS records situation-dependent tradeoffs about repair actions and schedule quality 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 contributions of the work lie in experimentally demonstrating in a domain where neither the user nor the program possess causal knowledge of the domain that (a) taking into consideration failure information in the form of failed cases or a repair history of a case improves schedule quality, (b) schedule quality improves with increasing case size and (c) preserving the case base rather than inducing rules gives better results.

1 Introduction

Recently there has been increased interest in approaches that incrementally modify an artifact (e.g., program, plan, design) by reusing previous expe-

[1]This research was partially supported by the Defense Advance Research Projects Agency under contract #F30602-91-C-0016.

[2]This paper appeared in the Proceedings of the AAAI-93 Case-Based Reasoning Workshop, July 11-12, Washington, D.C., 1993.

riences in order to accommodate changed artifact specifications or recover from failures. Most current approaches have the following common characteristics: (1) they are motivated by considerations of computational efficiency [5, 13, 12], (2) they are concerned with preserving artifact correctness not addressing optimization issues [5, 13, 12, 4], and (3) they assume the existence of a strong domain model that is utilized to guide artifact modification and repair [5, 13, 12, 4]. For example, CHEF [4] uses rules rather than CBR for repair tactic selection, uses a model-based simulator for detecting failures in a generated plan, and addresses plan correctness issues (recovery from a failed plan) but ignores issues of plan optimization. Such characteristics limit current approaches in their ability to handle interesting real world tasks since the existence of a strong domain model can almost never be assumed and improving artifact quality (as opposed to only correctness) in terms of a set of evaluation criteria is often a crucial consideration.

We present an approach, implemented in CABINS, that demonstrates that reuse of previous relevant experiences is effective not only to ensure artifact correctness but also to improve quality. Through case-based reasoning (CBR), CABINS learns two categories of concepts: (1) what heuristic repair actions to choose in a particular repair context, and (2) what combinations of effects of application of a particular repair action constitutes an acceptable or unacceptable repair outcome in terms of optimization criteria. In contrast to the knowledge acquisition task [1] where the program interacts with an expert teacher to acquire domain knowledge, in our approach neither the user nor the program possess causal domain knowledge. The user cannot predict the effects of modification actions on artifact correctness or quality. In the domain of scheduling, for example, a modification could result in worsening schedule quality or introducing constraint violations (see next section). The user's expertise lies in his/her ability to perform consistent evaluation of the results of problem solving and impart to the program cases of problem solving experiences and histories of evaluation tradeoffs.

CABINS has been evaluated in the domain of iterative improvement of job shop schedules. Experimental results reported in [7] have shown that CABINS substantially increased schedule quality along a variety of optimization criteria (improvements ranged from 30-70 percent) without undue degradation in efficiency as compared with (1) a state of the art constraint-based scheduler, and (2) a variety of well regarded dispatch heuristics that are used in production management.

In contrast to approaches that utilize a single repair heuristic [6] or use a statically predetermined model for selection of repair actions [8], our approach utilizes a repair model that is incrementally learned and encoded in the case base. Learning allows dynamic selection and application of repair actions depending on the repair context. In [15] plausible explanation based learning has been successfully used to learn schedule repair control rules for speed up. Our experimental results show that in the context of CABINS, keeping the case base rather than inducing rules gives better results in terms of schedule quality.

In this paper we experimentally demonstrate that (a) taking into consideration failure information improves performance results, (b) result quality improves with increasing case size and (c) preserving the case base rather than inducing rules gives better results.

1.1 Task Domain

Scheduling assigns a set of jobs to a set of resources with finite capacity over time. One of the most difficult scheduling problem classes is job shop scheduling. Job shop scheduling is a well-known NP-complete problem [3]. In job shop scheduling, each job consists of a *set of activities* to be scheduled according to a *partial activity ordering.* Each job is assigned a release date, the date that it will be ready for starting processing, and a due date, a date on which the job should finish. Each activity within a job is assigned a set of *substitutable* resources on which the activity can be performed, and an activity duration. For example a drilling activity could be performed utilizing either a drilling machine or a milling machine. The dominant constraints in job shop scheduling are: temporal precedence constraints that specify the relative sequencing of activities within a job and resource capacity constraints that restrict the number of activities that can be assigned to a resource during overlapping time intervals.

The activity precedence constraints along with a job's release date and due date restrict the set of acceptable start times for each activity. The capacity constraints restrict the number of activities that can use a resource at any particular point in time and create conflicts among activities that are competing for the use of the same resource at overlapping time intervals. The goal of a scheduling system is to produce schedules that respect temporal relations and resource capacity constraints, and *optimize a set of objectives*, such as minimization of job tardiness (i.e., how late a job will finish), minimization of weighted tardiness (the sum of tardiness of all jobs, each weighted by its importance), minimization of work in process inventory (WIP) (i.e., the time a job spends in a factory waiting to be processed), maximization of resource utilization, etc.

CABINS *incrementally revises a complete schedule* to improve its quality. Revision consists in identifying and moving activities in the schedule. Because of the tight constraint interactions, a revision in one part of the schedule may cause constraint violations in other parts. It is generally impossible to (a) predict in advance either the extent of the constraint violations resulting from a repair action, or the nature of the conflicts. or (b) judge a priori the effects of a repair action on the optimization objectives. Therefore, a repair action must be applied and its repair outcome must be evaluated in terms of the resulting effects on scheduling objectives. The evaluation criteria are often context dependent and reflect *user preferences* with respect to tradeoffs. For example, WIP and weighted tardiness are not always compatible with each other. There are situations where WIP is reduced, but weighted tardiness increases. Tradeoffs are context dependent and therefore difficult to fully describe a priori even for a human expert. In CABINS, evaluation feedback is used to incrementally acquire context dependent schedule evaluation tradeoffs and their justifications. These are recorded in the case base and can be re-used to evaluate future schedule revision outcomes. Hence, preferences are reflected in the case base in two ways: as *preferences for selecting a repair action*, and as *evaluation preferences* for the repair outcome that resulted from selection and application of a specific repair action.

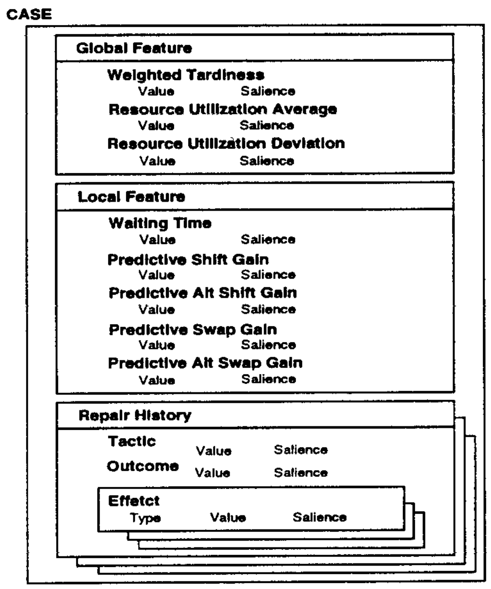


Figure 1: CABINS Case Representation

2 Overview of CABINS

2.1 Case representation

Within a job, repair is performed one activity at a time. At each iteration, the current job whose activity is being repaired is called the *focal_job* and the current activity being repaired is called the *focal_activity.* A case describes the application of a particular modification to an activity. Case indices are of three types (figure 1). First, there are features that reflect potential repair flexibility for the schedule as a whole, (global features). High resource utilization, for example, often indicates a tight schedule without much repair flexibility. High standard deviation of resource utilization indicates the presence of highly contended-for resources which in turn indicates low repair flexibility. Second, there are features that reflect flexibility for schedule revision within limited temporal bounds (local features). In particular, the temporal bound that CABINS uses is a time interval called *repair time horizon.* The repair time horizon of a focal_activity is the time interval between the end of the activity preceding the focal_activity in the same focal_job and the end of the focal_activity (see figure 2).

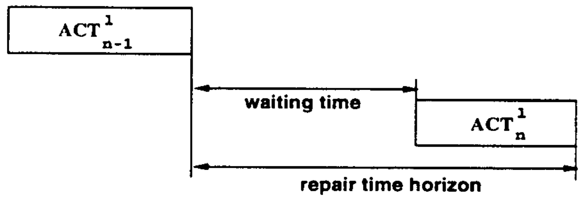


Figure 2: Repair time horizon of focal_activity(ACT^l_n)

Associated with the repair time horizon are local features that we have identified and which potentially are predictive of the effectiveness of applying a particular repair tactic. These features are in the same spirit as those utilized in [8]. For example, predictive-shift-gain predicts how much overall gain will be achieved by moving the current focal_activity earlier in its time horizon. In particular, it predicts the likely reduction of the focal_activity's waiting time when moved to the left within the repair time horizon. Because of the ill-structuredness of job shop scheduling, local and global features are heuristic approximations that reflect problem space characteristics.

The third category of case indices is a set of features that reflect the sequence of revisions to an activity (repair history). The repair history records the sequence of applications of successive repair actions, the repair effects and the repair outcome. Repair effects describe the impact of the application of a repair action on schedule optimization objectives (e.g., weighted tardiness, WIP). Typically these effects reflect tradeoffs among different objectives. A repair outcome is the evaluation assigned to the set of effects of a repair action and takes values in the set ['acceptable', 'unacceptable']. This judgement is made in the training phase and gets recorded in the case base. An outcome is 'acceptable' if the tradeoffs involved in the set of effects for the current application of a repair action is judged acceptable. If, during case acquisition, the outcome is judged as "unacceptable", the application of the repair tactic is considered a failure and an explanation that expresses tradeoffs with respect to balancing favorable and unfavorable outcomes on optimization objectives is provided. If during CBR repair the repair outcome is deemed unacceptable, another tactic is selected from success cases to repair the same activity, using as indices global and local case features, the failed tactic, and the indication of the failed outcome. This CBR invocation retrieves similar past failures of the tactic that were successfully repaired and the tactic that was eventually successful in fixing the past failure. The intuition here is that a similar outcome for the same tactic implies similarity of causal structure between the past and current case. Therefore, the eventually successful tactic of a similar failure can potentially be successful in the current problem.

2.2 Case acquisition

To gather enough cases, sample scheduling problems are solved by a constraint-based scheduler [11]. CABINS identifies jobs in a schedule that must be repaired. Those jobs are sorted according to the significance of defect, and repaired according to this sorting. For example, if the optimization criterion is to minimize job tardiness, the most tardy job is repaired first. A repair tactic is selected to be applied. Tactic application consists of two parts: (a) identify the activities, resources and time intervals that will be involved in the repair, and (b) execute the repair by applying constraint-based scheduling to reschedule the activities identified in (a). Repairing an activity, i.e., unscheduling it from its current position and re-scheduling at another time interval may cause conflicts with other activities. In each tactic application, the focal_activity and the conflicting activities are all re-scheduled. For details of the approach, see [7].

The tactics currently available in CABINS are:

left_shift : 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, so as to minimize the amount of resource capacity contention created by the move.

left_shift_into_alt : 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, so as to minimize the amount of resource capacity contention created by the move.

swap : swap the focal_activity with the activity on the *same* resource within the repair time horizon which causes the least amount of precedence constraint violations.

swap_into_alt : swap the focal_activity with the activity on a *substitutable* resource within the repair time horizon which causes the least amount of precedence constraint violations.

After tactic selection and application, the repair effects are calculated and evaluated. For example, repair of the current focal_activity may decrease WIP by 200 units and decrease weighted tardiness of the focal_job by

180 units while at the same time increasing weighted tardiness of another job by 130 units and increasing WIP by 300 units. If the repair outcome is evaluated as 'acceptable', CABINS proceeds to repair another activity and the process is repeated. If the evaluation of the repair outcome is "unacceptable", an explanation is supplied, the repair is undone and another repair tactic is selected for the same focal_activity. The process continues until an acceptable outcome for the current focal_activity is reached, or failure is declared. Failure is declared when there are no more tactics to be applied to the current focal_activity. The sequence of applications of successive repair actions, the effects, the repair outcome, and the explanation for failed application of a repair tactic are recorded in the repair history of the case. In this way, a number of cases are accumulated in the case base.

In the experiments reported here, we used a simple metric, minimizing weighted tardiness, [3] as an objective function to evaluate the performance of CABINS. Although there is no straightforward way to modify a schedule to optimize a realistic multi-criteria objective function, by using a single-criterion objective we built a rule-based reasoner (RBR) that goes through a trial-and-error repair process to optimize a schedule and forms an experimental baseline against which to compare CABINS. Since the RBR is constructed not to select the same tactic after tactic failure, it could go through all the tactics before giving up repairing an activity. For each repair, the repair effects are calculated and the repair outcome is correctly determined by comparing the change in the objective function. Since a clearly-defined objective function (which is available only in a user's mind) was used for

[3]Of course, CABINS does not know this metric but had to induce it from the contents of the case base.

evaluation, RBR can work as an emulator of a human scheduler, whose expertise lies in the ability of consistent evaluation. Therefore, we used RBR not only to make a comparison baseline for the CABINS experiment results but also to generate the case base for CABINS. So far, CABINS has been trained with 1,000 cases.

Once a case base is created, CABINS can repair a suboptimal schedule through CBR. CABINS repairs a schedule by (1) recognizing schedule suboptimalities, (2) focusing on an activity to be repaired in each repair cycle, (3) invoking CBR with global and local features as indices to decide the most appropriate repair tactic to be used for each activity, (4) invoking CBR using the repair effect features (type, value and salience) as indices to evaluate the repair result, and (5) in case of failure, deciding whether to give up or which repair tactic to use next by using global and local features and the repair history as indices. In the experimental study section, we report results about the effectiveness of indexing schemes that in situations of failure utilize different types of failure information.

2.3 Case retrieval

In CABINS concepts are defined extensionally by a collection of cases. As a case retrieval mechanism, CABINS uses a variation of k-Nearest Neighbor method (k-NN). [2] where not the frequency but the sum of similarity of k-nearest neighbors is used as a selection criterion. The similarity between i-th case and the current problem is calculated as follows :

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

where SL_j^i is the salience of j-th feature of i-th case in the case-base. Salience and values of features are numeric and have been heuristically defined by the user. CF_j^i is the value of j-th feature of i-th case, PF_j is the value of j-th feature in the current problem, E_D_j is a standard deviation of j-th feature value of all cases in the case-base. Feature values are normalized by division by a standard deviation of the feature value so that features of equal salience have equal weight in the similarity function.

3 Experimental Studies

To evaluate CABINS, we performed a set of controlled experiments where job shop schedule parameters, such as number of bottlenecks, range of due date, and activity durations were varied to cover a broad range of job shop scheduling problems. To ensure that we had not unintentionally hardwired knowledge of the problem into the solution strategies, we generated 60 job shop scheduling problems at random from problem generator functions where the above problem parameters were varied in controlled ways. Each problem has 5 resources and 10 jobs of 5 activities each. Each job has a process routing specifying a sequence where each job must visit bottleneck resources after a fixed number of activities, so as to increase resource contention and make the problem more difficult. We also varied job due dates and release dates, as well as the number of bottleneck resources (1 and 2). Six groups of 10 problems each were randomly generated by considering three different values of the due date range parameter (static, moderate, dynamic), and two values of the bottleneck configuration (1 and 2 bottleneck problems). The slack was adjusted as a function of the due date range and bottleneck parameters to keep demand for bottleneck resources close to 100 percent over the major part of each problem. Durations for activities in each job were also randomly generated. These problems are variations of the problems originally reported in [10]. Our problem sets are different in two respects: (a) we allow substitutable resources for non-bottleneck resources, and (b) the due dates of jobs in our problems are more constrained by 20 percent.

To make an accurate determination of CABINS' capabilities, we applied a two-fold cross-validation method. Each problem set in each class was divided in half. One half was repaired by the RBR emulator to gather cases. These cases were used to iteratively repair the other half of the problem set. We repeated the above process interchanging the sample set and the test set. Our results are the average of the two sets of results using case-based repair.

3.1 Evaluation of three repair strategies

Our hypothesis is that CBR enables CABINS to improve its competence both in repair quality and efficiency compared with RBR by utilizing different types of failure information recorded in the cases.

We experimentally compared three repair strategies:

(1) *one-shot repair*, where CABINS selects a repair tactic, applies it to a focal_activity and proceeds to repair the next focal_activity regardless of repair outcome.

(2)*exhaustive repair*, where CABINS selects a repair tactic and applies it to repair an activity. If the repair outcome is deemed unacceptable, another tactic is selected from success cases to repair the same activity, using as indices global and local case features, the failed tactic, and the indication of the failed outcome. This CBR invocation retrieves similar past failures of the tactic that were successfully repaired and the tactic that was eventually successful in fixing the past failure. The intuition here is that similar outcome for the same tactic imply similarity of causal structure between the past and current case. Therefore, the eventually successful tactic of a similar failure can potentially be successful in the current problem.

(3) *limited exhaustive repair*, where CABINS gives up further repair when it determines that it would be a waste of time. To decide whether to give up further repair, previous repair failed cases are utilized in conjunction with repair successes to determine case similarity. If the most similar case is a failure, CABINS gives up repair of the current activity and switches its attention to another activity. Since, in difficult problems, such as schedule repair, failures usually outnumber successes, if both case types are weighted equally, overly pessimistic results could be produced (i.e., CBR suggests giving up too often.) To avoid this, we bias (negatively) usage of failures by placing a threshold on the similarity value. Failure experiences whose similarity to the current problem is below this threshold are ignored in similarity calculations. Since the similarity metric selects the tactic which maximizes the sum of the most similar k cases, this biases tactic selection in favor of success cases which are moderately similar to the current problem.

The graphs in figure 3 show comparative results with respect to schedule quality improvement (weighted tardiness) and repair efficiency (in CPU secs) among limited exhaustive repair, exhaustive repair, one-shot repair and rule-based repair. The results show that one-shot repair is the worst in quality (even compared to rule-based repair) but best in efficiency. Exhaustive repair outperformed one-shot repair and rule-based repair in quality. But, the efficiency of exhaustive repair was worse than that of one-shot repair or

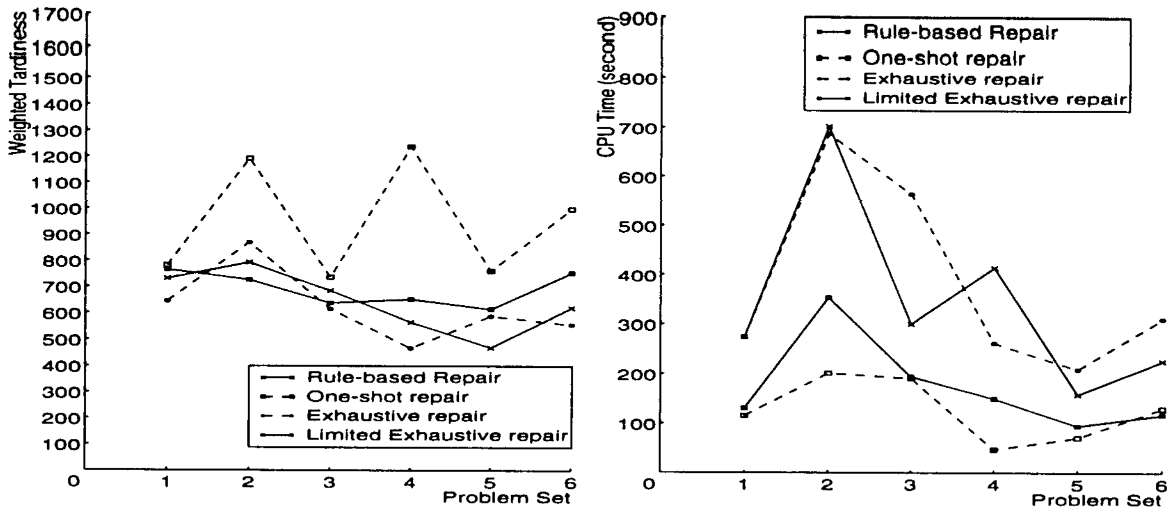
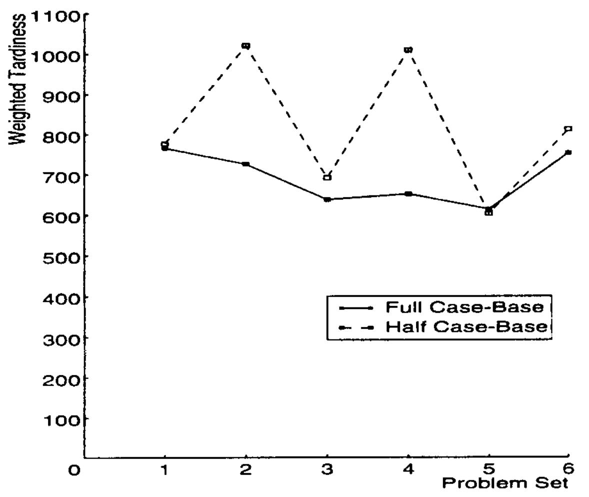


Figure 3: Effect of repair strategies in quality and efficiency

rule-based repair. We believe that this result stems from the following two reasons: (1) greediness - exhaustive repair applies the tactic from the most similar cases no matter how small their similarity is, and (2) stubbornness - exhaustive repair continues to repair an activity without giving up when the problem seems difficult. The quality of repair by limited exhaustive repair is only slightly worse than that by exhaustive repair, but is still comparable with that of rule-based repair. The efficiency of limited exhaustive repair is much higher than both rule-based repair and exhaustive repair. Although the efficiency of limited exhaustive repair is comparable with that of one-shot repair, the quality of repairs by limited exhaustive repair is much better than that of one-shot repair. With respect to repair quality, we can observe the following: (1) one shot repair does not have enough information to induce an adequate repair model, and (2) prediction accuracy can be improved by using information about failed application of a repair tactic as an additional index feature.

3.2 Comparison with different sized case-bases



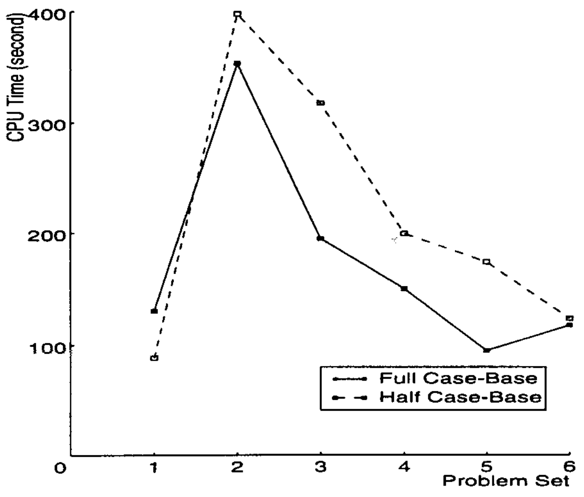


Figure 4: Effects of case-base size in quality and efficiency

The graphs in figure 4 show the comparison of CABINS' performance with different sized case-bases. In the experiments, we randomly chose half of the cases in the original case-base (used in the comparative repair strategy experiments) and created a new case-base. Then, we solved the same sixty problems by limited exhaustive repair with each of the case-bases. The graphs depict that CABINS with full case-base outperforms CABINS with half case-base both in quality and efficiency. This means that, as the case base of CABINS is enriched, its competence increases.

3.3 Comparison of CBR and rule induction

We tested the hypothesis that keeping the cases rather than inducing rules for repair tactic selection would result in better quality repairs. The graphs in figure 5 show the comparison of CABINS' performance with case-based rea-

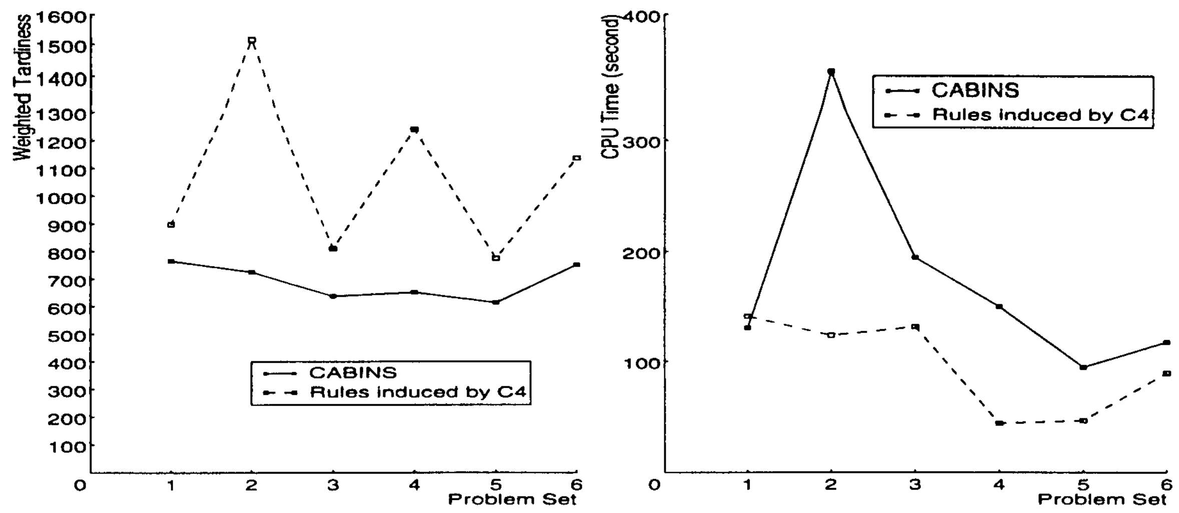


Figure 5: Comparison of CABINS and C4 in quality and efficiency

soning and CABINS' performance with rules induced from the case base by C4, a decision tree induction algorithm (descendant of ID3) [9]. The results show that CABINS' performance with C4-induced rules is better with respect to efficiency but much poorer in terms of quality than CABINS' performance with case-base. This drawback of C4 stems from the fact that geometrically, C4 (and most of other decision-tree induction programs) can't produce non-rectangular decision regions in the decision space. In the rules used for repairing schedules and creating cases, there are many conditions specifying the relationships of attributes, such as `If attribute-A is greater than attribute-B, then C`. To approximate the decision behavior of the nonrectangular regions produced by those rules, C4 has to fit many small rectangle sections in the form of a staircase function, which requires more training data.[14, 9]

4 Conclusions

We described a framework for acquisition and reuse of past problem solving experiences for plan revision in domains, such as job shop scheduling, without a strong domain model. Our experimental results show that our methodology can outperform rule based methods, and improve its own performance by: (1) using failures and their repairs as additional indices, and (2) trading off the use of success and failure cases depending on the context in which a repair tactic is applied. In addition, our experimental results showed that increasing case base size improves both quality and efficiency. Finally, CBR techniques used in CABINS, though lower in efficiency, result in superior solution quality compared with rule induction.

5 Acknowledgments

This work was done when the first author was a visiting scientist at the Robotics Institute of Carnegie Mellon University under the support of Matsushita Electric Industrial Co... C4 program used in this paper was implemented by Dr. Wray Buntine at NASA Ames Research Center as a module of his IND software package.

References

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

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

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

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

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

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

[7] Kazuo Miyashita and Katia Sycara. Adaptive case-based control of schedule revision. In M. Fox and M. Zweben, editors, *Knowledge-Based Scheduling.* Morgan Kaufmann, San Mateo, CA, 1993.

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

[9] J. Ross Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publisher, Inc., San Mateo, CA, 1993.

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

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

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

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

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

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

Distributed Problem Solving through Coordination in a Society of Agents[1][2]

JyiShane Liu Katia Sycara
jsl@cs.cmu.edu katia@cs.cmu.edu
Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a methodology, called Constraint Partition and Coordinated Reaction (CP&CR), where a problem solution emerges from the evolving computational process of a group of diverse, interacting, and well-coordinated reactive agents. ·Problem characteristics are utilized to achieve problem solving by asynchronous and well coordinated local interactions. The coordination mechanisms guide the search space exploration by the society of interacting agents, facilitating rapid convergence to a solution. Our domain of problem solving is constraint satisfaction. We have applied the methodology to job shop scheduling with non-relaxable time windows, an NP-complete constraint satisfaction problem. Utility of different types of coordination information in CP&CR was investigated. In addition, experimental results on a benchmark suite of problems show that CP&CR performed consid-

[1]In the Proceedings of the 13th International Workshop on Distributed Artificial Intelligence (1994)

[2]This research was partially supported by the Defense Advance Research Projects Agency under contract #F30602-91-C-0016.

erably well as compared to other centralized search scheduling techniques, in both computational cost and number of problems solved.

1 Introduction

Distributed AI (DAI) has primarily focused on Cooperative Distributed Problem Solving [4] [5] [10] by sophisticated agents that work together to solve problems that are beyond their individual capability. Another trend has been the study of computational models of agent societies [13], composed of simple agents that interact asynchronously. With few exceptions (e.g. [1][7][23]), these models have been used to investigate the evolutionary behavior of biological systems [12] [17] rather than the utility of these models in problem solving. We have developed a computational framework for problem solving by a society of simple interacting agents and applied it to solve job shop scheduling Constraint Satisfaction Problems (CSPs). Experimental results, presented in section 4, show that the approach performs considerably well as compared to centralized search methods for a set of benchmark job shop scheduling problems. These encouraging results indicate good problem solving potential of approaches based on distributed agent interactions.

Many problems of theoretical and practical interest (e.g., parametric design, resource allocation, scheduling) can be formulated as CSPs. A CSP is defined by a set of *variables* $X = \{x_1, x_2, \cdots, x_m\}$, each having a corresponding *domain* $V = \{v_1, v_2, \cdots, v_m\}$, and a set of *constraints* $C = \{c_1, c_2, \cdots, c_n\}$. A constraint c_i is a subset of the Cartesian product $v_l \times \cdots \times v_q$ which specifies which values of the variables are compatible with each other. The *variable set* of a constraint (or a set of constraints), denoted by *vs( )*, is the set of non-duplicating variables restricted by the constraint (or the set of constraints). A solution to a CSP is an assignment of values (an instantiation) for all variables, such that all constraints are satisfied. Numerical CSPs (NCSPs) [14] are a subset of CSPs, in which constraints are represented by numerical relations between quantitative variables usually with fairly large domains of possible values. Many CSPs of practical importance, such as scheduling, and parametric design, are NCSPs. Constraint satisfaction algorithms typically suffer from feasibility/efficiency problems for NCSPs due to their enormous search spaces.

