Constraint-Directed Search:
A Case Study of Job-Shop Scheduling

Mark S. Fox

CMU-RI-TR-83-22
CMU-CS-83-161

13 December 1983

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

This research was sponsored, in part, by the Air Force Office of Scientific Research under contract F49620-82-K0017, and the Westinghouse Corporation.

Table of Contents

LIST OF FIGURES

List of Figures

'.IST OF TABLES

List of Tables

Acknowledgements

Eight years is a long time to complete a Ph.D., a long time. In the course of those years, I have had to rely on many people, too many to mention, who have supported me while I have been off attending to other matters: participating in the creation of the Robotics Institute, and building the Intelligent Systems Laboratory. I would like to thank:

My parents, Sollie and Ida Fox, who provided unfailing support throughout the years, and my brothers and sister-in-law, Howard, Barry, and Cindy who never doubted that I could and would complete it.

Herb Simon and Raj Reddy who have acted as both my advisors and mentors. My intellectual debt to them can never be repaid.

Scott Fahlman and Tom Morton who have attempted to keep my work clear and honest. One day I may yet rise to their expectations.

The members of the ISIS project: Brad Allen, Paul Kohlbrenner, Steve Smith, and Gary Strohm, whose contributions to this project are uncountable, and without whom none of this would be possible.

The Westinghouse Electric Corporation for using their corporation as a testbed. In particular Jose Isasi for his undying faith, and Bob Baugh, Bob King, Dwight Mize, and George Yates for their continued support.

Alas, I am not married, hence do not have a wife to thank. Never-the-less, I do wish to thank the women whom I have known over the last eight years and have considered themselves of secondary interest when placed in apposition to this thesis.

Abstract

This thesis investigates the problem of constraint-directed reasoning in the job-shop scheduling domain. The job-shop scheduling problem is defined as: selecting a sequence of operations whose execution results in the completion of an order, and assigning times (i.e., start and end times) and resources to each operation. The number of possible schedules grows exponentially with the number of orders, alternative production plans, substitutable resources, and possible times to assign resources and perform operations. The acceptability of a particular schedule depends not only on the availability of alternatives, but on other knowledge such as organizational goals, physical limitations of resources, causal restrictions amongst resources and operations, availability of resources, and preferences amongst alternatives. By viewing the scheduling problem from a constraint-directed search perspective, much of this knowledge can be viewed as constraints on the schedule generation and selection process. The problem of scheduling orders in a job-shop under these constraints raises a number of issues of interest to the artificial intelligence community such as:

- knowledge representation semantics for organization modeling,
- extending knowledge representation techniques to include the variety of constraints found in the scheduling domain,
- integrating constraints into the search process -- in particular, determining how to use constraints to bound the generation and focus the selection of alternative solutions,
- relaxing constraints when conflict occurs, and
- analyzing the interaction between constraints to diagnose poor solutions.

In this thesis, we present a system called ISIS. ISIS uses a constraint-directed search paradigm to solve the scheduling problem. ISIS provides:

- a knowledge representation language (SRL) for modeling organizations and their constraints,
- hierarchical, constraint-directed scheduling of orders, which includes:
    - constraint-directed bounding of the solution space,
    - context-sensitive selection of constraints, and
    - weighted interpretation of constraints.
- analytic and generative constraint relaxation, and
- techniques for the diagnosis of poor schedules.

In addition, the ISIS system has been designed to provide complete facilities for practical use in the factory. These facilities include: interfaces for updating factory status, incremental

scheduling in response to changes in the factory, interfaces for altering the factory model, and interactive, scheduling with flagging of poorly satisfied constraints. Versions of the ISIS program have been tested on a model of a real factory using simulated orders.

Chapter 1 Introduction

This thesis investigates the problem of constraint-directed reasoning in the job-shop scheduling domain. The job-shop scheduling problem is defined as:

- selecting a set of operations whose execution results in the completion of an order, and

- assigning times (i.e., start and end times) and resources to each operation.

The number of possible schedules grows exponentially with the number of orders, alternative production plans, substitutable resources, and possible times to assign resources and perform operations. The acceptability of a particular schedule depends not only on the availability of alternatives, but on other knowledge such as:

- **Organization Goals**: due date requirements, work-in-process time requirements, cost restrictions, and machine utilization goals,

- **Physical Limitations**: machine capabilities, product size and quality limitations,

- **Causal Restrictions**: precedence of operations, and resource requirements to perform an operation,

- **Availability**: availability of resources (e.g., tools, fixtures, NC programs, and operators) to perform an operation, and

- **Preferences**: qualitative preferences for operations, machines, and other resources.

By viewing the scheduling problem from a constraint-directed search perspective, much of this knowledge can be viewed as constraints on the schedule generation and selection process. The problem of scheduling orders in a job-shop under these constraints raises a number of issues of interest to the artificial intelligence community such as:

- knowledge representation semantics for organization modeling,

- extending knowledge representation techniques to include the variety of constraints found in the scheduling domain,

- integrating constraints into the search process -- in particular, determining how to use constraints to bound the generation and focus the selection of alternative solutions,

- relaxing constraints when conflict occurs, and

- analyzing the interaction between constraints to diagnose poor solutions.

In this thesis, we present a system called ISIS. ISIS uses a constraint-directed search paradigm to solve the scheduling problem. ISIS provides:

- a knowledge representation language (SRL) for modeling organizations and their constraints,

- hierarchical, constraint-directed scheduling of orders, which includes:
    - constraint-directed bounding of the solution space,
    - context-sensitive selection of constraints, and
    - weighted interpretation of constraints.

- analytic and generative constraint relaxation, and

- techniques for the diagnosis of poor schedules.

In addition, the ISIS system has been designed to provide complete facilities for the practical use in the factory. These facilities include: interfaces for updating factory status, incremental scheduling in response to changes in the factory, interfaces for altering the factory model, and interactive scheduling with flagging of poorly satisfied constraints.

Versions of the ISIS program have been tested on a model of a real factory, using simulated orders. Experiments were conducted comparing: beam and hierarchical search, forward and backwards search, optimistic and pessimistic constraints, and eager and wait-and-see resource reservation selection.

The objective of this dissertation, from a job-shop scheduling perspective, is to focus on the *representation and utilization of all relevant constraints* in the scheduling process, *to design and construct an interactive system for the modeling and scheduling* of general job-shops, and to bridge the gap between scheduling systems which simply guide the human scheduler, to a scheduling *system that can control operations in realtime.*

The rest of this chapter provides a detailed description of the scheduling problem, followed by a brief discussion of the issues and a summary of the ISIS system.

1.1. The Scheduling Problem

The goal of this section is to illustrate the variety of different constraints found in the scheduling problem. Many of which have not been covered by previous techniques. This inability to model all the constraints of the problem has often led to systems of limited ability, leaving much of the work to the human scheduler. In this section we will illustrate the nature of the scheduling problem and the constraints which affect it. A categorization of constraints is provided.

The job-shop scheduling problem involves selecting a sequence of operations (i.e., a process routing) whose execution results in the completion of an order, and assigning times (i.e., start and end times) and resources to each operation. Historically, the scheduling problem is divided into two separate steps. Process routing selection is typically the product of a planning process, and the assignment of times and resources is typically the product of scheduling. Actually, the distinction between planning and scheduling is fuzzy. The choice of routing cannot be made without generating the accompanying schedule. The admissibility of a process routing, is determined by the feasibility of each selected and scheduled operation. An operation is feasible when its resource requirements are satisfied during the scheduled time of the operation. Resource requirements for an operation are determined by the operation and, in turn, by the machines that may perform the operation.

This problem has been described as NP-hard. The sequencing of 10 orders through 5 operations has $(10!)^5$ or more than 10^{30} possible schedules (without gaps or alternative routings). Adding more orders, operations, and resources to the selection process multiplies the complexity of the scheduling problem. The complexity of the job-shop scheduling problem can be illustrated by examining the problems of a real plant.

A Westinghouse Electric Corporation Turbine Component Plant (WTCP) was selected as a test case. The primary product of the plant is steam turbine blades. A turbine blade is a complex three dimensional object produced by a sequence of forging, milling and grinding operations to tolerances of a thousandth of an inch. Thousands of different blades are produced in the plant; much of them as replacements in turbines currently in service.

The plant continuously receives orders for one to a thousand blades at a time. Orders fall into at least six categories:

1. Forced outages (FO): Orders to replace blades which malfunctioned during operation. It is important to ship these orders as soon as possible.

2. Critical replacement (CR) and Ship Direct (SD): Orders to replace blades during scheduled maintenance. Advance warning is provided, but the blades must arrive on time.

3. Service and shop orders (SO, SH): Orders for new turbines. Lead times of up to three years may be known.

4. Stock orders (ST): Order for blades to be placed in stock for future needs.

The part of the plant considered by ISIS has 100 to 200 orders in process at any time.

Parts are produced according to a process routing or lineup. A routing specifies a sequence of operations on the part. An operation is an activity which defines:

- the resources required such as tools, materials, fixtures, and machines,
- machine setup and run times, and
- labor requirements.

In the plant, each part number has one or more process routings containing ten or more operations. Process routings may be as simple as substituting a different machine, or as complex as changing the manufacturing process. Further more, the resources needed for an operation may also be needed by other operations in the shop.

During our discussions, we found that orders are not scheduled in a uniform manner. Each scheduling choice entails side effects whose importance varies by order. One factor that continuously appears is the reliance of the scheduler on information other than due dates, process routings, and machine availability. The types and sources of this information were found by examining the documents issued by the scheduler. A schedule is distributed to persons in each department in the plant. Each person on the distribution list can provide information which may alter the existing schedule. In support of this observation, we found that the scheduler is spending 10% to 20% of his time scheduling, and 80%-90% of his time communicating with other employees to determine what additional "constraints" may affect an order's schedule. These constraints include operation precedence, operation alternatives, operation preferences, machine alternatives and preferences, tool availability, fixture availability, NC program availability, order sequencing, setup time reduction, machine breakdowns, machine capabilities, work-in-process time, due dates, start dates, shop stability, cost, quality, and personnel capabilities/availability.

From this analysis, we may conclude that the object of scheduling is not only meeting due dates, but satisfying the many constraints found in various parts of the plant. Scheduling is not a distinct function, separate from the rest of the plant, but is highly connected to and dependent upon decisions being made elsewhere in the plant. The added complexity imposed by these constraints leads schedulers to produce inefficient schedules. Indicators such as high work-in-process, tardiness, and low machine utilization support this conclusion[1]. Hence, any solution to the job-shop scheduling problem must identify the set of scheduling constraints, and their affect on the scheduling process. In the following, we examine the variety of constraints uncovered in the WTCP plant.

1.1.1. Constraint Categories

The first category of constraint encountered in the factory is called an *Organizational Goal*. Part of the organization planning process is the generation of measures of how the organization is to perform. These measures act as constraints on one or more organization variables. An organization goal constraint can be viewed as an expected value of some organization variable. For example:

Due Dates: A major concern of a factory is the meeting of due dates. The lateness of an order affects customer satisfaction.

[1]It is unfair to measure a scheduler's preformance based on the above measures alone. Our analysis has shown that scheduling is a complex constraint satisfaction problem, where the above indicators illustrate only a subset of constraints that the scheduler must consider. Schedulers are expert in acquiring and "juggling" the satisfaction of constraints.

Work-In-Process: Work-in-process (WIP) inventory levels are another concern. WIP inventory represents a substantial investment in raw materials and added value. These costs are not recoverable until delivery. Hence, reducing WIP time is desirable.

Resource Levels: Another concern is maintaining adequate levels of resources necessary to sustain operations. Resources include personnel, raw materials, tools, etc. But each resource may have constraints, for example, labor size must be smoothed over a month's interval, or raw materials inventory may have to be limited to a two day supply..

Costs: Cost reduction can be another important goal. Costs may include material costs, wages, and lost opportunity. Reducing costs may help achieve other goals such as stabilization of the work force.

Production Levels: Advance planning also sets production goals for each cost center in the plant. It serves two functions: it designates the primary facilities of the plant by specifying higher production goals, and also specifies a preliminary budget by predicting how much the plant will produce.

Shop Stability: Shop stability is a function of the number of revisions to a schedule and the amount of preparation disturbed by these revisions. It is an artifact of the time taken to communicate change in the plant and the preparation time.

One can view all organizational goal constraints as being approximations of a simple profit constraint. The goal of an organization is to maximize profits. Scheduling decisions are then made on the basis of current and future costs incurred. For example, not meeting a due date may result in the loss of a customer and, inturn, further profits. The longer the work in process time, the greater the carrying charge for raw materials and value added operations. Maintaining a designated production level may spread the cost of the capital equipment in a uniform manner. In practice, most of these costs cannot be accurately determined, but must be approximated.

A second category of constraint is *physical*. Physical constraints may specify characteristics which limit functionality. For example, a milling machine may be limited in the size of turbine blade it can work on due to the length of its workbed. A drill may have a graph which defines how long the drill can run at a particular speed in a particular material.

A third category of constraint is *causal*. It defines what conditions must be satisfied before initiating an operation. Examples of causal constraints include:

Precedence: A process routing is a sequence of operations. -A precedence constraint on an operation states that another operation must take place before (or after) it. There may be further modifiers on the constraint in terms of minimum or maximum time between operations, product temperature to be maintained, etc.

Resource Requirements: Another causal constraint is the specification of resources to be present before or during the execution of a process. For example, a milling operation requires the presence of certain tools, an operator, fixtures, etc.

A fourth category of constraint is *preference.* A preference constraint can also be viewed as an abstraction of other types of constraints. Consider a preference for a machine. It expresses a floor supervisor's desire that one machine be used instead of another. The reason for the preference may be due to cost or quality, but sufficient information does not exist to derive actual costs. In addition, machine preferences, operation preferences, and queue position preferences are examples of this type of constraint.

A fifth category of constraint is concerned with the *availability* of a resource. During the production of a schedule, as each resource is assigned to an operation, it has attached to it a constraint that defines it unavailability for other uses during that time period.

The following lists the variety of constraints and their categories, found in the scheduling domain:

Constraint	**Org. Goal**	**Physical**	**Causal**	**Pref.**	**Avail.**
Operation alternatives			x		
Operation Preferences				x	
Machine alternatives			x		
Machine Preferences				x	
Machine physical constraints		x			
Set-up times	x	x			
Queue ordering preferences				x	
Queue stability	x				
Due date	x				
Work-in-process	x				
Tool requirement			x		
Material requirement			x		
Personnel requirement			x		
Resource reservations					x
Shifts	x				x
Down time					x
Productivity achieved	x				
Cost	x				
Productivity goals	x				
Quality	x	x			
Inter-operation transfer times			x		

A review of commercial scheduling systems found that most provide simple capacity analysis with an emphasis on meeting due dates. This was found to be unacceptable for WTCP. These systems are also batch oriented, meant to be run weekly or monthly, and do not provide realtime control. Nor do they provide full constraint representation and utilization facilities. On the other hand, Management Science research focuses on optimal results for artificial problems, or dispatch rules for meeting due dates or makespan (i.e., facility utilization) which also have been found to be unsatisfactory for the real life job-shop scheduling problem.

WTCP uses a commercial scheduling system. Having recognized its limitations and those of manual scheduling, they continue to pursue the identification and acquisition of software to support the scheduling process.

1.2. Constraint-Directed Search: Issues and Objectives

Job-shop scheduling provides a rich environment for investigating the theory of constraint-directed search. The scheduling problem can be characterized as having a large search space which may be reduced through the examination and utilization of the relevant constraints. It is the purpose of this section to present the constraint-directed search issues, and the objectives of ISIS as related to them.

At present, there do not exist general models for the representation of constraints. Hence, the first objective of this research is to identify and represent the variety of constraints, and knowledge about them. As an example of constraint knowledge, consider a due date. One method of representing a due date would be by date alone. The implication being that the job would be shipped on that date. In reality, not all due dates can be met. Information in addition to the due date is required if a scheduling system is to construct a satisfactory schedule. For example:

- what alternative dates are satisfactory if the original cannot be met,
- what the preferences exist for these alternative dates,
- who specified the due date, when, and why,
- is the satisfaction of the due date more important than other constraints such as costs,
- does the satisfaction of the due date constraint positively or negatively affect the satisfaction of other constraints,
- under what circumstances should the due date constraint be considered, and
- if there are two or more due date constraints specified for an order, which should be used?

The constraint representation problem is concerned with the representation of this knowledge for effective utilization during search.

One of the first issues to be faced in the representation of constraints is *conflict*. Consider cost and due-date constraints, the former may require reduction of costs while the latter may require shipping the order in a short period of time. To accomplish the latter may require using faster, more expensive machines, hence conflicting with the former. If the conflict cannot be solved, one or both constraints must "give ground" or be *relaxed*. This is implicitly accomplished in mathematical programming and decision theory by means of utility functions and the specifications of relaxation through bounds on a variable's value. In AI, bounds on a variable are usually specified by predicates (Stefik, 1981a; Engleman et al., 1980) or choice sets (Steele, 1980; Waltz, 1975). In ISIS, our objective is to extend the general representation of knowledge to include the specification of constraints and their relaxations.

A second issue is constraint *importance*. Not all constraints are of equal importance. A due date constraint for a high priority order may be more important than an operation preference constraint. Any constraint-directed search system, before it attempts to choose a constraint to relax, must determine which is the least important. The importance of constraints may also differ from order to order. In one order, the due date may be important, and in another, cost may be important. A reasoning system should be able to perform these two types of differentiation. Our objective in ISIS is to extend the representation of constraints to include the specification of relative importance.