In general, CSPs are solved by two complementary approaches, backtracking and network consistency algorithms [16][2][21]. Recently, heuristic revision [18] and decomposition [3][8] techniques for CSPs have been proposed. On the other hand, recent work in DAI has considered the distributed CSPs [11] [25] [27] in which variables of a CSP are distributed among agents. Each agent has an *exclusive* subset of the variables and has sole responsibility to instantiate their values. Instead, our approach provides a decomposition scheme in which constraint type as well as constraint connectivity are used. This results in no inter-agent constraints, but each variable may be instantiated by more than one agent. While satisfying its own constraints, each agent instantiates/modifies variable values based on coordination information supplied by others. Coordination among agents facilitates effective problem solving.

In this paper, we present an approach, called Constraint Partition and Coordinated Reaction (CP&CR), in which a job shop scheduling NCSP is collectively solved by a set of agents with simple local reactions and effective coordination. CP&CR divides an NCSP into a set of subproblems according to constraint type and assigns each subproblem to an agent. Interaction characteristics among agents are identified. Agent coordination defines agent roles, the information they exchange, and the rules of interaction. The problem solution emerges as a result of the evolving process of the group of interacting and coordinating agents. The remainder of the paper is organized as follows. In Section 2, we define the CP&CR model, in which problem decomposition, coordination mechanisms, and asynchronous search procedure are specified. In Sections 3 and 4, we describe an application of CP&CR to job shop scheduling with non-relaxable time windows, and present comparative results on previously studied test problems. Finally, in Section 5, we conclude the paper and outline our current work on CP&CR.

2 CP&CR Model

CP&CR is a problem-solving framework in which a society of specialized and well-coordinated agents collectively solve a problem. Each agent reacts to others' actions and communicates with others by leaving and perceiving particular messages on the objects it acts on. A solution emerges from the evolutionary interaction process of the society of diverse agents. Specifically, CP&CR provides a framework to decompose an NCSP into a set of subproblems based on constraint type and constraint connectivity, identify their interaction characteristics and, accordingly construct effective coordination mechanisms. CP&CR assumes that an NCSP has at least two types of constraints.

2.1 Constraint Partition & Problem Decomposition

Constraints label relations between variables that specify how the values of variables are restricted for compatibility. We formally define constraint characteristics (e.g., constraint type, constraint connectivity) for NCSPs.

Definition 1: Constraint Type - In CP&CR, quantitative constraints are classified by differences in the numerical compatibility between two variables. We identify four types of quantitative constraints. In Figure 1, a black dot represents a value, v_i, that has been assigned to a variable, x_i. An empty dot represents the only possible value for the other variable, x_j. A shaded region (either open or closed) represents an interval within which an instantiation of the other variable, x_j, will violate the constraint.

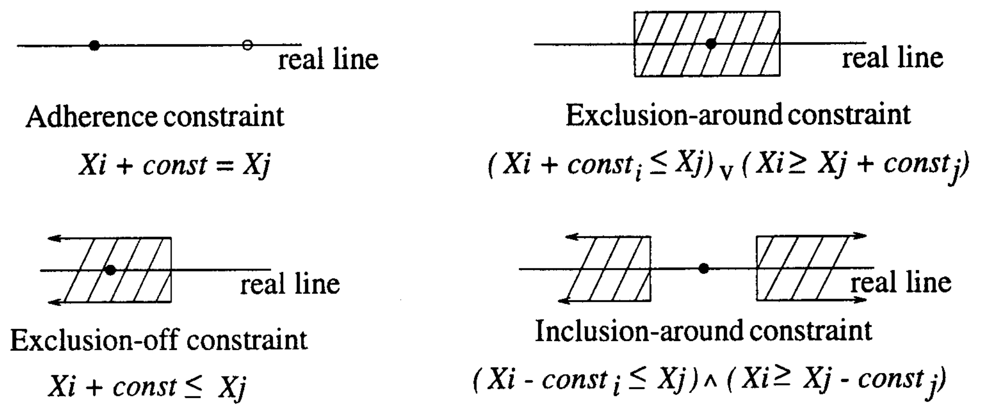

real line
Adherence constraint
$Xi + const = Xj$
real line
Exclusion-around constraint
$(Xi + const_i \leq Xj) \vee (Xi \geq Xj + const_j)$
real line
Exclusion-off constraint
$Xi + const \leq Xj$
real line
Inclusion-around constraint
$(Xi - const_i \leq Xj) \wedge (Xi \geq Xj - const_j)$

Figure 1: Constraint types classification

1. *adherence type* - A constraint is of adherence type if the instantiation of a variable, x_i, to the value v_i restricts the instantiation of another variable. x_j, to an individual point in the domain. For example, $x_i + const = x_j$.

2. *exclusion-around type* - A constraint is of exclusion-around type if the instantiation of a variable, x_i, to the value v_i restricts the instantiation of another variable, x_j, from a subsection within certain distances from v_i. For example, $x_i + const \neq x_j$, or $(x_i + const_i \leq x_j) \vee (x_i \geq x_j + const_j), const_i, const_j > 0$.

3. *exclusion-off type* - A constraint is of exclusion-off type if the instantiation of a variable, x_i, to the value v_i restricts the instantiation of another variable, x_j, from a connected subsection of the domain with a boundary set by v_i. For example, $x_i + const \leq x_j$.

4. *inclusion-around type* - A constraint is of inclusion-around type if the instantiation of a variable, x_i, to the value v_i restricts the instantiation of another variable, x_j, within a connected subsection of the domain with boundaries set by v_i. For example, $(x_i - const_i \leq x_j) \wedge (x_i \geq x_j - const_j), const_i, const_j > 0$.

We illustrate how our definitions can describe the constraints of some well known CSPs. In the N-Queen problem, both vertical and diagonal attack constraints are of exclusion-around type. In the Zebra problem, association constraints (e.g. the Englishman lives in the red house.) are of adherence type, and single-occupancy constraints (e.g. each attribute, such as pet, color, etc., must be assigned to each house.) are of exclusion-around type.

Definition 2: Constraint Connectivity - Two constraints are said to be *connected* iff the intersection of their variable sets is not empty. This implies that they have constrained variables in common.

c_p and c_q are connected $\equiv vs(c_p) \cap vs(c_q) \neq \emptyset$.

Definition 3: Constraint Partition is a scheme to decompose an NCSP into a set of subproblems by constraint type and constraint connectivity (see Figure 2). Two types of constraint grouping, *constraint bunch*, and *constraint cluster*, are introduced by the decomposition scheme.

A constraint bunch, C_i, is a set of constraints of the same constraint type. Define an operator, *pb( )*, which partitions the constraint set C of an NCSP into a set of constraint bunches, C_i, according to constraint type. Denote the resulting set of constraint bunches by C'. Define an operator, denoted by *type*( ), which identifies the constraint type of a constraint bunch. A constraint bunch has the following properties.

- $pb(C) = \{C_i\} = C'$
- C_i *partition* C
- $type(C_i) \neq type(C_j), i \neq j$

A constraint cluster, $C_{i,m}$, is a set of constraints which are of the same constraint type and are connected to each other. Define an operator, *pc( )*, which partitions a set of constraint bunches into a set of constraint clusters, C'', according to constraint connectivity. A constraint cluster has the following properties.

- $pc(C') = \{C_{i,m}\} = C''$
- *Constraint clusters of the same constraint type have no variables in common*
- *If a constraint cluster contains more than one constraint, each constraint is connected to at least one other constraint in the constraint cluster*

By constraint partition, an NCSP is decomposed into a set of subproblems, each of which is concerned with the satisfaction of constraints in a constraint cluster, and is assigned to an agent. A solution to a subproblem

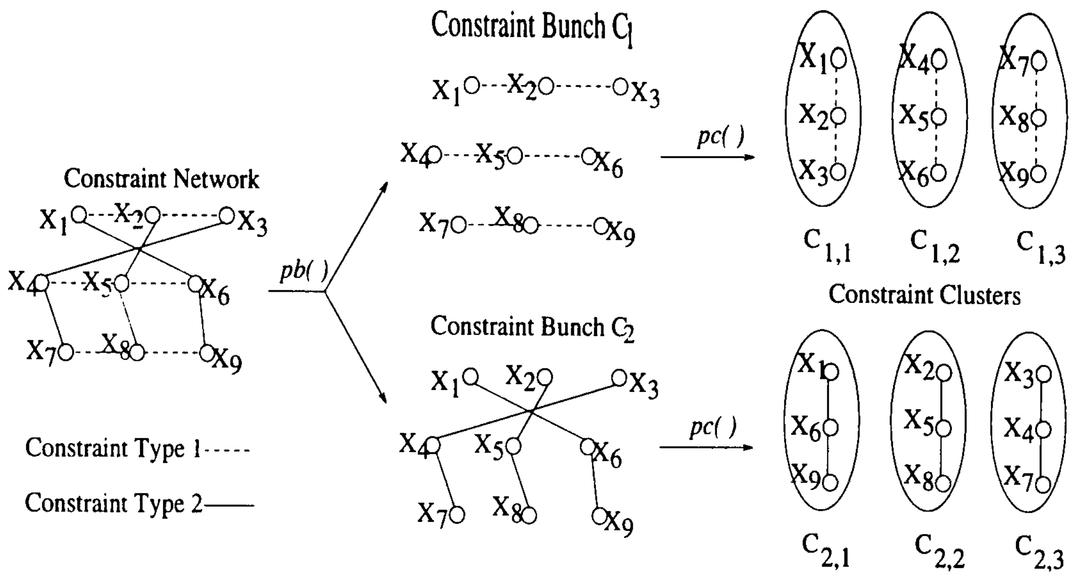


Figure 2: Constraint partition

is an instantiation of the variables in the constraint cluster such that none of the constraints in the subproblem are violated. Each agent has full jurisdiction over variables in the variable set of the assigned constraint cluster. A variable constrained by more than one type of constraint is under the jurisdiction of more than one agent. Agents responsible for the same variable have the same authority on its value, i.e. they can independently change its value. Therefore, a given value of a given variable is part of a solution, if it is *agreed* upon by all its responsible agents in the sense that no agent seeks to further change it. When all subproblems are solved, a solution of the NCSP is found.

2.2 A Society of Reactive Agents

In the framework of CP&CR, problem solving of an NCSP is transformed into collective behaviors of reactive agents. Variables of an NCSP are regarded as objects which constitute agents' environment. An instantiation of the variables characterizes a particular state of the environment. Each agent examines and makes changes to only local environment (variables under its responsibility), and seeks for satisfaction by ensuring that no constraint in its assigned constraint cluster is violated. When an agent detects constraint violations, it reacts by changing the instantiated values of some of the variables under its jurisdiction so that it becomes satisfied.

Agents are equipped with only primitive behavior. When activated, each agent reacts to the current state of the environment by going through an Examine-Resolve-Encode cycle (see Figure 3). It first examines its local view of current environment, i.e. the values of the variables under its jurisdiction. If there are constraint violations, it changes variable instantiations to resolve conflicts according to innate heuristics and coordination information.

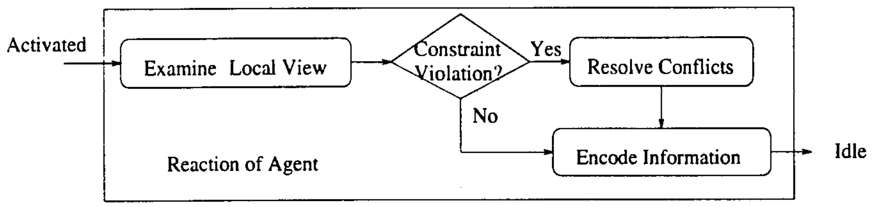


Figure 3: Agent's reactive behavior

Agents coordinate by *passive* communication. They do not communicate with each other directly. Instead, each agent reads and writes coordination information on objects under its jurisdiction. Coordination information on an object represents an agent's partial "view" on the current state of the environment and is consulted when other agents are considering changing the current instantiation of the variable to resolve their conflicts. Each time an agent is activated and has ensured its satisfaction, it writes down its view on current instantiations on each variable under its jurisdiction as coordination information.

Agents are divided into subgroups according to perspective (e.g., constraint type). For example, in job shop scheduling problems, one agent subgroup is responsible for resolving capacity constraints, whereas another agent subgroup is responsible for resolving temporal constraints. A variable is under the jurisdiction of agents from different perspectives. Agent subgroups of different perspectives are activated in turn, while agents within a subgroup can be activated simultaneously. An *iteration cycle* is an interval in which all agents are activated once. An initial instantiation of all variables is constructed by a subset of agents. The agents, then, arrive at a solution through collective and successive modifications.

2.3 Coordination Strategy

In a coordinated group of agents, individual behavior is regulated by policies so that the agents' collective actions achieve the common goals. Given the tasks of solving complex, large-scale NCSPs, our coordination mechanisms emphasize convergence efficiency by exploiting characteristics of agent group structure, agent tasks, and communicated information. We have developed coordination strategies that promote rapid convergence by considering the following principles of interaction control.

1. *Least disturbance* - When an agent is resolving conflicts, interactions should be initiated only when necessary and, in such a way as to reduce

the chances of causing other concerned agents to subsequently initiate further interaction.

2. *Island of reliability* - Consensus should be reached by a process of evolving coherent group decision-making based on *islands of reliability*, and modifying *islands of reliability* only when group coherence is perceived as infeasible under current assumptions.

3. *Loop prevention* - Looping behaviors, such as oscillatory value changes by a subset of agents, should be prevented.

Least disturbance Least disturbance corresponds to an attempt to minimize ripple effects of agents' actions. To reach group coherence, the number of unsatisfied agents within an operation cycle must be gradually reduced to zero. While an agent always becomes satisfied in an iteration cycle since it instantiates its variables to satisfy only its own constraints, its actions may cause conflicts to instantiations of other agents. Therefore, an agent should resolve conflicts in a way that minimizes the extent of causing disturbances to other agents. Least disturbance is incorporated in agent's heuristics of conflict resolution (see section 3.3). The least disturbance principle is operationalized during conflict resolution in two ways. First, an agent changes the instantiated values of as few variables as possible. Second, for a given selected variable, an agent changes the instantiated value as little as possible.

Island of reliability An *island of reliability* is a subset of variables whose consistent instantiated values are more likely than others to be part of the solution. In particular, islands of reliability should correspond to the most critical constraint clusters, i.e. clusters whose variables have the least flexibility in satisfying their constraints. Islands of reliability provide anchoring for reaching group coherence in terms of propagating more promising partial solutions and are changed less often.[3] For example, in job shop scheduling, a bottleneck resource is an island of reliability. A variable which is a member of an island of reliability is called a *seed variable.* A variable which is not a seed variable is a *regular variable.* Division of seed and regular variables reflects the inherent structure of the problem. The division is static and is complemented by the dynamic interactions among different kinds of agents.

Each agent assumes a role depending on the types of variables it controls. *Dominant agents* are responsible only for seed variables and therefore, are in charge of making decisions within islands of reliability. *Intermediate agents* control variable sets including both seed variables and regular variables. *Submissive agents* are responsible for only regular variables. Intermediate agents interact with submissive agents in a group effort to evolve an instantiation of regular variables compatible with the decisions of dominant agents regarding seed variables. A counter associated with each regular variable records the number of times that a submissive agent has changed the value of the regular variable and, thus, provides an estimate of the search efforts of intermediate and submissive agents to comply with islands of reliability. Intermediate agents monitor the value of the counter associated with the regular variables

[3]Blackboard systems (e.g., Hearsay-II speech-understanding system [6]) have used the notion of solution *islands* to conduct an incremental and opportunistic problem solving process. Partial solution islands emerge and grow into larger islands, which it is hoped will culminate in a hypothesis spanning the entire solution structure. In CP&CR, islands of reliability refer to partial solutions from some local perspectives and are used as anchors of interaction during the iterative solution repairing process from different local perspectives.

under their jurisdiction. When the counter of a conflicting regular variable exceeds a threshold. the intermediate agent, instead of changing the conflicting regular variable again. changes the value of its seed variables. In response to value changes in seed variables that result in conflicts, the dominant agent modifies its decisions on islands of reliability. All counters are reset to zero and, therefore, intermediate and submissive agents resume the efforts to evolve a compatible instantiation of regular variables.

Loop prevention Under the principles of least disturbance and islands of reliability, the system exhibits only two types of cyclic behavior. First, a subset of intermediate and submissive agents may be involved in cyclic value changes in order to find a compatible instantiation with dominant agents' decisions. Secondly, a dominant agent may be changing the value of its seed variables in a cyclic way.

The first type of looping behavior is interrupted by intermediate agents when the counter of a conflicting regular variable exceeds a threshold. To prevent the second type of looping behavior, a dominant agent keeps a history of its value changes so that it does not repeat the same configuration of variable instantiations.

2.3.1 Coordinated Group Search

From the point of view of search, the collective problem solving process is a coordinated, localized heuristic search with partially overlapping local search spaces (the values of variables that are the common responsibility of more than one agent). The process starts from an initial instantiation of all variables. The search proceeds as the agents interact with each other while seeking their own goals. Islands of reliability provide the means of anchoring the search, thus providing long term stability of partial solutions. The principle of least disturbance provides short term opportunistic search guidance. The search space is explored based on local feedback. The group of agents essentially performs a search through a series of modifications of islands of reliability. Within each configuration of islands of reliability, intermediate and submissive agents try to evolve a compatible instantiation on regular activities. The search ends when a solution is found or when dominant agents have exhausted all possible instantiation of the seed variables.

CP&CR provides a general framework that is potentially applicable to many NCSPs. We have applied it to solve the Zebra problem (classical test problem for constraint satisfaction algorithms). Experimental results show that CP&CR obtained a favorable performance in terms of the number of variable instantiations required as compared to a number of constraint satisfaction algorithms. In this paper, we focus on the application of CP&CR in job shop scheduling problems.

3 Job Shop Scheduling

Job shop scheduling with non-relaxable time windows involves synchronization of the completion of a number of jobs on a limited set of resources (machines). Each job is composed of a sequence of activities (operations), each of which has a specified processing time and requires the exclusive use of a designated resource for the duration of its processing (i.e. resources have only unit processing capacity). Each job must be completed within an interval (a time window) specified by its release and due time. A solution of the problem is a schedule, which assigns start times to each activity, that satisfies all *temporal activity precedence*, *release and due date*, and *resource capacity* constraints. This problem is known to be NP-complete [9], and has been considered as one of the most difficult CSPs. Traditional constraint satisfaction algorithms are shown to be insufficient for this problem [22].

3.1 Problem Decomposition and Transformation

Job shop scheduling with non-relaxable time windows is an NCSP, in which each activity is viewed as a quantitative *variable* with a *value* corresponding to the start time of the activity, and all constraints are expressed as numerical relations between variables. CP&CR, by applying the *pb( )* operator, partitions the constraint set into two constraint bunches: a constraint bunch of *exclusion-off* constraints to express temporal precedence constraints on activities within each job[4], and a constraint bunch of *exclusion-around* constraints to express capacity constraints on resources.

By applying the *pc( )* operator, CP&CR further partitions the constraint bunches into a set of constraint clusters corresponding to jobs or resources. Each job is a constraint cluster of exclusion-off constraints and is assigned to a *job agent*. Each job agent is responsible for enforcing temporal precedence constraints within the job. Similarly, each resource is a constraint cluster of exclusion-around constraints and is assigned to a *resource agent*. Each resource agent is responsible for enforcing capacity constraints on the resource. Therefore, for a given scheduling problem, the number of subproblems (and

[4]Release and due dates constraints are considered as temporal precedence constraints between activities and fixed time points and are included in the exclusion-off constraint bunch.

the number of agents) is equal to the sum of the number of jobs plus the number of resources.

An activity is governed both by a job agent and a resource agent. Manipulation of activities by job agents may result in constraint violations for resource agents and vice-versa. Therefore, coordination between agents is crucial for prompt convergence on a final solution. A *bottleneck resource* is the most contended resource among the resources, and corresponds to the most critical constraint cluster. The set of activities contending for the use of a bottleneck resource constitute an island of reliability and, therefore, are seed variables. A bottleneck resource agent assumes the role of a dominant agent, and a regular resource agent is a submissive agent. With the assumption that each job has at least one activity contending for the bottleneck resources. a job agent is an intermediate agent.

3.2 Coordination Information

Coordination information *written* by a job agent on an activity is referenced by a resource agent, and vice-versa.

Job agents provide the following coordination information for resource agents.

1. *Boundary* is the interval between the earliest start time and latest finish time of an activity (see Figure 4). It represents the overall temporal flexibility of an activity and is calculated only once during initial activation of job agents.

2. *Temporal Slack* is an interval between the current finish time of the previous activity and current start time of the next activity (see Figure

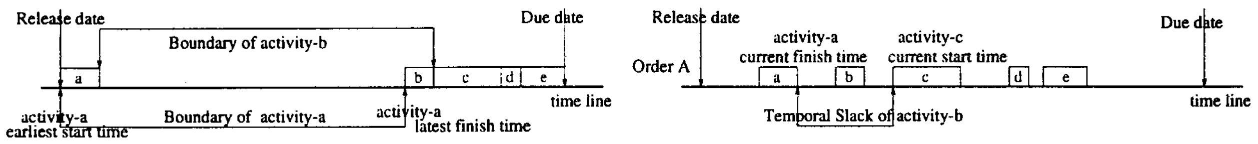

Release date
Boundary of activity-b
Due date
a
b
c
d
e
time line
activity-a earliest start time
Boundary of activity-a
activity-a latest finish time
Release date
Order A
activity-a current finish time
activity-c current start time
Due date
a
b
c
d
e
time line
Temporal Slack of activity-b

Figure 4: Coordination information: Boundary and Temporal Slack

4). It indicates the temporal range within which an activity may be assigned to without causing temporal constraint violations. (This is not guaranteed since temporal slacks of adjacent activities are overlapping with each other.)

3. *Weight* is the weighted sum of relative temporal slack with respect to activity boundary and relative temporal slack with respect to the interval bound by the closest seed activities (see Figure 5). It is a measure of the likelihood of the activity "bumping" into an adjacent activity, if its start time is changed. Therefore, a high weight represents a job agent's preference for not changing the current start time of the activity. In Figure 5, activity-p of job B will have a higher weight than that of activity-a of job A. If both activities use the same resource and are involved in a resource capacity conflict, the resource agent will change the start time of activity-a rather than start time of activity-p.

Resource agents provide the following coordination information for job agents.

1. *Bottleneck Tag* is a tag which marks that this activity uses a bottleneck resource. It indicates the seed variable status of the activity.

2. *Resource Slack* is an interval between the current finish time of the previous activity and the current start time of the next activity on the

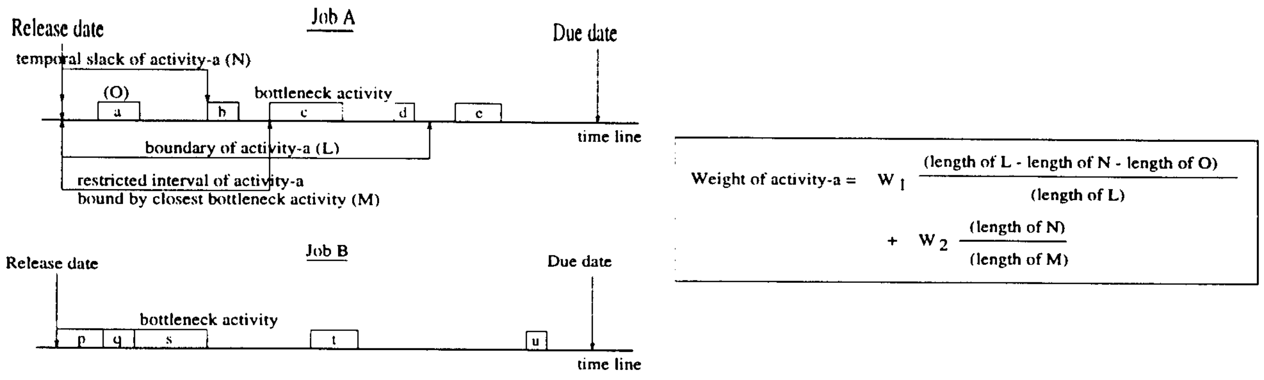

Release date
Job A
Due date
temporal slack of activity-a (N)
(O)
bottleneck activity
a
b
c
d
e
time line
boundary of activity-a (L)
restricted interval of activity-a
bound by closest bottleneck activity (M)
Release date
Job B
Due date
bottleneck activity
p
q
s
t
u
time line
Weight of activity-a = W1 (length of L - length of N - length of O) / (length of L) + W2 (length of N) / (length of M)

Figure 5: Coordination information: Weight

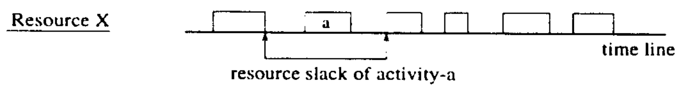


Figure 6: Coordination information: Resource Slack

resource timeline (see Figure 6). It indicates the range of activity start time in which an activity may be changed without causing capacity constraint violations. (There is no guaranteed since resource slacks of adjacent activities are overlapping with each other.)

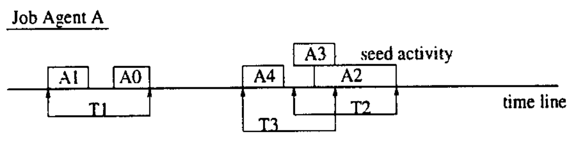
3. *Change Frequency* is a counter of how frequently the start time of this regular activity set by a job agent is changed by a submissive resource agent. It measures the search effort of job and regular resource agents between each modification on islands of reliability.

3.3 Reaction Heuristics

Agents' reaction heuristics attempt to minimize the ripple effects of causing conflicts to other agents as a result of fixing the current constraint violations. Conflict minimization is achieved by minimizing the number and extent of activity start time changes. The reaction heuristics utilize perceived coordination information and incorporate coordination strategies of group behaviors.

3.3.1 Reaction Heuristics of Job Agent

Job agents resolve conflicts by considering conflict pairs. A conflict pair involves two adjacent activities whose current start times violate the precedence constraint between them (see Figure 7). Conflict pairs are resolved one by one. A conflict pair involving a seed activity, i.e., an activity with tighter constraints, is given a higher conflict resolution priority. To resolve a conflict pair, job agents essentially determine which activity's current start time should be changed. If a conflict pair includes a seed and a regular activity, depending on whether the change frequency counter on the regular activity in the conflict pair is still under a threshold, job agents change the start time of either the regular or the seed activity. For conflict pairs of regular activities, job agents take into consideration additional factors, such as value changes feasibility of each activity, change frequency, and resource slack.

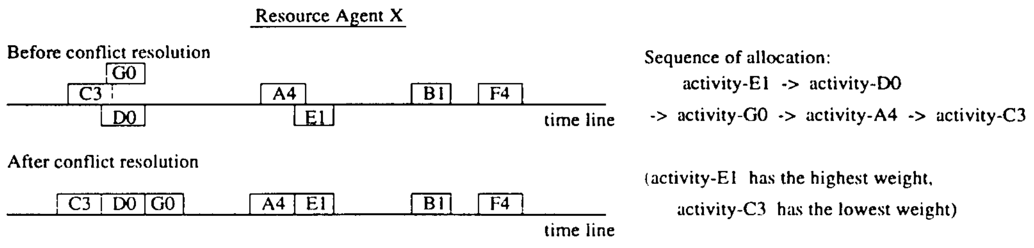


	considered change of start time	
bottleneck conflict pair:	activity-A2	minus T2
	activity-A3	plus T2
regular conflict pair:	activity-A0	minus T1
	activity-A1	plus T1
regular conflict pair:	activity-A3	minus T3
	activity-A4	plus T3

Figure 7: Conflict Resolution of Job Agent

In Figure 7. the conflict pair of activity-A2 and activity-A3 will be resolved first since activity-A2 is a seed variable. If the change frequency of activity-A3 is still below a threshold. start time of activity-A3 will be changed by an addition of T2 (the distance between current start time of activity-A3 and current end time of activity-A2) to its current start time. Otherwise, start time of activity-A2 will be changed by a subtraction of T2 from its current start time. In both cases, start time of activity-A4 will be changed to the end time of activity-A3. To resolve the conflict pair of activity-A0 and activity-A1, either start time of activity-A0 will be changed by a subtraction of T1 from its current start time or start time of activity-A1 will be changed by an addition of T1 to its current start time. If one of the two activities can be changed within its boundary and resource slack, job agent A will change that activity. Otherwise, job agent A will change the activity with less change frequency.

3.3.2 Reaction Heuristics of Regular Resource Agents



Resource Agent X
Before conflict resolution
G0
C3
A4
B1
F4
D0
E1
time line
After conflict resolution
C3
D0
G0
A4
E1
B1
F4
time line
Sequence of allocation:
activity-E1 -> activity-D0
-> activity-G0 -> activity-A4 -> activity-C3
(activity-E1 has the highest weight,
activity-C3 has the lowest weight)

Figure 8: Conflict Resolution of Regular Resource Agent

To resolve constraint violations, resource agents re-allocate the over-contended resource intervals to the competing activities in such a way as to resolve the conflicts and, at the same time, keep changes to the start times of these activities to a minimum. Activities are allocated according to their weights, boundaries, and temporal slacks. Since an activity's weight is a measure of the desire of the corresponding job agent to keep the activity at its current value, activity start time decisions based on weight reflect group coordination. For example, in Figure 8, activity-A4 was preempted by activity-E1 which has higher weight. Start time of activity-A4 is changed as little as possible. In addition, when a resource agent perceives a high resource contention during a particular time interval (such as the conflict involving activity-C3, activity-D0, and activity-G0), it allocates the resource intervals and assigns high change frequency to these activities. and thus dynamically changes the priority of these instantiation.

3.3.3 Reaction Heuristics of Bottleneck Resource Agents

A bottleneck resource agent has high resource contention. This means that most of the time a bottleneck resource agent does not have resource slack between activities. When the start time of a seed activity is changed, capacity constraint violations are very likely to occur. A bottleneck resource agent considers the amount of overlap of activity resource intervals on the resource to decide whether to right-shift some activities (Figure 9 (i)) or re-sequence some activities according to their current start times by swapping the changed activity with an appropriate activity. (Figure 9 (ii)). The intuition behind the heuristics is to keep the changes as minimum as possible.