A third issue is the understanding of how constraints *interact*. Conflicting constraints may negatively affect each other. For example, removing a machine's second shift may decrease costs but cause an order to miss its due date. Knowledge of these interactions before scheduling may help in ruling out poor choices. How to use the knowledge of interactions between constraints effectively is not well understood. Our objective in ISIS is to include in the representation of constraints, the existence of interactions with other constraints, and their effect when choosing among alternative relaxations.

A fourth issue is constraint *obligation*. Our experience with factories has uncovered problems in the *practical* application of constraints. Organizations change, and constraints change with it. Hence, for how long, and during which activities is a reasoning system obliged to follow a constraint? A second aspect of obligation is resolving inconsistencies within a constraint type. ISIS may be used by a number of departments in the factory; the same constraint type with different values may be created and applied to the same object. For example, both the material and marketing departments may place different and conflicting due date constraints on the same order. The system must determine which one it is obliged to follow. Another objective is to represent the knowledge necessary to determine the obligation of ISIS in satisfying constraints.

A fifth issue is constraint *generation*. Constraints have many sources. Many may be defined by the user during the creation of the plant model. Others may be defined dynamically as the production proceeds. For example, the constraint on the mass of metal removed during an operation is dependent on the mass of the metal going into the operation. Hence, the constraint is determined at the time the operation is performed. The issue of constraint generation is defining the ability to create and use constraints. An objective of ISIS is to enable the specification of constraint generators in the representation.

Aside from the issues of representation, there are other issues of importance. For example, how should ISIS react to poor results from the search process. All heuristic-based problem-solving methodologies suffer from the periodic generation of poor solutions. Our objective is to *recognize* poor solutions and to *suggest* ways of finding better solutions. In particular, how to use constraints to: remove poor partial solutions during the search process, diagnose poor final solutions, and suggest relaxations to related constraints which may result in better solutions.

Another important issue is efficiency. As described earlier, job-shop scheduling is combinatorially explosive; optimal solutions are generally not attainable. Yet, in the factory, a scheduling system must provide near realtime response. It cannot take hours to develop a schedule. Hence, any solution to the above issues must also be computationally efficient. Our objective is to design a system which will provide "good" results with a reasonable amount of processing time.

1.3. Thesis Summary

The remainder of the thesis is composed of a review of the literature followed by chapters on:

- Scheduling and constraint-directed reasoning review.
- Plant modeling,
- Constraint representation,
- Constraint-directed scheduling methods,
- Analysis, and
- Observations and conclusions.

The following summarizes these chapters:

1.3.1. Modeling

The ISIS modeling system is the repository of all the knowledge necessary to plan and schedule production. It should be able to model the concepts of activities, factory state, orders, etc. in a manner which is machine interpretable. The system is built using SRL (Fox, 1979; Wright & Fox, 1983), a knowledge representation system, which allows the user to mold the language to his needs. SRL is a frame-based language which encodes concepts as schemata. A schema is a collection of slots and values. Each schema, slot, and/or value may have meta-information attached to it. In addition to attribute knowledge, slots define inter-schema relations, through which slots and values may be inherited. The inheritance semantics of a relation is user definable. SRL has been used to support a number of different Intelligent Management System functions (Fox, 1981) including simulation (Reddy & Fox, 1983) , diagnosis (Fox Lowenfeld & Kleinosky, 1983), graphics, project management, and long range planning (Kosy & Dhar, 1983).

The ISIS modeling system is a multi-layer system for modeling manufacturing organizations in SRL. Its layers are: structural, basic semantics, world semantics, and domain semantics. The basic concepts are that of states, objects, and acts. Acts transform states and objects. Time and causality are primitive concepts in the language. Time relations provide time ordering amongst states and acts. Causal relations define how states enable acts, and acts cause states. A manufacturing operation is defined as an act, and time and causality relations link it to other states and acts. Prototypes, instances, and manifestations are used to distinguish classes, elements and states of objects and acts respectively. SRL's relation definition mechanism permits the construction of new relations which are abstractions of these conceptual primitives. Hence, the relation **next-operation** is defined in terms of time and causal relations. Operations may also be defined in multiple levels of abstraction. Resources are defined as objects. Attributes and physical structure may be defined for an object. Allocation of resources is defined as a state of possession by some operation or resource with a specified time relation (e.g., duration). Orders are also represented with these primitives. An order is simply a goal state (e.g., shipped) to be achieved by scheduling the appropriate operations.

The modeling language has an active interpretation. Its semantics supports the simulation of parallel, discrete events (acts).

1.3.2. Constraints

The ISIS modeling system provides primitives for describing the common actions and states in the scheduling domain. Experience has shown that a major part of our knowledge about a plant is an understanding of constraints on the plant's operation, and alternatives to perform when the constraints cannot be met. The modeling system is extended by providing the capability to attach constraints to a schema, its slots, and values. Hence, any concept representable in SRL, can also have constraints attached to it.

In scheduling, constraints may not always be satisfiable. Hence the representation of alternatives is important. The modeling system adds to the representation of constraints the specification of *relaxations*. Relaxations may be defined either as predicates, or choice sets which can be discrete or continuous. Associated with a relaxation is a preference measure which determines the preferred relaxations among those available. The representation of constraints and their *preferred* relaxations is also a solution to the pattern specification and matching problem.

The representation of constraints must not only cover what the constraint is, but when and how to use it. The first problem in using constraints focuses on their relative importance. Depending on the order, some constraints are more important than others. ISIS can represent relative importance by either an absolute measure of importance, or by the partioning of constraints into importance classes. Selecting which constraint to relax also depends on how the relaxation will affect other constraints. For example, reducing the number of shifts in the plant may be preferred, but it may negatively affect the due date constraint of many orders. Interactions of this nature are represented as relations in ISIS. It may also be the case, that the set of known constraints, may not be applicable to the particular decision. The system's *obligation* to satisfying a constraint depends on a number of factors: the time over which the constraint is applicable, the consistency of the constraint with others, the source of the constraint, and the context. ISIS provides a representation for all of this information. Lastly, constraints may be generated dynamically by attaching constraint generators to relations in the model.

1.3.3. Scheduling

The goal of ISIS is to construct schedules which satisfy as many constraints as possible in near realtime. To achieve this, ISIS uses constraints to bound, guide and analyze the scheduling/search process. The ISIS system performs a hierarchical, constraint-directed search in the space of alternative schedules. Level 1 selects an order to be scheduled according to a prioritization algorithm based on the category of the order, and its due date. Level 2 performs a current capacity analysis of the plant. It determines the earliest start time and latest finish time for each operation of the selected order, as bounded by the order's start and due date. The times generated at this level are codified as constraints which are passed to level 3. These operation time bound constraints constrain the start and end times of operations at the next level. Level 3 performs a detailed scheduling of all resources necessary to produce the selected order. Pre-search analysis begins by examining the constraints associated with the order to determine the scheduling direction (forward vs backward), whether any additional constraints should be created (e.g., due dates, work-in-process), and the search operators which will generate the search space. A beam search is then performed using the selected search operators. The beam search sequences the application of operators. Each application of an operator generates another "ply" in the search space. At each ply only the "n" highest rated states are selected for extension to the next ply. The most often selected operators generate alternative operations, machines, and queue positions for an order in the plant. Starting with a null schedule, alternative partial schedules are generated either forward from the start date or backward from the due date. An operation operator generates alternative states which represent alternative operations in either the forward or backward direction. Once the operation is known for a state, other operators extend the search by creating new states which bind the machine and/or the execution time of the operation. A variety of alternatives exist for each type of operator. For example, two operators have been tested for choosing the execution time of an operation. The "eager reserver" operator chooses the earliest possible reservation for the operation's required resources, and the "wait and see" operator tentatively reserves as much time as available, leaving the final decision to level 4. This enables the adjustment of reservations in order to reduce work in process time. Alternative resources are generated (e.g., tools, materials, etc.) by other operators. Each state in the search space is rated by the set of constraints found (resolved) to be relevant to the state and its ancestors. Constraints defined to be in the set are those which are attached to any resource (e.g., machine, tool, order, etc.) specified by the state. Each constraint assigns a utility between 0 and 2 to a state; zero signifies that the state is not admissible, 1 signifies indifference, 2 maximal support. The rating of a state with multiple constraints is the weighted (by importance) average of the constituent constraints. The importance of a constraint is defined statically or derived dynamically according to goal information. Once a set of candidate schedules have been generated, a rule-based post search analysis examines the candidates to determine if one is acceptable. Currently, any schedule with a rating greater than one is accepted. If no acceptable orders are found, then diagnosis is performed. First, the schedules are examined to determine a type of scheduling error. The error is then fed back to pre-analysis in order to select new operators which are used to re-schedule the same order. The diagnosis of poor solutions caused by constraint satisfaction decisions made at another level can be performed by analyzing the interaction relations linking constraints. A poor constraint decision at a higher level can be determined by the utilities of constraints affected by it at a lower level, and an alternative value chosen. Level 3 outputs reservation time bounds for each resource required for the operations in the chosen schedule. Level 4 selects the actual reservations for the resources required by the selected operations which minimize the work-in-process time.

The scheduling of ISIS is also reactive. The invalidation of reservations by actions such as machine breakdowns or other orders taking too long on a machine, results in a minimal re-scheduling of only the affected orders, while attempting to maintain previous reservations. ISIS's scheduling is also suggestive. If constraints cannot be met, it attempts to generate a schedule which satisfies as many constraints as possible. For example, if the due date of an order cannot be met by backwards scheduling, it attempts to schedule in the forward direction and suggests an alternative due date.

1.3.4. Analysis

Two series of experiments are performed, all based on a portion of the turbine plant defined by the plant scheduler. Each experiment in a series, tests a different combination of constraints, scheduling direction, search operators, and levels of reasoning. One experiment removes the capacity analysis and reservation selection levels. The resulting schedule displayed high tardiness (65 of 85), and high work in process times with a makespan of 857 days. This is due to horizon effect caused by the coupling of beam search and insufficient machine capacity. The experiment which included the capacity analysis and wait-and-see reservation selection levels displayed fewer tardy lots (17 of 85), and very low work in process time with a makespan of 588.8 days. At this point, machine capacity is the principal limitation affecting tardiness. Experiments with all lots being scheduled backwards from the due date and added capacity were also performed.

1.3.5. Conclusions

The contributions of this thesis are in three areas: representation, constraint-directed search and job-shop scheduling. In the representation area, a more complete semantics for the modeling of organizations is provided which includes: states, acts, time, causality, multi-level representation, and support for discrete simulation. In the area of search, ISIS introduces a number of new concepts:

- A general representation for constraints with particular attention paid to the representation of relaxations, interactions, and obligations.
- Constraint-based pre- and post-search analysis to bound the solution space before performing search, and diagnose poor constraint decisions at other levels.
- The generation and evaluation of constraint relaxations during the search process.

- The resolution and differentiation among constraints in evaluating states in the search space.

- Hierarchically constrained search. A level in the hierarchy communicates only constraints in order to guide search at the next level.

The contribution to job-shop scheduling made by this thesis is that it provides, for the first time, a system which can represent and consider all the domain constraints during the construction of a schedule. And do so in a reasonable amount of processing time. It also provides incremental scheduling in reaction to changes in the plant's status, and suggests alternative schedules when constraints cannot be satisfied.

Chapter 2
Review

2.1. Introduction

Scheduling has been a major research topic in management science for many years. In contrast, it has received little attention in artificial intelligence. But this is not to say that AI has little to bring to the table. As management science has recognized, it is not practical to separate planning from scheduling. The inability to construct an acceptable schedule may require the construction of alternative plans. So a scheduling system must also consider the planning problem. Secondly, much AI research is concerned with general reasoning processes. Hence many of the results are transferable among domains.

In the following, both Management Science (MS) and AI research are briefly reviewed. The review of the MS literature focuses on some of the relevant research in the area of scheduling and sequencing. The review of the AI literature focuses on general reasoning (problem-solving) research, and on constraint research.

2.2. Management Science

Management science research in scheduling has focussed on understanding the variety of scheduling environments that exist, and constructing scheduling algorithms specific to them. Four types of "shops" are distinguished in the literature:

- single machine - single operation
- parallel machines - single operation
- flow shop series of machines - multiple operations
- job shop network of machines - multiple operations

A job is defined as having:

- one or more operations
- a processing time for each operation
- a due date

And the utility of a scheduled is measured in terms of:

- lateness

- tardiness
- flowtime
- makespan

It was recognized early in management science that scheduling is an example of a constraint satisfaction problem which could be optimally solved using mathematical programming techniques. Integer programming approaches, while theoretically valid are useless practically. Such approaches are members of the class of problems described as NP (Cook, 1971). This problem of algorithmic complexity has forced a bifurcation of the field.

One branch of research focuses on the attainment of optimal results, but algorithmic complexity has restricted these results to the one and two machine cases (Lenstra & Rinnooy Kan, 1980). And the achievement of these results requires the removal of much of the constraints, and the focus on a single criterion for measuring schedule efficacy.

The second branch takes a heuristic approach called priority dispatch rules. A dispatch rule is a local decision rule which determines the next job to be processed on a machine from the set queued at the machine. Extensive simulation analyses have shown that the apparent urgency rule (Rachamadugu, 1982), and the weighted shortest processing time rule (WSPT) provide reasonable schedules with respect to a single criterion such as tardiness, while also ignoring many of the constraints found in a typical factory (Conway, 1965).

The dispatch rule approach suffers in that it does not provide predictive information about future operations and their machine reservations; it does not consider alternative paths; it does not incorporate other constraints. By contrast, the integer programming approach is combinatorially explosive and can handle linear constraints on integer variables.

One approach to the solving the complexity problem is to take a hierarchical approach to scheduling (Hax & Golovin, 1978). Each level may incorporate a mathematical programming approach, but considers only a subset of the total information. The results of one level, restrict the processing of the next. By using levels of abstraction, the number of variables and constraints is minimized, at the cost of ignoring possibly important information. Secondly, the flow of information is from the top down. Information at the lower levels cannot be fed back into the higher levels to affect their processing.

All of the above systems can be described as "guidance" systems. They construct schedules with are meant to guide the actual scheduling decision making performed on the shop floor. They are limited to guiding because they lack the information necessary to make detailed decisions. To circumvent this problem, there has been intense investigation of interactive scheduling systems (Godin, 1978). The interactive job-shop schedulers suffered from the same problems listed above since they incorporated those algorithms (Ferguson & Jones, 1969; Godin, 1968). But they did allow users to interactively modify schedules. More recent work in interactive flowshop scheduling uses resource-usage leveling to measure scheduling effectiveness, and a swapping heuristic to construct schedules (McDonald & Hodgson, 1980).

2.3. Artificial Intelligence

Two areas of research in artificial intelligence potentially impact the scheduling problem. The first is planning research, the second is constraint analysis. As depicted earlier, scheduling is a two part process: the first part is generating a sequence of tasks or actions to be accomplished, the second is the assignment of resources over time to accomplish tasks. Obviously, the latter impacts the choice of the former.

Before reviewing the planning literature, and at the risk of being pedantic, it is useful to consider a simple model for planning, and see how research over the last two decades has extended it. The most basic AI method is *heuristic search*. A search is made within a space of possible solution states for a state that satisfies some goal description. A state can be transformed into another state by applying a heuristic (operator) to it. Planning can be viewed as a form of heuristic search. The first problem in creating a planning system is to generate the states relevant to reaching the goal. Given a description of the initial state, goal state, and a set of operators, the operators can be iteratively applied to the initial state, and its successors, until a goal state is found. The path from the initial state to the goal state is a solution path of operations, or plan. Depending on the "strength" of the operators, the space elaborated can be large or small; however the better heuristics generate smaller search spaces and find the solution faster. Planning, and related research, has focussed on a number of issues: for instance, choosing what state to elaborate next, choosing which operator to expand a state, and choosing alternative state representations and operators (Amarel, 1967).

Planning research in AI can be traced back to the LT and GPS systems. LT (Newell & Simon, 1956) introduced the concept of heuristic search and its problem-solving states, goal states, and the operators that transform states into other states. It worked backward from the goal, splitting the problem into subproblems which it proved separately. GPS (Newell & Simon, 1963) generalized the approach into what they termed Means End Analysis. By use of a difference table, they were able to reduce the search space by focussing on only the most relevant operations in achieving the goal state.

Another research area, not normally classified as planning, is that of game playing. Many of the game playing programs in chess and checkers (Samuel, 1963) use heuristic search. The selection of moves to make during a game is equivalent to constructing a plan. An important contribution such systems have made to planning and scheduling is the rating of plans.[2] Due to the large size of the search space, game playing systems are required to prune the examined states. To achieve this, an evaluation function is used to rate states, in effect answering: of all the legal moves that can be made, what are the ***preferred*** moves? A variety of search algorithms such as min-max, A* (Nilsson, 1971), and B* (Berliner, 1979), have been used.

[2]It is not clear that many of the planning systems constructed to date have benefited form this research. This is due mainly to the "toyness" of the problems attacked.

Early robot planning research resulted in the formalization of operators in the predicate calculus. The STRIPS system (Fikes & Nilsson, 1971) represented operators as rules with pre-conditions and post conditions. GPS-like means-end analysis was used to plan tasks.