3.4 System

3.4.1 System Operations

System initialization is done as follows: (1) decomposition of the input scheduling problem according to resource and job constraints, (2) creation of the corresponding resource and job agents, (3) activation of the agents (see

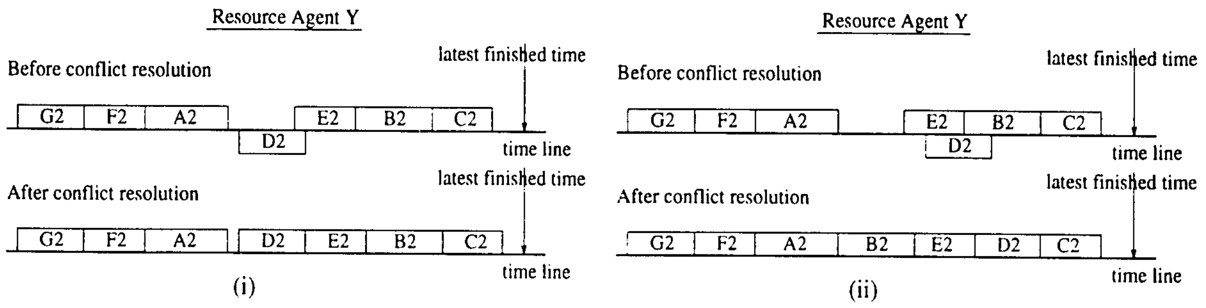


Figure 9: Conflict Resolution of Bottleneck Resource Agent

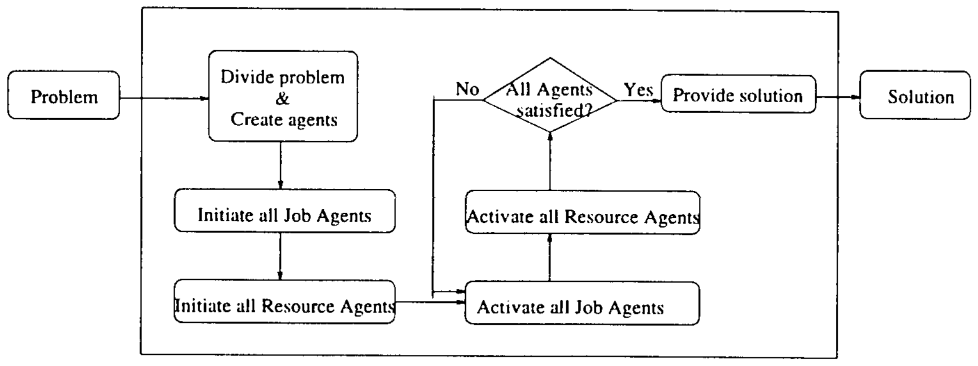


Figure 10: System Control Flow

Figure 10). Initially each job agent calculates boundary for each variable under its jurisdiction considering its release and due date constraints. Each resource agent calculates the contention ratio for its resource by summing the durations of activities on the resource and dividing by the interval length between the earliest and latest time boundary among the activities. If this ratio is larger than a certain threshold, a resource agent concludes that it is a bottleneck resource agent.[5] [6]

Activities under the jurisdiction of a bottleneck resource agent are marked as seed activities by the agent. Each resource agent heuristically allocates the earliest free resource interval to each activity under its jurisdiction according to each activity's boundary. After the initial activation of resource agents, all activities are instantiated with a start time. This initial instantiation of all variables represents the initial configuration of the solution.[7]

Subsequently, job agents and resource agents engage in an evolving process of reacting to constraint violations and making changes to the current instantiation. In each operation cycle, job and resource agents are activated alternatively, while agents of the same type are activated simultaneously, each working independently. When an agent finds constraint violations under its

[5]If no bottleneck resource is identified, threshold value is lowered until the most contended resource is identified.

[6]In job shop scheduling, the notion of bottleneck corresponds to a particular resource interval demanded by activities that exceeds the resource's capacity. Most state-of-the-art techniques emphasize the capability to identify *dynamic* bottlenecks that arise during the construction of solution. In our approach, the notion of bottleneck is *static* and we exploit the dynamic local interactions of agents.

[7]We have conducted experiments with random initial configurations and confirmed that the search is barely affected by its starting point, i.e. the search procedure has equal overall performance with heuristic and random initial configurations.

jurisdiction, it employs local reaction heuristics to resolve the violations. The process stops when none of the agents detect constraint violations during an iteration cycle. The system outputs the current instantiation of variables as a solution to the problem.

3.4.2 Solution Evolution

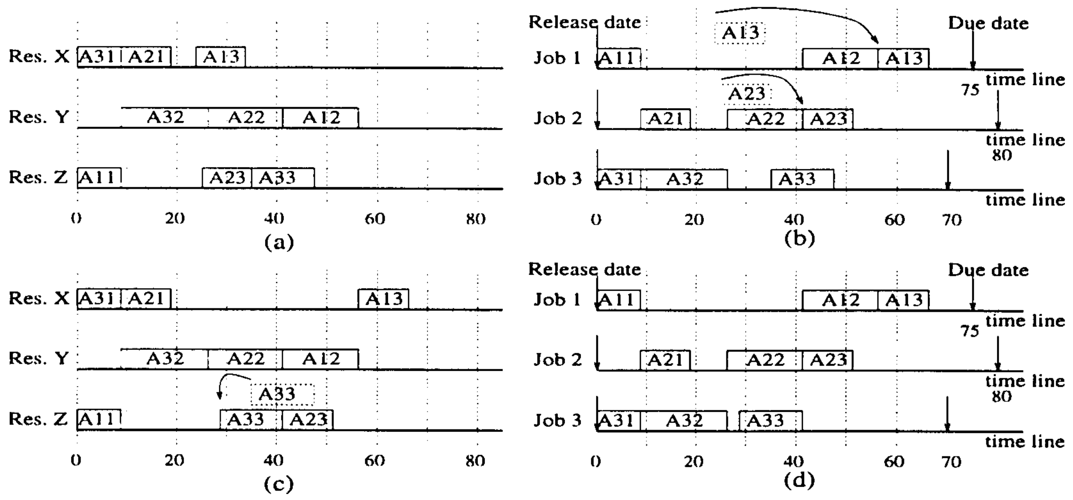


Figure 11: A Simplified Scenario

Figure 11 shows a solution evolution process of a very simple problem where resource Y is regarded as a bottleneck resource. In (a), resource agents allocate their earliest possible free resource intervals to activities, and thus construct the initial configuration of variable instantiation. In (b), Job1 and Job2 agents are not satisfied with current instantiation and change the start times of A13 and A23, respectively. In (c), Res.Z agent finds a constraint violation and changes the start time of A33. All agents are satisfied with the current instantiation of variables in (d) which represents a solution to the problem.

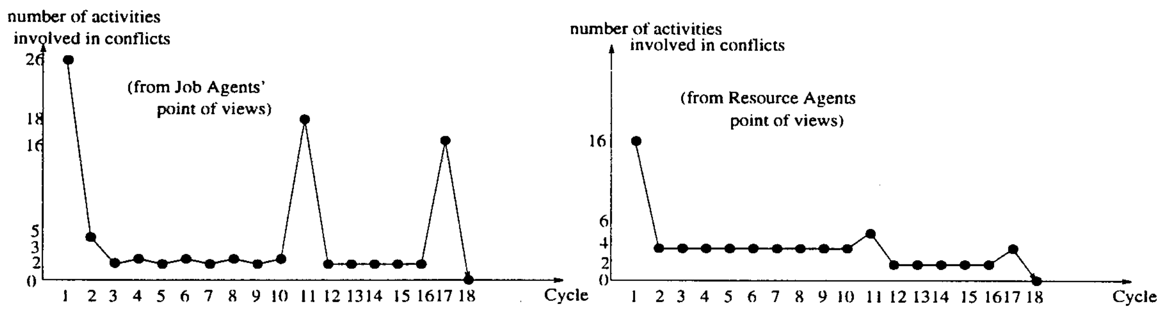


Figure 12: Conflicts Evolution of a more difficult problem

Figure 12 shows a solution evolution process in terms of occurred conflicts for a more difficult problem which involves 10 jobs on 5 resources. In cycle 0, resource agents construct an initial instantiation of variables that includes islands of reliability set by dominant (bottleneck resource) agents. During cycle 1 to cycle 9, intermediate (job) agents and submissive (regular resource) agents try to evolve a compatible instantiation with islands of reliability, i.e., the instantiation of variables (activities) on the bottleneck resource. In cycle 10, some job agents perceive the effort as having failed and change the values of their seed variables. Bottleneck resource agents respond to constraint violations by modifying instantiation on the islands of reliability. This results in a sharp increase of conflicting activities for job agents in cycle 11. Again, the search for compatible instantiation resumes until another modification on islands of reliability in cycle 16. In cycle 18, the solution is found.

4 Evaluation on Experimental Results

We evaluated the performance of CP&CR on a suite of job shop scheduling CSPs proposed in [22]. The benchmark consists of 6 groups, representing different scheduling conditions, of 10 problems, each of which has 10 jobs of 5 activities and 5 resources. Each problem has at least one feasible solution. CP&CR has been implemented in a system, called CORA (COordinated Reactive Agents). We experimentally (1) investigated the effects of coordination information in the system. (2) compared CORA's performance to other constraint-based as well as priority dispatch scheduling methods, (3) investigated CORA's scaling up characteristics on problems of larger sizes. The effectiveness of different types of coordination information was reported in [15]. We focus on the remaining aspects of evaluation in this paper.

CORA was compared to four other heuristic search scheduling techniques, ORR/FSS, MCIR, CPS, and PCP. ORR/FSS [22] incrementally constructs a solution by chronological backtracking search guided by specialized variable and value ordering heuristics. ORR/FSS+ is an improved version augmented with an intelligent backtracking technique [26]. Min-Conflict Iterative Repair (MCIR) [18] starts with an initial, inconsistent solution and searches through the space of possible repairs based on a *min-conflicts* heuristic which attempts to minimize the number of constraint violations after each step. Conflict Partition Scheduling (CPS) [20] employs a search space analysis methodology based on stochastic simulation which iteratively prunes the search space by posting additional constraints. Precedence Constraint Posting (PCP) [24] conducts the search by establishing sequencing constraints between pairs of activities using the same resource based on *slack-based* heuristics. In addition, three frequently used and appreciated priority dispatch rules from the field of Operations Research: EDD, COVERT, and R&M [19], are also included for comparison.

	CORA	CPS	MCIR	ORR/ FSS	ORR/ FSS+	PCP	EDD	COVERT	R&M
w/1	10	10	9.8	10	10	10	10	8	10
w/2	10	10	2.2	10	10	10	10	7	10
n/1	10	10	7.4	8	10	10	8	7	9
n/2	10	10	1	9	10	10	8	6	9
0/1	10	10	4.2	7	10	10	3	4	6
0/2	10	10	0	8	10	8 ~ 10	8	8	8
Total	60	60	24.6	52	60	58 ~ 60	47	40	52
AVG. CPU time	4.8 seconds	13.07 * seconds	49.74 * seconds	39.12 * seconds	21.46 * seconds	N/A	0.9 seconds	0.9 seconds	0.9 seconds

Table 1: Performance Comparison

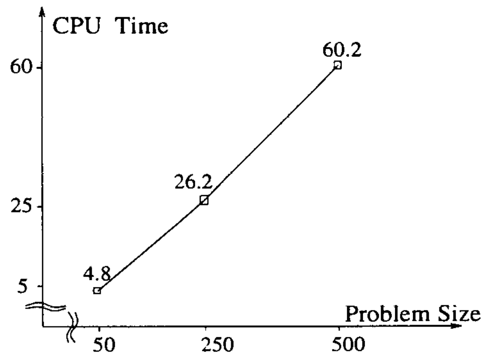


Figure 13: CORA's Scaling Up Property

Table 1 reports the number of problems solved[8] and the average CPU time spent over all the benchmark problems for each technique. Note that the results of ORR/FSS, ORR/FSS+, MCIR, CPS, and PCP were obtained from published reports, of mostly the developers of the techniques. MCIR is the only exception, which is implemented by Muscettola who reported its results based on randomly generated initial solutions [20]. All CPU times were obtained from Allegro Common Lisp implementations on a DEC 5000/200. In particular. CORA was implemented in CLOS (Common Lisp Object System). CPS. MCIR. ORR/FSS, and ORR/FSS+ were implemented using CRL (Carnegie Representation Language) as an underlying frame-based knowledge representation language. CPU times of CPS, MCIR, ORR/FSS, and ORR/FSS+ were divided by six from the published numbers as an estimate of translating to straight Common Lisp implementation.[9] PCP's CPU times are not listed for comparison because its CPU times in Common Lisp are not available. Its reported CPU times in C are 0.3 second [24]. Although CORA can operate asynchronously, it was sequentially implemented for fair comparison. The results show that CORA works considerably well as compared to the other techniques both on feasibility and efficiency in finding a solution. In addition, the same problem generator function producing the benchmark problems was used to produce problem sets of 250 and 500 variables (e.g. 100 factory jobs on 5 machines which is a problem of realistic size). Figure 13 shows CORA's performance on these larger sized problems, which exhibits favorable. near-linear scaling-up characteristics.

[8]PCP's performance is sensitive to the parameters that specify search bias [24].

[9]ORR/FSS and ORR/FSS+ obtained 30 times speedup in C/C++ implementation. We assumed a factor of five between Common Lisp and C/C++ implementations.

As a scheduling technique, CORA performs a heuristic *approximate* search in the sense that it does not systematically try all possible configurations. Although there are other centralized scheduling techniques that employ similar search strategies, CORA distinguishes itself by an interaction driven search mechanism based on well-coordinated asynchronous local reactions. Heuristic approximate search provides a middle ground between the generality of domain-independent search mechanisms and the efficiency of domain-specific heuristic rules. Instead of the rigidity of one-pass attempt in solution construction (either it succeeds or fails, and the decisions are never revised) in approaches using heuristic rules, CORA adapts to constraint violations and performs an effective search for a solution. As opposed to generic search approaches, in which a single search is performed on the whole search space and search knowledge is obtained by analyzing the whole space at each step, CORA exploits local interactions by analyzing problem characteristics and conducts well-coordinated asynchronous local searches.

The experimental results obtained by various approaches concur with the above observations. Approaches using generic search techniques augmented by domain-specific search-focus heuristics (ORR/FSS, ORR/FSS+, MCIR, CPS) required substantial amount of computational effort. Some of them could not solve all problems in the sense that they failed to find a solution for a problem within the time limit set by their investigators. Approaches using dispatch rules (EDD, COVERT, R&M) were computationally efficient, but did not succeed in all problems. PCP relies on heuristic rules to conduct one-pass search and its performance is sensitive to parameters that specify search bias. CORA struck a good balance in terms of solving all problems with considerable efficiency. Furthermore, with a mechanism based on collective operations, CORA can be readily implemented in parallel processing such that only two kinds of agents are activated sequentially in each iteration cycle, instead of 10 job agents and 5 resource agents under current implementation. This would result in an approximate time-reducing factor of 7 (i.e., 15/2) and would enable CORA to outperform all other scheduling techniques in comparison.

CORA exploits local interactions based on the notion of islands of reliability and has showed to perform quite well on problems with clear resource bottlenecks. For problems with no clear bottlenecks and all resources are loosely utilized (say, below 50 percents of utilization), we expect CORA perform with the same efficiency by selecting the most utilized resource as islands of reliability. However, CORA's current mechanism based on *dominant coordination* may not be sufficient for problems in which all resources are at least moderately utilized (say, above 60 percents of utilization) and there is no outstanding bottleneck. We are interested in developing a more sophisticated mechanism based on *competing coordination* and investigate its utility in various scheduling conditions.

5 Conclusions

In this paper, we have presented a collective problem solving framework, where problem solving is viewed as an emergent functionality from the evolving process of a society of diverse, interacting, well-coordinated reactive agents. We show that large-scaled NCSPs can be decomposed and assigned to different problem solving agents according to disjoint functionality (constraint types) and overlapping responsibility (variable subsets). This decomposition results in utilization of interaction characteristics to achieve problem solving by asynchronous and well coordinated local interactions. Application of the methodology to job shop scheduling with non-relaxable time windows results in very good performance. Our experimental results show that the coordination mechanism (1) incorporates search knowledge and guides the search space exploration by the society of interacting agents, facilitating rapid convergence to a solution, and (2) is independent of initial configuration. In addition, the search complexity grows only linearly with problem size. We are currently applying the CP&CR methodology to Constraint Optimization Problems (COPs). Preliminary experiments show encouraging results compared to both heuristic search and simulated-annealing-based techniques. We are also investigating the utility of CP&CR in other domains with different problem structures.

References

[1] Rodney A. Brooks. Intelligence without reason. In *Proceedings of the IJCAI-91*, pages 569–595, 1991.

[2] Rina Dechter. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.

[3] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[4] Keith S. Decker. Distributed problem-solving techniques: A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(5):729–739, 1987.

[5] Edmund H. Durfee. *Coordination of Distributed Problem Solvers.* Kluwer Academic Publishers, 1988.

[6] L. D. Erman, F. A. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The hearsay-II speech-understanding system: integrating knowledge to resolve uncertainty. *Computer Survey*, 12(2):213–253, 1980.

[7] J. Ferber and E. Jacopin. The framework of Eco Problem Solving. In Demazeau and Muller, editors, *Decentralized AI 2.* Elsevier, North-Holland, 1991.

[8] Eugene C. Freuder and Paul D. Hubbe. Using inferred disjunctive constraints to decompose constraint satisfaction problems. In *Proceedings of the IJCAI-93*, pages 254–260, 1993.

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman and Co., 1979.

[10] Les Gasser and Randall W. Hill, Jr. Engineering coordinated problem solvers. *Annual Review of Computer Science*, 4:203–253, 1990.

[11] M. Huhns and D. Bridgeland. Multiagent truth maintenance. *IEEE Transactions on System, Man, and Cybernetics*, 21(6):1437–1445, 1991.

[12] C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, editors. *Artificial Life II.* Addison-Wesley, 1991.

[13] Christopher G. Langton, editor. *Artificial Life.* Addison-Wesley, 1989.

[14] Olivier Lhomme. Consistency techniques for numerical CSPs. In *Proceedings of IJCAI-93*, pages 232–238, 1993.

[15] JyiShane Liu and Katia P. Sycara. Distributed constraint satisfaction through constraint partition and coordinated reaction. In *Proceedings of the 12th International Workshop on Distributed AI*, 1993.

[16] Alan K. Mackworth. Constraint satisfaction. In S. C. Shapiro, editor, *Encyclopedia in Artificial Intelligence*, pages 205–211. Wiley, New York, 1987.

[17] Jean-Arcady Meyer and Stewart W. Wilson, editors. *Proceedings of the First International Conference on Simulation of Adaptive Behavior - From Animals To Animats.* MIT Press, 1991.

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

[19] Thomas E. Morton and David W. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management.* John Wiley & Sons, New York, 1993.

[20] Nicola Muscettola. HSTS: Integrated planning and scheduling. In Mark Fox and Monte Zweben, editors, *Knowledge-Based Scheduling.* Morgan Kaufmann, 1993.

[21] Bernard A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.

[22] Norman Sadeh. Look-ahead techniques for micro-opportunistic job shop scheduling. Technical Report CMU-CS-91-102, School of Computer Science, Carnegie-Mellon University, 1991.

[23] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proceedings of AAAI-92*, pages 276–281, 1992.

[24] Stephen F. Smith and Cheng-Chung Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of AAAI-93*, pages 139–144, 1993.

[25] Katia Sycara, Steve Roth, Norman Sadeh, and Mark Fox. Distributed constraint heuristic search. *IEEE Transactions on System, Man, and Cybernetics*, 21(6):1446–1461, 1991.

[26] Yalin Xiong, Norman Sadeh, and Katia Sycara. Intelligent backtracking techniques for job shop scheduling. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 14–23, 1992.

[27] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the 12th IEEE International Conference on Distributed Computing Systems*, pages 614–621, 1992.

On the Utility of Bottleneck Reasoning for Scheduling

Nicola Muscettola
RECOM Technologies
NASA Ames Research Center
AI Research Branch, Mail Stop: 269-2
Moffett Field, CA 94035-1000
e-mail: mus@ptolemy.arc.nasa.gov

Abstract

The design of better schedulers requires a deeper understanding of each component technique and of their interactions. Although widely accepted in practice, bottleneck reasoning for scheduling has not yet been sufficiently validated, either formally or empirically. This paper reports an empirical analysis of the heuristic information used by bottleneck-centered, opportunistic scheduling systems to solve constraint satisfaction scheduling problems. Different configurations of a single scheduling framework are applied to a benchmark set of scheduling problems and compared with respect to number of problems solved and processing time. We show superior performances for schedulers that use bottleneck information. We also show that focusing at the bottleneck might not only provide an effective "most constrained first" heuristic but also, unexpectedly, increase the utility of other heuristic information.

1 Introduction

Problem solvers often use combinations of several different heuristics and reasoning methods (*e.g.*, constraint propagation, search). Empirical comparisons of performances of different problem solvers can show that one combination of techniques is superior to another. However, to design better problem solvers we need a deeper understanding of the importance of each component technique and of how different techniques interact.

This paper reports an empirical analysis of the performance of heuristic information typically used to solve constraint satisfaction scheduling problems. We will focus on bottleneck-centered, opportunistic schedulers, a class of systems that has shown better performance than other kinds of schedulers [1, 13, 11]. We will show that there is strong empirical evidence on the effectiveness of reasoning about bottlenecks. Moreover, we will show evidence of the fact that heuristic information gathered at the bottleneck is "more useful" than average. This allows a bottleneck centered scheduler to make several decisions without having to re-evaluate its heuristic information too often.

Although widely accepted in practice (*e.g.*, manufacturing scheduling), bottleneck reasoning has not yet been sufficiently validated, either formally or empirically. Although formal validation would be most desirable, at present no formal model realistically captures the deep structure of scheduling problems. In its absence, strong evidence of performance can be gathered with empirical studies. We believe that such studies will also be extremely useful to discover new "phenomena" which will guide the search for an appropriate formal model.

The goal of an opportunistic scheduler is to build an assignment of time and resources to a network of activities and a set of resources such as to avoid resource over-subscriptions. The goal is achieved by repeatedly applying the following basic *opportunistic scheduling cycle*:

1. *Analyze*: analyze the current problem solving state;

2. *Focus*: select one or more activities that are expected to participate in a critical interaction among problem constraints;

3. *Decide*: add constraints to reduce negative interactions among critical activities.

Opportunistic schedulers differ on the specific techniques used to implement each phase [15, 3, 13, 1, 11] but share several fundamental characteristics. The *Analyze* phase consists of building estimates of demand/supply ratios for the different resources and/or activities. These estimates are usually conducted on relaxed versions of the problem obtained by dropping some of its original temporal and/or resource constraints. During the *Focus* phase, all opportunistic schedulers use bottlenecks as the primary means of selecting critical interactions. While the exact definition of a bottleneck may vary, all opportunistic schedulers agree on relating this concept to a resource and time interval with a high demand/supply ratio. Critical activities are usually defined as those that are likely to request the use of a bottleneck. The constraints posted during the *Decide* phase impose arbitration among conflicting capacity and time requests. This is the phase where opportunistic schedulers differ the most with respect to the type of constraint posted (assigning a value to a variable versus imposing a precedence among activities) and to the granularity of the decision making process (the number of decisions taken at each cycle).

The study in this paper was conducted on Conflict Partition Scheduling (CPS) [11], a scheduling method that implements the opportunistic scheduling paradigm. We first describe CPS and the stochastic simulation method used to compute heuristic information. Then we analyze two steps of the procedure: bottleneck detection and scheduling decision making. We discuss modifications of these steps and make hypotheses on how these modifications might affect performance. We then verify our hypotheses against the results of an experimental analysis.

We believe that our results are typical of the performance of other opportunistic schedulers. We base our belief on the similarity of each CPS step to other opportunistic schedulers

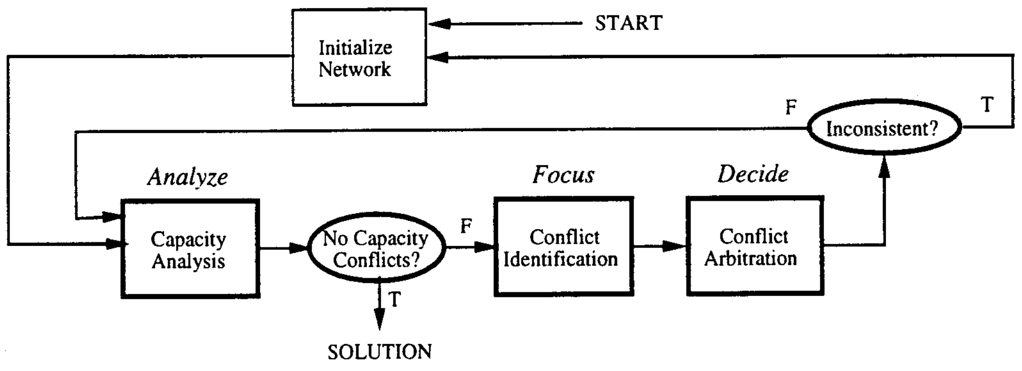


Figure 1: Conflict Partition Scheduling

and on the applicability of the performance hypotheses to comparable modifications of other schedulers.

2 Conflict Partition Scheduling

CPS adopts a *constraint posting* approach to scheduling, *i.e.*, it operates by posting temporal precedences among activities (activity α_i must precede activity α_j). During problem solving, each activity has an associated window of opportunity for its execution; these can be deduced by propagating activity durations and metric temporal constraints (absolute and relative) across the activity network [5]. Previous empirical studies have shown that constraint posting schedulers perform better than schedulers that proceed by assigning precise values to the start and end time of each activity [2, 11, 14].

Figure 1 shows how CPS organizes its computation. In relation to the opportunistic cycle described in the introduction. Capacity Analysis corresponds to *Analyze*, Conflict Identification to *Focus*, and Conflict Arbitration to *Decide*. The consistency test is a propagation of the metric temporal constraints in the activity network.

The algorithm described in the diagram follows an iterative sampling approach to search [10, 9]. If the consistency test fails, the activity network is reset to the initial state and the procedure is re-started. As we will see later, CPS' capacity analysis is stochastic in nature; therefore, each repetition can explore a different path in the problem solving space. If a solution has not been found after a fixed number of repetitions (in our case, 10), CPS terminates with an overall failure. The choice of iterative sampling is consistent with our interest in isolating the information content of the heuristic information generated by the Capacity Analysis. However, it is also possible to use the internal CPS cycle in a systematic search approach. For example, Conflict Arbitration could sprout several alternative ways of adding constraints among conflicting activities. Prioritization of these alternatives could make use of Capacity Analysis information and a backtracking scheme would ensure continuation when reaching a dead end.

The Capacity Analysis computes all the heuristic information used for decision-making by generating estimates of the structure of the remaining search space without engaging in detailed problem solving. Such estimates are statistics computed from a sample of complete time assignments to activity start times. These assignments are consistent with all the temporal constraints explicitly represented in the current activity network[1], but do not usually result in consistent schedules since they do not necessarily satisfy all the constraints of the problem (*i.e.*, those capacity constraints that have not yet been explicitly enforced). CPS uses a stochastic simulation process to generate each complete time assignment. This process differs from other stochastic simulation techniques [6, 7] used to estimate the possible outcomes of executing a detailed schedule in an uncertain environment. Having to insure executability, these simulations must introduce additional constraints to complete an intermediate problem solving state into a consistent schedule. Therefore, they end up considering many more details than are useful or necessary for an aggregate capacity analysis. Instead, CPS' stochastic simulation [12] considers only the constraints that are explicit in the current intermediate state, with very weak assumptions on how it will be extended into a complete schedule.

In the following, *EST* (α) and *LFT* (α) will denote, respectively, the earliest start time and the latest finish time of the activity α, H will denote the overall scheduling horizon, and R will be the set of resources.

CPS' stochastic simulation proceeds by repeating the following cycle. Before the simulation starts, a temporal constraint propagation establishes the range of possible start times for each activity. At each simulation cycle i, an activity α_i is selected according to a given strategy. After the selection, a start time is randomly chosen among α_i's possible start times and assigned to the activity. The random choice follows a given probability distribution, or *selection rule*. The consequences of the start time assignment are then propagated through the network to restrict the range of other activities' start times, and the simulation cycle is repeated. The simulation terminates when all the activities of the network have been assigned a start time.

Different implementations of CPS can choose different activity selection strategies and start time selection rules. A typical activity selection strategy is *forward temporal dispatching* which selects α_i among the set of activities whose predecessors have all start times assigned by previous simulation cycles. A possible start time selection rule is a linearly biased distribution (*i.e.*, the weight of the currently available times increases or decreases linearly over the time bound) for each activity in the network. These choices are crucial to the performance of CPS since they determine: (1) the computational cost of each cycle and (2) the probability of generating each of the possible total start time assignments and, therefore, the bias of the sampling base.

Figure 2 illustrates a single simulation cycle using a linear selection rule. In the figure, activity α_i precedes activity α_{i+1} in the activity network. Figure 2 (a) shows the time bounds for each activity; α_i has a linear value selection rule superimposed on its time bound. In Figure 2 (b), a start time has been selected for α_i and time has been reserved for its execution;

[1]Although CPS can deal with activities with flexible durations (*i.e.*, the duration of α must fall in the range $[d_\alpha, D_\alpha]$), we will only consider activities with fixed durations to simplify the presentation.

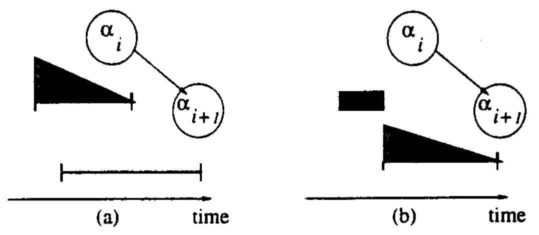


Figure 2: Simulation step: (a) before step i; (b) after step i

the reservation is represented by the black rectangle. This causes α_{i+1}'s time bound to shrink. A triangular selection rule is now superimposed on α_{i+1}'s time bound and the simulation cycle can start again.