Early on in planning research, Simon (1962) recognized that a planning system in a real domain will have to struggle with the size of the search space. He proposed that planning be done at differing levels of abstraction. By designating planning hierarchies, planning can proceed at the highest, least detailed level and use the plan to constrain planning at the next, more detailed level, and so on. One could view the ordering of differences and operators in GPS's difference table as an implicit hierarchy. The first explicit manifestation of this concept was ABSTRIPS (Sacerdoti, 1974). By separating pre-condition variables into levels of importance, the pre-conditions would contain only the variables important at the specified for the current level of planning. While achieving good performance, a problem with this method was the a priori designation of variable levels.

Another issue of concern in planning was that of *goal protection*. The result of one operation may be undone by another operation, *before* the result could be utilized in the overall achievement of the goal. To deal with this, the HACKER system (Sussman, 1975) used a *debugging* approach to fix a plan after it was constructed. A set of *critics* were dynamically constructed to recognize errors and suggest corrections. Though an interesting approach, it lacked extensibility. On the other hand, the NOAH system (Sacerdoti, 1975), took a *least-commitment* approach to planning. NOAH would not sequence operations unless forced to. This approach reduced the amount of backtracking necessary to secure a legal plan because the current plan did not make any unnecessary sequencing decisions.

Reasoning at multiple levels of representation can also be found in Hearsay-II. Though the Hearsay-II speech understanding system (Erman et al., 1980) cannot be viewed as a planning system, its architecture has had a major effect on more current planning systems. Some of the ideas that Hearsay-II incorporated, are:

- multiple levels of representation,
- data and goal directed (bidirectional) problem-solving and,
- island-driving.

Hayes-Roth & Hayes-Roth (1980) call the combination of bidirectional problem-solving and the ability to start problem-solving at any point in the search space (island-driving as opposed to left to right), *opportunistic* reasoning. Opportunistic reasoning reduces the search space by focussing the planning effort in areas that are of high certainty and/or highly constrained. By extrapolating these "islands", further constraints on the more uncertain parts of the planning space will most-likely be generated.

Hearsay-II did incorporate a planning mechanism. Its policy modules (Hayes-Roth & Lesser, 1976) combined to form a *focus of attention* system that determined the sequence of knowledge source executions. When parts of the utterance remained uninterpreted, it dynamically determined what parts of the search space required more attention and turned the systems resources towards reducing the uncertainty in those areas. By understanding what problem-solving methods it had available (i.e., knowledge sources) and its resource constraints, it would decide the best next action. The ability to reason about "how to reason" (or plan) has been called *meta-planning* in MOLGEN (Stefik, 1981b) and also appeared as *meta-rules* in TEIRESIAS (Davis, 1976; Davis & Buchanan, 1977) As in conventional hierarchical planning the ultimate result of these techniques is to reduce the space of states that is searched by the system.

While much of the planning research mentioned above was concerned with how to reduce the search space, there are other aspects of the planning problem that we have yet to consider in this review. Game playing systems introduced search techniques for adversary-oriented games. That is, the search would consider both the programs' moves and the opponents moves in determining a next move. The concept of *adversary-oriented planning* has reappeared as *counter-planning* in the POLITICS system (Carbonell, 1979). This research can be viewed as a form of goal-protection where the system has to consider what the adversary may do to prevent the system from achieving its goals.

Another type of planning is concerned with the satisfaction of *multiple, possibly competing goals*. All of the above planning research is concerned with achieving a single goal. One of the few pieces of research in the multiple goal satisfaction area is the system NUDGE (Goldstein & Robert, 1977). A heuristic approach was developed for the domain of appointment calendar maintenance. What was unique about this research was that it included rules for the *relaxation* of constraints. When a schedule could not be found that satisfied the existing constraints, it used the rules to propose alternatives (possibilities) by relaxing certain constraints such as preferences. In this case, the preference constraint was simply removed. Other rules peculiar to the appointment domain were used to alter existing calendar requirements until a viable schedule was produced.

At this point in the review, we turn away from planning and look at the field of constraint analysis. Much of the constraint analysis research is recent, hence there is not a great deal of literature to review. But one can view planning as constraint analysis in the sense that operators incorporate constraints in their pre-conditions.

One of the earlier works in constraint analysis was REF-ARF (Fikes, 1970). Its task was similar to the linear programming task. Given a set of linear equations that restrict the possible values of a set of variables, can value assignments be found for them? Rather than doing a brute force search for a set of bindings that satisfied all the constraints (equations), it used the constraints to reduce the generated binding set. Hence, the system can be viewed as a classical generate and test, where the system was able to take the constraints and use them in the generator to reduce the size of the search space.

Another type of constraint which has existed for many decades can be called a *binary compatibility constraint*. Consider a grammar. It defines the legal sentences that can be formed from a symbol set. The grammar can be viewed as a constraint on the symbols that will be recognized and/or generated. It defines what symbols are compatible with other symbols when linearly ordered. Another example is the conceptual hierarchy of the SEMANT knowledge source of Hearsay-II (Fox & Mostow, 1977). It is similar to a grammar, but relaxes the sequence constraint at the phrase level, allowing ungrammatical sentences, and sentence fragments to be understood. A third instance is the 3D space description network used in ARGOS (Rubin, 1978). In this case, a network was used to define adjacencies of objects in a visual scene. This was then used to constrain the set of acceptable labelings of an image.

In many real-world applications, constraints are not binary, but are continuous. A *continuous compatibility constraint* imparts a rating of how one symbol is compatible with another. For example, in image understanding, how a pixel is to be labeled is determined by the labels of neighboring pixels. The knowledge of how to do neighborhood based labelling is at best uncertain, hence the constraints that tie pixels together return a certainty rating for each of the possible labelings of the pixel. The higher the rating, the more probable that the label is correct. This type of constraint is the chief mechanism of *relaxation* (Zucker, 1976). Relaxation can also be viewed as a network constraint system. The goal is to assign a value to each node. A nodes value is constrained by the compatibility rules on the incident arcs. CONSTRAINTS (Steele, 1980) can be viewed (loosely) as the dual of relaxation. Behavior rules are associated with nodes, and values with arcs. When an arc value changes, a node's rules determine its effect (i.e., value) on the other incident arcs. The system could recognize inconsistencies in arc values due to the lack of uncertainty in rule knowledge.

As planning moves from single level to hierarchical, so have relaxation and relaxation-like processes. Single level relaxation often does not have enough information to adequately label a scene. By creating multiple levels of representation, higher levels of knowledge could be incorporated (Zucker, 1977).

The next step was to combine both binary and continuous constraints in a hierarchical system. Again, image understanding research has been the area for this research (Ballard et al., 1977; Russell, 1979). The representation of constraints in image understanding has also been extended to predicate calculus. Davis (1980) makes the case for predicate calculus as a better representation for discrete relaxation constraints.

MOLGEN combined planning with constraint-analysis (Stefik, 1981a). As plans were broken into sub-problems, variable value constraints determined in one subproblem were propagated to other subproblems. Hence variables would accumulate constraints across subproblems before an actual binding was chosen (a least commitment approach).

Engelman et al. (1980) in interactive frame instantiation associates constraints with groups of slots. An interesting feature of their approach is that constraints form buckets, each having its own priority. Hence, constraints have a priority ordering.

In getting closer to the scheduling problem, McCalla (1978), in planning driving paths through a town, considered constraints such as possible routes, and time and space restrictions.

One of the few AI scheduling systems was in the domain of train scheduling (Fukumori, 1980). It used a constraint-based approach to determine the arrival and departure times of trains at stations. Trains initially had fuzzy times assigned (i.e., a time span or *belt*). Constraints then reduced the size of the belt. The problem was much simpler than the general scheduling problem in that trains had only one route, and two resources, a track and stations. The fuzziness of times was similar to that used in Hearsay-II in denoting the time span of an hypothesis when its boundaries were uncertain.

A second AI scheduling study was that of Vere (1981). In it plans are constructed, and times associated with each step in the plan. A sophisticated algorithm for time propagation based on interactions is described.

Lastly, Bullers et al. (1980) describe a logic-based factory modeling, question-answering and scheduling system. While the approach is viable for question answering, they ignore the complexity issue involved in scheduling job shops. The approach suffices for small problems only.

2.4. Relationship to Previous Research

Our primary concern in this research is the construction of schedules for job-shops in a near realtime manner. If factories with hundreds of orders, machines and tools are to be scheduled then, scheduling algorithms must be found that find satisfactory solutions quickly, while considering all the necessary constraints. Our approach is to perform constraint-directed heuristic search; constraints are used to bound and guide the search (scheduling) process.

Towards this goal, the most relevant research in the AI literature comes more from general search and focus of attention research, than in any of the constraint research. The research describes how to search in multiple levels of representation, and how to smooth the effects of terms in a polynomial evaluation function (Berliner, 1980), but does not describe how to resolve dynamically the set of constraints by which to rate a state.

Much of the constraint research to date is application specific. There does not exist any general theories of constraint representation, nor approaches to constraint relaxation. The relaxation of constraints is important. Experience has shown that many constraints conflict, making the construction of a schedule which satisfies all constraints impossible. Though NUDGE (Goldstein, 1977) is the only system that explicitly worries about constraint relaxation, its approach is domain specific.

The fuzzy times of Hearsay-II and Fukumori's (1980) train scheduling are an interesting approach to reducing the representation of alternatives. And the focus of attention (Hayes-Roth & Lesser, 1976) research restricts where in the search space to attend. But neither solve the problem of having to bound the solution space in order to make the size of the solution space feasible to search.

In conclusion, many theories of search and constraint-based reasoning have been proposed. But none have been combined to deal jointly with the problem of constraint representation and relaxation, and the constraint-directed construction of schedules in near realtime.

Chapter 3
ISIS Modeling System

Summary

The ISIS modeling system is the repository of all the knowledge necessary to plan and schedule production. It should be able to model the concepts of activities, factory state, orders, etc. in a manner which is machine interpretable. The system is built using SRL (Fox, 1979; Wright & Fox, 1983), a knowledge representation system, which allows the user to mold the language to his needs. SRL is a frame-based language which encodes concepts as schemata. A schema is a collection of slots and values. Each schema, slot, and/or value may have meta-information attached to it. In addition to attribute knowledge, slots define inter-schema relations, through which slots and values may be inherited. The inheritance semantics of a relation is user definable. SRL has been used to support a number of different Intelligent Management System functions (Fox, 1981) including simulation (Reddy & Fox, 1983) , diagnosis (Fox Lowenfeld & Kleinosky, 1983), graphics. project management, and long range planning (Kosy & Dhar, 1983).

The ISIS modeling system is a multi-layer system for modeling manufacturing organizations in SRL. Its layers are: structural, basic semantics, world semantics, and domain semantics. The basic concepts are that of states, objects, and acts. Acts transform states and objects. Time and causality are primitive concepts in the language. Time relations provide time ordering amongst states and acts. Causal relations define how states enable acts, and acts cause states. A manufacturing operation is defined as an act, and time and causality relations link it to other states and acts. Prototypes, instances, and manifestations are used to distinguish classes, elements and states of objects and acts respectively. SRL's relation definition mechanism permits the construction of new relations which are abstractions of these conceptual primitives. Hence, the relation **next-operation** is defined in terms of time and causal relations. Operations may also be defined in multiple levels of abstraction. Resources are defined as objects. Attributes and physical structure may be defined for an object. Allocation of resources is defined as a state of possession by some operation or resource with a specified time relation (e.g., duration). Orders are also represented with these primitives. An order is simply a goal state (e.g., shipped) to be achieved by scheduling the appropriate operations.

The modeling language has an active interpretation. Its semantics supports the simulation of parallel, discrete events (acts).

3.1. Introduction

Computers as decision aids are proliferating in today's organization. Organizations are purchasing more programs at a faster rate than ever before. With the advent of personal computers, each user becomes a computer installation with their own databases and programs, and has the ability to tailor programs and data to suit his needs. Productivity gains may be significant, but possibly short lived. Given the ability to create software systems in their own "image", the form and content of this software will diverge to the point of becoming a "network of babel". In the days when computers were expensive, such problems arose less often. There was a small set of analysts who designed and built programs; the content and form of information being (somewhat) standardized throughout the dp shop. Returning to the days of the large dp shop is not the solution. A better approach would be to provide these mini-installations with a shared language in which to build models. That is, the esperanto of modeling.

Why would users want to share models? A model is more than a simple database. It is an accumulation of expertise: a repository in which both historical information, and the diverse perspectives of members of the organization, are placed. Hence, it is a tool with which to analyze the organization and to instruct new members in its operation. As the distribution of computing grows within an organization, so will the need to share and integrate the knowledge generated therein.

The major task in the construction of a *sharable* modeling language is the choice of conceptual primitives. Not only must these primitives span the set of concepts germane to the application (e.g., planning and scheduling), but the language must satisfy other criteria as well:

Generality: The model should support a variety of functions in addition to scheduling, such as: question-answering, simulation, and graphics.

Accessibility: The model should be accessible in two senses. First, it must be easily perused and altered. Second, it must be perspicuous. That is, the contents of the model must be easily and unambiguously understood by the user.

Extensibility: Organizations change, and so does the way we think about them. A modeling system which does not adapt to these changing views will restrict the ways in which we may analyze it. Ultimately, the level of intelligent behaviour a system will display may be limited by the model. Hence, the model must be extendible, incorporating new ways of viewing and describing the organization.

In this chapter, the ISIS modeling system is described. The modeling system is composed of multiple layers, with the core being an AI knowledge representation system called SRL: Schema Representation Language (Fox, 1979; Wright & Fox, 1983). It provides the structural primitives in which the domain's conceptual entities are defined. With these SRL structures, a set of conceptual primitives which is basic to many domains is defined. On top of this, concepts germaine to scheduling are defined, and these are instantiated as a model for a particular plant. The key feature of SRL which allows this layered approach is the user-definable relations. High level, domain dependent relations may be constructed from low level, domain independent primitives. The efficacy of this approach will become clear in the example of this chapter.

Throughout this chapter, an example of an order and a couple of operations required to produce it will be used to explicate the modeling concepts. In particular, the sequence of a milling and drilling operation will be described as a method for producing a turbine blade. An order is a specification of a product to be produced for some customer. It describes parameters such as due date, cost and quality to be satisfied by the shop. An operation is an activity in which resources may be transformed, through one or more actions, into new forms of resources. In deciding which operations to perform and when, the following types of knowledge are required:

- The range of durations of the operation, including a probability density function.
- The operations which may precede or follow the current operation.
- The resources required: materials, machines, tools, fixtures, software, etc.
- The period of time during which the above resources are required.
- The transformations applied to the resources. For example, is the cutting fluid on a milling machine totally consumed?
- Are there any constraints on the usage of the resources?
- Who may perform the operation (i.e., operator).
- Substitutability of resources. If a machine is not available, can another be used?
- A description of how the operations are performed. What are the components, i.e., suboperations, comprising the operation.

The purpose of the ISIS modeling language is to represent much of the information described above.

3.2. Layer 1: Structure

The ISIS modeling system is based on the knowledge representation system SRL: Schema Representation Language (Fox, 1979; Wright & Fox, 1983). SRL has its basis in schemata (Bartlett, 1932), which have come to been known as frames (Minsky, 1975), Concepts (Lenat, 1976), and Units (Bobrow & Winograd, 1977; Stefik, 1979).

3.2.1. Schema Syntax

The basic unit for representing objects, processes, ideas, etc. is the **Schema**. Physically, a schema is composed of a schema name (printed in the bold font) and a set of slots (printed in small caps). A schema is always enclosed by double braces with the schema name appearing at the top.

```
{{ operation
    NEXT-OPERATION:
    PREVIOUS-OPERATION:
    MACHINE:
    OPERATOR:
    DURATION:  }}
```

Figure 3-1: operation Schema

The **operation** schema (figure 3-1) contains slots defining attributes of the schema such as, next-operation, duration, and operator.

SRL provides the user with a standard set of relations for defining classes of concepts and their instantiations. For example, a **milling-operation** (figure 3-2) can be defined as an instance of **machining-operation**, and the **machining-operation** is a sub-class (i.e., is-a) of **operation**:

```
{{ machining-operation
    IS-A: operation   }}

{{ milling-operation
    creator: Mark.Fox
    { INSTANCE machining-operation
        NEXT-OPERATION: drilling-operation
                              creation-date: 18-jan-83
        MACHINE: excello   }    }}
```

Figure 3-2: milling-operation Schema

3.2.2. Meta-Information

Meta-information may be attached to any part of a schema. It provides the user with a means of documenting the information in a schema, and also for defining the semantics of schema, slots, and values. In figure 3-2 slots in *italics* are meta-information attached to the schema, slot, or value depending on their indentation. In this example, the *creator* of the schema is "Mark.Fox", and the *creation-date* of the value in the NEXT-OPERATION slot is 18-jan-83.

3.2.3. Inheritance

Schemata provide a "case" or "frame" approach to knowledge representation. Encapsulated in a schema is the information which defines the concept embodied by the schema and meta-information about it. In a schema, slots play a dual role. They define the concept, and *relate* it to other schemata in the knowledge base. As a relation, a slot allows information to flow between schemata. In the above example, the **instance** relation allowed information to flow from **machining-operation** to **milling-operation**.

Both **is-a** and **instance** are system-defined relations provided to the user. SRL provides the user not only with the ability to define schemata, but also the ability to define new relations and their inheritance semantics. All relations in SRL are defined as subtypes of the **relation** schema.

```
{{ relation
    DOMAIN: "schemata in the domain of the relation"
    RANGE: "schemata in the range of the relation"
    INVERSE: "inverse of the relation"
    PATH: "transitivity definition of relation"

    INCLUSION: "slots & values which may be passed from the range to domain"
    EXCLUSION: "slots & values which may NOT be passed from the range to domain"
    ELABORATION: "specialization of slots in range into new slots in the domain"
    MAP: "functional mapping of slots in range onto the domain"
    INTRODUCTION: "definition of new slots in the domain"    }}
```

Figure 3-3: Relation Schema

To define a new relation, the user simply creates a schema of type **relation** and fills its slots with the appropriate information. This is defined further in (Fox, 1979; Wright & Fox, 1983).

In figure 3-2, the occurrence of single brace brackets denotes the *opening* of a slot. In this case, the INSTANCE slot with the value **machining-operation** is opened. The NEXT-OPERATION and MACHINE slots are inherited along the INSTANCE relation from **machining-operation** and have their values defined relative to this perspective, i.e., being an operation. Slot openings distinguish among the versions of the same slot being inherited along more than one relation, where each perspective having a different value.

3.3. Level 2: Basic Semantics

The semantics of the ISIS modeling systems is defined using a layered approach (Brachman, 1982). Level 1 defined the structure of schemata and inheritance semantics. Level 2 expands the semantics to those more epistemological in nature.

3.3.1. Classes, Sets and Instances

Classification hierarchies are found throughout manufacturing organizations. For example, Group Technology classification hierarchies (Opitz, 1969) are used to classify products and the operations that produce them. In order to differentiate among classes of objects, sets of objects, and objects which exist physically, the concepts of intension, extension and sets are adopted.

Schemata are divided into three types: prototype, instance, and set. A "prototype" schema provides a description of a prototypical member of a set. An "instance" schema defines a member of the set defined by the prototype. A "set" schema defines the set of instances defined by a prototype. The schema type is specified in the meta-schema attached to the schema, in the TYPE slot. Hence, the **operation** schema is further defined as:

{{ **operation**
    type: prototype }}

Figure 3-4: Prototype **operation** Schema

(the *type* slot actually resides in **operation**'s meta-schema.)

Class hierarchies are defined by relating prototypes with an **is-a** relation.

{{ **machining-operation**
    type: prototype
    IS-A: operation }}

Figure 3-5: Prototype **machining-operation** Schema

And an individual member of a class is related to its class by an **instance** relation.

```
{{ milling-operation
    type: instance
    INSTANCE: machining-operation  }}
```

Figure 3-6: Instance of a **milling-operation**

Sets of objects are defined by a schema of type set. For example, a set of turbine blades can be defines as:

```
{{ turbine-blade
    type: prototype
    IS-A: object      }}
```

Figure 3-7: Prototype **turbine-blade** Schema

```
{{ turbine-blades
    type: set
    PROTOTYPE: turbine-blade
    MEMBER: TS831 CS495 }}
```

Figure 3-8: Set of **turbine-blades**

The PROTOTYPE slot (inverse: prototype-of) contains the name of the prototype schema(ta) defining the characteristics of the set's members. The actual members of the set are contained in the MEMBER slot (inverse: member-of).

3.3.2. Attributes, Parts and Structure

The primary goal of ISIS is to choose and schedule the application of shop resources for the production of an order. To achieve this the modeling system must provide ISIS with the ability to represent the relevant information about a resource. The description of resources should provide at least two types of information:

1. A physical description of the resource. This provides the prototypical description of an resource which is invariant over time. This information is used to determine if the resource can be used in a particular operation. For example, a machine may be too small to produce a large turbine blade.

2. A state description of the resource. This provides a time dependent description of the resource. For example, a description of its availability. Much of the resources in a shop are not consumable and exist in small numbers. Their allocation must be known to those who may want to use them.

The physical description of an object contains many types of information such as attributes (e.g., mass, material type, color), part descriptions (i.e., physically composition), and structure (i.e., where the parts are located). These are defined as slots in a schema. Each type of information is formally distinguished by making each slot a one of the following slot types:

attribute/attribute-of: It defines an attribute of a schema that is true of the schema as a whole. Concepts of mass and color are examples of this.

has-part/part-of: It specifies that the value of the slot is a constituent of the definition of the schema. For example, a left-leg is part of a human, or B-52 bombers are part of the nuclear defense triad.

structure/sub-structure-of: It specifies a structuring of the slots (relations) that are of type part. For example, the right-leg is *right-of* the left-leg.

Consider the description of a **milling-machine**. It may specify a capacity (e.g., number of orders it may work on at one time), the cost-center it is part of, and a description of its workbed.

```
{{ milling-machine
    CAPACITY: 1
        type: attribute
    COST-CENTER: milling-center
        type: part-of
    WORD-BED: milling-bed
        type: has-part
    STATUS:
        type: attribute    }}
```

Figure 3-9: **milling-machine** Schema with attributes

```
{{ milling-bed
    IS-A: work-bed
    WORK-BED-OF: milling-machine
        type: part-of
    LENGTH: 5.ft
        type: attribute    }}
```

Figure 3-10: **milling-bed** Schema

The **milling-bed** and **milling-machine** are related by the work-bed/work-bed-of relations, and **milling-bed** has a length of 5 feet.

These primitives provide a relational approach to the attribute and structural description of objects. This suffices for ISIS's planning and scheduling needs. Other functions such as generative process planning, or graphics display may require other representations (e.g., oct-tree). These should be integratable with the SRL representation.

3.3.3. Abstraction

Reasoning often occurs at many levels of abstraction. Simon (1962) recognized that the complexity of problem solving could be reduced by reasoning at different levels of abstraction. Sacerdoti (1974) constructed a system which performs exactly that. In order to facilitate such reasoning, a **refines/refined-by** relation is used to denote that a set of schemata refine the concept represented by another schema.

For example a **milling-operation** may be refined into **milling-setup** and **milling-run.**

```
{{ milling-operation
    INSTANCE: machining-operation
    REFINED-BY: milling-setup milling-run    }}
```

Figure 3-11: Abstract **milling-operation** Schema

```
{{ milling-setup
    INSTANCE: machining-operation
    REFINES: milling-operation }}
```

Figure 3-12: **milling-setup** refines **milling-operation** Schema

```
{{ milling-run
    INSTANCE: machining-operation
    REFINES: milling-operation }}
```

Figure 3-13: **milling-run** refines **milling-operation** Schema

---

3.4. Layer 3: World Model

The goal of the world model level is to define a set of primitive concepts upon which the concepts found in the scheduling domain may be defined. The approach taken in ISIS is to define the production environment as a world where actions are performed which transform the world from its current state into a new state. The actions are operations, the states are descriptions of products and assignments of resources. Included in this world model are primitives for modeling time and causality, parallelism and repetition of acts.

3.4.1. State/Acts

ISIS has as its most basic schema classes, an **object**, **act**, and **state**. Informally, an **object** is a physical entity, an **act** transforms one **state** or **object** into another, and a **state** is a compound description of **states** and manifestations of **objects**. Hence, an **operation** can be further defined as a sub-class of **act**:

```
{{ operation
     creator: "Mark Fox"
     type: prototype

     IS-A: act  }}
```

Figure 3-14: operation Schema

3.4.2. Time

An important part of the modeling of organizations is the representation of time. Consider the representation of a set of activities. Time is described in two ways. The first is an absolute specification of when an activity is to begin and for how long. The second is a relative specification in which an activity is to precede or follow another activity. Duration is a *rational* conceptualization of time, and precedence is an *ordinal* conceptualization of time.

A relative specification of time is used most often in the description of prototypical concepts. For example, the operation graph for producing a product defines precedence and duration, but does not specify an absolute time; this representation documents the production process, and is used to guide scheduling. On the other hand, schedules specify actual times for operations. Hence, the representation and reasoning about time must be supported at each of these levels by the representation.

The concept of time in artificial reasoning systems has been explored (Bruce, 1972; Kahn & Gorry, 1977; McDermott, 1982). The approach taken in ISIS is to represent time as intervals and relations amongst intervals (Allen, 1981; Smith, 1983). The time that a state or act may occur is defined by a time interval as opposed to a single point, or by reference to known time intervals.

The first step towards the representation of time is to specify the units of time, a scale and the functions to manipulate time. This is defined by the **time-line** schema (Smith, 1983).

```
{{ time-line
     POINT-FORM: "specifies the form of a point of time"
     START-POINT: "starting point of time line"
     END-POINT: "end point of time line"
     GRANULARITY: "granularity of time line"
     ADD: "function for adding two time points"
     DIFF: "function for subtracting two time points"  }}
```

Figure 3-15: time-line Schema

A time interval is defined by a schema as having a start time, and end time and a duration:

```
{{ time-interval
     BEGIN-TIME:
     END-TIME:
     DURATION:
     DATED-BY:  }}
```

Figure 3-16: time-interval schema

Given values for two of the three slots, the third may be derived. Hence absolute descriptions of when an act may occur is provided by specifying the begin and end times. If only the duration of an act is known, this is specified by filling only the DURATION slot. The DATED-BY slot points to the time line in which the interval is defined.

Ordinal (relational) specification of time is represented by slots (relations) in a schema. If act A is to occur before act B, then a BEFORE slot will exist in A with a value of B. The following time relations are defined in ISIS:

before (figure 3-17). Specifies that an act or state takes place before another act or state in time.

```
{{ before
     { IS-A time-relation
          DOMAIN:
               range: (or (type is-a act) (type is-a state))
          RANGE:
               range: (or (type is-a act) (type is-a state))
          INVERSE: after    }
     TIME: before      }}
```

Figure 3-17: Before Schema

The DOMAIN of **before** is a subtype of either **act** or **state**. The RANGE is a subtype of either **act** or **state**. The inverse of **before** is the **after** relation.

during. Specifies that an act or state takes place during another act or state in time. Its inverse is **includes**.

meet. Specifies that an act or state takes place before, but without any intervening time, another act or state in time. Its inverse is **met-by**.

overlap. Specifies that an act or state begins before another act or state in time, but ends after the second begins and before it ends. Its inverse is **overlapped-by**.

time-equal Specifies that an act or state shares the same time interval with another act or state in time. Its inverse is **time-equal**.

The representation of the **milling-operation**'s duration can be defined as follows:

```
{{ milling-operation
     INSTANCE: machining-operation
     WORK-CENTER: milling-center
     DURATION: {{ INSTANCE time-interval
                      DURATION:  5    }}   }}
```

Figure 3-18: **milling-operation** Duration Specification

The contents of the DURATION slot is an instance of a **time-interval** whose duration is 5 hours (time units are defined in the time-line schema).

Operation precedence may be defined by means of the time relations. In this example **milling-operation** is to occur before the **drilling-operation**.

```
{{ milling-operation
     INSTANCE: machining-operation
     WORK-CENTER: milling-center
     DURATION: {{ INSTANCE time-interval
                      DURATION:  5    }}
     BEFORE: drilling-operation }}
```

Figure 3-19: **milling-operation** Precedence Specification

And the sub-operations which comprise the **milling-operation** and are performed during the same time period are defined by an INCLUDES relation. Hence, **milling-operation** is refined-by **milling-setup** and **milling-run**, and they are included in time.

```
{{ milling-operation
     INSTANCE: machining-operation
     WORK-CENTER: milling-center
     DURATION: {{ INSTANCE time-interval
                      DURATION:  5    }}
     BEFORE: drilling-operation
     REFINED-BY: milling-setup milling-run
     INCLUDES: milling-setup milling-run     }}
```

Figure 3-20: **milling-operation** sub-operation time specification

The definition of **milling-setup** and **milling-run**, in turn, have their own time relations which define their time-oriented sequencing.

3.4.3. Causality

The concept of precedence in the sequencing of operations encodes another primitive concept, that of causality. The difference between time and causality in this case is subtle, but important. Not only does the milling operation precede the drilling operation in time, but it *must* be performed *before* the drilling operation. If the milling operation is not performed, then the drilling operation cannot be performed; performance of the former enables the performance of the latter. If only the time relation existed, then the drilling operation could be performed even if the milling operation was never performed; non-performance of the operation is not inconsistent with the time relation. The approach taken in ISIS in representing causal information is a derivative of the common sense reasoning system defined by Rieger and Grinberg (1977) and London (1978).

Causality is represented as relations in SRL. The approach differs from that of Rieger & Grinberg in that time may be specified separately. On the other hand, causality implies a time ordering which is represented in the causal relation as a range restriction on the time slot.

The following are the five basic causal relations:

```
{{ cause
   { IS-A time-relation
       DOMAIN:
           range: (type is-a act)
       RANGE:
           range: (type is-a state)
       INVERSE: caused-by
       TIME:
           range: (or before meet overlap includes)  }
   CAUSALITY: cause
   CAUSAL-CONDITION:
       default: t }}
```

Figure 3-21: cause Schema

The **cause** relation (figure 3-21) links an **act** to a **state**. It defines that the execution of a specified act results in the specified state. The CAUSAL-CONDITION slot is filled by a condition schema which must be true in order for the cause to take effect (i.e., the state to follow from the act). The causal condition is, in essence, a gating condition on the interpretation of the relation. Note that time is included in the definition of cause, hence the time of the cause may be included in its specification. The inverse of **cause** is **caused-by**.

An **enable** relation is the reverse of the **cause** relation. It links a state to an act. Its inverse is **enabled-by**. Time is restricted to (or before meet overlap includes).

A third type of causal relation is a **statecouple**. It defines two states as being related, but the actual nature of the relation is unknown. Its inverse is **statecouple + inv**. Time is restricted to (or overlap includes time-equal).

The fourth type of causal relation is **cause-equal**. It states that two acts, or states may be equivalent. Its inverse is **cause-equal**. Time is restricted to time-equal.

The last type of causal relation is an **actcouple**[3]. It is similar to a **state-couple**, but links **acts**. Its inverse is **actcouple + inv**. Time is restricted to (or overlap includes time-equal).

The complete definition of precedence can now be represented:

```
{{ milling-operation
    INSTANCE: operation
    WORK-CENTER: milling-center
    DURATION: {{ INSTANCE time-interval
                 DURATION: 5    }}
    BEFORE: drilling-operation
    ACTCOUPLE: drilling-operation
    REFINED-BY: milling-setup milling-run
    INCLUDES: milling-setup milling-run      }}
```

Figure 3-22: milling-operation Causal Specification

Hence, two relations (slots) are used to define precedence. An alternative and more compact approach is to take advantage of SRL's relational capabilities. The user may define for their application new relations beyond the simple **is-a** and **instance** relations. For example, a **next-operation** schema may be defined which incorporates the time and causality information:

```
{{ next-operation
    { IS-A relation
        TIME: before
        CAUSALITY: actcouple    }    }}
```

Figure 3-23: next-operation Schema

And **milling-operation** may be defined as follows:

[3]Equivalent to action thread (London, 1978)

```
{{ milling-operation
    INSTANCE: operation
    WORK-CENTER: milling-center
    DURATION: {{ INSTANCE time-interval
                DURATION: 5    }}
    NEXT-OPERATION: drilling-operation
    REFINED-BY: milling-setup milling-run
    INCLUDES: milling-setup milling-run      }}
```

Figure 3-24: milling-operation next-operation Specification

3.4.4. States Revisted: Possession and Manifestations

Basic to the representation of an operation is the specification of resources that are used during its execution. These resources must be available *before* the operation may proceed. For example, tools, materials, fixtures, and even the operator must be present before the operation may be performed. The time it takes to assemble these resources is normally included in the setup definition of the operation. Conceptually, a state of the organization's world must exist in which the work center where the operation is to be performed possesses the resources before, and possibly during the operation. Hence, this state must exist **before** (in time) the operation is performed, and its existence **enables** the execution of the operation.

The concept of possession of resources is represented as a simple state.

```
{{ possess
    IS-A: state
    POSSESSOR:
    POSSESSION: }}
```

Figure 3-25: possess Schema

If the milling operation requires a wrench during its execution, it would be ENABLED-BY the possession of the wrench:

```
{{ milling-operation
     INSTANCE: operation
     WORK-CENTER: milling-center
     DURATION: {{ INSTANCE time-interval
                      DURATION: 5     }}
     NEXT-OPERATION: drilling-operation
     REFINED-BY: milling-setup milling-run
     INCLUDES: milling-setup milling-run
     ENABLED-BY: possess-wrench       }}
```

Figure 3-26: milling-operation next-operation Specification

where the possession of a wrench is defined as:

```
{{ possess-wrench
     { INSTANCE possess
          POSSESSOR: milling-center
          POSSESSION: wrench }
     INCLUDES: milling-operation }}
```

Figure 3-27: possess-wrench Schema

possess-wrench not only specifies that the milling center must possess a wrench, but it also specifies the time of its possession. That is, the time interval must span the time of the **milling-operation**. If this state specification is satisfied, then the state exists, and it enables the performance of the **milling-operation.**

A simple state description may also include the description of a resource which is true during some time period. In ISIS, this is viewed as a state dependent description and is represented by creating a schema which defines the time dependent information about a resource and linking it via a **manifestation** relation[4] to the instance. A manifestation of an object or resource denotes a description which is true for some specified time interval. Figure 3-28 depicts a manifestation of the **milling-machine** with a status of busy over the time interval defined by the week-38 schema.