Repeating the simulation N times yields a sample of N complete time assignments. CPS' Capacity Analysis uses this sample to estimate the following two problem space metrics:

- **activity demand**: for each activity α and for each time $EST(\alpha) \leq t_i < LFT(\alpha)$, the activity demand, $\Delta(\alpha, t_i)$, is equal to n_{t_i}/N, where n_{t_i} is the number of complete time assignments in the sample for which α is being executed at time t_i.

- **resource contention**: for each resource $\rho \in R$ and for each time $t_j \in H$, the resource contention, $X(\rho, t_j)$, is equal to n_{t_j}/N, where n_{t_j} is the number of complete time assignments in the sample for which ρ is requested by more than one activity at time t_j.

Activity demand and resource contention represent two different aspects of the current problem solving state. Activity demand is a measure of preference; it indicates how much the current constraints bias an activity toward being executed at a given time. Resource contention is a measure of potential inconsistency; it indicates how likely it is that the current constraints will generate a congestion of capacity requests on a resource at a given time.

3 Bottleneck Detection

In problem solving, a widely accepted principle is to focus on the most tightly interacting variables, *i.e.*, those with the smallest set of possible values. For example, in constraint satisfaction search [8] the 'most constrained first' heuristic minimizes the expected length of any path in the search tree and, therefore, increases the probability of achieving a solution in less time. In opportunistic scheduling this principle translates into looking for bottleneck resources.

In CPS each problem solving cycle focuses on a set of activities that are potentially in conflict, called the *conflict set*. More precisely, a conflict set is a set of activities that: (1) request the same resource, (2) have overlapping execution time bounds, and (3) are not necessarily totally ordered according to the precedence constraints of the current activity network.

To detect a conflict set CPS first identifies a bottleneck using resource contention:

- **Bottleneck**: Given the set of resource contention functions $\{X(\rho, t)\}$ with $\rho \in R$ and $t \in H$, we define a bottleneck to be a pair $< \rho_b, t_b >$ such that:

$$X(\rho_b, t_b) = \max\{X(\rho, t)\}$$

  for any $\rho \in R$ and $t \in H$ such that $X(\rho, t) > 0$.

The conflict set is then extracted among the activities requesting ρ_b with current time bounds overlapping t_b.

Although focusing on bottlenecks is widely accepted in opportunistic scheduling, there is little quantitative evidence of its effectiveness. One could wonder if the performance of the scheduler would remain unaffected if it focussed on *any* set of activities, either associated with a bottleneck or not. If this were true, one could save the additional cost required to compute resource contention and base all decision making on activity demand alone.

To answer this question we consider two different configurations of CPS' Bottleneck Identification step.

1. **Maximum contention bottleneck (BTL)**: The original method used in CPS; the set of conflicting activities is selected around the bottleneck.

2. **Random (RAND)**: The conflict set is selected around a randomly chosen resource and time.

4 Conflict Arbitration

At each Conflict Arbitration step, CPS introduces additional precedence constraints between pairs of activities to restrict their mutual position and their time bounds. An important differentiating aspect among schedulers is the granularity of decision making. At one end of the spectrum there are schedulers that make the minimum possible decision at each scheduling cycle; this follows the spirit of micro-opportunistic scheduling [13]. At the other end there are schedulers that make decisions that eliminate any possibility of conflict among all the activities in the conflict set; this follows the spirit of macro-opportunistic approaches [15, 1].

Within CPS we can explore the consequences of different decision making granularities. For example, a micro-opportunistic approach translates into adding a single precedence constraint between two activities in the conflict set. Conversely, a macro-opportunistic approach could be implemented by imposing a total ordering on all activities in the conflict set.

The current implementation of CPS [11] proposes an intermediate granularity approach by partitioning the conflict set into two subsets, A_{before} and A_{after}, and then constraining every activity in A_{before} to occur before any activity in A_{after}. The bi-partition of the

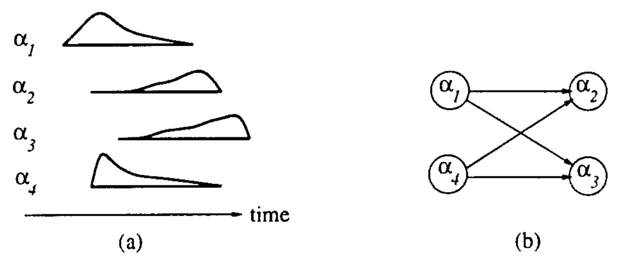


Figure 3: A Conflict Arbitration step

original conflict set relies on a clustering analysis of activity demands. Figure 3 (a) shows four conflicting activities and their demand profiles; figure 3 (b) shows the new precedence constraints added by Conflict Arbitration.

To choose the appropriate decision granularity one needs to evaluate a trade-off. The greater the number of scheduling decisions in a step, the greater the pruning of the search space and, therefore, the faster the scheduler. However, the more decisions that are made, the greater the change in the topology of the activity network after the step. Therefore, an analysis done before the step could give little information on the structure of the destination state. This can increase the likelihood of failure and consequent backtracking, and therefore slow down the scheduler. In summary, an important trade-off involves, the speed of convergence vs. the number of restarts needed during iterative sampling. This trade-off rests on the reasonable assumption that making fewer decisions at each cycle is always at least as accurate as making several; in other words, although possibly slower, a micro-opportunistic scheduler should solve at least as many problems as a larger granularity scheduler [13].

We therefore consider two distinct Conflict Arbitration rules:

1. **conflict bi-partition (BIP)**: The technique originally used in CPS; the conflict set is separated into two A_{before} and A_{after} subsets.

2. **most separated activities (MS)**: A *micro* arbitration technique; it introduces a single precedence between two activities extracted from the conflict set. To minimize the impact of sampling noise, we select the two activities whose demand profiles are maximally separated.

5 Experimental Results

The experimental analysis made use of the Constraint Satisfaction Scheduling benchmark proposed in [13]. The benchmark consists of 6 groups of 10 problems, each with 50 activities, 5 resources, and non-relaxable release and due date constraints. The groups vary according to their expected difficulty. Each group is identified by two parameters: (1) the spread of the release and due dates, which can assume the three levels (in increasing order of expected difficulty) *W* for wide, *N* for narrow and *0* for null; (2) the number of expected bottleneck resources, either *1* or *2*. For more details see [13].

	< BTL, BIP >	< BTL, MS >	< RAND, BIP >	< RAND, MS >
W/1	10.00	10.00	9.95	7.68
W/2	10.00	10.00	9.95	9.10
N/1	10.00	10.00	9.10	8.70
N/2	10.00	10.00	8.20	6.26
0/1	10.00	9.95	8.30	8.60
0/2	9.25	10.00	4.85	4.05
TOT	59.25	59.95	50.35	46.30

Table 1: Experimental results: number of problem solved

	< BTL, BIP >	< BTL, MS >	< RAND, BIP >	< RAND, MS >
W/1	44.94	120.90	61.74	281.49
W/2	44.59	134.00	90.82	476.97
N/1	47.02	127.40	106.67	360.72
N/2	46.03	138.90	142.81	754.39
0/1	50.27	140.40	118.24	405.10
0/2	64.86	161.10	212.96	908.89
AVG	49.62	137.10	122.20	531.26

Table 2: Experimental results: processing time

Tables 1 and 2 report the performance of all possible combinations of the alternative settings described in the previous sections. Table 1 shows the average number of problems solved over 20 independent runs of the procedure. Table 2 reports the corresponding average processing times. In order to factor out the effects of known implementation inefficiencies, processing times are given as the number of opportunistic cycles needed either to find a solution or to fail. The number of iterations was weighed differently depending on the conflict identification method, with each **RAND** cycle taking 85.64% of the time of a **BTL** cycle. The speed-up results from avoiding the computation of resource contention.

The cardinality of the Capacity Analysis sample was $N = 10$. We used *forward temporal dispatching* as the activity selection strategy. The start time selection rule was linearly biased over the time bound, with highest preference to the earliest time and lowest (0) to the latest.

All of the following conclusions have been tested for statistical significance using the methods available in the S statistical package [4]. For the average number of problems solved we fitted the results as a function of the configuration; this was done through a *logit* generalized linear regression. An analysis of deviance and a Chi-squared test yielded the desired measure of significance (see [4], chapter 6). For the processing time we used a standard analysis of variance (see [4], chapter 5). Unless otherwise noted, differences in performance are statistically significant at a 1% level.

To test the importance of bottleneck information, let us compare each < **RAND, ?x** > entry with the corresponding < **BTL, ?x** > entry. Differences in performance are always significant except for the number of problems solved for groups *W/1* and *W/2* when using bi-partition for Conflict Arbitration (**BIP**). These are the two problem groups with lowest expected difficulty. In every other case, random focusing performs significantly worse than bottleneck focusing, with an average slowdown of approximately 3.1 times. Therefore, these results show that, all things remaining equal, there is a substantial advantage in focusing problem solving on what CPS characterizes as bottlenecks.

To test the effect of decision making granularity, let us compare each < **BTL, BIP** > configuration with the corresponding < **BTL, MS** > configuration. With respect to the number of problem solved, only for group *0/2* there is a statistically significant advantage in using most-separated-pair for Conflict Arbitration; *0/2* is the the group with the highest expected difficulty. This advantage is due to a single problem that **MS** always solves while **BIP** solves less than 50% of the time. Although small, this advantage is consistent with the expectation of better problem solving accuracy with smaller decision making granularity, especially on difficult problems. However, when comparing processing times we see an average slowdown of approximately 3.2 times going from **BIP** to **MS** which makes the cost of micro-granularity scheduling prohibitive (except for one problem).

Unexpectedly, the trend toward better accuracy with lower decisions granularity is completely reversed when comparing < **RAND, BIP** > to < **RAND, MS** >. In fact, for all problem groups, the average number of problems solved tend to *decrease* when going from bi-partition to most-separated-pair. This trend is statistically significant for groups *W/1* (at a 2% level), *W/2*, and *N/2*. This result contradicts our expectations. After all, a macro decision step can always be seen as a sequence of micro steps without additional intermediate capacity analyses. A macro-granularity approach should be less informed than a micro-granularity approach and therefore more prone to errors.

However, worse performance with lower granularity can be explained by assuming that at each step there is a probability p of selecting a misleading conflict set, *i.e.*, one for which the preferential information leads to a wrong ordering among activities. The overall probability of following a dead-end path is the sum of the probabilities of failing after x cycles, with x less or equal to maximum path length in the search tree. When the decision making granularity decreases, the expected path length increases. Correspondingly, if p does not substantially decrease, the overall probability of failure increases. The experimental results seem to indicate that the decrement of p is adequate only when using the bottleneck information for focusing. In other words, the expected utility of preferential information at the bottleneck is higher than average.

6 Conclusions

In this paper we experimentally analyzed the role of bottleneck reasoning in opportunistic schedulers. The aim was to go beyond a bulk comparison of systems and to identify important design trade-offs among system components. The results of the study empirically validate the importance of bottlenecks to focus problem solving. The results seem to indicate that preferential information at bottlenecks has a higher expected utility than average. Therefore the utility of bottleneck-focusing goes beyond the classical view of a "most constrained first" heuristic in a constraint satisfaction search. This is an unexpected result that will further focus the search for a plausible formal model of the performance of schedulers.

Acknowledgements

The author thanks the following people for reviewing earlier drafts of the paper: John Allen, John Bresina, Mark Drummond. Keith Swanson. This work was carried out when the author was at the Center for Integrated Manufacturing and Decision Systems, The Robotics Institute, Carnegie Mellon University. This work was sponsored in part by the National Aeronautics and Space Administration under contract # NCC 2-707, the Defense Advanced Research Projects under contract # F30602-91-F-0016, and the Robotics Institute.

References

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

[2] D. Applegate and W. Cook. A computational study of job-shop scheduling. Technical Report CMU-CS-90-145, School of Computer Science, Carnegie Mellon University, 1990.

[3] E. Biefeld and L. Cooper. Bottleneck identification using process chronologies. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 218–224, Menlo Park, California, 1991. AAAI Press.

[4] J.M. Chambers and T.J. Hastie. editors. *Statistical Models in S.* Wadsworth and Brooks/Cole, 1992.

[5] R. Dechter, I Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, May 1991.

[6] M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 138–144. AAAI Press, 1990.

[7] S. Hanks. Practical temporal projection. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 158–163. AAAI Press, 1990.

[8] R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, October 1980.

[9] P. Langley. Systematic and nonsystematic search strategies. In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, pages 145–152. Morgan Kaufmann, 1992.

[10] S. Minton, M. Drummond, J.L. Bresina, and A.B Philips. Total order vs. partial order planning: Factors influencing performance. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 83–92. Morgan Kaufmann, 1992.

[11] N. Muscettola. Scheduling by iterative partition of bottleneck conflicts. In *Proceedings of the 9th Conference on Artificial Intelligence for Applications*, pages 49–55, Los Alamitos, California, March 1993. IEEE Computer Society Press.

[12] N. Muscettola and S.F. Smith. A probabilistic framework for resource-constrained multi-agent planning. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 1063–1066, Menlo Park, California, 1987. AAAI Press.

[13] N. Sadeh. Look-ahead techniques for micro-opportunistic job shop scheduling. Technical Report CMU-CS-91-102, School of Computer Science, Carnegie Mellon University, 1991.

[14] S.F. Smith and Cheng-Chung Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI 93)*, pages 139–144, Menlo Park, California, 1993. The AAAI Press.

[15] S.F. Smith, P.S. Ow, J.Y. Potvin, N. Muscettola, and D. Matthys. An integrated framework for generating and revising factory schedules. *Journal of the Operational Research Society*, 41(6):539–552, 1990.

Interactive Graphic Design Using Automatic Presentation Knowledge

Steven F. Roth, John Kolojejchick, Joe Mattis, Jade Goldstein

School of Computer Science

Carnegie Mellon University
Pittsburgh, PA 15213

(412) 268-7690

Steven.Roth@cs.cmu.edu

ABSTRACT

We present three novel tools for creating data graphics: (1) SageBrush, for assembling graphics from primitive objects like bars, lines and axes, (2) SageBook, for browsing previously created graphics relevant to current needs, and (3) SAGE, a knowledge-based presentation system that automatically designs graphics and also interprets a user's specifications conveyed with the other tools. The combination of these tools supports two complementary processes in a single environment: design as a constructive process of selecting and arranging graphical elements, and design as a process of browsing and customizing previous cases. SAGE enhances *user-directed* design by completing partial specifications, by retrieving previously created graphics based on their appearance and data content, by creating the novel displays that users specify, and by designing alternatives when users request them. Our approach was to propose interfaces employing styles of interaction that appear to support graphic design. Knowledge-based techniques were then applied to enable the interfaces and enhance their usability.

KEYWORDS: Graphic Design, Data Visualization, Automatic Presentation Systems, Intelligent Interfaces, Design Environments, Interactive Techniques

INTRODUCTION

Graphic displays of information have been valuable for supporting data exploration, analysis, and presentation. Still, current graphics packages remain very limited because: (1) they do not provide *integrative* displays for viewing the relations among several data attributes or data sets, (2) they have time-consuming and complex interfaces, and (3) they provide little guidance for the majority of users who are not experienced graphic designers.

Consider these problems in the context of two graphics in Roth Color Plate 1. In 1a, a sequence of indented text, charts, and a table are aligned to integrate six attributes of *activities* (organization, start, end, status, cost, resource).

In *Proceedings CHI'94*: Human Factors in Computing Systems. April 24-28, 1994. Boston, Massachusetts. This research was partially supported by DARPA under contract #F30602-91-C-0016.

All information about a single activity can be obtained by glancing horizontally across the graphic. Most packages do enable users to create charts and tables like these, but only as isolated displays. Even painstaking cutting, pasting, and resizing (usually the only means provided) are insufficient to layout and sort the bars and text in a coordinated way.

Similarly, current packages provide no way to create a single display with different graphical objects. In 1b, properties of lines, text strings and diamond-shaped marks vary to integrate ten data attributes. Also, graphical objects are *clustered* to express facts (i.e. each diamond is accompanied by two text labels to convey the geographic location, city, and date of battles). Together, these graphics illustrate the large number of possible combinations of object types, their graphical properties, the *encoding spaces* in which they occur (e.g. within a chart, map, table, or network), and the different ways they can be clustered and aligned. Clearly, current menu-style interfaces in spreadsheet packages would not support the creation of so many alternatives, nor could they help users assign data attributes to these graphics easily. Imagine the difficulty of conveying the relationship between data in spreadsheet columns and all the graphical objects and properties in 1b.

Furthermore, imagine the considerable design expertise required of users to produce these displays, including an awareness of the appropriateness of graphic choices for each data type. Even when users can judge the effectiveness of a particular display of their data, they often lack exposure to the many types and combinations of graphics that are possible. Systems that provide the ability to create new integrative designs will need to provide design guidance as well.

One approach to these problems is to build systems that are knowledgeable of graphic design, so they can generate a variety of effective displays based on descriptions of data and viewing goals [1,3,4,9,10]. This research has provided a vocabulary for describing the elements of graphics, knowledge about the appropriateness of their use for different data and tasks, and design operations for combining elements to form integrative displays.

Armed with this knowledge, automatic design systems should reduce the need for interaction and expertise, while providing great flexibility in customizing displays. However, previous automatic design research has not been concerned with supporting interaction with users and has focused on issues of identifying and encoding knowledge of data, tasks, and design. No paradigms have been developed for a collaborative process between human and automated designers.

This paper describes a novel approach to interactive graphic design, in which automatic mechanisms are used to support users, not replace them. The following sections describe an overview of our approach, two major components of the system that correspond to two complementary styles of design, and some sample design interactions which illustrate these capabilities.

OVERVIEW OF CURRENT APPROACH

Our approach to supporting design has been to integrate an evolving automatic presentation system called SAGE [9,10] with two new interactive design tools called SageBrush and SageBook. Both tools enable users to manipulate familiar objects in order to perform natural design operations, shielding users from the more complex representations and operations that SAGE uses to create graphics.

SageBrush (also called Brush) is representative of design tool interfaces in which users specify graphics by constructing sketches from a palette of primitives and/or partial designs. Our goal is to provide a flexible, generative, direct manipulation design interface, in which users can create a large number of possible combinations of graphical elements, customize their spatial and structural relationships, and map them to the data they wish to visualize.

SageBook (also called Book) is an interface for browsing and retrieving previously created pictures (i.e. complete, rendered designs) and utilizing them to visualize new data. Book supports an approach to design in which people remember or examine previous successful visualizations and use them as a starting point for designing displays of new data, extending and customizing them as needed. Our experiences in graphic design, as well as related research on engineering and software design [2,6], suggest that search and reuse of prior cases with customization is a common process. Therefore, our goal is to provide methods for searching through previously created pictures based on their graphical properties and/or the properties of the data they express. A picture found in this way can optionally be modified in Brush prior to sending it to SAGE, which creates a graphic for the new data.

SAGE is an automatic presentation system containing many features of related systems like APT, BOZ, and ANDD [1,3,4]. Inputs are a characterization of data to be visualized and a user's data viewing goals. Design operations include selecting techniques based on expressiveness and effectiveness criteria, and composing and laying out graphics appropriate to data and goals. A detailed discussion of automatic design capabilities, including the operations that produced Roth Color Plate 1a, can be found elsewhere [7,9].

The current version of SAGE goes beyond previous systems in several ways. SAGE can create graphics when users completely specify their designs as well as when they provide no specifications at all. Most importantly, it can accept *partial* specifications at any level of completeness between these two extremes and finish the design reasonably. User specifications serve as *design directives,* which constrain the path of a search algorithm that selects and composes graphics to create a design.

The ability to accept partial specifications from Brush is due to a rich *object representation* of the components of graphic displays, including their syntax (i.e. their spatial and structural relationships) and semantics (i.e. how they indicate the correspondence between data and graphics). The representation allows SAGE to produce combinations of a wide variety of 2D graphics (e.g. charts, tables, map-like coordinate systems, text-outlines, networks). It also enables SAGE to support Book's search for previous pictures with graphical elements specified by users.

The object representation is highly extensible, allowing new graphical objects (e.g. lines, polygons, custom icons) and encoder mechanisms (e.g. charts, color keys, maps) to be added incrementally. For example, when a line object is added to the library, each end-point is defined as having horizontal and vertical positions, enabling the line to be displayed against the axes of a chart. If a map-style is later defined in the library as an encoder that displays horizontal and vertical positions, then SAGE can automatically draw lines on maps (as in Roth Color Plate 1b).

SAGE also contains a richer representation of the *characteristics* of data (e.g. distinguishing scales of measurement, temperature, dates, spatial coordinates, etc). Data transformation operations enable the design of graphics without depending on the surface form of input data (e.g. in relational database terms, SAGE can display N-ary relations and is not dependent on whether data is expressed as multiple binary relations or as a single N-ary relation).

ARCHITECTURE

Figure 1 illustrates the conceptual relationships among SageBrush, SageBook, SAGE, and a Data Selector - a tool for indicating the mapping between data and graphics. The process of retrieving data needs to be integrated with graphic creation but is not the focus of this paper. We are exploring several interactive methods for retrieving and transferring data to the selector, where data appears as a table whose headers can be mapped to graphics (Figure 2).

Users interact with Brush to create graphic design *sketches,* which are schematic views of designs. These are translated into *design directives,* which are specifications expressed in SAGE's graphic representation language. Directives include:

- grapheme and property choices (e.g. color and size of circles, lines, text, and other graphical objects),
- encoding mechanisms that provide frames of reference against which properties of graphemes are interpreted (e.g. 2-axis chart, table, map, network),
- layout constraints (e.g. alignment of multiple charts horizontally; ordering of labels and graphemes),
- grouping constraints indicating that clusters of graphemes are being used to express a single fact (e.g. a bar annotated with a text string; a cluster of items around a city on a map),
- mappings between data and these graphic elements.

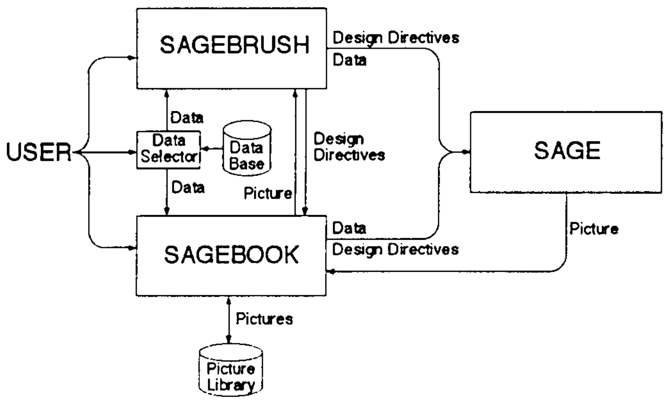


Figure 1: Architecture

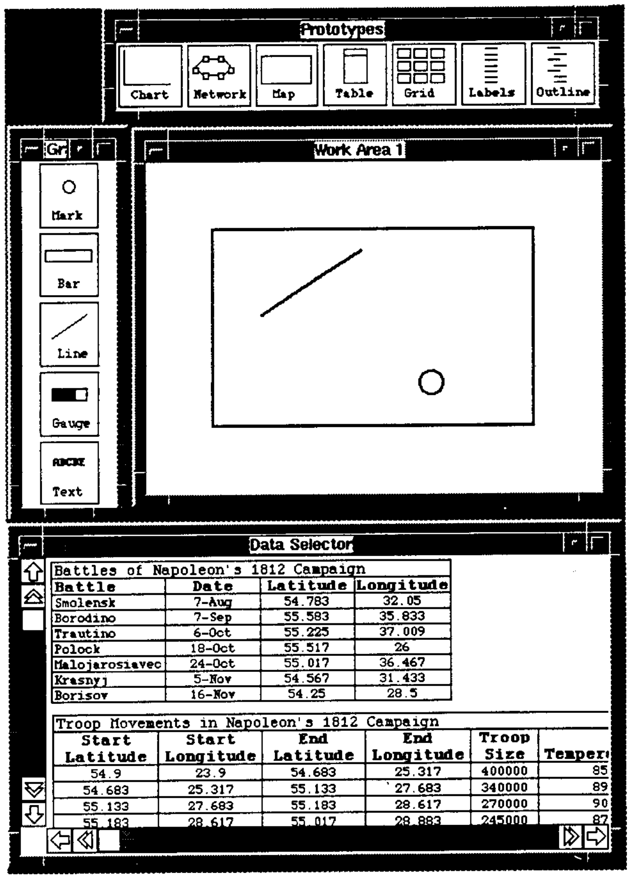
Design directives from Brush serve two purposes: they guide SAGE's automatic processes and provide criteria for Book to use in searching its library of previously designed pictures. Brush can also translate graphics produced by SAGE back into sketches so that users can modify them.

Users interact with Book to view and save pictures created by SAGE. The saved information includes a bit map scaled to a browsable size, a sequence of design operations that SAGE can use to reconstruct the picture efficiently (i.e. without redesigning), the picture's data and data type characteristics, and a complete representation of the rendered graphic. Book searches its picture library based on data users specify with the Selector and/or design directives derived from sketches created in Brush's work area (Figure 2). Users request the creation of a graphic based on a previously found one by transferring it to Brush (where they modify it as a sketch) or directly to SAGE. The next sections describe these components in detail.

SAGEBRUSH

Brush is representative of tools with which users sketch or assemble graphical elements to create designs and map them to data. Brush provides users with an intuitive and efficient language for sketching their designs, and translates these sketches into a form that can be interpreted by SAGE. There are other possible styles of graphic design interface that could be coordinated with SAGE's internal design mechanisms. One alternative is the demonstrational approach proposed for Gold [5], in which users draw examples of displays. Our claim is that any interactive design interface that attempts to provide complete coverage of graphics will require a knowledgeable system behind it to be successful.

An example: Figures 2, 3, and Roth Color Plate 1b illustrate a sequence for creating a new version of the famous graphic by Minard showing Napoleon's 1812 Campaign [11]. One data set describes the march segments (start and end latitudes/longitudes of each segment, the number of troops remaining, and the temperature). The other data set contains the city, date, and location of each major battle. These will be visualized by composing multiple graphemes and their properties on a map.



Battles of Napoleon's 1812 Campaign

Battle	Date	Latitude	Longitude
Smolensk	7-Aug	54.783	32.05
Borodino	7-Sep	55.583	35.833
Trautino	6-Oct	55.225	37.009
Polock	18-Oct	55.517	26
Malojarosiavec	24-Oct	55.017	36.467
Krasnyj	5-Nov	54.567	31.433
Borisov	16-Nov	54.25	28.5

Troop Movements in Napoleon's 1812 Campaign

Start Latitude	Start Longitude	End Latitude	End Longitude	Troop Size	Temper
54.9	23.9	54.683	25.317	400000	85
54.683	25.317	55.133	27.683	340000	89
55.133	27.683	55.183	28.617	270000	90
55.183	28.617	55.017	28.883	245000	87

Figure 2: Starting a design sketch in SageBrush.

Anchoring new designs with partial prototypes. The creation of a new design begins with a user's selection of a partial prototype. As illustrated in Figure 2, Brush's interface consists of a design work area (center) into which users drag prototypes (top), graphemes (left), and data names (bottom). *Prototypes* are partial designs, each with a spatial organization, graphemes, and/or encoders that commonly occur together. Encoders are frames of reference for interpreting properties of graphemes. For example, axes enable us to interpret (i.e. derive a data value from) the position of a bar in a chart.

The choice of prototypes to include in the top menu can be customized to applications and could include previously designed graphics. Although primarily a constructive interface, Brush still allows design to be viewed as a process of refining prior, effective graphics. The first prototype in the top-left of Figure 2 is a general one for constructing all charts. It is actually a composite of horizontal and vertical axes. Although users could construct charts by assembling separate axes, doing so requires more steps and appears less intuitive than selecting a chart prototype. A similar rationale led to a network prototype, consisting of both graphemes (i.e. lines) and an encoder against which the graphemes are interpreted (i.e. the nodes). This eliminates the need for users to construct networks from primitives each time. In the example, a map prototype (more precisely, a 2D spatial coordinate display) was dragged to the design work area.

Customizing by adding primitives to prototypes. Prototypes are extended by adding graphemes. While the chart and map prototypes have no graphemes, dragging them into the design work area creates an *encoding space* which supports new design choices. The encoding space of a chart or map is defined by the interior of the two axes or coordinate-frame, respectively. Dragging line and mark graphemes (to represent march segments and battles) from the left window into the map's encoding space results in directives to SAGE to include these grapheme types in a design, with their positional properties interpreted relative to the map's coordinate system.

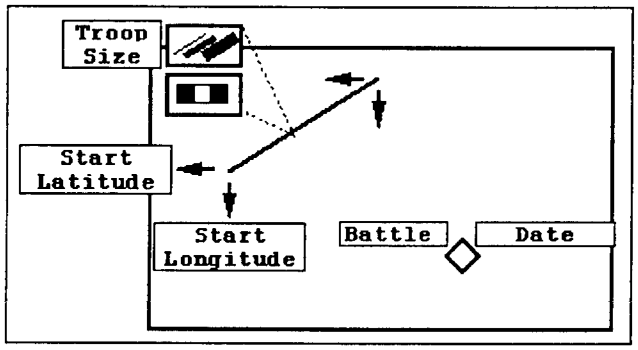


Figure 3: Property selection and data mapping in SageBrush's work area.

Customizing the properties of graphemes. Graphemes have other properties for encoding data besides position. Properties are chosen by selecting *property icons*, displayed by double-clicking a grapheme in the design work area. Double-clicking on the line in Figure 3 displays a menu of line properties (width and color) and arrows representing the positional properties of end-points. Selecting a property directs SAGE to use it to encode data in a design but does not indicate the data to which it corresponds. Double-clicking on a property icon allows users to convey specific directives (e.g. make all marks diamond-shaped or all lines blue; *reject* the use of color).

Completing the graphic requires a way to create *grapheme clusters*. As described above, dragging graphemes into an encoding space results in directives to use their positional properties in a design. When two or more graphemes are dropped close together in the same space, the position of one is interpreted relative to the axes or coordinate system, while the positions of others are interpreted to convey *association* by adjacency. In Figure 3, two text strings have been placed next to the mark (which has been customized to be diamond-shaped) to convey association. Note that Brush only determined that the two strings and diamond are associated. SAGE must infer which of the three is used to convey position in the coordinate system (using knowledge of data characteristics and graphic expressiveness criteria [8,9]). Of course, a user can explicitly double-click on the diamond and select its property icons for position (a pair of arrows).

Communicating the mapping of data to graphics. Dropping a grapheme in a chart and selecting its color result in directives to SAGE to generate a design where position and color encode data. It does not specify which data (i.e. relation domains) to assign to these properties. While SAGE could attempt to infer this (just as it could also make choices of graphemes and properties), users can explicitly make these choices by dragging data labels from the Data Selector (bottom Figure 2), and dropping them on property icons. In Figure 3, Troop Size was mapped to line thickness and Start Latitude and Start Longitude to the position of one end of the line. Battle and Date have been mapped to text labels adjacent to the diamond (dragging a data name into the space simultaneously specifies that a text grapheme be used and maps the data to it). The completed design resulting from this interaction is shown in Roth Color Plate 1b, which was generated by SAGE.

Coordinating multiple design spaces. In addition to defining encoding spaces, prototypes also define *layout spaces*, which enable users to specify the relative positions of prototypes with respect to each other. There are two types of layout spaces, reflecting adjacency and embedding relationships. Adjacency spaces enable horizontal and vertical alignments among charts, tables, maps and other prototypes. Two charts and a table in Figure 5 have been sequenced by placement adjacent to each others' layout spaces. Embedding spaces enable the placement of one prototype within another (e.g. a network placed within a map or chart; a list placed within a network node).

Finally, it is important to emphasize that all of these design choices are optional. Users only need to specify the data they wish to visualize, but may further specify (to any level of completion):

- prototypes only,
- prototypes and additional graphemes,
- graphemes and their properties,
- the mapping of data to graphemes, and
- the mapping of data to specific grapheme properties.

The Napoleon example illustrates that users needn't specify all mappings. The system inferred End Latitude, End Longitude, and Temperature (and could have made choices for the other data, possibly differing from those of the user). The strength of this approach is that it can (1) reduce the amount of work needed to convey design choices and map them to data, (2) enable the construction of composites that could not be created by menu-based approaches, (3) provide design expertise to supplement that of users, and (4) provide design directives for SageBook to use in searching its library of previously constructed pictures.

SAGEBOOK

The goal of Book is to provide users with the ability to create new pictures analogous to existing ones they consider useful. Our intent is to provide users with access to a growing portfolio of graphic designs to provide ideas for visualizing their data. Book capitalizes on the explicit representation of designs and data characteristics by SAGE to provide a vocabulary for expressing search criteria.

Book provides two mechanisms for browsing pictures. The first is a file-folder metaphor analogous to that used in the Macintosh system, in which pictures created by SAGE are named and stored in locations defined by users. The second mechanism provides browsing by two types of picture content: graphical elements and data. Search criteria are based on exact match or partial overlap with data in the Data Selector and/or design elements in Brush.

Figure 4 illustrates the interface for browsing pictures retrieved by a search based on data overlap. The data for the search were facts about activities in a project management database (the final picture is shown in Roth Color Plate 1a). Pictures in the library that expressed similar data were listed by the interface. As a user selects each picture name, its bitmap is displayed. Multiple full size pictures can be displayed and arranged by users for comparison.

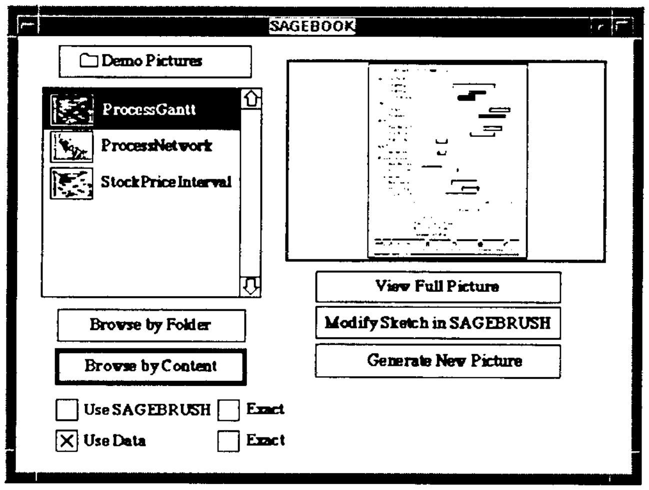


Figure 4: Browsing graphics by their data content in SageBook.

We have designed search criteria for several levels of match overlap based on data. These involve retrieving pictures which:

- show exactly the same data relations/attributes as in the data selector (e.g. find pictures of Activity End),
- contain the selected data in addition to other data,
- show different data that have the same underlying data characteristics.

For example, a *data relation* (to use relational database terms) representing quarterly expenses for a company's departments (Department, Business Quarter, Operating Cost) may have the same properties as another relation for stock market data (Stock, Calendar Date, Shares Traded). Both relations contain three domains with identical data characteristics: a nominal type, a temporal coordinate, and a quantity. There is also exactly one quantity for each nominal-time pair in both relations (i.e. functional dependency). See [8] for a more complete treatment of data characterization relevant to graphic design.

We have designed search criteria for several levels of match overlap based on graphical elements as well. These involve retrieving pictures that (1) show exactly the same design elements as those in the Brush sketch and (2) contain the Brush elements as a subset of a more complex design.

Our current work is addressing the problem of defining match criteria for combinations of data and graphical properties. We are also exploring similarity criteria for defining close matches with partial overlaps. For example, we need criteria for determining whether a network where the *color* of links encodes data is more similar to a chart using the color of bars or to another network where the *widths* of links encode data (i.e. what graphical elements are salient to users). Our intuitions suggest the latter, but a cognitive model based on user studies is needed to define similarity, as well as to verify the appropriate graphical primitives for the Book and Brush interfaces.

Our preliminary view is that searches based on different criteria serve different purposes for users, including:

- discovering how basic techniques can be expanded with additional graphical elements (e.g. how a network can encode using additional text or marks along its links or within its nodes),

- quickly retrieving a picture whose name has been forgotten, but some of whose elements are known,

- minimizing the effort of sketching a new design by retrieving a picture similar to the one desired and then modifying it in Brush.

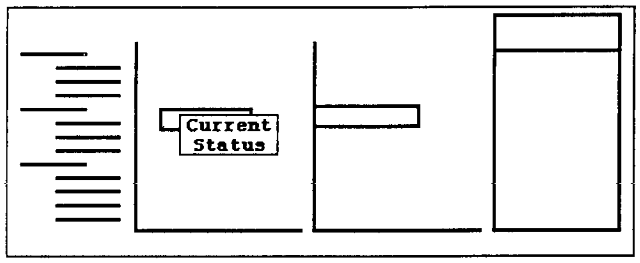


Figure 5: Adding graphics in SageBrush to a picture found using SageBook (see Figure 4).

The last case is illustrated in Figures 4 and 5. Book found an *indented chart* with *color-coded interval bars* for data matching only part of a large data selection (activity, organization, start, end, current-status, labor-cost, resource). The chart was converted to a sketch in Brush, and the user added a bar chart and table aligned with the original interval chart. The user also mapped Current-Status to the interval grapheme, leaving it to automatic mechanisms in SAGE to map it to color (because the original picture used color). SAGE can automatically assign Activity to the Y-axis, dates to the interval bar, and Labor-Cost to the horizontal position of the bars in the added chart, based on expressiveness rules for these graphical properties. The resulting picture is shown in Roth Color Plate 1a. SAGE integrated all design elements and determined appropriate data mappings. Notice that Resource is placed in the table, while Organization is placed in the indentation of the Y-axis...an arbitrary choice that a user can easily reverse. The operations that produced Roth Color Plate 1a can be found in [9].

SUMMARY AND CONCLUSIONS

Our approach views the task of creating visualizations of data as a combination of two interrelated processes:

- *constructing* designs from graphical elements, and
- finding and customizing relevant prior examples.

The extent to which each process occurs varies with user and context. Consequently, we created two tools that play flexible, mutually supportive roles to enable design. SageBrush provides users with an interface for constructing graphic designs and customizing graphics found with SageBook. Brush also enables users to compose graphical queries to be searched using Book.

Another central theme of our approach is the use of automated design knowledge in SAGE to provide new display capabilities, to enhance the usability of graphic design interfaces, and to provide design expertise when needed by users. These are realized in several ways.

First, SAGE enables users to create a wide variety of *integrative* displays, which coordinate multiple spaces, graphemes, and visual properties to show the relationships among several data attributes or data sets. This is possible because SAGE recognizes and parses the structure and semantics of sketches that users construct.

Second, knowledge enables a system to automatically design a graphic when requested by users. This can occur when users do not know how to represent data (i.e. they lack expertise in general or for a specific problem) or when they want to compare alternative designs with the ones they have created.

Third, SAGE reduces the work of designing a graphic by completing it automatically when partially specified. This often eliminates the need for users to assign data to elements of the graphic, select graphical properties once objects are specified, or perform other repetitive selections.

Fourth, SAGE makes it possible to search displays created previously based on meaningful criteria: the data and graphic elements they contain. Without this knowledge, Book would be limited to browsing graphics based on file attributes.

There are many research problems remaining, especially for supporting users with limited graphics expertise. First, the operation of any automatic presentation system depends on the existence of data characterizations [8]. In this research, data characterizations were already present in the database or spreadsheet. We will be exploring ways to infer them or obtain them interactively.

Second, although SAGE considers user information-seeking goals or tasks [1,8,9], no attempt was made to provide users with the ability to specify these. We are considering creating a goal-selection interface so users can convey their intentions as design directives.

Finally, there are numerous new graphic design problems to address, including the design of interactive mechanisms for manipulating data displays, displays of large data sets, and graphical techniques such as animation and 3D. See [7] for a more complete discussion of research problems in this area.

REFERENCES

1. Casner, S. M. A Task-Analytic Approach to the Automated Design of Information Graphic Presentations. *ACM Transactions on Graphics*, 10, 2 (Apr. 1991), 111-151.

2. Fischer, G. Cognitive View of Reuse and Redesign. *IEEE Software*, (July, 1987), 60-72.

3. Mackinlay, J. D. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics*, 5, 2 (Apr. 1986), 110-141.

4. Marks, J. W. Automating the Design of Network Diagrams. Ph.D. thesis, Harvard University, 1991.

5. Myers, B. A., Goldstein, J., and Goldberg, M. A. Creating Charts by Demonstration. *Proceedings SIGCHI'94 Human Factors in Computing Systems,* Boston, MA, ACM, April, 1994.

6. Navin-Chandra, D. *Exploration and Innovation in Design: Towards a Computational Model.* Springer-Verlag, 1991.

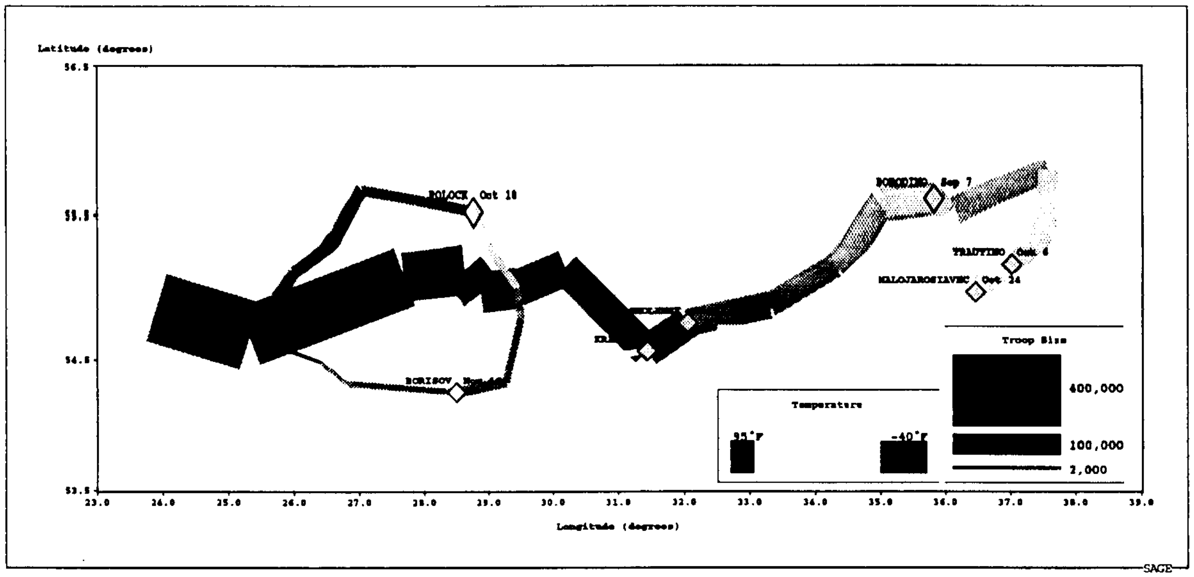
7. Roth, S. F. and Hefley, W.E. Intelligent Multimedia Presentation Systems: Research and Principles. In Mark Maybury (Ed.) *Intelligent Multimedia Interfaces,* AAAI Press, 1993, pp. 13-53.

8. Roth, S. F. and Mattis J. Data Characterization for Intelligent Graphics Presentation. *Proceedings SIGCHI'90 Human Factors in Computing Systems*, Seattle, WA, ACM, April, 1990, pp. 193-200.

9. Roth, S. F. and Mattis, J. Automating the Presentation of Information. *Proceedings IEEE Conference on AI Applications*, Miami Beach, FL, Feb. 1991, pp. 90-97.

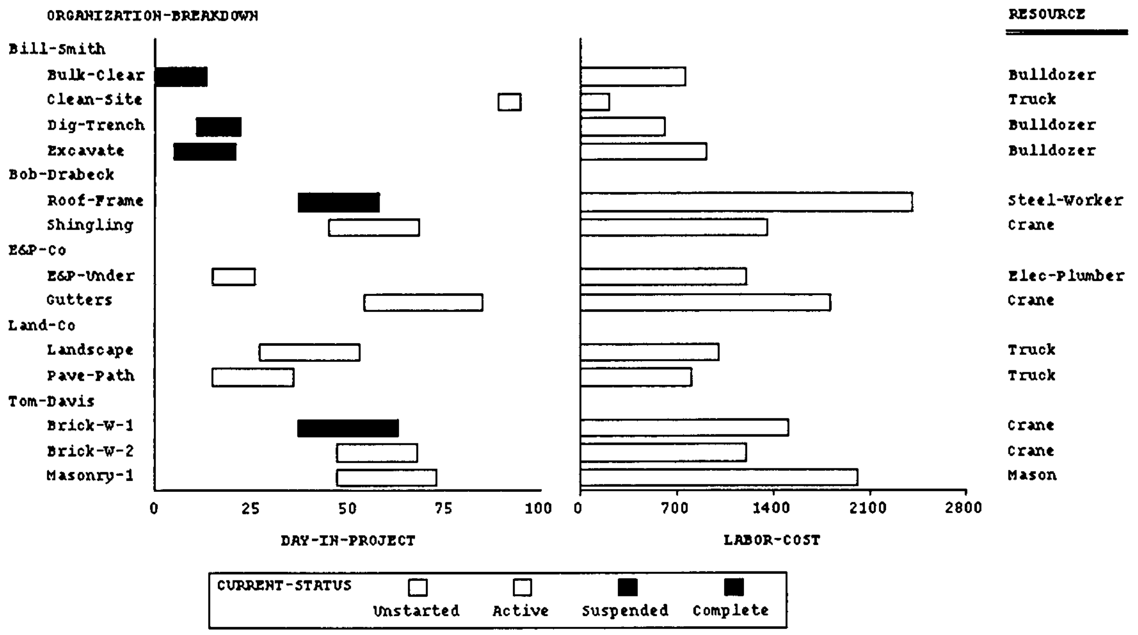
10. Roth, S. F., Mattis, J., and Mesnard, X. Graphics and Natural Language Generation as Components of Automatic Explanation. In Sullivan and Tyler (Ed.), *Intelligent User Interfaces*, Addison-Wesley, Reading, MA, 1991, 207-239.

11. Tufte, E. R. *The Visual Display of Quantitative Information.* Graphic Press, Cheshire, CT, 1983.



Roth, Color Plate 1b: "Napoleon's 1812 Campaign", designed interactively using SageBrush. In all, ten data attributes are integrated in a single map-like coordinate space using several kinds of graphical objects. The lines trace the path of Napoleon's eastward advance and westward retreat, including the path of troops that branched north to protect his main force. Line thickness conveys the number of troops traveling each segment, line color conveys the temperature, and the dates and sites of battles are signified by yellow diamonds and text.

Napoleon's eastward advance began in extreme heat, with the weather cooling as he approached Moscow (bright red at left fading to pink at upper right). During the westward retreat, the army circled back to retrace its route while the temperature dropped below freezing (the light blue segments overlying the red path). Upon reaching Krasnyj, the army veered south from its previous route. When the march ended, less than three percent of Napoleon's troops remained, as shown by the striking decrease in line thickness.



Roth, Color Plate 1a: Result of a SageBrush customizaton of a picture retrieved using SageBook. Six data attributes are integrated in a coordinated set of aligned displays that illustrate project management data.

A FRAMEWORK FOR KNOWLEDGE-BASED, INTERACTIVE DATA EXPLORATION

Jade Goldstein, Steven F. Roth, John Kolojejchick, and Joe Mattis

School of Computer Science

Carnegie Mellon University

Pittsburgh, PA 15213

(412) 268-7690

Steven.Roth@cs.cmu.edu

ABSTRACT

In this paper, we propose a framework that combines the functionality of data exploration and automatic presentation systems to create a knowledge-based, interactive, data exploration system. The purpose of a data exploration system is to enable users to uncover and extract relationships hidden in large data sets. The purpose of an automatic presentation system is to reduce the need for users and application developers to have graphic design expertise and to spend much time interacting with graphics packages to view their data. Previous work on data exploration was limited to query mechanisms that were often complex to learn and difficult to use, data manipulation mechanisms that did not provide complete coverage of the operations needed by users (especially the ability to form ad hoc groupings of data), and graphics that were restricted to a small set of predefined visualizations. Automatic presentation research, although addressing these issues, has been limited to the display of small data sets. This research has also not developed approaches to combine interactive, user-directed processes of design and data manipulation with automatic presentation mechanisms. We propose a framework that overcomes these limitations of current data exploration systems and integrates new interactive capabilities with automatic presentation components. This approach to supporting data exploration integrates recent work on SageTools, an environment for interactive and automatic presentation design, with a prototypical interactive data manipulation system called IDES. In this paper, we present our work on the IDES data manipulation capabilities and discuss requirements for coordinating them with automatic presentation of large data sets.

KEYWORDS: Data Exploration, Data Visualization, Intelligent Interfaces, Automatic Presentation Systems, Graphic Design, Computer-supported Design, Large Data Sets.

1. INTRODUCTION

The widespread use of databases and computers is requiring growing numbers of people to use and understand increasing amounts of information. These databases contain diverse data, including combinations of quantitative, temporal, categorical, hierarchical, geographic, and other types of information. Users of these large data repositories will not be limited to scientists and technically oriented professionals. Thus, there is a need for software that assists users with diverse levels of expertise in their data exploration tasks without substantial training and/or effort. The tasks users need to perform with information go beyond retrieving simple facts and

To Appear in Journal of Visual Languages and Computing, December 1994. This research was partially supported by DARPA under contract #F30602-91-C-0016.

answering focused questions. Instead, the tasks involve solving problems and making decisions based on the current state of the data, which is often repeatedly refined and extracted. These decisions depend upon the user's understanding of the relationships latent within the data. Once an interesting relationship is discovered, the user can use it to guide the next instruction to the system. This iterative and interactive process, which Brachman [5] calls data archaeology, is initiated and controlled by people. In contrast, data mining [10] emphasizes the use of automatic mechanisms to search for patterns.

We propose a framework for building a *knowledge-based interactive data exploration* system that will support the data archaeology process. Doing so requires understanding the interactive and iterative processes of data exploration. Specifically, tools must support three kinds of exploration subtasks:

1. *data visualization operations,* which include finding or designing and creating effective graphics.
2. *data manipulation operations,* which include selecting data for display, focusing on particular attributes of the data, and grouping or reorganizing data.
3. *data analysis operations*, which include statistical testing, summarization, and transformation for understanding properties of the data.

Of course, these subtasks are interdependent and overlapping. For example, a user may wish to select data directly from a visualization (a data manipulation operation performed on a visualization). Data exploration systems will need to employ other user interface techniques that provide easy interaction and communication with the components of the system that generate displays, such as direct manipulation [11]. They also need to employ query mechanisms that allow the user to focus on their data and results and not on the process of creating a query.

Data Visualization. Automating portions of the data exploration process requires specific knowledge relevant to each of the three exploration subtasks. For data visualization, research has focused on systems that can automatically generate a display of data composed of graphics and text or developing new techniques customized to specific tasks or types of analyses. The intent of automatic presentation systems is to relieve users and application programmers of the need for graphic design knowledge and of the task of designing and specifying displays. This lets users concentrate on their goals for viewing information. Furthermore, for complex combinations of data, automatic presentation systems have the ability to generate graphics that users might not even consider. However, research on existing automatic presentation systems has been narrow, primarily focusing on representing knowledge of graphic design and not on interactive mechanisms for performing design. Recent work [19] has applied this technology to create computer supported data-graphic design tools, in which users can interactively specify and/or search and choose from a library of previously created graphics. This system, called SageTools, builds on an automatic presentation system called SAGE [16, 17, 18] and provides an approach to situating automatic technology in an environment which supports iterative and interactive data-graphic creation.

Previous automatic presentation systems have also been limited to the display of small data sets - those that fit well in a single computer window [6, 12, 14, 16, 17, 19]. These systems are unable to support many applications that use hundreds or even thousands of data elements. When dealing with such large data sets, it is no longer sufficient for the presentation system to create only a display. The system must also provide mechanisms for interactive manipulation and analysis of large data sets.

Data Manipulation. The functionality needed for large data set manipulation and analysis in an automatic presentation system is an extension of that needed in a conventional data exploration system. Thus, it is useful to first explore standard data exploration systems in which there are several popular approaches to providing interactive data manipulation techniques. Data base query systems, statistical packages and electronic spreadsheets provide various levels of support for accessing, modifying and reorganizing data. However, these are not well integrated with effective graphics techniques, nor do they provide flexible, low-effort tools that can be used by a broad cross-section of users. Query systems have great flexibility but require learning programming skills and lack convenient tools for organizing and summarizing information. Spreadsheets have greater intuitive appeal but lose the flexibility of query languages for selecting data. They also have a limited ability to rapidly reorganize information. Attempts to provide spreadsheets with these capabilities have often resulted in new programming environments rather than effective interface mechanisms.

In exploring the nature of data manipulation techniques, we classify data manipulation goals into three categories: *controlling the scope* (selecting desired portions of data), *choosing the level of detail* (creating and decomposing aggregates of data), and *selecting the focus of attention* (concentrating on the attributes of data that are relevant to current analysis). We have used this classification to evaluate the functionality of existing data manipulation interface techniques.

Based on these results, we have expanded an interface mechanism called the Aggregate Manipulator (AM) [15] and combined it with Dynamic Queries (DQ) [1] in a prototype system called IDES (Interactive Data Exploration System). We use the results of our experience with IDES to propose extensions to SageTools to handle large data sets. The goal of these extensions and the integration of IDES with SageTools is a knowledge-based interactive data exploration system, a tool for users who are investigating relationships in large data sets.

In this paper, we concentrate on an overview of a framework for accomplishing this goal and the details of the IDES system. Section 2 discusses the proposed integration of SAGE and IDES. Section 3 discusses the data manipulation operations: controlling the scope, choosing the level of detail, and selecting the focus of attention. Section 4 explains how we selected the interface mechanisms used in IDES. Section 5 gives an overview of the design of IDES. Section 6 provides examples of how IDES is an effective tool for covering the data manipulation operations, due to the complementary nature of AM and DQ, as well as IDES's inherent flexibility in methods of exploring large data sets. This is demonstrated in two domains: shipping and real estate. Section 7 highlights the extensions required for SAGE as illustrated by the properties of large data sets in IDES.

2. A FRAMEWORK FOR KNOWLEDGE-BASED INTERACTIVE DATA EXPLORATION

In this section, we propose a framework for a knowledge-based interactive data exploration system. Figure 1 shows the conceptual architecture that will be discussed in this section. The framework is composed of system modules, communication protocols, and knowledge of data and task characteristics and graphic design. Users communicate through direct manipulation interfaces, which generate appropriate directives to other components. The components access stored knowledge relevant to their functionality and the data. The following subsections will expand on these three concepts.

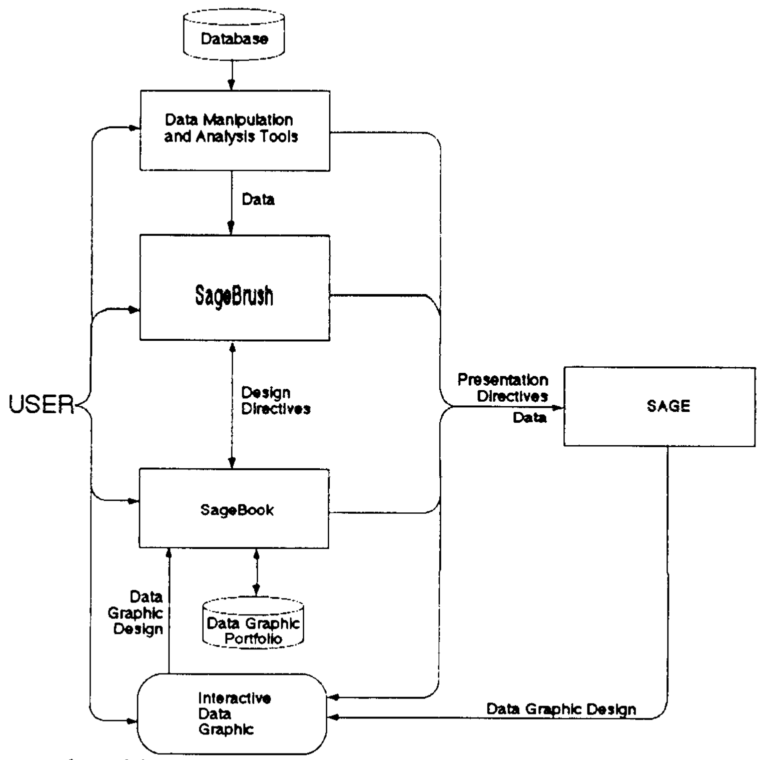


Figure 1: Proposed architecture for knowledge-based interactive data exploration.

2.1 System Components

All the components in the architecture provide one or more mechanisms for supporting data visualization, data manipulation, or data analysis. SAGE [16, 17, 18] is an automatic presentation system containing many features of related systems like APT, BOZ, and ANDD [12, 6, 14]. SAGE uses a characterization of data [16] to be visualized and a user's data viewing goals to design graphics. Design operations include selecting techniques based on expressiveness and effectiveness criteria and composing and laying out graphics appropriate to the data and user goals.

SageTools [19], an extension of SAGE, goes beyond previous presentation systems in several ways. SAGE can create graphics when users completely specify their designs (using SageBrush) as well as when they provide no specifications at all. Most importantly, it can accept *partial* specifications at any level of completeness between these two extremes and finish the design in a reasonable manner. User specifications generate *design directives*, which constrain the path of a search algorithm that selects and composes graphics to create a design. The ability to accept partial specifications is due to a rich *object representation* of the components of graphic displays, including their syntax (i.e., their spatial and structural relationships) and semantics (i.e., how they indicate the correspondence between data and graphics). The representation allows SAGE to produce combinations of a wide variety of 2D graphics (e.g., charts, tables, map-like coordinate systems, text-outlines, networks). It also enables SageBook to support search for previously created pictures with graphical or data elements specified by users. A detailed discussion of automatic design capabilities can be found elsewhere [6, 12, 16, 17, 18, 19].

SageBrush is a tool with which users sketch or assemble graphical elements to create designs and map them to data. SageBrush provides users with an intuitive and efficient language for sketching their designs, then translates these sketches into a form that can be interpreted by SAGE. The assumption is that any interactive design interface that attempts to provide complete coverage of graphics will require a knowledgeable system behind it to be successful.

New designs begin with a user's selection of a partial prototype. As illustrated in Figure 2, SageBrush's interface (left window) consists of a design work area (center) into which users drag prototypes (top), graphemes (left), and data attribute names (bottom). *Prototypes* are partial designs, each with a spatial organization, graphemes, and/or encoders that commonly occur together. Encoders are frames of reference for interpreting properties of graphemes. For example, axis encoders enable us to interpret (i.e., derive a data value from) the position of a bar (a grapheme) in a chart (a spatial framework).

Prototypes are extended by adding graphemes and selecting properties of them to assign to data attributes (e.g., their color, shape, size, position, etc.). While the chart and map prototypes have no graphemes, dragging them into the design work area creates an *encoding space* which supports new design choices. The encoding space of a chart or map is defined by the interior of the two axes or coordinate-frame, respectively. Dragging a mark grapheme into a chart's encoding space results in directives to SAGE to include these grapheme types in the design, with their positional properties interpreted relative to the chart's coordinate system.