[4]The concept of manifestations was elaborated by Hayes (1973).

```
{{ milling-machine-man-1
    { MANIFESTATION milling-machine
        STATUS: busy
        TIME-EQUAL: week-38 }    }}
```

Figure 3-28: Manifestation of **milling-machine**

Manifestations of resources, and objects in general, are considered to be state descriptions.

3.4.5. Composite States and Parallelism

Operations, and acts in general, may occur in parallel. While the representation of enablement and causality can specify more than one subsequent state or act, its interpretation is ambiguous; it is unclear whether all, some or one act is to occur. The representation must be extended to represent not only what may occur next, but the conditions of their joint occurrence.

The specification of parallelism in the modeling system is accomplished by introducing composite states. A composite state is constructed out of simple state descriptions. Three types of composite state schemata are defined:

and: specifies that all of its sub-states must be true in order for the composite to be true.

or: specifies that one or more of its sub-states must be true in order for the composite to be true.

xor: specifies that only one sub-state is to be true in order for its composite to be true.

The **and** state is represented schematically as:

```
{{ and
    IS-A: state
    SUB-STATE:
        type: has-part
        range: (set (type is-a state))    }}
```

Figure 3-29: and schema

or and **xor** are defined similarily.

Composite states may be created using these state schemata. For example, the requirement of a wrench and an operator is defined as follows:

```
{{ enable-milling
     { INSTANCE and
          SUB-STATE: possess-wrench possess-operator }     }}
```

Figure 3-30: enable-milling Schema

```
{{ possess-operator
     { INSTANCE possess
          POSSESSOR: milling-center
          POSSESSION: milling-operator      }
     OVERLAP: milling-operation }}
```

Figure 3-31: possess-operator Schema

Note that the **possess-operator** state is a SUB-STATE of the **enable-milling** state. The time over which the milling center must possess an operator overlaps the performance of the operation. Hence, the operator is not required throughout the operation.

If a sub-state of a composite state does not have a time relation associated with it, the time relations associated with its enclosing state descriptions are inherited along the **sub-state** relation. That is, a more complete definition of **possess-wrench** is:

```
{{ possess-wrench
     SUB-STATE-OF: enable-milling
     { INSTANCE possess
          POSSESSOR: milling-center
          POSSESSION: wrench }
     OVERLAP: milling-operation }}
```

Figure 3-32: possess-wrench Schema

Where the **sub-state-of** relation is defined by:

```
{{ sub-state-of
    { IS-A relation
        DOMAIN:
            range: (type is-a state)
        RANGE:
            range: (type is-a state)
        INVERSE: sub-state
        INCLUSION: {{ INSTANCE inclusion-spec
                            SLOT: (type is-a time-relation)   }}
    }   }}
```

Figure 3-33: **sub-state-of** Relation Schema

This definition of **sub-state-of** defines that any slot that **is-a** a **time-relation**, may have its values inherited along it.

3.4.6. Repetition

Rieger and Grinberg (1976) recognized that the concept of causality must include the concepts of discreteness or continuity in both state and act. Their approach to this problem was to elaborate a variety of "continuous" and "one shot" causal links. In our approach, the separation of time from causality covers much of these concepts. For example, if a state must persist over the time of the act, then it will be related by a **time-equal** or **includes** relation.

Another issue is the extent to which a state or act may be repeated. For example, an operations graph may show a re-work cycle when the product is defective. It is commonly the case, that the amount of re-work which may be performed is limited, hence the number of times an act may be repeated is constrained.

Limitations on the number of times an act may occur is specified indirectly by state descriptions. The definition of a state is extended to include an upper bound on the number of times it may occur (see section 3.7 for the description of the procedural interpretation of repetitive states).

```
{{ state:
    MAXIMUM-MANIFESTATION:   }}
```

The **maximum-manifestation** slot specifies the number of occurrences of the state. Note that the term manifestation is used. The prediction or appearance of a state in time is linked to the original state description via a **manifestation** relation. Hence, the operations graph is defined in terms of intances of operations and states, but predicted (reservations) or actual (history) occurrences of states and acts are manifestations of the instances. See section 3.6 for further details.

3.4.7. Time Revisited: The Future and the Past

The purpose of job shop scheduling is to select a sequence of operations to produce a product. For each operation, a set of resources are assigned and a time of execution is specified. In other words, scheduling specifies, or more realistically, predicts a sequence of states that should occur in the plant's future. It follows then, that the representation of states should include the ability of specifying when the state will occur, or be "true".

Conversely, the ISIS modeling system must also record historical information. That is, records of past performance of the shop in order to ascertain how well schedules are met, determine time standards, etc. Hence, the modeling system must also be able to specify states of the system which existed in the past.

Fortunately, the method of specifying time is general enough to support both. Each state may have specified a time interval over which the state was known or is predicted to exist. In order to determine whether a state description refers to the past or future, one must simply compare the time-interval to where "now" is specified on the corresponding time line.

In order to represent the prediction of a state in which an act occurs, the use of manifestations is extended to acts. *A manifestation of an act is a state which describes the act as having taken place or is predicted to take place.*

3.4.8. Scheduling Goals

The first step in creating a system to perform planning and scheduling is the definition of goals. The meta-goal of ISIS is to schedule the shop. This is achieved through the satisfaction of the subgoals of scheduling each order. Each **order** in ISIS is a **goal** state. That is, it is a state which is to be achieved by ISIS. An **order** specifies parameters which are to be satisfied by ISIS, such as the product to be produced, quantity, delivery date, cost and customer.

```
{{ goal
    IS-A: state }}
```

Figure 3-34: goal Schema

3.5. Layer 4: Domain Semantics

A major goal in our use of SRL for plant modeling is to construct a language which closely resembles that used by shop personnel. If the model is able to represent concepts in the same language as used by shop personnel then the hypothesis is that it should be easier for them to create and maintain such models. On the other hand, if ISIS is to be usable across multiple plants which have different terminology, then its representation should be as plant independent as possible. On the surface, these two goals appear to be contradictory. In reality they are not. By taking a layered approach in which plant dependent terminology is defined in terms of plant independent terminology, it is possible for ISIS to interpret a plant's model.

Level three defines the plant independent terminology. Level four defines the plant dependent terminology in terms of level one through three concepts. In the rest of this section, examples of scheduling terminology are defined.

3.5.1. Operations

As described earlier, the concept of an operation is basic to scheduling. It can be defined in terms of an act:

```
{{ operation
    type: prototype
    IS-A act
    WORK-CENTER:
        type: attribute  }}
```

Figure 3-35: operation Schema Re-visited

The **operation** schema is a prototype with a single attribute WORK-CENTER. As seen in figure 3-2, a **milling-operation** can be defined as an instance of an **machining-operation,** which is a sub-class of **operation**. As shown earlier, an operation may be related to other acts and states by means of the time and causal relations (see figure 3-26).

Conceptually, **milling-operation** is the abstraction of a more detailed set of operations to be performed. In this case, they are **setup-milling** and **run-milling**. This is equivalent to a "stepwise refinement" approach to the definition of acts (Wirth, 1971). The **refines** relation provides the user with the ability to implement models at various levels of detail. Hence, the **milling-operation** may be defined as:

```
{{ milling-operation
     INSTANCE: operation
     WORK-CENTER: milling-center
     DURATION: {{ INSTANCE time-interval
                    DURATION:  5     }}
     NEXT-OPERATION: drilling-operation
     INCLUDES: milling-setup milling-run
     REFINED-BY: milling-setup milling-run
     ENABLED-BY: enable-milling        }}
```

Figure 3-36: milling-operation refined-by Specification

This defines the **milling-setup** and **milling-run** operations as being refinements of the **milling-operation**, and being performed during the same time-interval. These two relations may be combined into a single **sub-operation** relation:

```
{{ sub-operation
     { IS-A refined-by
          DOMAIN:
               range: (type is-a operation)
          RANGE:
               range: (set (type is-a operation)) }
     TIME: includes     }}
```

Figure 3-37: sub-operation Schema

Allowing the milling operation to be equivalently defined by substituting SUB-OPERATION for REFINED-BY and INCLUDES:

```
{{ milling-operation
     INSTANCE: operation
     WORK-CENTER: milling-center
     DURATION: {{ INSTANCE time-interval
                    DURATION:  5     }}
     NEXT-OPERATION: drilling-operation
     SUB-OPERATION: milling-setup milling-run
     ENABLED-BY: enable-milling        }}
```

Figure 3-38: milling-operation sub-operation Specification

In the discussion of composite states, the possession of a wrench was defined as overlapping the time interval of the milling operation. At the milling operation level of abstraction, that was all that is known and could be explicitly defined. But with the definition of the milling operation's sub-operations, the exact time of possession of the wrench can be defined more specifically. That is, the wrench is required only during the milling-setup operation:

```
{{ possess-wrench
     SUB-STATE-OF: enable-milling
     { INSTANCE possess
          POSSESSOR: milling-center
          POSSESSION: wrench
     OVERLAP: milling-operation
     INCLUDES: milling-setup  }     }}
```

Figure 3-39: **possess-wrench** Schema

Note that the **possess-wrench** schema now has time relations linking the possession of the wrench at two levels of abstraction. It is obvious that the overlap relation may be deduced from the includes relation.

Figure 3-40 graphically depicts a sub-set of an operations graph.

3.5.2. Reservations

An important attribute of resources for scheduling purposes is the specification of their allocation to scheduled operations. This is accomplished by adding a RESERVATION slot (i.e., the inverse of possesion in **posses** schema) to an object and filling it with a manifestation of a **possess** schema. Hence, the full effect of creating a manifestation of a **possess-wrench** schema is to add to the POSSESSION and POSSESSOR schemata the name of the manifestation:

```
{{ milling-center
     INSTANCE: work-center
     POSSESSES: m-possess-wrench }}

{{ wrench
     INSTANCE: tool
     RESERVATION: m-possess-wrench     }}
```

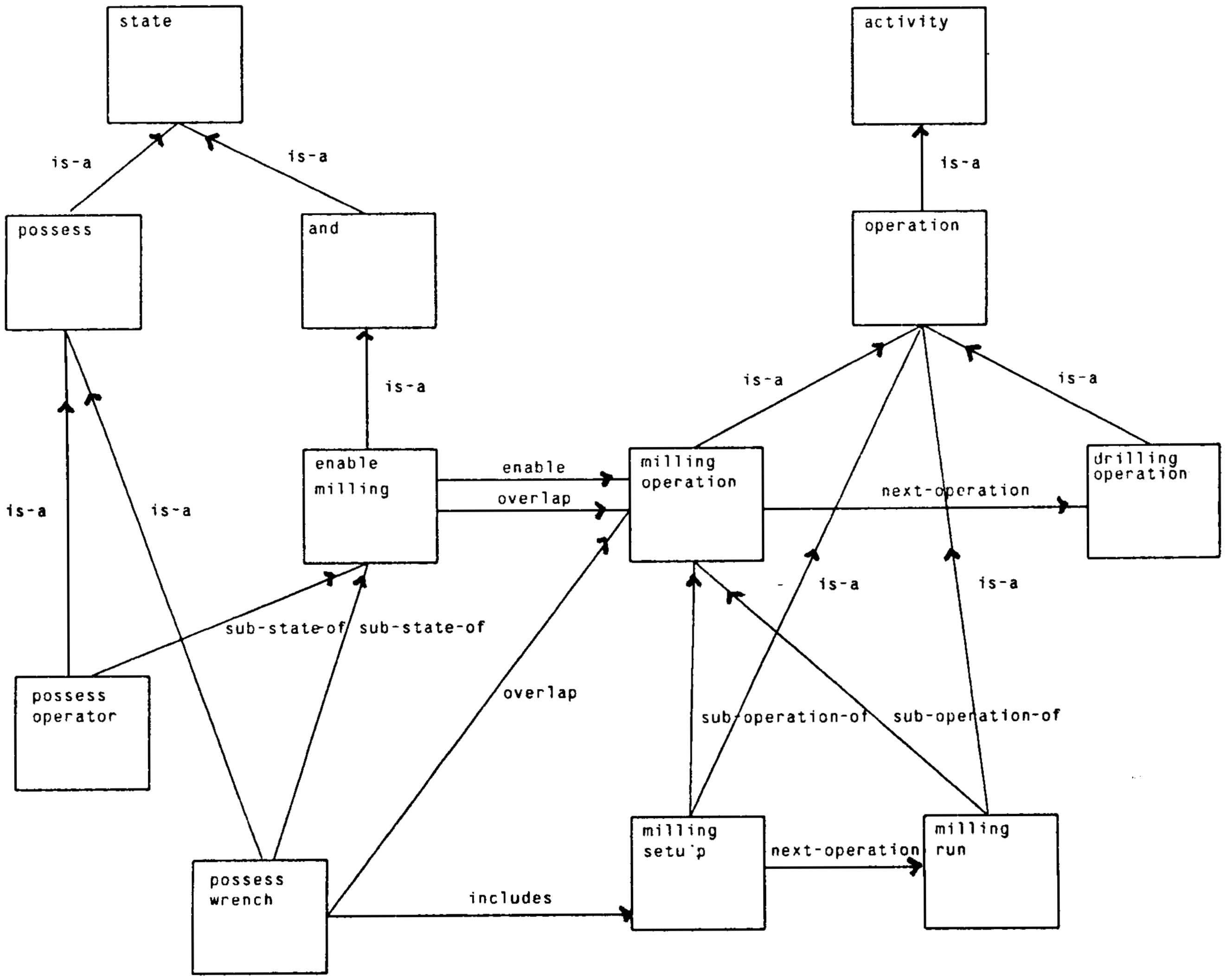


Figure 3-40: Operations Graph

3.5.3. Products

A product is yet another type of object. One additional slot is added, a pointer to the first operation to produce it.

{{ **product**
  IS-A: object
  FIRST-OPERATION: }}

Figure 3-41: **product** Schema

Hence, a turbine blade is a product with the first operation being the **milling-operation**.

```
{{ turbine-blade
    { IS-A product
        FIRST-OPERATION: milling-operation }    }}
```

Figure 3-42: **turbine-blade** Schema

3.5.4. Orders

An order is a sub-type of the **shipped** state which defines the PRODUCT, QUANTITY, CUSTOMER, and SHIP-DATE. An **order** for a **turbine-blade** would have the following structure:

```
{{ shipped
    IS-A: goal
    PRODUCT:
    QUANTITY:
    SHIP-DATE:
    CUSTOMER:  }}
```

Figure 3-43: **shipped** Schema

```
{{ order
    IS-A: shipped     }}
```

Figure 3-44: **order** Schema

```
{{ turbine-blade-order
    { INSTANCE order
        PRODUCT: turbine-blade
        SHIP-DATE: 28-feb-83
        QUANTITY: 128
        CUSTOMER: ontario-hydro
        FIRST-OPERATION:  }    }}
```

Figure 3-45: **turbine-blade-order** Schema

3.6. Schedule

A product is defined to be an object which may be produced by interpreting the operations graph attached to it. The operations graph is composed of instances of states and acts. The question then is how is a schedule defined if instances cannot themselves have instances? As described earlier, states, objects, and acts may have manifestations. A manifestation defines a state during some time period which has or will occur. The view implemented in the modeling system is that actual schedules are manifestations of the states and acts found in the product's operations graph; where product points to instances of states and acts, an order points to manifestations of those instances. For example, the schedule for the drilling and milling operations would be composed of manifestations:

```
{{ turbine-blade-order
    { INSTANCE order
        FIRST-OPERATION: m-milling-operation      }    }}

{{ m-milling-operation
    { MANIFESTATION milling-operation
        NEXT-OPERATION: m-drilling-operation
        ENABLED-BY: m-enable-milling      }    }}

{{ m-enable-milling
    { MANIFESTATION enable-milling
        SUB-STATE: m-possess-wrench m-possess-operator    }    }}

{{ m-possess-wrench
    { MANIFESTATION possess-wrench
        POSSESSION: wrench
        POSSESSOR: milling-center  }    }}
```

3.7. Model Interpretation/Simulation

Another goal of our modeling effort is to construct a language which can support multiple functions. Scheduling is just one of the many functions found in manufacturing organizations. Of late, there is increasing use of simulation systems in analyzing current and future modes of production. Given the depth of information contained in the ISIS model, it should be possible to write an interpreter which can simulate the multitude of activities described.

The ISIS modeling system can support discrete event simulation. The Knowledge-Based Simulation (KBS), system is an SRL interpreter for discrete event simulation (Reddy & Fox, 1982). It performs simulations based on event rules embedded in schemata. Hence, the semantics of an event are defined by rules in event slots in a schema. KBS does not interpret schema nor relations types directly, but relies on event rules in schemata to define their behavior. The rationale for this approach was to keep KBS independent of the domain semantics.

An alternative approach to providing discrete event simulation is to build an interpreter which understands level three semantics. The interpretation of the ISIS model is similar to the token passing method employed in Petri nets (Petri, 1966). The interpretation proceeds as follows:

1. The simulation clock is set to its initial time.

2. Tokens are placed at all initial (true) states.

3. If the state at which the token is located is a composite state, then copies of the token are placed at the constituent states if it is an **and** and all the constituents are true, at only the true states if an **or**, or a single true state if an **xor**. A token may wait at either of these composite states if there do not exist the appropriate set of true constituents.

4. Tokens situated at states which are either state-coupled with another state, or enable an act are passed from the current state to the related state or act.

5. For all acts to which a token has just arrived, the exit time of the token is calculated from the CALCULATE-DURATION slot in the act. The exit time of each token is placed on the simulation calendar. Any action associated with the act is executed.

6. Current simulation time is set to the time of the earliest calendar event (token exit from an act).

7. The act associated with the earliest calendar event has its token transferred along a cause link to an act or state respectively (act-couple links are ignored due to the ambiguity they introduce in interpretation).

8. Go to step 3.

The above steps sketch how the ISIS model may be interpreted to support simulation. Much detail is left out, e.g., how to calculate duration times, due to the focus of this chapter.

3.8. Conclusion

The modeling system described in this chapter does not represent a complete theory of organization modeling, but is a synthesis of many basic concepts found in artificial intelligence and operations management which are necessary to support the scheduling process in factories. It has been shown by example that the primary concepts necessary to support scheduling can be easily represented. The next question is whether the system also supports the criteria mentioned in the introduction:

Generality: SRL provides the user with the ability to mold the semantics of the language to their individual needs. It does not force a particular perspective on the user. For the ISIS application, as set of conceptual primitives was chosen to be domain independent. The independence or universality of these concepts support a variety of applications in addition to scheduling.

Accessibility: The set of conceptual primitives was also chosen to be understandable. If the user understands the primitives then he should understand the model. Using the relation definition capabilities of SRL, the user may construct slots whose names match those used in the domain (e.g., next-operation), but whose semantics are defined by these primitives. Perusal and alteration capabilities are already provided through SRL.

Extensibility: The availability of the representation of meta-information and user-definable relations enables the user to extend the language in any direction they wish. The layered approach taken by the modeling system shows how the system can be extended in the appropriate directions.

We expect that the ISIS modeling system will continue to evolve as more experience is acquired in the factory.

Chapter 4
Constraints

Summary

The ISIS modeling system provides primitives for describing the common actions and states in the scheduling domain. Experience has shown that a major part of our knowledge about a plant is an understanding of constraints on the plant's operation, and alternatives to perform when the constraints cannot be met. The modeling system is extended by providing the capability to attach constraints to a schema, its slots, and values. Hence, any concept representable in SRL, can also have constraints attached to it.

In scheduling, constraints may not always be satisfiable. Hence the representation of alternatives is important. The modeling system adds to the representation of constraints the specification of *relaxations*. Relaxations may be defined either as predicates, or choice sets which can be discrete or continuous. Associated with a relaxation is a preference measure which determines the preferred relaxations among those available. The representation of constraints and their *preferred* relaxations is also a solution to the pattern specification and matching problem.

The representation of constraints must not only cover what the constraint is, but when and how to use it. The first problem in using constraints focuses on their relative importance. Depending on the order, some constraints are more important than others. ISIS can represent relative importance by either an absolute measure of importance, or by the partioning of constraints into importance classes. Selecting which constraint to relax also depends on how the relaxation will affect other constraints. For example, reducing the number of shifts in the plant may be preferred, but it may negatively affect the due date constraint of many orders. Interactions of this nature are represented as relations in ISIS. It may also be the case, that the set of known constraints, may not be applicable to the particular decision. The system's *obligation* to satisfying a constraint depends on a number of factors: the time over which the constraint is applicable, the consistency of the constraint with others, the source of the constraint, and the context. ISIS provides a representation for all of this information. Lastly, constraints may be generated dynamically by attaching constraint generators to relations in the model.

4.1. Introduction

ISIS implements a constraint-directed search approach in its construction of job-shop schedules. Though the approach is complex, it can be viewed as a variation of generate and test. That is, there exist a set of operators which define a space of states which represent partial and full schedules, and there exists knowledge about the domain which is used to rate each state in the search space. The view taken in ISIS concerning the representation and utilization of this domain knowledge, is that it can be represented in a single formalism of constraints. That is, knowledge such as:

- organizational goals,
- physical characteristics and capabilities,
- causality,
- preferences of action and selection, and
- availability of resources,

may all be viewed as constraints. Additionally, there may exist knowledge at the search level which may constrain the reasoning process. For example, certain types of orders, e.g., forced outage, may be scheduled in a manner differently than other orders, e.g., shop orders. These can be viewed as *meta-constraints*, since they operate at a level above the actual scheduling problem.

Consider the due date associated with a lot. In reality, the due date is a constraint which the scheduling system should attempt to meet; it constrains the set of admissible schedules. If ISIS is to reason effectively with this type of constraint, the following information should be known to the system:

- How important is the constraint relative to the other known constraints? Is it more important to satisfy the cost constraint than the due date?
- If I cannot find a schedule which satisfies the constraint, are there relaxations of the constraint which can be satisfied. I.e., is there another due date which is almost as good?
- If there are relaxations available for the constraint, are any more preferred? Perhaps I would rather ship the order early rather than late?
- If I chose a particular relaxation, how will it affect the other constraints I am trying to satisfy? Will meeting the due date negatively or positively affect the cost of the order?
- Under what conditions am I obliged to satisfy a constraint? What if there are two constraints specified for the same variable? I.e., two different due date for the same lot? Or there may two different due dates depending on the time of year.

In essence, a constraint is not simply a restriction on the value of a slot etc., but the aggregation of a variety of knowledge used in the reasoning process. Hence, the implementation of a theory of constraint-directed search requires an answer to the following questions:

1. How are constraints to be represented?
2. How are they to be interpreted during the search process?

The first question is the focus of this chapter, the second is the focus of chapter 5.

his chapter begins with a basic description of constraints, followed by a description of their relaxations. In the following, we focus on applicability issues of constraints, in particular, issues in the representation of:

- the relative importance of multiple constraints,
- the obligation to satisfy constraints according to time, context, and source,
- the interactions among constraints, and
- the dynamic generation of constraints.

This is followed by some examples.

4.2. Structural Constraints

Constraints come in many guises. Section 1.1.1 provided a glimpse of the many kinds of constraints found in the job-shop scheduling domain. If ISIS is to represent these constraints, and more importantly, any new constraints not previously identified, then the constraint representation should be as domain independent as possible.

Many of the constraints found in the scheduling domain restrict the choice of value for some attribute, e.g., the due date of an order, or the relation between two or more schemata, e.g., the next operation to be performed. Constraints of this type may be described independently of the application. That is the constraints may be defined in terms of the structural level of representation, i.e., the representation language, as opposed to the domain level.

SRL/1.5 provides a good starting point for solving this problem; it provides a syntax for the representation of concepts. If constraints can be defined at the structural level of SRL/1.5, then they should subsume any constraint representable at the domain level. The following defines three types of constraints which correspond to the three primary structures found in SRL/1.5.

4.2.1. Range Constraint

A range constraint constrains the value(s) a slot may have. This constraint is probably the most common, and is basis of other constraint-directed reasoning systems (Zucker, 1976; Steele, 1981; Stefik, 1981). Range constraints may be specified as predicates, by enumeration, or through bounding. They may be unary in form; providing an absolute restriction such as the due date must be before the end of the month. They may be n-ary in form; restricting the value based on information within the schema (e.g., the due date must be no more than 2 weeks after the start date), or based on information in other schemata (e.g., the due date for the order must follow the due date of their previous order). Examples of range constraints include: due dates, work-in-process, cost, and production levels.

A constraint on a relation's range is represented by a **range-constraint.**

```
{{ range-constraint
    { IS-A constraint
        VALUE:    }
    DOMAIN:
    RELATION:
    CONSTRAINED-BY:         }}
```

Figure 4-1: range-constraint Schema

A **range-constraint** is a **constraint** whose properties are defined in the rest of the chapter and summarized in section 4.8. It specifies a DOMAIN schema and RELATION (slot) pair whose range (value) is to be constrained. The actual value chosen is placed in the VALUE slot inherited from **constraint**. The CONSTRAINED-BY slot contains a schema which specifies the actual constraint. The contents of the CONSTRAINED-BY slot is an and/or combination of constraints:

```
<constrained-by-spec> :: =

        (and <constrained-by-spec>+) |
        (or <constrained-by-spec>+) |
        (not <constrained-by-spec>) |
        <relaxation-spec>
```

The form of the constraint schema will be provided later in the chapter.

Range constraints may be attached directly to the relation (slot) of the schema to which it applies. In SRL this is accomplished by placing the constraint directly in the slot's *range* facet, replacing SRL's standard range specification. An example of the use of a range constraint would be the due date of the a turbine blade order.

```
{{ order-38
    { INSTANCE order
        DUE-DATE:
            range: order-38-due-date-constraint }    }}
```

order-38 has attached to its DUE-DATE slot a facet[5] called *range* which specifies the constraint on its range.

4.2.2. Relation Constraint

A relation constraint defines a relation between domain and range schemata. Examples include: precedence among operations, e.g., NEXT-OPERATION, and time ordering constraints between activities, e.g., before and during. The simplist form of a relation constraint is the slot. The existance of a slot in a schema implies that the schema must and does contain the slot. For example, if an **operation** schema contains a WORK-CENTER slot, then it is constrained to always have the WORK-CENTER relation.

Relation constraints may be more complex. They may specify a predicate or bounds on the the type of relation. A relation constraint may be specified by the **relation-constraint** schema.

```
{{ relation-constraint
    { IS-A constraint    }
    DOMAIN:
    RANGE:
    CONSTRAINED-BY:  }}
```

Figure 4-2: relation-constraint Schema

The DOMAIN and RANGE is specified by the constraint. The RELATION is constrained by the contents of the CONSTRAINED-BY slot.

Similar to the range-constraint, the relation constraint is attached to the meta-schema of a schema. This is accomplished by adding to the definition of **schema**[6] a RELATION-CONSTRAINT.

[5] A facet is another name for a slot in the meta-schema attached to a slot.

[6] All meta-schemata attached to a schema are of type **schema**.

```
<slot> |
<relation-constraint>
```

The method of constraint representation described here also provides a flexible pattern matching language. In particular it defines a language for the specification of schema patterns and the (binary) conditions of a successful match.

The definition of structural constraints is a first but important step towards a theory of constraint representation. But there is much to be represented before the theory can be introduced into a production environment. These extensions are the subjects of the remaining sections.

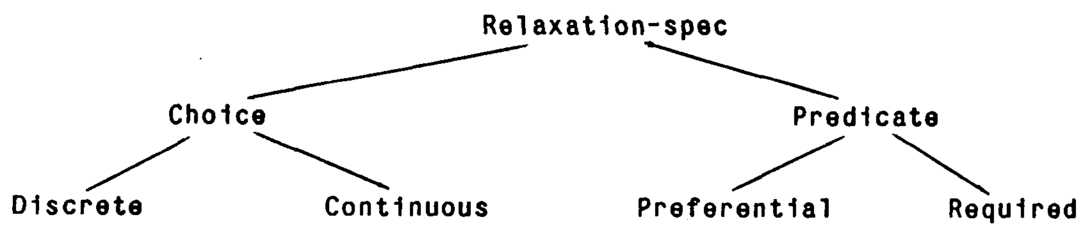
4.3. Relaxations and Preferences

During the construction of a schedule, it may be found that one or more constraints may not be satisfiable. For example, the NEXT-OPERATION of the **milling-operation** is constrained to be a **drilling-operation**. At some time during the plant's operation, drilling may not be performable due to lack of supporting resources (e.g., machine repair, operator sickness, etc.). Under these circumstances the system should look for alternative ways of achieving the order goal. This may be accomplished by choosing an alternative to drilling. The selection of an alternative to a specified constraint is called *constraint relaxation*.

In general, the variety and possibly conflicting nature of the constraints, may preclude the existence of a single schedule which satisfies all of them. How then can a schedule be chosen which satisfies as many constraints as possible? And if a constraint cannot be satisfied which *relaxation* of the constraint can be satisfied?

4.3.1. Relaxation Specification

The first step towards a theory of constraint relaxation is the categorization of relaxation types. ISIS divides constraints into the following categories:



Predicate constraints test the value chosen. Choice constraints specify alternatives from either a discrete or continuous set. This categorization only specifies that alternatives are available. It does not describe which alternative is preferred, if given a choice.

To enable ISIS to choose amongst relaxations, a system of utilities is provided. A constraint may have one of two effects on a schedule. It may determine the *admissibility* of a schedule, or it may determine the *acceptability* of a schedule. Admissibility determines the legality of a schedule against constraints that cannot be *relaxed*. Acceptability rates a schedule, allowing alternatives to be distinguished. The approach taken in ISIS in representing the satisfaction of a constraint, is to have each constraint impart a utility to a schedule. A constraint may return a value in the interval [0,2]. 0 implies that the schedule is rejected, 1 implies that the constraint is indifferent to the schedule, and 2 implies that the schedule is maximally rated[7].

Predicate constraints (figure 4-3) apply a predicate to the object under consideration.

```
{{ predicate-constraint
    { IS-A relaxation-spec
        RELAXATION-TYPE: predicate }
    PREDICATE:
    TRUE-UTILITY:
    FALSE-UTILITY:   }}
```

Figure 4-3: **predicate-constraint** Schema

If the predicate is evaluated to be true, then the contents of the TRUE-UTILITY slot is returned, otherwise the contents of the FALSE-UTILITY slot is returned.

A **required-constraint** determines the admissibility of a scheduling decision. For example, the choice of a machine for an operation may be rejected if the machine workbed size constraint rejects the order because it is too large. Either the machine is admissible for the order's operation or it is not. The **required-constraint** specializes the **predicate-constraint** by restricting the value of FALSE-UTILITY to zero. In ISIS, a zero utility results in the rejection of the schedule under consideration.

A **preferential-constraint** determines the acceptability of a scheduling decision by providing a constant rating if the constraint is satisfied. An example is a machine preference. The constraint may reflect the shop-floor supervisor's preference for one machine over another. This preference may have been derived from some cost or quality differential.

Both predicate constraints test a predicate to determine whether the constraint has been satisfied or not. They differ in their effects. One may reject a schedule, the other just changes its rating. The former prunes the space of schedules, the latter prioritizes the space. What has yet to be considered is the relaxation of constraints. In order to relax a constraint, ISIS must know the space of possible relaxations, and the preferences amongst them.

[7]The choice of [0,2] is a legacy of an earlier version of ISIS.

A **choice-constraint** specifies alternative values (relaxations), and the utility of the relaxation. The first type of choice constraint is the **discrete-constraint** (figure 4-4). It specifies the alternatives and their utility. For example, the capacity variable of an inspection center is dependent upon the number of inspectors. The capacity constraint of the center may specify three inspectors, but it may also specify choices defining the cost of each additional inspector in the center up to some maximum. The utility indirectly defines the cost of relaxing the number of inspectors constraint. The number of shifts for the center could be specified in an analagous manner.

```
{{ discrete-constraint
    { IS-A choice-constraint
        RELAXATION:
            range: (SET (TYPE INSTANCE choice-constraint))
        RELAXATION-UTILITY:
        RELAXATION-VALUE:
        DISCRETE-TYPE:
            range: (OR exclusive inclusive)
            default: exclusive }    }}
```

Figure 4-4: discrete-constraint Schema

A **discrete-constraint** extends the **choice-constraint** by defining a choice: RELAXATION-VALUE, and its utility: RELAXATION-UTILITY, and one or more alternatives: RELAXATION. The RELAXATION slot is filled with instances of **choice-constraints** which specify alternative discrete values and their utilities. The DISCRETE-TYPE slot defines whether the alternatives are to be exclusive or inclusive. The RELAXATION-UTILITY slot specifies the utility of the choice, which is then used to fill the constraint's UTILITY for the choice. Note that the form of the constraint allows relaxations to form a tree of alternatives. An example of a **discrete-constraint** is the specification of the number of shifts associated with a particular machine. Under normal operating procedures, the plant runs with only one shift, adding another shift results in overtime being paid. Adding another shift is not desirous.

A continuous constraint (figure 4-5) provides a rating of a schedule when it satisfies the constraint, and also provides a continuous rating when the constraint is not met, but relaxed. Consider the problem of meeting due dates. The due date is a constraint on the delivery date of the order. But it is not the case that being tardy or early is not admissible. Rather, it is not preferred. In this case, the due date constraint provides a rating for when the date is met, and also when the ship date is early or late.

```
{{ continuous-constraint
    { IS-A choice-constraint
        UTILITY-FUNCTION: }    }}
```

Figure 4-5: **continuous-constraint** Schema

The **continuous-constraint** (figure 4-5) has an additional slot, UTILITY-FUNCTION, which when passed the chosen value, returns the associated utility for the choice. Some constraints represented in ISIS as continuous constraints are due dates, start dates, work in process times, and queue stability.

Preferences among relaxations could also be specified relationally. The relation **more-preferred-than** could link two schemata to determine preferences among alternatives. No utilities need to be specified when using this relation, but could be derived from the ordering. This introduces two problems. The first is inconsistent preferences. Preference graphs may form cycles, hence display inconsistency in choice. A second problem is the lack of a scale against which to compare relaxations of other constraints. In view of these two problems, ISIS uses absolute utilities.

4.3.2. Relaxation Generation and Testing

The relaxation of a constraint specifies a set of alternatives with an associated utility. With this information, constraints may play an active role in the generation of the solution space. That is, constraints may act as generators of states in the search space in addition to testers.