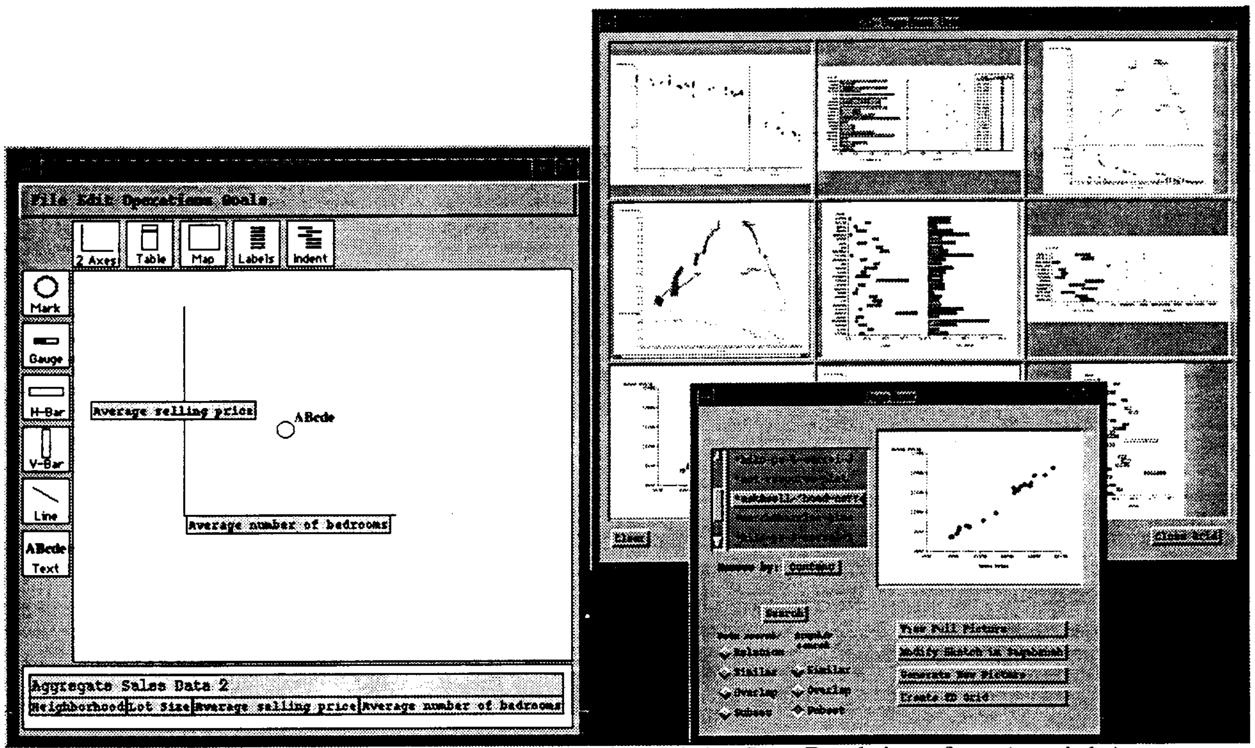


Figure 2: The SageBrush interface (on left) and the SageBook interface (on right)

SageBook (Figure 2, right windows) is a tool for browsing and retrieving previously created pictures (i.e. complete, rendered designs) and utilizing them to visualize new data. SageBook supports an approach to design in which people remember or examine previous successful visualizations and use them as a starting point for designing displays of new data. After selecting a visualization, users extend or customize it as needed. Our experiences in graphic design suggest that search and reuse of prior cases with customization is a common process. Therefore, our goal is to provide methods for searching through previously created pictures based on their graphical properties and/or the properties of the data they express. A picture found in this way can optionally be modified in SageBrush prior to sending it to SAGE, which creates a similar style graphic for the new data. SageBook capitalizes on the explicit representation of designs and data characteristics by SAGE to provide a vocabulary for expressing search criteria.

SAGE and SageBrush will need to be extended to handle large data sets. For example, SAGE will need to create visualizations that group portions of data to control level of detail (e.g., given screen constraints and user direction, SAGE might choose the picture in Figure 4b rather than in Figure 4a). Section 7.2 will briefly discuss how the syntactic and semantic representation of graphics and data characteristics will need be extended to create displays containing these data groupings. SageBrush will need to include new interactive graphemes, such as the aggregate gateway grapheme, the graphical representation for a grouping of data.

Most of all, SageTools will need a Data Manipulation component to provide a variety of interface mechanisms for grouping, filtering, transforming and performing other operations on data. Data manipulation operations are the focus of IDES, and Section 7 will discuss their role in the broader framework. Sections 3-6 explain the purpose and capabilities of IDES in detail.

2.2 Communication

Communication between the user and the components of the exploration system occurs via directives, which are messages that can result from user actions. Directives are generated through interactions with a variety of user interfaces, including menu commands, dialog boxes and direct manipulation techniques, such as sliders, button selection, and drag and drop mechanisms. There are several types of directive:

Presentation directives are the means by which users communicate their intent for the creation of graphics. There are four types of presentation directives: task, style, aesthetic, and data. Task directives are used to communicate a user's information-seeking goals (e.g., the data must be viewed very accurately). Style directives are display goals which affect the types of graphic techniques that are used to encode data (e.g., use saturation to show neighborhood as in Figure 4). Aesthetic directives influence the choice of *values* for graphical techniques (e.g. the color values, scale of axes, shape of objects). Data directives, discussed in subsequent sections, are the means by which the user communicates with the data manipulation and analysis component of the system. Following are some examples of each types of presentation directive.

Task Directives:

- indicate specific information-seeking goals for particular data attributes, e.g., looking up accurate values, detecting correlations, locating needs, scanning for differences.
- prioritize data attributes, e.g., the "cost" data attribute should be displayed more prominently than the "quantity" attribute, and the information-seeking goals for cost have a higher priority than the information-seeking goals for quantity.
- will need to include focus of attention and level of detail directives, because the choice of number of attributes and the groupings of data affect the choice of graphical techniques.

Display Directives:

- Style Directives: Convey combinations of graphical and textual techniques. For example, users may specify a display with two bar charts side by side, whose vertical axes are

identical, or that they don't want to use color to encode a data attribute. Design directives will also need to convey interface objects to be included in displays to perform data manipulation.
- Aesthetic Directives: Inform the presentation system of the user's preferences for the appearance of the picture, e.g., use red, blue, and white, or draw the picture in a 3" x 5" window.

Previous automatic presentation systems did not provide interfaces for users to convey directives. In SageTools, task, style and aesthetic directives are generated through SageBrush and SageBook specifications. For an interactive data exploration system, we need to expand the task and display directives to include operations associated with large data sets. We also need to introduce data directives for communication with the data manipulation and analysis component of the system. Section 7.1 will briefly discuss the data directives.

2.3 Data Characterization and Design Knowledge

A knowledge-based interactive data exploration system must be application-independent, yet able to interpret data sufficiently in each new application. It is able to do this because of a general vocabulary for describing the data and task characteristics of domains. It uses these characteristics along with design knowledge to generate graphics [16, 17, 18, 19].

The data characterization is provided by an application developer or user and can be modified or updated by the user through a data characterization generator. It is based on an underlying vocabulary for describing the semantic and structural properties of the data (actual and derived) that are relevant to presentation design [4, 16]. A complete set of data characteristics enables appropriate mappings between the data and graphics. All current work on automatic presentation is founded on a strong definition of the data characteristics. Data characteristics include:
- *sets of data objects,* including distinctions among quantitative, ordinal, and nominal scales of measurement; whether objects represent coordinates or amounts, where a coordinate is a point or location temporally, spatially or otherwise (e.g., calendar-date) and an amount is a value not embedded in a frame of reference (e.g., weight or number-of-days); ordering conventions among objects, etc.
- *attributes of objects,* e.g., whether an attribute or relation maps from one object to one or more other objects; whether missing values are meaningful; the arity of a relation; etc.
- *higher order relationships* among attributes, e.g., the semantics underlying the relation between the beginning, end and duration of a time interval or the relationship between longitude and latitude in a two-dimensional spatial representation.
- *domain of membership*, which refers to whether data measures time, space, temperature, currency, mass, etc., which enables a system to preserve such standard conventions as time displayed along a horizontal axis, and East-West appearing right to left.
- *algebraic dependencies* among database elements, e.g., each bar in the stacked bar chart displays the sum of its parts.

The data characterization is interpreted by using application-independent design knowledge. Design knowledge contains two components: a library of presentation techniques and mechanisms for selecting and combining techniques. The library of presentation techniques consists of: (1) graphical and textual techniques for displaying the components of various kinds of tables, charts, maps and network diagrams; (2) information describing the types of data for which the technique is suitable (e.g., the graphical technique color is suitable for nominal data with six or fewer items); and (3) the syntactic or structural relations among elements used in graphics (e.g., axes, lines, points, labels). The presentation design knowledge contains information about which techniques best satisfy which goals, how techniques can be combined, and which combinations of techniques create the most effective presentation for the users' data, according to their information seeking and display goals. It also has knowledge about how to structure, organize, and lay out displays and their components. Extensions to this knowledge and to the data and task characterization language are needed to support large data set exploration and are discussed in subsequent sections.

This section provided an overview of the components of a framework for data exploration environments. It was based on prior research on SageTools, which was an approach to computer supported data-graphic design and IDES, an experimental data manipulation tool for large data sets. Our future research will involve extending SageTools to incorporate the features of IDES so that visualizations created with SageTools include appropriate data manipulation operations. Sections 3-6 will discuss the implementation and functionality of IDES, and Section 7 will discuss the extensions necessary for large data set functionality.

3. DATA MANIPULATION: CONTROLLING SCOPE, FOCUS OF ATTENTION, & LEVEL OF DETAIL

The exploration goals that a user will have are clearly task dependent. These goals are also dynamic, changing as the user views various data and displays. Data manipulation is one of the processes that users perform in data exploration. Springmeyer [20] performed an extensive empirical analysis of the processes that scientists use when performing data analysis. Her category "data culling" is most similar to that of data manipulation. We have analyzed the data manipulation process in detail for object-attribute data (data which consists of an object such as a house and several attributes for that object, such as selling-price and number of bedrooms) and have identified three types of exploration operations: controlling the scope, selecting the focus of attention, and choosing the level of detail.

Controlling *scope* involves restricting the amount of data one wishes to consider. There are two ways this can be accomplished: (1) selecting a subset of values of a data attributes, such as only cities with a population over 2 million, or (2) disjunctively join subsets of the data, such as data for cities where the population is over 1 million (set 1) or cities which have a population between 2 million and 10 million and are in the Eastern US (set 2)

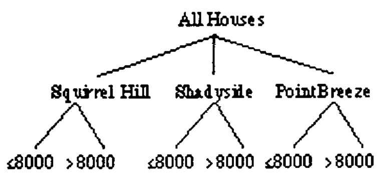
The second class of goals addresses *focus of attention,* which involves choosing the attributes of data one wishes to view in displays, to use for scope operations, and to use to control the level of detail. For example, a database of cars may consist of various attributes (car-model, year, company, cost, miles-per-gallon, repair-rating), but a user may wish to just focus on the average miles-per-gallon (for all car-models of a given company).

There is one specialized focus operation, the creation of *derived attributes*, which are attributes that do not occur in the original data and are defined by the user. For example, for our car data, we can create a derived-attribute called manufacturing-location (with values of American, European and Asian) by assigning a value to each car based on its manufacturer. The result is three groups, that can be displayed visually by coloring the cars on a display based on their manufacturer. Referred to as brushing [7] or painting [13], this technique control focus of attention using color. Another way to create derived attributes is to transform existing attributes by some filter [9], for example, create a binary attribute, fuel-efficient, from the car attribute miles-per-gallon by filtering the data by miles-per-gallon greater than 30.

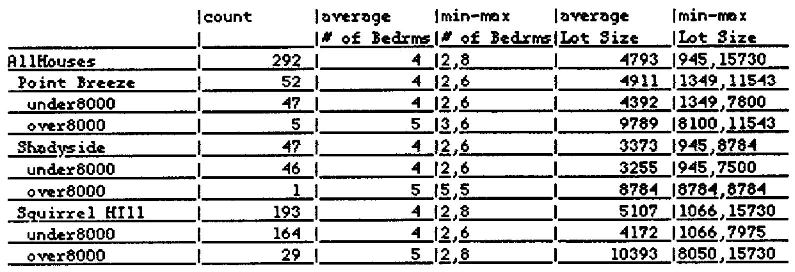
The third type of goal is choosing the *level of detail*, which involves changing the granularity of the data that the user wants to examine, either by *aggregation* (combining data into meaningful groups) or by *decomposition*: (breaking a larger data group into smaller groups). The process of aggregation is sometimes referred to as *composition*, when the process involves combining two data groups. Suppose we have house-sale data with the following attributes: number-houses-sold, total-sale-price, and date, where date represents a day of 1992. This involves 365 data points. The user may wish to change the level of detail by grouping dates into months and displaying the total sales per month. This reduces a display of 365 data points to one of 12 (aggregation). On any resultant aggregate, users might want to do data analysis operations, which involve examining *derived properties* of the data. These include defining *summary statistic*s, which are statistics that can be computed on the values of attributes (e.g., sum or average), such as the average total-sale-price for 1992.

Decomposition involves reducing a data group into smaller groups based on the same or different attributes of the included data objects. Figure 3a gives an example of how decomposition could occur for a real estate sales database with the following attributes: house-selling-price, neighborhood, number-of-bedrooms, lot-size. The neighborhood attribute has the values {Squirrel Hill, Shadyside, Point Breeze}. The grouping of "All Houses" is decomposed by neighborhood into three sub-groups of houses, one for each neighborhood. Each neighborhood group is partitioned further by lot size: lot sizes less than or equal to 8000 ft. and lot sizes greater than 8000 ft. Figure 3b shows the representation of the house data in the aggregate manipulator's outliner (described further in section 6).

(a)



(b)



	count	average # of Bedrms	min-max # of Bedrms	average Lot Size	min-max Lot Size
AllHouses	292	4	2,8	4793	945,15730
Point Breeze	52	4	2,6	4911	1349,11543
under8000	47	4	2,6	4392	1349,7800
over8000	5	5	3,6	9789	8100,11543
Shadyside	47	4	2,6	3373	945,8784
under8000	46	4	2,6	3255	945,7500
over8000	1	5	5,5	8784	8784,8784
Squirrel HIll	193	4	2,8	5107	1066,15730
under8000	164	4	2,6	4172	1066,7975
over8000	29	5	2,8	10393	8050,15730

Figure 3: Decomposition Representations: (a) Decomposition Tree (b) Aggregate Manipulator representation.

We could also perform the same aggregation using data visualizations. For the same real estate data (as in Figure 3b), Figure 4a shows the scatter plot depicting houses, where the number of bedrooms attribute is on the x-axis and price is on the y-axis. If we aggregate all the individual data points by neighborhood, we obtain the scatter plot in Figure 4b which consists of three data groups, each of which is plotted using the average bedroom and average price. Figure 4c shows the plot where each neighborhood aggregate has been decomposed into two lot-size partitions (under8000 and over8000).

As we can see from Figure 3a, the process of decomposition forms a *hierarchy*, which structures data into meaningful groupings specified by the user. We have identified four classes of decomposition:

- user-defined or pre-defined natural groupings: These can be defined interactively by the user or in advance as built-in knowledge. A possible pre-defined natural grouping is time, e.g., years -> quarters -> months. An example of a user-defined grouping is data on crimes, where each crime data object has a date attribute. An analyst may decide to break the year into holiday days and non-holiday days (as illustrated in Figure 5a).

- element frequency divisions: Divisions are computed by the system to have the same number of elements (*equi-frequency*). For example, if the user wants 20 divisions (partitioned by time) of the 1000 crimes committed in 1991, then there will be 50 crimes per time interval and each time interval can have a different size (Figure 5b).
- set interval divisions : Divisions are computed by the system to have the same interval size (*equi-interval.*). In the above example, if the user wants weekly divisions of the data, the system would divide the data into weeks: 1/1-1/7, 1/8-1/14, etc. (Figure 5c).
- system-provided statistical methods: The system can use clustering statistics or other methods to partition the data into groups.

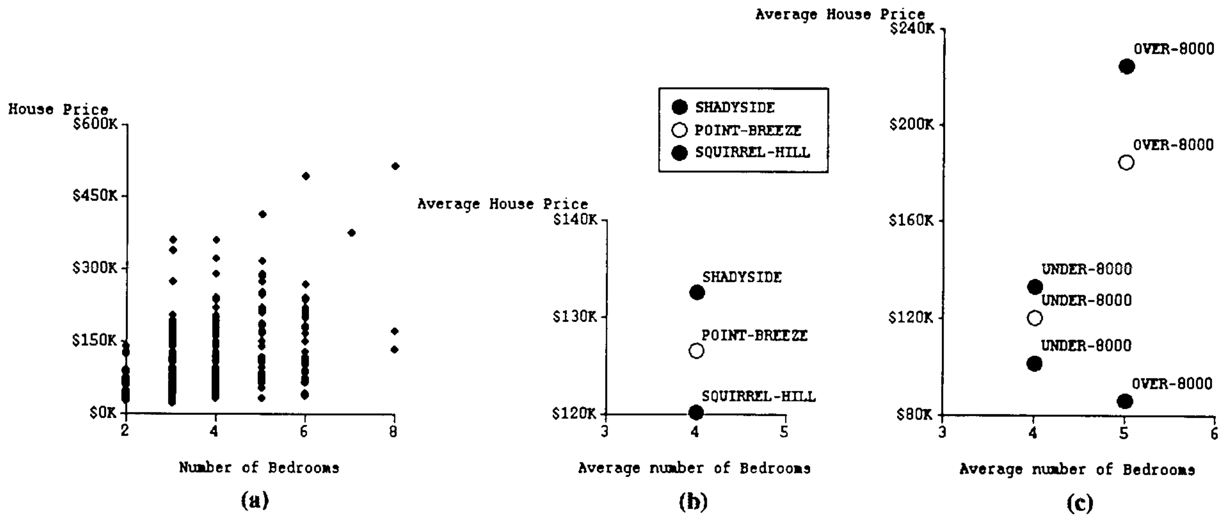


Figure 4: Example of Aggregation in Visualization. (a) The raw data. (b) Aggregating the raw data by neighborhood. (c) Decomposing the neighborhood aggregates by the lot-size attribute.

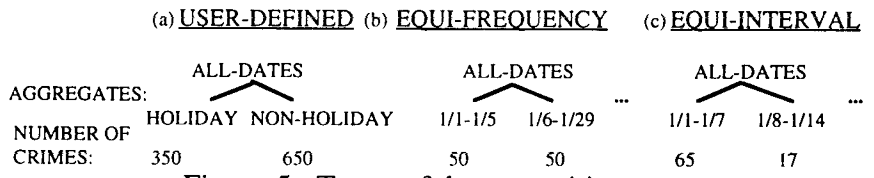


Figure 5: Types of decomposition.

4. SELECTING INTERFACE MECHANISMS

In the last section, we discussed a categorization of data manipulation operations - methods of selecting, grouping, and transforming data. In previous work [8], the authors have evaluated data exploration software according to this classification and discussed their advantages and disadvantages; the results are summarized in Table 1.

Dynamic Query or Queries (DQ) is an interactive technique which allows the user to manipulate sliders to control the amount of data displayed [1]. Each slider as shown in Figure 11, corresponds to a data attribute. The Aggregate Manipulator mechanism of Webs [15] was designed for level of detail operations, that of aggregation (grouping) and decomposition (partitioning groups), along with the display of group summary statistics. Iconographer [9] uses directed graphs that are programmed visually by the user to control scope, level of detail, and focus of attention operations. Powerplay [3] allows decomposition in pre-defined hierarchical structures. Excel allows level of detail operations through an "outline" mechanism in which the user can create groupings of data sets through a cumbersome process of individually linking cells in the database. SQL and other database query interfaces provide the most expressive means for specifying control of scope, but require learning complex programming languages to perform these and other operations.

	DATA MANIPULATION OPERATIONS	Dynamic Query	Aggregate Manipulator	Iconographer	Powerplay	Excel
SCOPE	- filter data using attribute(s)	xx	x	x		
	- select multiple disjunctive subsets		xx	x		x
FOCUS OF ATTENTION	- select attribute(s) for viewing operations	xx	x	xx	x	x
	- select attribute(s) for level of detail operations		xx	x	x	x
	- derive attribute(s) from existing attributes		xx	x		x
LEVEL OF DETAIL	- predefined aggregation & decomposition		xx	x	x	x
	- flexible aggregation & decomposition		xx	x		x

Table 1: The data exploration operations provided by different software or techniques. If the software (technique) allowed the operation in a simple, straightforward manner, we assigned the value "xx". If the operation involved non-intuitive operations, lots of steps, or steps bordering on programming, we assigned the value "x".

Table 1 suggests that the aggregate manipulator provides complete coverage of desired data manipulation operations. However, the AM does not perform the scope operations of filtering data or selecting attributes for viewing operations as well as DQ does. For filtering data, the AM requires creating user-defined partitions, which might have to be re-created for a slightly different choice of data (e.g., if users decide they want to view data for houses which sold for \$125,000-\$200,000 instead of \$100,000-\$200,000). Furthermore, if the user partitions the data set several times, it can be confusing what portion of the data (i.e., what range of values for the various attributes) is being displayed. In the case of DQ, determining these values is straightforward, since each attribute has its own slider or selector mechanism. However, DQ does not have the ability to disjunctively combine sets (e.g., display houses which sold for \$50,000-\$100,000 in the neighborhood Squirrel Hill and those that sold for \$50,000-\$150,000 in Shadyside) without creating methods that have multiple sets of dynamic queries linked to the same display. Thus, there is a need for a mechanism such as the AM to perform this and other operations (e.g. displaying summary statistics). Examples of the interactions between DQ and the AM will be given in Section 6.

In order to integrate AM and DQ in a prototype for exploration of large data sets IDES [8], we needed to extend them to function for many types of data. For the AM, this required exploring the types of operations that users would want to do with their data and then extending the AM so it could perform these types of decompositions and summary statistics based on a general data characterization rather than application-specific mechanisms. For DQ, we needed to be able to create a slider on demand and to have a method to select elements of nominal (non-numeric, unordered elements) data rather than just ranges of quantitative (numeric) data. For nominal data we use a scrolling list of elements (see Figure 11) and allow the user to select multiple elements of the list. Since the combination of these new versions of AM and DQ is not data specific, it is easily generalized to any new object-attribute data set.

5. SYSTEM DESIGN

The main functionality and dataflow of IDES is summarized in Figure 6. Decoupling the display area and the AM (which were linked in Webs) has the advantage that the user can explore and manipulate the data in the display area or the AM without affecting the other workspaces. This allows maintaining multiple perspectives on the data at different levels of detail. However, this has the drawback that aggregates that appear in the AM may or may not appear in the Display Area and vice versa. Whether or not this causes problems for users will be evaluated in user studies.

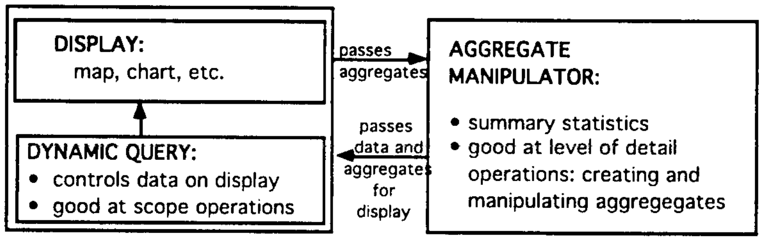

DISPLAY:
map, chart, etc.
DYNAMIC QUERY:
• controls data on display
• good at scope operations
passes aggregates
passes data and aggregates for display
AGGREGATE MANIPULATOR:
• summary statistics
• good at level of detail operations: creating and manipulating aggregegates

Figure 6: Data Flow for the AM & DQ.

The AM, DQ, and display comprise three of the four IDES workspaces (see Figure 11). The AM is a workspace for creating, decomposing, and directing the display of aggregates in other areas. The Display Area is both a work area for creating aggregates and a place to view the elements of the aggregates created by the AM. Dynamic query sliders are always connected to the current display. Changing the sliders changes the portion of the data that is displayed. Above the Display Area are menus that allow the user to create an aggregate, show an aggregate, clear the Display Area, and perform related functions. New aggregates can be created in the Display Area by selecting data points (represented by icons or graphical symbols) individually or as groups. Selected icons can be composed into a new aggregate, by the "Create Aggregate" command from the Display options. The user gives the aggregate a name, and the points representing the individual data objects are replaced by an icon representing an the aggregate data object, which we call an *aggregate gateway*. Users can move aggregates into the AM to be worked on further and optionally transferred back to the Display Area. If the aggregate is selected in the Display Area and a corresponding aggregate exists in the AM, both are highlighted. Lastly, users can decompose an existing aggregate into its components in the Display Area by double-clicking on the aggregate gateway object. The number of objects displayed reflects the bounds of the existing dynamic query sliders. This change in detail is not mirrored in the AM.

The last workspace consists of the data detail area (lower right corner), which is used for showing selected attributes of subsets of aggregates or individual data objects.

6 APPLICATIONS

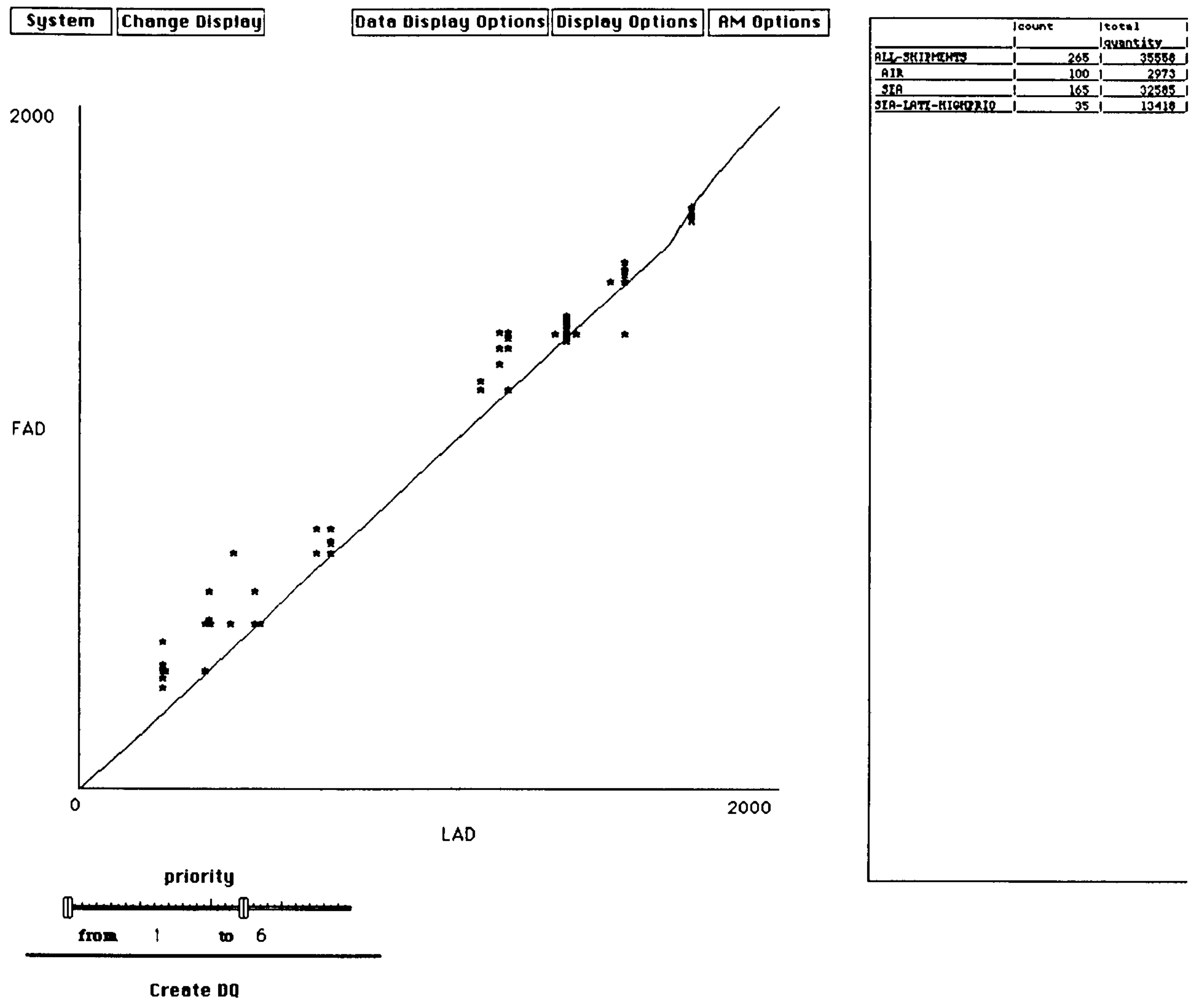
We have implemented this design for a shipping domain and a real estate domain. We will show advantages of using multiple visualizations, aggregation of data, and iterative processing to allow the user to find answers to data exploration questions. We will also show the usefulness of the AM for combining groupings of data, in particular joining sets disjunctively, and we will show the advantages of combining both DQ and the AM mechanisms.

6.1 Shipping

Consider a scenario in which the U.S. government is sending emergency supplies worldwide. Transportation planners use a database of shipments, where each record represents a description of a single order to be transported. An order has the following attributes: shipment ID, origin port, destination port, mode of transportation (air/sea), priority, quantity (shipment weight in tons), scheduled arrival date (FAD), due date (LAD), and lateness (FAD - LAD). The initial state of IDES is a scatterplot display (Figure 7 shows the scatterplot after some operations have been performed) and a single aggregate of all individual records, called ALL-SHIPMENTS, in the aggregate manipulator. All points above the diagonal line in Figure 7 are late. We want to know which destination seaports have large quantities of high priority late shipments, so that we can send additional personnel and equipment to assist with the situation.

We first want to display the number of items in ALL-SHIPMENTS and the total quantity (in tons) of shipments in the AM. We obtain these values by pressing on the top column header area of the AM table and selecting the summary statistic "count". This gives the total number of items, i.e. 265 (Figure 7). For the second column we select the summary statistic "total", then obtain a list of the possible attributes that can be used with "total" and from this list select "quantity". This procedure could just as easily have been done in reverse, choosing the attribute "quantity" and then selecting from a list of possible summary statistic options. Because the system has built-in knowledge data characteristics for each attribute (refer to Section 2.3), the choices are limited to those appropriate for the attribute. For example, since "total" is not a valid summary statistic for the attribute "due-date" (dates are coordinates rather than quantities and hence can't be totaled) it wouldn't be on the list if the "due-date" attribute were already chosen.

We then decompose the ALL-SHIPMENTS aggregate by mode, by pressing on the ALL-SHIPMENTS aggregate in the aggregate manipulator, which gives a pop-up menu of attributes, the decomposition options. We choose "mode" which creates two aggregates, AIR and SEA (Figure 7). These new aggregates are indented from the initial group ALL-SHIPMENTS in the *outliner* (the leftmost column) of the AM. We want to display only the sea shipping data, so we clear the display, select and move the SEA aggregate from the AM into the display We then double click on the SEA aggregate (i.e. gateway) in the display area to display the individual shipment data.



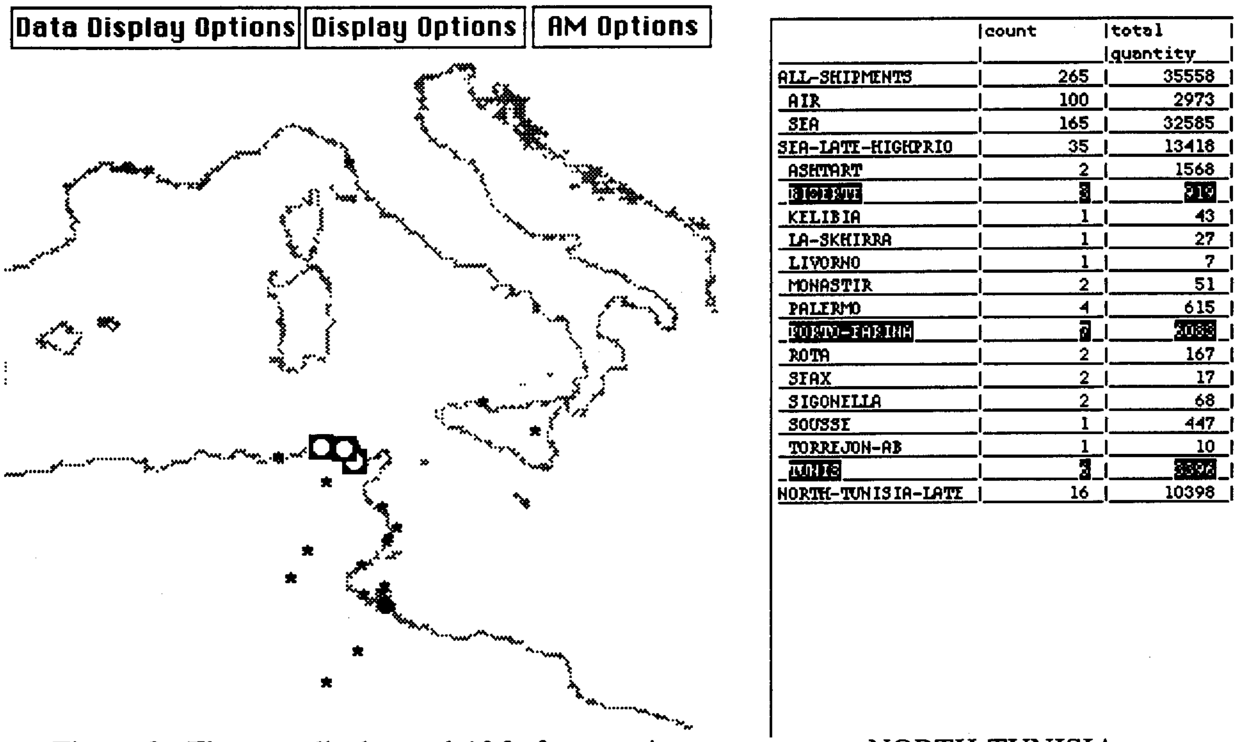
	count	total quantity
ALL-SHIPMENTS	265	35558
AIR	100	2973
SEA	165	32585
SEA-LATE-HIGHPRIO	35	13418

Figure 7: Scatterplot display after a series of operations. In the upper right hand corner is the outliner portion of the AM and in the upper left hand corner are the option buttons. Pressing on an option button gives a pop-up menu of available commands.

We now create a slider bar for priority by pressing on "Create DQ" in the DQ area. We choose the attribute "priority" from the list of options and use the resulting slider to limit the shipments displayed to those with priority 1 to 6. Figure 7 shows the results of these operations. Additional sliders for other attributes could also be created.

We decide to focus only on the first cluster of the two late shipment clusters (the lower left region of the display in Figure 7). We select all shipments above the diagonal line using bounding boxes . When we have the desired group, we choose the command Create Aggregate from the Display Options and name the aggregate SEA-LATE-HIGHPRIO. We then display this aggregate in the aggregate manipulator and decompose it by all values of the attribute destination port (Figure 8).

We change to the map display, then select the four aggregates in the AM that represent ports with the largest total quantity of late shipments. We display them on the map (their representation is a larger graphical object than the individual shipments) to see their location. We notice that three of them are in North Tunisia, and we select those three aggregates on the display, which highlights (selects) them in the AM. We then create an aggregate in the AM and name it NORTH-TUNISIA-LATE. Figure 8 shows the results of these operations. We can see the total quantity (in tons) of late shipments, which gives us an idea of the extent of the problem and what resources we might need to allocate to the situation.



	count	total quantity
ALL-SHIPMENTS	265	35558
AIR	100	2973
SEA	165	32585
SEA-LATE-HIGHPRIO	35	13418
ASHTART	2	1568
BIZERTE	[illegible]	219
KELIBIA	1	43
LA-SKHIRRA	1	27
LIVORNO	1	7
MONASTIR	2	51
PALERMO	4	615
PORTO-FARINA	7	3088
ROTA	2	167
SFAX	2	17
SIGONELLA	2	68
SOUSSE	1	447
TORREJON-AB	1	10
TUNIS	3	[illegible]
NORTH-TUNISIA-LATE	16	10398

Figure 8: The map display and AM after creating an aggregate NORTH-TUNISIA-LATE out of the ports in North Tunisia with the largest weight of late shipments (Bizerte, Porto-Farina, and Tunis).

This example has shown the value of multiple visualizations for aggregate creation and decomposition to support the user in the exploration process. Formulating SQL-like queries for this situation would be difficult because the user is unaware at the beginning where the problem destination ports are located and whether or not these ports are close enough geographically to be grouped together. For example, the set of shipments represented by the NORTH-TUNISIA-LATE aggregate would have been retrieved by a query like: "the set of shipments, whose mode is SEA, priority is between 1 and 6, lateness is positive, LAD (i.e. due-date) is 'among the lowest values', and which are delivered to ports where 'there are a lot of late shipments and some of which are close geographically'."

One of the features of the AM is its ability to create (compose) an aggregate by just selecting and combining multiple aggregates in the AM. For example, suppose we are interested in the combination of high-priority (<4) sea shipments and all air shipments (we assume air is high priority). We can decompose SEA into different priority levels (or choose only one priority level if we so desire) and then compose the high-priority sea aggregate (SEA-HIGHPRIO<4) with the aggregate representing all air shipments (AIR) in the AM (see Figure 9) to form a new aggregate

(HIGH-PRIO). Composed aggregates, such as HIGH-PRIO are placed in the outliner one step to the left of their leftmost child aggregate to avoid confusing them as parts of other aggregates.

Since AIR and SEA are distinct categories their combination contains no overlapping elements. Suppose we decide that North Tunisia is a high priority area and thus any shipment with a destination port there is considered high priority. We could aggregate all shipments in North Tunisia on the map, name it NORTH-TUNISIA-PRIO and move the aggregate to the AM. We then compose NORTH-TUNISIA-PRIO with HIGH-PRIO to form INTEREST-SHIPMENTS (Figure 9). There may be overlapping shipments between these two sets. The system is aware of any overlapping shipments and makes sure that the resulting aggregate references each appropriately. Notice that the count for the INTEREST-SHIPMENTS aggregate is less than the sum of HIGH-PRIO and NORTH-TUNISA-PRIO.

	count	total quantity
ALL-SHIPMENTS	265	35558
AIR	100	2973
SEA	165	32585
SEA-HIGHPRIO<4	64	16050
HIGH-PRIO	164	19023
NORTH-TUNISIA-PRIO	109	16067
INTEREST-SHIPMENTS	206	24708

Figure 9: Combining aggregates.

Note that the integration of AM and DQ supports the formation of disjunctive queries which are impossible with DQ alone. Forming complex queries in the aggregate manipulator is intuitive -- users do not even have to recognize that they are creating queries. Such a straightforward mechanism helps the users to concentrate on their data and goals, rather than on the formation of queries.

These examples have shown how the AM is a useful mechanism for discovering properties of information in a large data set and how the use of various visualizations can assist the user in this process. In the real estate example, we will elaborate on how AM and DQ synergistically operate in our system and how they are useful for different purposes.

6.2 Real Estate

Our real estate data consists of attributes from an actual real estate database of houses sold. There are 27 attributes (Figure 10) with varied data types: quantitative (e.g., selling price), nominal (e.g., neighborhood), and interval (e.g., date of sale). The attributes of the house data have three natural hierarchical relationships. City can be decomposed into neighborhoods or zip codes. Companies can be decomposed into offices (selling or listing), and offices can be decomposed by agents. There are many possible user-defined partition options as discussed in Section 3.

• address	• number of rooms	• lot size	• selling price	• selling office
• neighborhood	• number of bedrooms	• living room size	• asking price	• listing office
• city	• number of bathrooms	• dining room size	• date of sale	• listing agent
• zip code	• style of house	• kitchen size	• assessment	• company
• age of house	• fireplace	• master bedroom size	• tax	• days on market
• type of house	• garage			

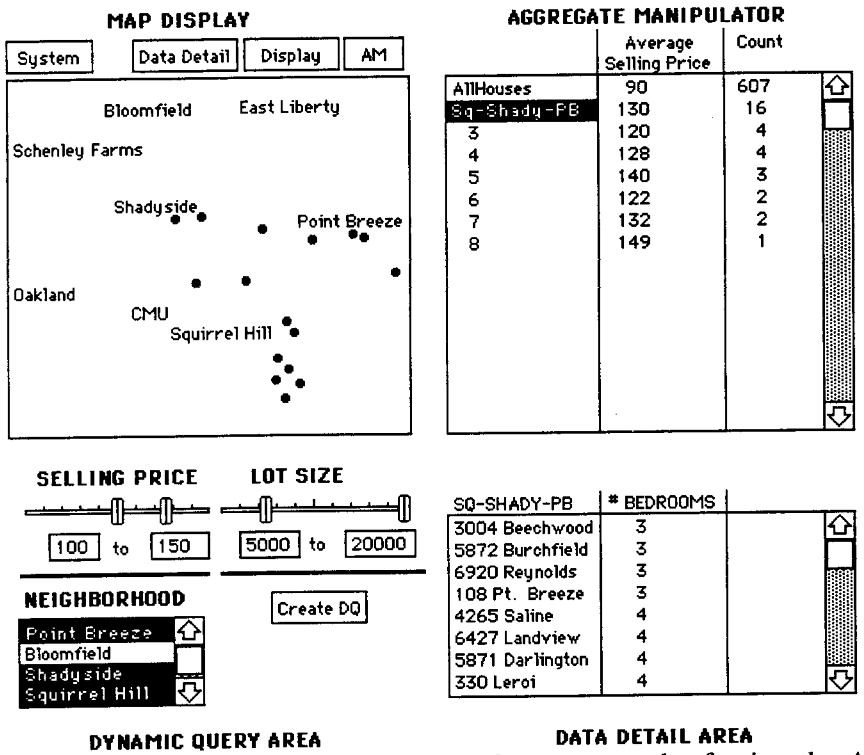
Figure 10: Attributes of the real estate data set.

In this section, we will discuss two scenarios. The first shows the weaknesses of using the AM alone. The second scenario shows how using DQ alone would require substantial work for the user. For both these cases, we show how using the combination of the AM and DQ is most effective.

Consider the following scenario. Jennifer is new to the Pittsburgh area and has the following goals for a house: (a) in the price range $100,000 to $150,000, (b) a lot size of at least 5000 sq. ft., because she wants a nice back yard, (c) at least 4 bedrooms, and (d) close to Carnegie Mellon University (i.e. in the neighborhoods of Shadyside, Squirrel or Pt. Breeze).

Jennifer would like to see houses which match these criteria and their map locations. The initial state of the system has the aggregate "AllHouses" in the AM and all houses displayed on the map (Figure 11). First, Jennifer creates dynamic query sliders for the attributes Selling Price, Lot Size and Neighborhood. After she selects the appropriate ranges or values for these queries, the map displays houses in Shadyside, Squirrel Hill and Point Breeze, with a price between 100-150K and lot size over 5000 sq. ft. Jennifer then selects all the data on the map and creates a new aggregate "Sq-Shady-PB".

Note that if Jennifer wanted a group with more or fewer houses, she could change the sliders until she had approximately the number she desired. This is an awkward procedure in the AM because it requires creating a new user-defined partition for each revision and then looking either at the summary statistics or displaying the new partition on the map.



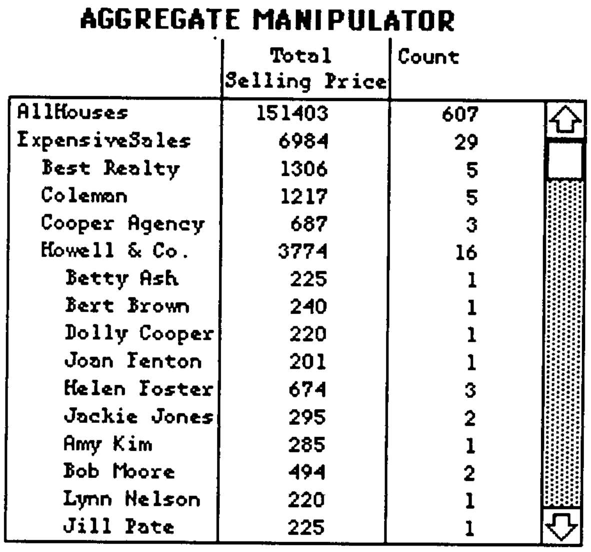
	Average Selling Price	Count
AllHouses	90	607
Sq-Shady-PB	130	16
3	120	4
4	128	4
5	140	3
6	122	2
7	132	2
8	149	1

SQ-SHADY-PB	# BEDROOMS
3004 Beechwood	3
5872 Burchfield	3
6920 Reynolds	3
108 Pt. Breeze	3
4265 Saline	4
6427 Landview	4
5871 Darlington	4
330 Leroi	4

Figure 11: A readable representation of the interface as a result of using the AM and DQ to partition house data. There are four workspaces with various options accessible via pop-up menus from the buttons in the upper left hand corner.

Figure 11 shows the results of the operations and summary statistics and data for the aggregate "Sq-Shady-PB". To partition "Sq-Shady-PB", she presses on it in the outliner of the AM, which gives a pop-up menu of attributes and selects the attribute "# Bedrooms." She chooses to partition the attribute "# Bedrooms" into individual values. All non-empty aggregate groups are displayed in the AM (in this case 3-8) along with summary statistics for any specified columns (in this case Average Selling Price and Count). To get the display in the Data Detail Area, she selects the aggregate "Sq-Shady-PB", uses an AM option to move it to the Detail Area and then chooses the attribute "# Bedrooms" from a column header pop-up menu in the same manner as that for the summary statistics.

The second scenario involves another situation in which we show the combination of the AM and DQ is superior to either method alone. John wants to sell his house and is looking for possible real estate agents. He believes his house will sell for around $250,000. He wants to know which company and then which sales agent has sold the most houses in the price range $200,000-$300,000 in his neighborhood in the last year. DQ alone is quite awkward to use because John would have to select all combinations of company and sales agents and then count the number of houses that appear on the map. However, DQ is easy to use for simple selection of the neighborhood and ranges for the price and date. From this he creates an aggregate ExpensiveSales (Figure 12). He then partitions this aggregate by the attribute company and selects two summary statistics: "Total" (for the attribute "Selling Price") and "Count". After finding that Howell & Co. sold the most houses, he decomposes this aggregate of houses by sales agent. The result of this decomposition is that John can quickly see that Helen Foster sold the most houses.



AGGREGATE MANIPULATOR

	Total Selling Price	Count
AllHouses	151403	607
ExpensiveSales	6984	29
Best Realty	1306	5
Coleman	1217	5
Cooper Agency	687	3
Howell & Co.	3774	16
Betty Ash	225	1
Bert Brown	240	1
Dolly Cooper	220	1
Joan Fenton	201	1
Helen Foster	674	3
Jackie Jones	295	2
Amy Kim	285	1
Bob Moore	494	2
Lynn Nelson	220	1
Jill Pate	225	1

Figure 12: The AM as a result of decomposing the aggregate ExpensiveSales by the attribute company and decomposing the partition Howell & Co. by the attribute sales agent.

These examples have shown how AM and DQ synergistically operate in our system and how they are useful for different purposes. The process of aggregation provides control over the level of detail of data shown. Slider mechanisms allow rapid filtering of the data. We will now discuss how these techniques can be used in our framework and how the components, knowledge, and directives modules need to be extended.

7. EXTENSIONS REQUIRED FOR LARGE DATA SETS

The IDES system and the examples above show how effective data manipulation tools can assist users in the data exploration process. For a computer-supported data graphic design system to handle these capabilities, the directives and knowledge must be extended and a data manipulation and analysis tool must be provided. In addition, we propose adding a data characterization generator, which allows the user to specify their own hierarchical decompositions. This section will discuss these extensions.

7.1 Presentation Directives

As discussed in section 2.2, presentation directives are the means by which users communicate their intent for the creation of graphics. In order to support interactive data exploration, presentation directives must be extended to allow users to communicate their data exploration intent as well.

Additional data directives must be generated when users communicate with the data manipulation and analysis component of the system to control their data. Examples of data directives include specifying the scope of the data to be displayed and specifying desired summary statistics for a group of data.

Task directives must be extended to allow users to specify when control over scope, focus of attention or level of detail is needed in a visualization. For example, to allow control over focus of attention a directive specifying the users need to actively filter visualized data based on values of a specified attribute could be generated. Such a request might result in the design of a visualization with built-in dynamic query sliders.

To allow control over level of detail, a directive might allow users to specify a particular data hierarchy to be used when data needs to be aggregated. For example, the user might specify when presenting data on people that if there are too many individuals to display, the system should display groups of people by income, age or other breakdown which accomplishes a directive (like minimizing overlap or achieving better distribution of attribute values). Such directives must be expressed in a vocabulary that is sufficiently general and atomic to enable their use in interface mechanisms which control the addition and removal of visible data as part of the process of controlling scope or expanding aggregates.

7.2 Data Characterization for Aggregates

Exploration of large data sets requires two features with which we have experimented in IDES: aggregates and hierarchies. Aggregates represent groupings of data and abstractions or summaries of their attributes. Hierarchies help define and organize aggregates. Each aggregate data object has an associated set of data elements, a data characterization that describes the set, and representative data values for each attribute based on global properties of the set. A system can use representative values and display them using aggregate gateways, which are the graphic representation of aggregate data objects that enable interactive expansion to greater detail. A system selects representative values of attributes and visualizes them as gateways appropriately is due to its knowledge of aggregate data characteristics.

An aggregate data object is an abstraction and grouping of similar individual data objects and its characteristics can be derived partly from those of the data elements from which it is composed.

For example, an aggregate of shipment data objects will express abstractions of shipment attributes: weight, port, and date. Weight is a quantitative attribute that can be summarized by a mean, mode, total, range, or a distribution frequency (i.e. number of elements with each value or the number of different values). In contrast, port is a nominal attribute, which can only be summarized with a distribution-count (i.e. the frequency of different values in the set, or possibly just the most frequent value). Dates are interval attributes, which can be represented with the same summary statistics as quantitative attributes except total.

Knowledge of data characteristics can also guide the choice of an attribute to partition data elements into aggregates, independent of the manner in which their attributes are summarized. For example, nominal attributes can be used to group data objects which have the same value (e.g. shipments destined for the same port). Quantitative and temporal attributes can be used to group data objects with common values, but they can also group data based on intervals (i.e. ranges). We discussed the use of equi-frequency and equi-interval methods of grouping data into aggregates in the section on IDES.

These observations enable a knowledge-based system to automatically select (or support user selection of) attributes and values to define groupings to create aggregates when the level of detail must be reduced. They also enable a system to select appropriate graphic techniques to visualize groups and the representative values which summarize their attributes. For example, they enable a system to select bar lengths for totals or medians of quantitative attributes but not for the *medians* of sets of dates - a median of a set of dates is still a date and must be expressed as a coordinate rather than an amount [16]. Similar knowledge enables interval bars to be selected to express quantitative or temporal ranges.

In addition to knowledge of aggregate characteristics derived from element data, a data exploration system requires knowledge of strategies for hierarchically structuring data aggregates. There are three types of knowledge of hierarchical relationships:

- Domain independent: knowledge of universal hierarchical relationships, e.g., that days are grouped into weeks or calendar months, and then into years.
- Domain dependent: knowledge specific to an application, e.g. that dates in a college's academic year can be aggregated by months with semesters (Fall, Spring and Summer semesters); dates in a business calendar might be aggregated by quarter (Jan-Mar, Apr-Jun, etc.).
- User-defined: knowledge that a user provides for a particular data set. For example, for population data, the user may want to use a new partition of the states which divide the states into coastal and non-coastal ones or alphabetically organized groups.

Just as a system can choose attributes to group data elements and summarize other attributes, it can use hierarchical structures like these to control the level of detail. Hierarchies provide intuitive methods for moving flexibly across levels of detail, enabling users to control aggregation and decomposition and keep track of the relationships among aggregates at different levels (i.e. parents and siblings in a tree).

A mechanism which must be added to support manipulation is a data characterization builder. Such a tool would enable users to interactively create and store new hierarchical structures for grouping data and controlling level of detail (e.g. to provide the ability to create the holiday/non-holiday breakdown of all-dates depicted in Figure 5a). It updates the data characterization so that the presentation system will be aware of the new hierarchy for aggregation purposes. The ability to quickly and easily create hierarchies allows users to group their data in partitions they want to explore repeatedly and in ways that are meaningful for specific instances of data.

7.3 Extensions to the Design Knowledge

Besides being used to display data, the IDES display area can be used to create and manipulate aggregates as we have shown. If the level of detail is too great (i.e. there are too many overlapping data points), the presentation system can choose to group and aggregate data by using the aggregate gateway graphical object. Once displayed, the user can expand an aggregate gateway to increase detail. If the aggregate gateway contains too much data to fit into a newly generated display, the presentation system must choose between three options: permit overly dense and overlapping data representations, change the scale and use scrolling controls to expose portions of the data, or reduce the visible data by creating new aggregates (i.e., those that do not overlap as much or that create an intermediate number of aggregates).

In addition, there are several possibilities for the graphic resulting from the expansion of the aggregate gateway:

- expand the aggregate data into the same display.
- create a separate display for the more detailed data, using the same graphic techniques as the parent display.
- create a separate display, using new techniques which shows only the detailed data.
- create a new display with new techniques which enable both the expanded and unexpanded aggregate data from the original display to coexist effectively.
- create a new display following some presentation directives that the user provides. For example, the user might want to emphasize a particular dimension of the aggregate differently from the previous display.

Design knowledge can also be used to select interface techniques in the data exploration tools. For example, instead of the outliner form of the aggregate manipulator as shown in Figure 11, a hierarchical graph (node-link diagram) might be used to show additional relations between aggregates or a Tree Map [21] to better show quantitative attributes of aggregates. Another example is the case where the data includes nominal attributes with a large number of values. Based on knowledge of the task and data characteristics, a system could select a form of DQ called the Alphaslider [2], which allow users to rapidly choose nominal values.

8. SUMMARY AND CONCLUSION

One important component in the design of user interfaces for exploring large data sets is that of data manipulation techniques. In this paper we explored these techniques with respect to a classification for data manipulation user goals, that of scope, focus of attention, and level of detail. We integrated into our interactive data exploration system, IDES, the technique of dynamic query, whose strength is scope operations, with the aggregate manipulator, whose strength is control of level of detail. We demonstrated how the combination of these tools can enable people to efficiently answer questions that are typical in data exploration.

Another important component for exploration large data sets is data visualization techniques. Automatic presentation systems are useful in that they relieve the user of the need to design and construct pictures. However, they must have features that allow users to construct graphics (as with SageBrush) and find previously constructed graphics to reuse or modify (as with SageBook).

In this paper, we have proposed a framework for knowledge-based, interactive data exploration. We have discussed the components that such a system should have in terms of our research on

IDES and SageTools. Implementing this framework would require addressing all the issues we have presented:

- Developing richer directives that refer to characteristics of data, tasks, design choices, and aesthetic preferences.
- Extending characterization vocabulary to describe both the hierarchical structure of data as well as the data exploration tasks that users perform.
- Developing approaches to visualizing information using aggregate graphical objects, to serve both as useful representations of data and as mechanisms to explore data, i.e. as a gateway to more detailed data.
- Developing and testing interface mechanisms that support data exploration, including filtering, dynamic query, painting, hierarchy expansion and contraction, scrolling, and mechanisms for structuring, partitioning, and aggregating data.
- Developing interface mechanisms for interacting with automatic presentation systems, such as SageBrush and SageBook, which enable users to communicate presentation directives in a natural way.
- Developing and understanding the processes of data manipulation, data analysis and data visualization and their relationships.

In addition, future research will involve performing user studies to ascertain how well people are able to use the various components of these systems. In regard to IDES, we plan to explore how our object-attribute paradigm needs to be expanded to relational data that does not fall in this paradigm. We also plan to incorporate other visualization techniques, such as brushing, which supports coordination of attributes across multiple displays.

REFERENCES

1. Ahlberg, C., Williamson, C. and Shneiderman, B. Dynamic Queries for Information Exploration: An Implementation and Evaluation, in Proceedings of the CHI '92 Conference (May 1992), ACM Press, pp. 619-626.

2. Ahlberg, C. and Shneiderman, B. The Alphaslider: A Compact and Rapid Selector, in Proceedings of the CHI '94 Conference (Boston, April 1994), ACM Press, pp. 365-371.

3. Barr, R. Using Graphs to Explore Databases and Create Reports, in SIGCHI Bulletin (July 1990), p. 24-27.

4. Bertin, .J. *Semiology of Graphics*, The University of Wisconsin Press 1983.

5. Brachman, R.J. et. al. Intelligent Support for Data Archaeology, in proceedings of Workshop on Intelligent Visualization Systems, IEEE Visualization'93 Conference (San Jose, October 1993) , p. 5-19.

6. Casner, S. A Task-Analytic Approach to the Automated Design of Graphic Presentations, in ACM Transactions on Graphics, 10, 2 (April 1991), 111-151.

7. Cleveland, W.S. and McGill, M.E. *Dynamic Graphics for Statistics,* Wadsforth Inc., Belmont, CA 1988.

8. Goldstein, J. and Roth, S.F. Using Aggregation and Dynamic Queries for Exploring Large Data Sets, in Proceedings of the CHI'94 Conference (Boston, April 1994), ACM Press, pp. 23-29.

9. Gray, P.D., Waite, K.W., and Draper, S.W. Do-It Yourself Iconic Displays, in *Human-Computer Interaction - INTERACT '90,* D. Diaper et al., Elsevier Science Publishers B.V., 1990, pp. 639-644.

10. Holsheimer, M. and Siebes, A. Data Mining: The Search for Knowledge in Databases, Report CS-R9406, ISSN 0169-118X, Amersterdam, The Netherlands 1991.

11. Hutchins, E.L., Hollan, J.D., and Norman, D.A. Direct Manpulation Interfaces, in *User Centered System Design*, D.A Norman and S.W. Draper eds., 1986, pp. 87-124.

12. Mackinlay, J.D. Automating the Design of Graphical Presentations of Relational Information, in ACM Transactions on Graphics, 5, 2 (April 1986), ACM Press, pp. 110-141.

13. McDonald, J.A. and Stuetzle, W. Painting multiple views of complex objects, in Proceedings of the. ECOOP/OOPSLA'90 European Conference on Object Oriented Programming (Oct. 21-25, 1990), ACM Press, pp. 245-257.

14. Marks, J.W. *Automating the Design of Network Diagrams*, Ph.D. Dissertation, Harvard University, 1991.

15. Maya Design. *The Webs Data Exploration Tool*, Maya Design Technical Report, Maya Design, Pittsburgh, PA 1993

16. Roth, S.F. and Mattis, J.A. Data Characterization for Intelligent Graphics Presentation, in Proceedings of the CHI'90 Conference (Seattle, April 1990), ACM Press, pp. 193-200.

17. Roth, S.F. and Mattis, J.A. Automating the Presentation of Information, in Proceedings of the Conference on Artificial Intelligence Applications (Miami Beach, Feb. 1991), IEEE Press, pp. 90-97.

18. Roth, S.F., Mattis, J.A., and Mesnard, X.A. Graphics and Natural Language as Components of Automatic Explanation, in *Architectures for Intelligent Interfaces: Elements and Prototypes*. Addison-Wesley, Reading, Mass., 1991.

19. Roth, S.F., Kolojejchick, J., Mattis, J. and Goldstein, J. Interactive Graphic Design Using Automatic Presentation Knowledge, in Proceedings of the CHI'94 Conference (Boston, April 1994), ACM Press, pp. 112-117.

20. Springmeyer, R.R., Blattner, M.M., and Max., N.L. Developing a Broader Basis for Scientific Data Analysis Interfaces. In *Proceedings of Visualization '92* (October 19-23, 1992, Boston, MA), pp. 235-242.

21. Turo, D. and Johnson, B. Improving the Viusalization of Hierarchies with Treemaps: Design Issues and Experimentation. In *Proceedings of Visualization '92* (October 19-23, 1992, Boston, MA), pp. 124-131.