The specification of how states (see section 5.5) may be generated from a constraint is contained in the GENERATOR slot of the constraint. It contains a function which takes two parameters: the current search state and the constraint, and generates zero or more successor states:

```
{{ constraint
    GENERATOR:
    TESTER:  }}
```

Figure 4-6: Basic CONSTRAINT Schema

Similarily, the interpretation of a constraint as a test of a state is determined by a function placed in the TESTER slot of the constraint. It takes the search state and the constraint as parameters, and returns a utility for the state. Consequently, these two slots proceduralize the knowledge contained in a constraint for both the generation and testing of states in the space of schedules. For example, the function in the tester slot for a due date constraint would, for each constraint in the CONSTRAINED-BY slot, apply the contents of the UTILITY-FUNCTION slot to the state's proposed due date, and combine the utilities according to the form of the and/or.

4.4. Importance

Constraints are not created equal. Depending on the context, some constraints are more important than others. For example, a "forced outage" order of blades is a class of orders with a very short lead time and a due date which must be met at all costs. A "stock" order is an order which has no definite due date but should efficiently utilize resources in the factory. The importance of an individual constraint may differ from order to order. Hence, ISIS must provide a method by which the relative importance of constraints may be determined.

The importance of a constraint is specified by a value between 0 and 1. An importance of zero would imply that the constraint should not be considered, i.e., it is unnecessary. An importance greater than zero would signify an increasing importance. The actual level of importance is relative to the importance of the other constraints under consideration. Hence a constraint's importance may be directly compared. A constraint's importance is specified in its IMPORTANCE slot. The importance of a constraint may be derived manually or automatically. The following defines the methods available in ISIS.

Manual. A simple way of specifying constraint importance is to assign to each constraint a constant importance which is some number chosen from a rational scale. This importance could be changed by the user for each order (lot) to be scheduled. This places the burden of differentiation on the user, requiring their input for each scheduling run.

Partitioning. Orders in W-S fall into definite classes. i.e., forced outage, ship direct, stock, etc. Experience has shown that each class has a preferred set of constraints to satisfy. These preferences, in effect, partition the constraint set such that the importance of a constraint is determined by the partition in which it is a member. Partitioning is implemented in ISIS by **scheduling-goal** (see section 4.9.4).

Relational. A third method of differentiation relies on relations to specify an ordering among constraints. To simply say one constraint is more important than another does not solve the problem of how much more important. The ISIS method of schedule rating requires a rational, as opposed to an ordinal, system of measurement. Hence, relative differentiation is not used.

Since ISIS will be attempting to satisfy many constraints in parallel, some method of comparing their satisfaction must be provided. A constraint's utility provides a rational measure of the acceptability of the particular relaxation to ISIS; if a due date constraint's utility is greater than the utility of a cost constraint, this implies that it has been "better satisfied". The measure of importance of a constraint may be viewed as a weight. It may be combined with a constraint's utility to form a weighted combination of utilities.

4.5. Interactions

Until now, constraints have been treated as being independent; the selection of a constraint's value does not depend on nor affect another constraint's choice. But consider the choice of the number of shifts to run a machine. If the machine is a bottleneck, then reducing the number of shifts will increase possibly an order's work-in-process time and/or miss its due date. The choice of a budget constraint for the overall operation of the plant may affect the choice of shifts available. It is obvious that constraints do affect each other. The relaxation of one may affect the possible relaxations of others, and there by reduce or increase their corresponding utilities. Such knowledge would be useful both in pre-selecting constraint values, and analyzing the resultant utility of a schedule (see section 5.5.3.2).

To aid ISIS in its analysis of schedules, the **constrains** relation is defined (figure 4-7). This relation links **constraints**. A value of pos in the DIRECTION slot denotes that a positive shift in the utility of the domain constraint will result in a positive shift in the utility of the range constraint. A value of neg denotes that a positive shift in the utility of the domain constraint will result in a negative shift in the utility of the range constraint. An unknown direction implies no direct correlation. Each of these relations has a **sensitivity** slot which denotes the sensitivity of the range constraint to a change in the domain constraint (i.e., approximation of a linear coefficient).

```
{{ constrains
    { IS-A relation
        DOMAIN:
            range: (type "is-a" "constraint")
        RANGE:
            range: (type "is-a" "constraint") }
    DIRECTION:
        range: (or pos neg unknown)
    SENSITIVITY:    }}
```

Figure 4-7: constrains Schema

4.6. Obligation

Obligation defines the conditions underwhich a particular constraint must be considered and possibly satisfied. Some forms of obligation are obvious such as the context sensitivity of a constraint. Other forms of obligation are more subtle and did not become apparent until field implementation of ISIS became imminent. The following defines various types of obligation a constraint may specify.

4.6.1. Duration

[8] Organizations are organic; they change over time. Hence constraints may only have a limited time span for which they are applicable. This specification of time can be absolute, or it can be defined relative to some other span of time. An example of the latter is the specification that a constraint is applicable only during the execution of an activity such as an operation, or before or after it. To represent the concept of absolute and relative time specification, the time representation described by Allen (1982) is used. It is interval based and contains the following basic relations: before, overlap, during, meet, and time-equal (see section 3.4.2). An example of the during relation is as follows:

```
{{ need-box-gauge
   { INSTANCE possess
       POSSESSION: box-gauge
       DURING: airfoil-inspection }        }}
```

Figure 4-8: need-box-gauge Schema

Constraints may now be linked to activities which span a time interval by means of these relations. Constraints may also be directly linked to a time interval in order specify an explicit time during which they should be considered. See (Smith, 1983) for a complete description of time-based selection of constraints.

4.6.2. Consistency

The use of ISIS in a factory may result in the same constraint being supplied by more than one source. Consider the possibility that both the marketing representative and the scheduler have access to ISIS. Each may specify a possibly different due date in line with their own perception of "reality". What should ISIS do when faced with inconsistent constraints (as opposed to conflicting)? In this case it could determine which constraint is to be ignored either by asking one of the users to yield, or by referring to other information. One approach to the latter is to generate an authority model (Meehan, 1981). This model would specify the authority relations between people and departments which exist within the corporation, and whose specifications of constraints have precedence over others. These relations must also specify precedence among automatically generated constraints, and those manually generated.

To support this, ISIS must first recognize that there cannot exist more than one constraint for a range, relation, or schema. This is accomplished by means of the CONSISTENCY slot in each type constraint. At present the slot may have two values: exclusive or inclusive. An

[8] Much of this subsection evolved during discussions with the ISIS research group: Brad Allen, Stephen Smith, and Gary Strohm.

exclusive value signifies that there can only be a single constraint, and if there is more than one, the system must determine which to ignore.

The second step toward removing inconsistencies is the recording of adequate information with which to make decisions. Each constraint may have meta-information which records its creator. This is easily handled by the meta-schema.

4.6.3. Residency

A third type of obligation is *residency*. Simply through attachment of a constraint to an object or process, ISIS is obliged to consider the constraint whenever it uses the object and/or process in a schedule. Attachment is accomplished by either adding a CONSTRAINT slot to a schema and filling it with one or more constraints, or elaborating a CONSTRAINT slot into one or more types of constraints (e.g., shift, capacity).

4.6.4. Context

Duration, authority, and residency are part of the basic specification of the context underwhich a constraint may be applicable. Other obligation types remain to be determined and their relations constructed. In lieu of further analysis, a catch-all CONTEXT slot is specifiable in a constraint. The contents of this slot is a lisp function which takes two paramenters, the search state and the constraint. The function must return non nil in order for the constraint to be used.

4.6.5. Obligation Resolution

The resolution of the set of constraints to be evaluated during the scheduling process is defined in the next chapter. Once the ISIS search algorithm has determined the initial set, the constraints are filtered according to the following tests:

- remove all constraints whose time specification does not coincide with the current time,
- remove all constraints whose contexts evaluate to nil,
- partition constraints into inconsistency sets and reduce to single constraint in each set according to authority/preference relations:
    1. the creators of the inconsistent constraints are retrieved from the attached meta-schemata,
    2. constraints without creators are ignored,
    3. the **has-authority-over** relation is followed to determine the creator with the greatest authority, and the corresponding constraint is returned.

4.7. Generation

With the definition of constraints complete, it is time to discuss their generation. Until now, it has been assumed that constraints exist statically in the model; their source being the system user. Instead, constraints may be generated dynamically. Consider the problem of a milling operation which transforms the shape of metal. The final mass of the metal may be a function of the initial mass, and the operation. Hence, the *constraints* on the the metal's final mass must be determined dynamically; the operation does *not* produce a value for the metal's mass, but one or more constraints.

At this point it is important to differentiate among a classical algorithmic approach to problem solving, and a constraint-directed approach. In the former, algorithms/functions would produce a single value for the attribute of a state. In the latter, constraints are produced, and the attribute's value must be deduced from them. These approaches are equivalent only when the constraint limits choice to a single value.

Existing approaches to the dynamic generation of constraints are either highly specific to the domain, for example the CONSTRAINTS system (Steele, 1980) produces a constraint on the nth pin of a circuit, given constraints on the n-1 pins, and the definition of the circuit. Or more general to the point of being simple production rules (McDermott & Steele, 1981; Stanfill, 1981).

The approach taken in ISIS is somewhere between the two. The generation of constraints can be embedded in the relational model of the domain. Consider the milling operation again. The performance of the operation *causes* a state in which the metal's mass is reduced. In ISIS, the constraint generator would be attached directly to the **cause** relation linking the milling operation to its post-state. Hence, when the operation (act) is performed, the system is not only able to deduce what states are caused, but any constraints on those states. The approach may be extended by allowing constraint generators to be attached to any relation.

The representation of a relation is amended to include constraint generators:

```
{{ relation
    DOMAIN:
    RANGE:

    ...

    CONSTRAINT-GENERATOR:    }}
```

Figure 4-9: **relation** Schema with Constraint Generator

4.8. The Constraint Schema

The following is the general form of a constraint, from which the others are derived:

```
{{ constraint
    VALUE:
    UTILITY:

    CONTEXT:
    IMPORTANCE:
    INTERACTION:
    DURATION:
    CONSISTENCY:

    GENERATOR:
    TESTER:       }}
```

Figure 4-10: General **constraint** Schema

In the VALUE slot is placed the value chosen for the constraint. In the UTILITY slot is placed the utility of the value chosen. Each of the other slots refer to the basic issues raised in the preceding sections. They are either used directly or elaborated in some manner. For example, the IMPORTANCE slot is filled manually and/or by interpreting scheduling goals. The DURATION slot is elaborated into relative time relations, or filled with a time span.

The next section provides examples of different types of constraints.

4.9. Examples

4.9.1. Predicate Range Constraint

Consider a constraint that restricts the length of a turbine blade that can be milled on a machine to less that 28.5 inches. This can be represented by the schema **product-length-requirement** which is a combination of a **required-constraint** and a **binary-attribute-constraint** (see section 4.11):

```
{{ product-length-requirement
    { IS-A range-constraint
        DURING: airfoil-operation
        CONSTRAINS: airfoil-machine-preference
            direction: neg

        DOMAIN:
            range: (type "is-a" "blade")
        RELATION: foil-length
        CONSTRAINED-BY: product-length-constraint }    }}
```

Figure 4-11: **product-length-requirement** Schema

```
{{ product-length-constraint
    CONSTRAINT-OF: product-length-requirement

    { INSTANCE required-constraint
        RELAXATION-TYPE: required
        TRUE-UTILITY: 1.2
        PREDICATE: product-length-predicate }    }}
```

Figure 4-12: **product-length-constraint** Schema

```
{{ product-length-predicate
    PREDICATE-OF: product-length-constraint

    { INSTANCE binary-attribute-predicate
        RANGE-2: 28.5
        PREDICATE: lessp  }    }}
```

Figure 4-13: **product-length-predicate** Schema

product-length-requirement specifies that the foil-length of a blade is being constrained. It is obligated to being used during an airfoil-operation, and it negatively affects the airfoil-machine-preference constraint. The actual constraint is specified in **product-length-constraint**. If the constraint is satisfied then the utility returned will be 1.2, otherwise 0. The predicate of the requirement is specified by the **product-length-predicate** It specifies that any blade must have a foil-length less than 28.5 units. One potential problem in constructing this constraint is enabling the predicate to refer to slots in the root constraint (i.e., product-length-requirement). You will notice that the predicate schema is linked to the requirement schema via a PREDICATE-OF relation (inverse of predicate), and that the requirement schema is linked to the range constraint by a CONSTRAINT-OF relation (inverse of constrained-by). Each of these relations allow the inheritance of slots and values. Hence the **product-length-predicate** inherits the DOMAIN, and RELATION slots from **product-length-requirement.**

The **product-length-requirement** is not attached to the FOIL-LENGTH slot of all products, but is attached instead to the **airfoil-operation** schema (i.e., contained in the **constraint** slot). It is up to ISIS to retrieve the constraint from the operation's CONSTRAINT slot, and apply its tester function to the search state and constraint. The tester retrieves the blade being scheduled and places it in the domain slot, and applies the contents of the APPLY slot in the predicate to its schema name. This is described further in the next chapter.

4.9.2. Discrete Range Constraint

Another type of constraint often found in manufacturing organizations is the specification of shifts. A shift defines the time that a work center is availabe for work. Historically, it has been discrete, specifying one, two, or three shifts during a work day. In addition, the number of shifts on a week end may differ from that during a week day. Therefore, a shift constraint should specify what the normal available shifts are, what the relaxations are, and the period during which the shift constraint should be interpreted.

A shift specification may be specified as a **discrete-constraint** (figure 4-14). The CONSISTENCY of the slot is exclusive, specifying that only one shift constraint may exist for the slot. No alternatives are specified at this point.

```
{{ shift-constraint
    { IS-A range-constraint
        DOMAIN:
            range: (or (TYPE is-a machine) (TYPE is-a work-center))
        RELATION: shift  }  }}
```

Figure 4-14: **shift-constraint** Schema

```
{{ shift-constraint-spec
    { is-a discrete-constraint
        CONSISTENCY: exclusive   }   }}
```

Figure 4-15: **shift-constraint-spec** Schema

```
{{ shift
    START-TIME:
    END-TIME:
    WORK-WEEK: }}
```

Figure 4-16: **shift** Schema

An example of a shift constraint is that specified for a wmf1 machine (figure 4-17).

```
{{ wmf1-shift
    { IS-A shift-constraint
        DOMAIN: wmf1
        RELATION: shift
        CONSTRAINED-BY: wmf1-shift-constraint   }   }}
```

Figure 4-17: **wmf1-shift** Schema

The range constraint specifies the domain of the constraint and relation. That is, the constraint affects the SHIFT slot of the **wmf1**. The contents of the CONSTRAINED-BY slot is the name of the constraint: **wmf1-shift-constraint** (figure 4-18). It describes a start-time, end-time, and day for the shift.

```
{{ wmf1-shift-constraint
    { INSTANCE shift-constraint
        RELAXATION-VALUE: {{ INSTANCE shift
                              START-TIME:    8:00
                              END-TIME:     16:00
                              WORK-WEEK:      (OR monday tuesday wednesday
                                              thursday friday)    }}
        RELAXATION-UTILITY:    2
        RELAXATION: wmf1-shift-relaxation  }     }}
```

Figure 4-18: **wmf1-shift-constraint** Schema

The shift constraint is not a schema constraint. Each relaxation completely specifies the start-time, end-time, and work week. They cannot be relaxed individually. The contents of the RELAXATION slot specify another shift, **wmf1-shift-relaxation**, to be used in addition to the first constraint (the DISCRETE-TYPE of the constraint is inclusive).

```
{{ wmf1-shift-relaxation
    { INSTANCE shift-constraint
        RELAXATION-VALUE: {{ INSTANCE shift
                              START-TIME:    16:00
                              END-TIME:     24:00
                              WORK-WEEK:      (OR monday tuesday wednesday
                                              thursday friday)    }}
        RELAXATION-UTILITY:    1.2
        DISCRETE-TYPE:         inclusive    }}
```

Figure 4-19: **wmf1-shift-relaxation** Schema

The constraint is interpreted by taking the value of the TESTER slot from **wmf1-shift** (not shown) and applying it to the pair (<state> **wmf1-shift**). The tester will retrieve the discrete constraint and find the value which matches the value under consideration (i.e., specified inthe state) and return the relaxation utility.

4.9.3. Continuous Range Constraint

A **due-date-constraint** is represented as a continuous range constraint.

```
{{ due-date-constraint

   { IS-A range-constraint
       DOMAIN:
           range: (type "is-a" "lot")
       RELATION: due-date
       CONSTRAINED-BY:
           range: (type "is-a" "due-date-constraint") }

   TESTER: due-date-tester
   PRIORITY-CLASS:  }}
```

Figure 4-20: **due-date-constraint** Schema

```
{{ due-date-constraint-spec
   { IS-A continuous-constraint
       CONSISTENCY: exclusive
       UTILITY-FUNCTION: interpolate    }
   PIECE-WISE-LINEAR-UTILITY: }}
```

Figure 4-21: **due-date-constraint-spec** Schema

The basic **due-date-constraint** is a continuous value constraint which constrains the due-date slot of a lot. The choice of a due-date has a utility specified by the PIECE-WISE-LINEAR-UTILITY. The utility is specified by (shipping-lateness utility) pairs. An example of its use is a due date for forced outage orders. The tester for **due-date-constraints** takes the search state and the constraint as parameters, retrieves the due date being considered int the state, or predicts one, and applies the value of the utility function slot to the due date. The utility function uses the PIECE-WISE-LINER-UTILITY value to interpolate and return a utility.

```
{{ fo-due-date
   { IS-A due-date-constraint
       PRIORITY-CLASS: forced-outage
       CONSTRAINED-BY: {{ INSTANCE due-date-constraint
                              PIECE-WISE-LINEAR-UTILITY: ( (0 2) (7 0.2) )    }}
       }  }}
```

fo-due-date specifies that the utility of the due date chosen is 2 if it less than or equal to the the requested due date. It is linearly decreasing to 0.2 if it greater than 0 days late and less than 7. And is 0.2 if greater than 7 days late.