☆U.S. GOVERNMENT PRINTING OFFICE: 1995-610-126-50108

DISTRIBUTION LIST

addresses	number of copies
ROME LABORATORY/C3C ATTN: NORTHRUP FOWLER III 525 BROOKS ROAD GRIFFISS AFB NY 13441-4505	15
ROBOTICS INSTITUTE ATTN: DR NORMAN SADEH CARNEGIE MELLON UNIVERSITY PITTSBURGH PA 15213-3890	5
RL/SUL TECHNICAL LIBRARY 26 ELECTRONIC PKY GRIFFISS AFB NY 13441-4514	1
ADMINISTRATOR DEFENSE TECHNICAL INFO CENTER DTIC-FDAC CAMERON STATION BUILDING 5 ALEXANDRIA VA 22304-6145	2
ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
RL/C3AB 525 BROOKS RD GRIFFISS AFB NY 13441-4505	1
NAVAL WARFARE ASSESSMENT CENTER GIDEP OPERATIONS CENTER/CODE QA-50 ATTN: E RICHARDS CORONA CA 91718-5000	1
HQ ACC/DRIY ATTN: MAJ. DIVINE LANGLEY AFB VA 23665-5575	1

ASC/ENEMS WRIGHT-PATTERSON AFB OH 45433-6503	1
WRIGHT LABORATORY/AAAI-4 WRIGHT-PATTERSON AFB OH 45433-6543	1
WRIGHT LABORATORY/AAAI-2 ATTN: MR FRANKLIN HUTSON WRIGHT-PATTERSON AFB OH 45433-6543	1
AFIT/LDEE 2950 P STREET WRIGHT-PATTERSON AFB OH 45433-6577	1
WRIGHT LABORATORY/MTEL WRIGHT-PATTERSON AFB OH 45433	1
AAMRL/HE WRIGHT-PATTERSON AFB OH 45433-6573	1
AIR FORCE HUMAN RESOURCES LAB TECHNICAL DOCUMENTS CENTER AFHRL/LRS-TDC WRIGHT-PATTERSON AFB OH 45433	1
AUL/LSE BLDG 1405 MAXWELL AFB AL 36112-5564	1
US ARMY STRATEGIC DEF CSSD-IM-PA PO BOX 1500 HUNTSVILLE AL 35807-3801	1

COMMANDING OFFICER NAVAL AVIONICS CENTER LIBRARY D/765 INDIANAPOLIS IN 46219-2189	1
COMMANDING OFFICER NCCOSC RDTE DIVISION CODE 0274B, TECH LIBRARY 53560 HULL STREET SAN DIEGO CA 92152-5001	1
CMDR NAVAL WEAPONS CENTER TECHNICAL LIBRARY/C3431 CHINA LAKE CA 93555-6001	1
SPACE & NAVAL WARFARE SYSTEMS COMM WASHINGTON DC 20363-5100	1
CDR, U.S. ARMY MISSILE COMMAND REDSTONE SCIENTIFIC INFO CENTER AMSMI-RD-CS-R/ILL DOCUMENTS REDSTONE ARSENAL AL 35898-5241	2
ADVISORY GROUP ON ELECTRON DEVICES ATTN: DOCUMENTS 2011 CRYSTAL DRIVE,SUITE 307 ARLINGTON VA 22202	2
REPORT COLLECTION, RESEARCH LIBRARY MS P364 LOS ALAMOS NATIONAL LABORATORY LOS ALAMOS NM 87545	1
AEDC LIBRARY TECH FILES/MS-100 ARNOLD AFB TN 37389	1
COMMANDER/USAISC ATTN: ASOP-DO-TL BLDG 61801 FT HUACHUCA AZ 85613-5000	1

AIR WEATHER SERVICE TECHNICAL LIB FL 4414 SCOTT AFB IL 62225-5458	1
AFIWC/MSO 102 HALL BLVD STE 315 SAN ANTONIO TX 78243-7016	1
SOFTWARE ENGINEERING INST (SEI) TECHNICAL LIBRARY 5000 FORBES AVE PITTSBURGH PA 15213	1
DIRECTOR NSA/CSS W157 9800 SAVAGE ROAD FORT MEADE MD 21055-6000	1
NSA ATTN: D. ALLEY DIV X911 9800 SAVAGE ROAD FT MEADE MD 20755-6000	1
DOD R31 9800 SAVAGE ROAD FT. MEADE MD 20755-6000	1
DIRNSA R509 9800 SAVAGE ROAD FT MEADE MD 20775	1
DOD COMPUTER CENTER C/TIC 9800 SAVAGE ROAD FORT GEORGE G. MEADE MD 20755-6000	1
ESC/IC 50 GRIFFISS STREET HANSCOM AFB MA 01731-1619	1

ESC/AV 1
20 SCHILLING CIRCLE
HANSCOM AFB MA 01731-2816

DCMAO/GWE 1
ATTN: JOHN CHENG
US COURTHOUSE/SUITE B-34
401 N MARKET
WICHITA KS 67202-2095

FL 2807/RESEARCH LIBRARY 1
OL AA/SULL
HANSCOM AFB MA 01731-5000

TECHNICAL REPORTS CENTER 1
MAIL DROP D130
BURLINGTON ROAD
BEDFORD MA 01731

DEFENSE TECHNOLOGY SEC ADMIN (DTSA) 1
ATTN: STTD/PATRICK SULLIVAN
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

SOFTWARE ENGR'G INST TECH LIBRARY 1
ATTN: MR DENNIS SMITH
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213-3890

SOFTWARE OPTIONS, INC. 1
ATTN: MR TOM CHEATHAM
22 HILLIARD STREET
CAMBRIDGE MA 02138

USC-ISI 1
ATTN: DR ROBERT M. BALZER
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292-6695

KESTREL INSTITUTE 1
ATTN: DR CORDELL GREEN
1801 PAGE MILL ROAD
PALO ALTO CA 94304

ROCHESTER INSTITUTE OF TECHNOLOGY ATTN: PROF J. A. LASKY 1 LOMB MEMORIAL DRIVE P.O. BOX 9887 ROCHESTER NY 14613-5700	1
WESTINGHOUSE ELECTRONICS CORP ATTN: MR DENNIS BIELAK ELECTRONICS SYSTEMS GROUP P.O. BOX 746, MAIL STOP 432 BALTIMORE MD 21203	1
AFIT/ENG ATTN: PAUL BAILOR. MAJOR, USAF WPAFB OH 45433-6583	1
THE MITRE CORPORATION ATTN: MR EDWARD H. BENSLEY BURLINGTON RD/MAIL STOP A350 BEDFORD MA 01730	1
UNIV OF ILLINOIS, URBANA-CHAMPAIGN ATTN: SANJAY BHANSALI DEPT OF COMPUTER SCIENCES 1304 WEST SPRINGFIELD URBANA IL 61801	1
ANDERSEN CONSULTING ATTN: MR MICHAEL E. DEBELLIS 100 SOUTH WACKER DRIVE CHICAGO IL 60606	1
UNIV OF ILLINOIS, URBANA-CHAMPAIGN ATTN: DR MEHDI HARANDI DEPT OF COMPUTER SCIENCES 1304 W. SPRINGFIELD/240 DIGITAL LAB URBANA IL 61801	1
HONEYWELL, INC. ATTN: MR BERT HARRIS FEDERAL SYSTEMS 7900 WESTPARK DRIVE MCLEAN VA 22102	1
SOFTWARE ENGINEERING INSTITUTE ATTN: MR WILLIAM E. HEFLEY CARNEGIE-MELLON UNIVERSITY SEI 2218 PITTSBURGH PA 15213-38990	1

UNIVERSITY OF SOUTHERN CALIFORNIA 1
ATTN: DR W. LEWIS JOHNSON
INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY/SUITE 1001
MARINA DEL REY CA 90292-6695

COLUMBIA UNIV/DEPT COMPUTER SCIENCE 1
ATTN: DR GAIL E. KAISER
450 COMPUTER SCIENCE BLDG
500 WEST 120TH STREET
NEW YORK NY 10027

SOFTWARE ENGINEERING INSTITUTE 1
ATTN: KYO CHUL KANG
CARNEGIE-MELLON UNIVERSITY
PITTSBURGH PA 15213-3890

SOFTWARE PRODUCTIVITY CONSORTIUM 1
ATTN: MR ROBERT LAI
2214 ROCK HILL ROAD
HERNDON VA 22070

AFIT/ENG 1
ATTN: DR GARY B. LAMONT
SCHOOL OF ENGINEERING
DEPT ELECTRICAL & COMPUTER ENGRG
WPAFB OH 45433-6583

NSA/OFC OF RESEARCH 1
ATTN: MS MARY ANNE OVERMAN
9800 SAVAGE ROAD
FT GEORGE G. MEADE MD 20755-6000

THE MITRE CORPORATION 1
ATTN: MR HOWARD REUBENSTEIN
BURLINGTON ROAD
BEDFORD MA 01730

ANDERSEN CONSULTING 1
ATTN: DR WILLIAM C. SASSO
CENTER FOR STRATEGIC TECH RSCH
100 SOUTH WACKER DRIVE
CHICAGO IL 60606

AT&T BELL LABORATORIES 1
ATTN: MR PETER G. SELFRIDGE
ROOM 3C-441
600 MOUNTAIN AVE
MURRAY HILL NJ 07974

VITRO CORPORATION ATTN: MR ROBERT A. SMALL 14000 GEORGIA AVENUE SILVER SPRING MD 20906-2972	1
ODYSSEY RESEARCH ASSOCIATES, INC. ATTN: MS MAUREEN STILLMAN 301A HARRIS B. DATES DRIVE ITHACA NY 14850-1312	1
WRDC/AAAF-3 ATTN: JAMES P. WEBER, CAPT, USAF AERONAUTICAL SYSTEMS CENTER WPAFB OH 45433-6543	1
TEXAS INSTRUMENTS INCORPORATED ATTN: DR DAVID L. WELLS P.O. BOX 655474, MS 238 DALLAS TX 75265	1
BOEING COMPUTER SERVICES ATTN: DR PHIL NEWCOMB MS 7L-64 P.O. BOX 24346 SEATTLE WA 98124-0346	1
LOCKHEED SOFTWARE TEHNOLOGY CENTER ATTN: MR HENSON GRAVES ORG. 96-10 BLDG 254E 3251 HANOVER STREET PALO ALTO CA 94304-1191	1
REASONING SYSTEMS ATTN: DR GORDON KOTIK 3260 HILLVIEW AVENUE PALO ALTO CA 94304	1
TEXAS A & M UNIVERSITY ATTN: DR PAULA MAYER KNOWLEDGE BASED SYSTEMS LABORATORY DEPT OF INDUSTRIAL ENGINEERING COLLEGE STATION TX 77843	1
KESTREL DEVELOPMENT CORPORATION ATTN: DR RICHARD JULLIG 3260 HILLVIEW AVENUE PALO ALTO CA 94304	1

AEROSPACE CORPORATION 1
ATTN: DR. KIRSTIE BELLMAN
M1/102 COMPUTER SCI & TECH SUBDIV
P. O. BOX 92957
LOS ANGELES CA 90009-2957

LOCKHEED O/96-10 B/254E 1
ATTN: JACKY COMBS
3251 HANOVER STREET
PALO ALTO CA 94304-1191

NASA/JOHNSON SPACE CENTER 1
ATTN: CHRIS CULBERT
MAIL CODE PT4
HOUSTON TX 77058

SAIC 1
ATTN: LANCE MILLER
MS T1-6-3
PO BOX 1303 (OR 1710 GOODRIDGE DR)
MCLEAN VA 22102

STERLING IMD INC. 1
KSC OPERATIONS
ATTN: MARK MAGINN
BEECHES TECHNICAL CAMPUS/RT 26 N.
ROME NY 13440

NAVAL POSTGRADUATE SCHOOL 1
ATTN: BALA RAMESH
CODE AS/RS
ADMINISTRATIVE SCIENCES DEPT
MONTEREY CA 93943

KESTREL INSTITUTE 1
ATTN: MARIA PRYCE
3260 HILLVIEW AVENUE
PALO ALTO CA 94304

HUGHES AIRCRAFT COMPANY 1
ATTN: GERRY BARKSDALE
P. O. BOX 3310
BLDG 618 MS E215
FULLERTON CA 92634

FORWISS UNIVERSITY OF ERLANGEN 1
ATTN: ERNST LUTZ
AM WEICHSELGARTEN 7
8520 ERLANGEN, GERMANY

THE MITRE CORPORATION 1
ATTN: HOWARD REUBENSTEIN
BURLINGTON ROAD, K302
BEDFORD MA 01730

SCHLUMBERGER LABORATORY FOR 1
COMPUTER SCIENCE
ATTN: DR. GUILLFRMO ARANGO
8311 NORTH FM620
AUSTIN, TX 78720

PARAMAX SYSTEMS CORPORATION 1
ATTN: DON YU
8201 GREENSBORO DRIVE, SUITE 1000
MCLEAN VA 22101

MOTOROLA, INC. 1
ATTN: MR. ARNOLD PITTLER
3701 ALGONQUIN ROAD, SUTE 601
ROLLING MEADOWS, IL 60008

DECISION SYSTEMS DEPARTMENT 1
ATTN: PROF WALT SCACCHI
SCHOOL OF BUSINESS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CA 90089-1421

SOUTHWEST RESEARCH INSTITUTE 1
ATTN: BRUCE REYNOLDS
6220 CULEBRA ROAD
SAN ANTONIO, TX 78228-0510

NATIONAL INSTITUTE OF STANDARDS 1
AND TECHNOLOGY
ATTN: CHRIS DABROWSKI
ROOM A266, BLDG 225
GAITHSBURG MD 20899

EXPERT SYSTEMS LABORATORY 1
ATTN: STEVEN H. SCHWARTZ
NYNEX SCIENCE & TECHNOLOGY
500 WESTCHESTER AVENUE
WHITE PLANS NY 20604

NAVAL TRAINING SYSTEMS CENTER 1
ATTN: ROBERT BREAUX/CODE 252
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3224

CENTER FOR EXCELLENCE IN COMPUTER- 1
AIDED SYSTEMS ENGINEERING
ATTN: PERRY ALEXANDER
2291 IRVING HILL ROAD
LAWRENCE KS 66049

SOFTWARE TECHNOLOGY SUPPORT CENTER 1
ATTN: MAJ ALAN K. MILLER
OGDEN ALC/TISE
BLDG 100, BAY G
HILL AFB, UTAH 84056

MS. KAREN ALGUIRE 1
RL/C3CA
525 BROOKS RD
GRIFFISS AFB NY 13441-4505

JAMES ALLEN 1
COMPUTER SCIENCE DEPT/BLDG RM 732
UNIV OF ROCHESTER
WILSON BLVD
ROCHESTER NY 14627

MS TIFFANY WALKER 1
DIGITAL SYSTEMS RSCH INC
4301 NORTH FAIRFAX DRIVE
SUITE 725
ARLINGTON VA 22203

YIGAL ARENS 1
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

MR. RAY BAREISS 1
THE INST. FOR LEARNING SCIENCES
NORTHWESTERN UNIV
1890 MAPLE AVE
EVANSTON IL 60201

MR. JEFF BERLINER 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

MARIE A. BIENKOWSKI 1
SRI INTERNATIONAL
333 RAVENSWOOD AVE/EK 337
MENLO PRK CA 94025

DR MARK S. BODDY 1
HONEYWELL SYSTEMS & RSCH CENTER
3660 TECHNOLOGY DRIVE
MINNEAPOLIS MN 55418

PIERO P. BONISSONE 1
GE CORPORATE RESEARCH & DEVELOPMENT
BLDG K1-RM 5C-32A
P. O. BOX 8
SCHENECTADY NY 12301

MR. DAVID BROWN 1
MITRE
EAGLE CENTER 3, SUITE 8
O'FALLON IL 62269

MR. MARK BURSTEIN 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

MR. GREGG COLLINS 1
INST FOR LEARNING SCIENCES
1890 MAPLE AVE
EVANSTON IL 60201

MR. RANDALL J. CALISTRI-YEH 1
ORA CORPORATION
301 DATES DRIVE
ITHACA NY 14850-1313

DR STEPHEN E. CROSS 1
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213

MS. JUDITH DALY 1
ARPA/ASTO
3701 N. FAIRFAX DR., 7TH FLOOR
ARLINGTON VA 22209-1714

THOMAS CHEATHAM 1
HARVARD UNIVERSITY
DIV OF APPLIED SCIENCE
AIKEN, RM 104
CAMBRIDGE MA 02138

MS. LAURA DAVIS CODE 5510 NAVY CTR FOR APPLIED RES IN AI NAVAL RESEARCH LABORATORY WASH DC 20375-5337	1
MS. GLADYS CHOW COMPUTER SCIENCE DEPT. UNIV OF CALIFORNIA LOS ANGELES CA 90024	1
THOMAS L. DEAN BROWN UNIVERSITY DEPT OF COMPUTER SCIENCE P.O. BOX 1910 PROVIDENCE RI 02912	1
WESLEY CHU COMPUTER SCIENCE DEPT UNIV OF CALIFORNIA LOS ANGELES CA 90024	1
MR. ROBERTO DESIMONE SRI INTERNATIONAL (EK335) 333 RAVENSWOOD AVE MENLO PRK CA 94025	1
PAUL R. COHEN UNIV OF MASSACHUSETTS COINS DEPT LEDERLE GRC AMHERST MA 01003	1
MS. MARIE DEJARDINS SRI INTERNATIONAL 333 RAVENSWOOD AVENUE MENLO PRK CA 94025	1
JON DOYLE LABORATORY FOR COMPUTER SCIENCE MASS INSTITUTE OF TECHNOLOGY 545 TECHNOLOGY SQUARE CAMBRIDGE MA 02139	1
DR. BRIAN DRABBLE AI APPLICATIONS INSTITUTE UNIV OF EDINBURGH/80 S. BRIDGE EDINBURGH EH1 LHN UNITED KINGDOM	1

MR. SCOTT FOUSE 1
TSX CORPORATION
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

MR. STU DRAPER 1
MITRE
EAGLE CENTER 3, SUITE 8
O'FALLON IL 62269

MARK FOX 1
DEPT O INDUSTRIAL ENGRG
UNIV OF TORONTO
4 TADDLE CREAK ROAD
TORONTO, ONTARIO, CANADA

MR. GARY EDWARDS 1
4353 PARK TERRACE DRIVE
WESTLAKE VILLACA 91361

MS. MARTHA FARINACCI 1
MITRE
7525 COLSHIRE DRIVE
MCLEAN VA 22101

MR. RUSS FREW 1
GENERAL ELECTRIC
MOORESTOWN CORPORATE CENTER
BLDG ATK 145-2
MOORESTOWN NJ 08057

MICHAEL FEHLING 1
STANFORD UNIVERSITY
ENGINEERING ECO SYSTEMS
STANFORD CA 94305

MR. RICH FRITZSON 1
CENTER OR ADVANCED INFO TECHNOLOGY
UNISYS
P.O. BOX 517
PAOLI PA 19301

MR KRISTIAN J. HAMMOND 1
UNIV OF CHICAGO
COMPUTER SCIENCE DEPT/RY155
1100 E. 58TH STREET
CHICAGO IL 60637

MR. ROBERT FROST 1
MITRE CORP
WASHINGTON C3 CENTER, MS 644
7525 COLSHIER ROAD
MCLEAN VA 22101-3481

RICK HAYES-ROTH 1
CIMFLEX-TEKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303

RANDY GARRETT 1
INST FOR DEFENSE ANALYSES (IDA)
1801 N. BEAUREGARD STREET
ALEXANDRA VA 22311-1772

MR. JIM HENDLER 1
UNIV OF MARYLAND
DEPT OF COMPUTER SCIENCE
COLLEGE PARK MD 20742

MS. YOLANDA GIL 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

MR.MAX HERION 1
ROCKWELL INTERNATIONAL SCIENCE CTR
444 HIGH STREET
PALO ALTO CA 94301

MR. STEVE GOYA 1
DISA/JIEO/GS11
CODE TBD
11440 ISAAC NEWTON SQ
RESTON VA 22090

MR. MORTON A. HIRSCHBERG, DIRECTOR 1
US ARMY RESEARCH LABORATORY
ATTN: AMSRL-CI-CB
ABERDEEN PROVING GROUND MD
21005-5066

MR. MARK A. HOFFMAN 1
ISX CORPORATION
1165 NORTHCHASE PARKWAY
MARIETTA GA 30067

MR. RON LARSEN 1
NAVAL CMD, CONTROL & OCEAN SUR CTR
RESEARCH, DEVELOP, TEST & EVAL DIV
CODE 444
SAN DIEGO CA 92152-5000

DR. JAMES JUST 1
MITRE
DEPT. W032-M/S Z360
7525 COLSHIER RD
MCLEAN VA 22101

MR. CRAIG KNOBLOCK 1
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

MR. RICHARD LOWE (AP-10) 1
SRA CORPORATION
2000 15TH STREET NORTH
ARLINGTON VA 22201

MR. TED C. KRAL 1
BBN SYSTEMS & TECHNOLOGIES
4015 HANCOCK STREET, SUITEE 101
SAN DIEGO CA 92110

MR. JOHN LOWRENCE 1
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025

DR. ALAN MEYROWITZ 1
NAVAL RESEARCH LABORATORY/CODE 5510
4555 OVERLOOK AVE
WASH DC 20375

ALICE MULVEHILL 1
MITRE CORPORATION
BURLINGTON RD
M/S K-302
BEDFORD MA 01730

ROBERT MACGREGOR 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292

WILLIAM S. MARK, MGR AI CENTER 1
LOCKHEED MISSILES & SPACE CENTER
1801 PAGE MILL RD
PALO ALTO CA 94304-1211

RICHARD MARTIN 1
SOTWARE ENGINEERING INSTITUTE
CARNEGIE MELLON UNIV
PITTSBURGH PA 16213

DREW MCDERMOTT 1
YALE COMPUTER SCIENCE DEPT
P.O. BOX 2158, YALE STATION
51 PROPSPECT STREET
MEW HAVEN CT 06520

MS. CECILE PARIS 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DOUGLAS SMITH 1
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304

DR. AUSTIN TATE 1
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH
80 SOUTH BRIDGE
EDINBURGH EH1 1HN - SCOTLAND

EDWARD THOMPSON 1
ARPA/SISTO
3701 N. FAIRFAX DR., 7TH FL
ARLINGTON VA 22209-1714

MR. STEPHEN F. SMITH 1
ROBOTICS INSTITUTE/CMU
SCHENLEY PRK
PITTSBURGH PA 15213

LTCOL RAYMOND STACHA 1
DEPUTY SCIENTIFIC & TECHNICAL
 ADVISOR
HQ USCINCPAC/STA
CAMP H. M. SMITH HI 96861

DR. ABRAHAM WAKSMAN AFOSR/NM 110 DUNCAN AVE., SUITE B115 BOLLING AFB DC 20331-0001	1
JONATHAN P.STILLMAN GENERAL ELECTRIC CRD 1 RIVER RD, RM K1-5C31A P. O. BOX 8 SCHENECTADY NY 12345	1
MR. EDWARD C. T. WALKER BBN SYSTEMS & TECHNOLOGIES 10 MOULTON STREET CAMBRIDGE MA 02138	1
MR. BILL SWARTOUT USC/ISI 4676 ADMIRALTY WAY MARINA DEL RAY CA 90292	1
GIO WIEDERHOLD STANFORD UNIVERSITY DEPT OF COMPUTER SCIENCE 438 MARGARET JACKS HALL STANFORD CA 94305-2140	1
KATIA SYCARA/THE ROBOTICS INST SCHOOL OF COMPUTER SCIENCE CARNEGIE MELLON UNIV DOHERTY HALL RM 3325 PITTSBURGH PA 15213	1
MR. DAVID E. WILKINS SRI INTERNATIONAL ARTIFICIAL INTELLIGENCE CENTER 333 RAVENSWOOD AVE MENLO PARK CA 94025	1
DR. PATRICK WINSTON MASS INSTITUTE OF TECHNOLOGY RM NE43-817 545 TECHNOLOGY SQUARE CAMBRIDGE MA 02139	1
HUA YANG COMPUTER SCIENCE DEPT UNIV OF CALIFORNIA LOS ANGELES CA 90024	1

LTCOL DAVE NEYLAND ARPA/ISTO 3701 N. FAIRFAX DRIVE, 7TH FLOOR ARLINGTON VA 22209-1714	1
MR. RICK SCHANTZ BBN SYSTEMS & TECHNOLOGIES 10 MOULTON STREET CAMBRIDGE MA 02138	1
LTC FRED M. RAWCLIFFE USTRANSCOM/TCJ5-SC BLDG 1900 SCOTT AFB IL 62225-7001	1
JOHN P. SCHILL NAVAL COMMAND, CONTROL & OCEAN SURVEILLANCE CENTER/CODE 423 EVALUATION DIVISION SAN DIEGO CA 92152-5000	1
MR. DONALD F. ROBERTS RL/C3CA 525 BROOKS ROAD GRIFFISS AFB NY 13441-4505	1
ALLEN SEARS MITRE 7525 COLESHIRE DRIVE, STOP Z289 MCLEAN VA 22101	1
STEVE ROTH CENTER FOR INTEGRATED MANUFACTURING THE ROBOTICS INSTITUTE CARNEGIE MELLON UNIV PITTSBURGH PA 15213-3890	1
JEFF ROTHENBERG SENIOR COMPUTER SCIENTIST THE RAND CORPORATION 1700 MAIN STREET SANTA MONICA CA 90407-2138	1
YOAV SHOHAM STANFORD UNIVERSITY COMPUTER SCIENCE DEPT STANFORD CA 94305	1

MR. DAVID B. SKALAK UNIV OF MASSACHUSETTS DEPT OF COMPUTER SCIENCE RM 243, LGRC AMHERST MA 01003	1
MR. MIKE ROUSE AFSC 7800 HAMPTON RD NORFOLK VA 23511-6097	1
MR. DAVID E. SMITH ROCKWELL INTERNATIONAL 444 HIGH STREET PALO ALTO CA 94301	1
JEFF ROTHENBERG SENIOR COMPUTER SCIENTIST THE RAND CORPORATION 1700 MIN STREET SANTA MONICA CA 90407-2138	1
DR LARRY BIRNBAUM NORTHWESTERN UNIVERSITY ILS 1890 MAPLE AVE EVANSTON IL 60201	1
MR RANDALL J. CALISTRI-YEH ORA 301 DATES DR ITHACA NY 14850-1313	1
MR WESLEY CHU COMPUTER SCIENCE DEPT UNIVERSITY OF CALIFORNIA LOS ANGELES CA 9002	1
MR PAUL R COHEN UNIVERSITY OF MASSACHUSETTS COINS DEPT, LEDERLE GRC AMHERST MA 01003	1
MR DON EDDINGTON NAVAL COMMAND, CONTROL & OCEAN SURV CENTER RDT&E DIVISION, CODE 404 SAN DIEGO CA 92152-5000	1

MR. LEE ERMAN 1
CIMFLEX TECKNOWLEDGE
1810 EMBARCARDERO RD
PALO ALTO CA 94303

MR DICK ESTRADA 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON ST
CAMBRIDGE MA 02138

MR HARRY FORSDICK 1
BBN SYSTEMS AND TECHNOLOGIES
10 MOULTON ST
CAMBRIDGE MA 02138

MR MATTHEW L. GINSBERG 1
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE OR 97403

MR IRA GOLDSTEIN 1
OPEN SW FOUNDATION RESEARCH INST
ONE CAMBRIDGE CENTER
CAMBRIDGE MA 02142

MR MOISES GOLDSZMIDT 1
INFORMATION AND DECISION SCIENCES
ROCKWELL INTL SCIENCE CENTER
444 HIGH ST, SUITE 400
PALO ALTO CA 94301

MR JEFF GROSSMAN, CO 1
NCCOSC RDTE DIV 44
5370 SILVERGATE AVE, ROOM 1405
SAN DIEGO CA 92152-5146

JAN GUNTHER 1
ASCENT TECHNOLOGY, INC.
64 SIDNEY ST, SUITE 380
CAMBRIDGE MA 02139

DR LYNETTE HIRSCHMAN 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

MS ADELE E. HOWE 1
COMPUTER SCIENCE DEPT
COLORADO STATE UNIVERSITY
FORT COLLINS CO 80523

DR LESLIE PACK KAELBLING 1
COMPUTER SCIENCE DEPT
BROWN UNIVERSITY
PROVIDENCE RI 02912

SUBBARAO KAMBHAMPATI 1
DEPT OF COMPUTER SCIENCE
ARIZONA STATE UNIVERSITY
TEMPE AZ 85287-5406

MR THOMAS E. KAZMIERCZAK 1
SRA CORPORATION
331 SALEM PLACE, SUITE 200
FAIRVIEW HEIGHTS IL 62208

PRADEEP K. KHOSLA 1
ARPA/SSTO
3701 N. FAIRFAX DR
ARLINGTON VA 22203

MR CRAIG KNOBLOCK 1
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DR CARLA LUDLOW 1
ROME LABORATORY/C3CA
525 BROOKS RD
GRIFFISS AFB NY 13441-4505

DR MARK T. MAYBURY 1
ASSOCIATE DIRECTOR OF AI CENTER
ADVANCED INFO SYSTEMS TECH G041
MITRE CORP, BURLINGTON RD, MS K-329
BEDFORD MA 01730

MR DONALD P. MCKAY 1
PARAMAX/UNISYS
P O BOX 517
PAOLI PA 19301

DR KAREN MYERS 1
AI CENTER
SRI INTERNTIONAL
333 RAVENSWOOD
MENLO PARK CA 94025

DR MARTHA E POLLACK 1
DEPT OF COMPUTER SCIENCE
UNIVERSITY OF PITTSBURGH
PITTSBURGH PA 15260

RAJ REDDY 1
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213

EDWINA RISSLAND 1
DEPT OF COMPUTER & INFO SCIENCE
UNIVERSITY OF MASSACHUSETTS
AMHERST MA 01003

MR NORMAN SADEH 1
CIMDS
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213

MR ERIC TIFFANY 1
ASCENT TECHNOLOGY INC.
237 LONGVIEW TERRACE
WILLIAMSTOWN MA 01267

MANUELA VELOSO 1
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891

MR DAN WELD 1
DEPT OF COMPUTER SCIENCE & ENG
MAIL STOP FR-35
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195

MR CRAIG WIER 1
ARPA/SISTO
3701 N. FAIRFAX DR
ARLINGTON VA 22203

MR JOE ROBERTS ISX CORPORATION 2231 CRYSTAL DRIVE, SUITE 500 ARLINGTON VA 22202	1
COL JOHN A. WARDEN III ASC/CC 225 CHENNAULT CIRCLE MAXWELL AFB AL 36112-6426	1
DR TOM GARVEY ARPA/SISTO 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
MR JOHN N. ENTZMINGER, JR. ARPA/DIRO 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
LT COL ANTHONY WAISANEN, PHD COMMAND ANALYSIS GROUP HQ AIR MOBILITY COMMAND 402 SCOTT DRIVE, UNIT 3L3 SCOTT AFB IL 62225-5307	1
DIRECTOR ARPA/SISTO 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
MS LESLIE WILLIAMS DIGITAL SYSEMS RSCH INC 4301 NORTH FAIRFAX DRIVE SUITE 725 ARLINGTON VA 22203	1