4.9.4. Meta-Constraints: Importance Partitioning

A variety of constraints exist which are used to select the type of search to be performed, and the knowledge to be used. They are called meta-constraints. Examples of meta-constraints include:

- selecting the direction of the beam search based on the priority class of the order,
- selecting the operators to perform the search,
- resolving which constraints to use to rate a state in the search space.

One such meta-constraint is the **scheduling-goal**. It defines for each type of order the relative importance of constraints. During the scheduling process, ISIS retrieves the **scheduling-goal** for the order being scheduled and derives the importance of constraints under consideration. Hence, ISIS dynamically determines a constraint's IMPORTANCE while searching.

```
{{ scheduling-goal
    { IS-A constraint
        DOMAIN: search-state      }
    PRIORITY-CLASS:
    PARTITION-IMPORTANCE: }}
```

Figure 4-22: scheduling-goal Schema

The **scheduling-goal** constraint applies to any state generated by ISIS in it search for a schedule.

The **partition-importance** schema defines an importance for a schema or class of schemata. Given a set of constraints. ISIS partitions the constraints according to the **partition-importance** specifications (<partition-spec> is an AND/OR/NOT specification of a partition). And divides the partition's importance equally among the constraints in the partition.

```
{{ partition-importance
    PARTITION-SPEC: <partition-spec>
    IMPORTANCE: }}

<partition-spec> ::=  <schema>
              | (OR <partition-spec>+)
              | (AND <partition-spec>+)
              | (NOT <partition-spec>+)
```

An example of a goal in ISIS is a **Forced-Outage-Sched-Goal** (figure 4-23).

```
{{ Forced-Outage-Sched-Goal
    { INSTANCE scheduling-goal
        PRIORITY-CLASS: forced-outage
        PARTITION-IMPORTANCE: fo-partition other-partition }     }}

{{ fo-partition
    { INSTANCE partition-importance
        PARTITION-SPEC: (OR lead-time due-date)
        IMPORTANCE: 0.9          }    }}

{{ other-partition
    { INSTANCE partition-importance
        CONSTRAINT: (NOT (OR lead-time due-date))
        IMPORTANCE: 0.1          }    }}
```

Figure 4-23: Forced Outage Scheduling Goal

In this goal, all **due-date** and **lead-time** constraints are in the **fo-partition** which splits a importance of 0.9. The **other-partition** covers all the other constraints not in the **fo-partition** with a combined importance of 0.1.

The interpretation of scheduling goals is "hard-wired" into the resource analysis level's beam search algorithm. A general approach to the interpretation of meta-constraints has not been investigated.

4.10. Conclusion

This chapter introduced some of the principles of constraint-directed reasoning. In particular, it defined the types of knowledge required to reason about constraints, and proposed a representation. In ISIS, "constraint-directed search" is more than a phrase. It provides a shift in perspective. Instead of viewing the creation of a shop model as object-attribute-value triples, the focus shifts to specifying object-attribute-constraint triples. Values are simply the result of evaluating the constraint or constraints.

Constraint-directed search is also a theory of problem solving. More specifically, it is an extension of heuristic search paradigm. Where heuristic search specifies a state-space approach to problem-solving, constraint-directed search extends this theory by providing meta-information about the problem and its search space. Constraints may:

- specify possible operators. For example, a next operation constraint can be an operator.

- evaluate states. The utility associated with a constraint and its relaxations can be used to evaluate the choices made in a state.

- stratify the search space. Interactions among constraints my define a structuring of the search space.

- structure our knowledge about the problem. Obligation and relaxation provide additional structure as to when and where to use this knowledge.

Much of the problem in performing problem-solving is the selection of state description and operators. That is, the determining what the structure of the search space will be. A poor choice of either may result in infinite search. Constraints provide a solution to this problem by explicitly representing the dimensions to be searched (i.e., constraints and their relaxations), and preferences among them. With this information, it may be possible to construct a system which can structure the search space and select the operators to search it. This issue is explored in the next chapter.

4.11. Appendix: Predicate Specification

This appendix examines how predicate constraints may be specified in SRL/1.5. The approach taken is a combination of declarative and procedural representations. That is, the description of the constraint is defined by slots in a schema, and the application or testing of the constraint is performed by procedural interpretation. For example, the concept of a function can be represented as follows:

```
{{ function
    PARAMETER:
    BODY:
    APPLY: (lambda (schema)
                (apply (valueg1? schema "body") (list (valueg1? schema "parameter")))) }}
```

Figure 4-24: Function Schema

The contents of the APPLY slot provides the procedural interpretation of the **function** schema. It simply applies the contents of the BODY slot to the contents of the PARAMETER slot.[9]

It follows that constraints which have a procedural interpretation such as enabling and preference constraints can be represented in this fashion. For example, a **binary-attribute-predicate** may be represented as follows:

[9]The function valueg1? returns the first value in the slot. But if the slot is empty, it returns the name of the slot. The rationale for this is a result of the user's ability to elaborate any slot into new slots, in SRL. By elaboration, the user may redefine the BODY slot into more detailed slots until the level of specification reaches lisp functions. At this point the slot name is equivalent to the value of the slot.

```
{{ binary-attribute-predicate
     DOMAIN:
     RELATION:
     RANGE:
     RANGE-2:
     PREDICATE:
     APPLY: (lambda (bap)

               ; place in the range slot the value defined in domain.range
               (valuec1 bap "range"
                 (valueg1 (valueg1? bap "domain") (valueg1? bap "relation")))

               ; apply the predicate to the two ranges
               (apply (valueg1? bap "predicate")
                   (list (valueg1? bap "range") (valueg1? bap "range-2"))))      }}
```

Figure 4-25: binary-attribute-predicate

The procedural interpretation takes the name of the schema as its only parameter (bap). It assumes that the schema has the DOMAIN, RELATION, and RANGE-2 slots filled. It fills the RANGE slot from the designated DOMAIN and RELATION. It then applies the PREDICATE to both values. Other predicates can be defined in a similar manner. This allows ISIS to "execute" any predicate in a predicate-constraint in schema form.

Chapter 5
Constraint-Directed Scheduling

Summary

The goal of ISIS is to construct schedules which satisfy as many constraints as possible in near realtime. To achieve this, ISIS uses constraints to bound, guide and analyze the scheduling/search process. The ISIS system performs a hierarchical, constraint-directed search in the space of alternative schedules. Level 1 selects an order to be scheduled according to a prioritization algorithm based on the category of the order, and its due date. Level 2 performs a current capacity analysis of the plant. It determines the earliest start time and latest finish time for each operation of the selected order, as bounded by the order's start and due date. The times generated at this level are codified as constraints which are passed to level 3. These operation time bound constraints constrain the start and end times of operations at the next level. Level 3 performs a detailed scheduling of all resources necessary to produce the selected order. Pre-search analysis begins by examining the constraints associated with the order to determine the scheduling direction (forward vs backward), whether any additional constraints should be created (e.g., due dates, work-in-process), and the search operators which will generate the search space. A beam search is then performed using the selected search operators. The beam search sequences the application of operators. Each application of an operator generates another "ply" in the search space. At each ply only the "n" highest rated states are selected for extension to the next ply. The most often selected operators generate alternative operations, machines, and queue positions for an order in the plant. Starting with a null schedule, alternative partial schedules are generated either forward from the start date or backward from the due date. An operation operator generates alternative states which represent alternative operations in either the forward or backward direction. Once the operation is known for a state, other operators extend the search by creating new states which bind the machine and/or the execution time of the operation. A variety of alternatives exist for each type of operator. For example, two operators have been tested for choosing the execution time of an operation. The "eager reserver" operator chooses the earliest possible reservation for the operation's required resources, and the "wait and see" operator tentatively reserves as much time as available, leaving the final decision to level 4. This enables the adjustment of reservations in order to reduce work in process time. Alternative resources are generated (e.g., tools, materials, etc.) by other operators. Each state in the search space is rated by the set of constraints found (resolved) to be relevant to the state and its ancestors. Constraints defined to be in the set are those which are attached to any resource (e.g., machine, tool,

order, etc.) specified by the state. Each constraint assigns a utility between 0 and 2 to a state; zero signifies that the state is not admissible, 1 signifies indifference, 2 maximal support. The rating of a state with multiple constraints is the weighted (by importance) average of the constituent constraints. The importance of a constraint is defined statically or derived dynamically according to goal information. Once a set of candidate schedules have been generated, a rule-based post search analysis examines the candidates to determine if one is acceptable. Currently, any schedule with a rating greater than one is accepted. If no acceptable orders are found, then diagnosis is performed. First, the schedules are examined to determine a type of scheduling error. The error is then fed back to pre-analysis in order to select new operators which are used to re-schedule the same order. The diagnosis of poor solutions caused by constraint satisfaction decisions made at another level can be performed by analyzing the interaction relations linking constraints. A poor constraint decision at a higher level can be determined by the utilities of constraints affected by it at a lower level, and an alternative value chosen. Level 3 outputs reservation time bounds for each resource required for the operations in the chosen schedule. Level 4 selects the actual reservations for the resources required by the selected operations which minimize the work-in-process time.

The scheduling of ISIS is also reactive. The invalidation of reservations by actions such as machine breakdowns or other orders taking too long on a machine, results in a minimal re-scheduling of only the affected orders, while attempting to maintain previous reservations. ISIS's scheduling is also suggestive. If constraints cannot be met, it attempts to generate a schedule which satisfies as many constraints as possible. For example, if the due date of an order cannot be met by backwards scheduling, it attempts to schedule in the forward direction and suggests an alternative due date.

5.1. Introduction

Job-shop scheduling is a constraint-directed reasoning task. This has been determined by an analysis of the tasks performed by schedulers. How then must machine problem solving be organized in order to construct schedules which satisfy the constraints? A strawman proposal is to use the british museum algorithm approach: enumerate the solution space of possible schedules and choose the best by using constraints as rating functions. Regrettably, scheduling is a classic example of an NP-hard problem which grows exponentially along each dimension such as operations, machines, tools, personnel, orders, etc. By reducing the number of dimensions the size of the problem decreases, but just considering n orders on an m machine process sequence results in $(n!)^m$ different sequences. And this ignores time assignments with gaps. The solution lies not just in how to search the "solution space", but in finding the right problem in the "problem space". Selecting the "right problem" implies bounding the dimensions and size of the solution space.

The first step towards bounding the problem would be to analyze the complexity of the search space. *Meta-planning* is one possible approach (Hayes-Roth & Hayes-Roth, 1979;

Wilensky, 1980; Stefik, 1981b). Given the goal of a "shipped order", a meta-planning system should first decide how it can be achieved. For example, there is an **operation** which has its caused state restricted to a "shipped order". The meta-planner would examine the operation to identify slots which enable it. In the **finish-&-test** operation (figure 5-1), possession of a NDT-MACHINE and **ndt-fixture** enable the operation to be performed. In addition, **airfoil-machining** is act-coupled with **finish-&-test**, hence it must be performed before **finish-&-test**. The meta-planner would note that the latter constraint will require that planning be done to determine an operation sequence. It would also note that the former constraints require scheduling (time assignments) of resources to be performed.

```
{{ finish-&-test
   { IS-A operation
      CAUSE:
         range: (TYPE "is-a" "shipped order")
      ENABLED-BY: "f&t-resources"
      ACT-COUPLE+INV: "airfoil-machining" }    }}
```

Figure 5-1: finish-&-test Schema

```
{{ f&t-resources
   { INSTANCE and
      SUB-STATE: "possess-ndt" "possess-ndt-fixture"    }    }}

{{ possess-ndt
   { INSTANCE possess
      POSSESSION: ndt   }    }}

{{ possess-ndt-fixture
   { INSTANCE possess
      POSSESSION: ndt-fixture   }    }}

{{ airfoil-machining
   { INSTANCE operation
      ACTCOUPLE: "finish-&-test" }    }}
```

In this initial meta-planning or problem generation stage, it was found that planning in an operation space and scheduling in multiple resource spaces must be performed. Now that the extent of the problem space has been marked off, states in the space must be generated, keeping in mind that a problem definition must be found of tractable complexity.

One approach to reducing the complexity of the search space is to recognize that scheduling has *multiple foci.* That each non-perishable non-infinite resource, can be viewed as a separate scheduling focus with its own constraints to be considered and utilities to be maximized. Of the scheduling systems surveyed in the market place, it was found that each was a machine scheduling system. The focus being machine-operation assignments with bill of materials (i.e., tooling, material, and other requirements) exploded for each operation. None attempt to optimize tools or other resource assignments.

We now encounter the problem of whether to totally decompose scheduling into separate problems for each resource. And if so how are their results to be communicated to each other? Or should the scheduling of all resources be collapsed into a single search space?

Since complexity does not allow the merging of all scheduling foci into a single search space, it remains to choose how the problem is to be decomposed and what the operators are that elaborate the search space. In order to determine the dimensions along which to search the space of partial schedules, scheduling goals and constraints must be re-examined. A primary goal of the factory is to produce the product and to meet an order's due date. In order to produce a product, a process routing must be chosen. The choice of an operation directly affects all time related constraints since processing time is directly related to the operation. The choice of an operation restricts the choice of machine, and in turn the choice of queue position. And the choice of a machine and queue position for the machine also directly affect time-related constraints. So there are three dimensions along which the search space can be elaborated: operation, machine, queue position. Upon further examination, each of the resources have their availability constrained. For example, the number of SHIFTS constrains how long the machine is run each day. Each of these constraints may also define dimensions in the search space.

Should the search be restricted to the operation, machine, and queue position dimensions when the choice of the number of shifts available on a machine has a large affect on time-related constraints? In this case the number of shifts is a *hard* constraint. The degree of hardness of a constraint is related to the cost of relaxing it. Normally, the number of shifts associated with a machine or cost center is set in advance, and the addition of another shift is quite costly. It is not a decision that is normally considered except in certain situations. Hence, choosing the dimension along which partial schedules are elaborated is determined by its relevance to goals, primary constraints, and the cost of relaxation.

Once the problem has been defined, the method of search must be determined. As described earlier, the number of possible sequences of orders on a set of machines is very large. Hence, the search method is crucial.

Whenever search is to be performed, the first problem is to decide where to start. Search problems can be distinguished by whether they are *anchored* or not. Anchoring implies that there are one or more points in the search space that are known to the problem solver. These anchors can be used to guide or focus the search. Actually, anchors are instances of of meta-constraints, i.e., a *search* constraint. An anchor constrains the search process.

A problem with only an initial state is called a *single anchor problem*. Only one state is known. The goal state may not be unique, but can be recognized. The search begins at this anchor state and spreads. How the search spreads can be categorized in two ways: *rated* and *blind*. A rated search assumes that each state can be rated, and ordered by rating. Examples of rated searches are best-first, A^* (Nilsson, 1971), B^* (Berliner, 1979), and Beam (Lowerre, 1976). A blind search assumes that states cannot be differentiated other than by order of elaboration. Two such searches are breadth and depth searches.

Problems can have more than one anchor. A *two anchor problem* might have both an initial state and a goal state. The search is for a path between the two states. Means-end analysis (MEA) (Newell, 1969) is one technique that finds paths between anchors, and the parallel search of NOAH (Sacerdoti, 1975) is another. The existence of more than two anchors implies that there may be more than one initial or goal state, or that states along the solution path are also known. In the latter case, MEA can be used, or the *island-driving* technique of Hearsay-II (Erman et al., 1980) and HWIM (Woods et al., 1976).

Up to this point, it has been assumed that anchors are well defined. That they are points in the search space to be used or avoided, implying that they are discrete. But actually anchors may form a continuum. Obviously, if anchors are just constraints on how to search, then they should follow the constraint categorizations in section 1.1.1.

Search may be directed, hence reducing complexity, by exploiting *structure* in the search space. Two types of structure have been discerned. The first is the *stratification* of the solution space. That is, the solution space can be divided into levels. Each level can be searched separately or in some connected fashion. The Hearsay-II levels of representation is a *dependency* stratification where the higher levels (e.g., syntax) restrict the lower levels (e.g., lexical, phonemic). On the other hand, ABSTRIPS (Sacerdoti, 1974) stratified the conditions of its operators by partitioning the variables into *major factor* sets. Strata can be searched top-down, bottom-up, or opportunistically.

The second type of stratification is to *decompose* the space into separate sub-spaces that can be searched separately.

In our analysis of the problem, the search dimensions were narrowed to operations, machines, and queue positions. These dimensions formed a dependency stratification of the search space (figure 5-2). The space was single anchored at either a start date or a due date in conjunction with a first or last operation, and each state was rated (by the constraints). The question then was what search technique should be used?

The number of schedules elaborated can be reduced by taking a top-down approach to searching the machine space. This means that the operation space would first be searched to find a "reasonable" sequence of operations. Then the machine space would be searched along the direction imposed by the chosen operation sequence. Finally, the queue position space would be searched for only the machines in the chosen machine sequence. An alternative search method would be to *collapse* the spaces into one and search each

Operation Space

Machine Space

Queue Position Space

Figure 5-2: Machine Scheduling Search Space

dimension in parallel. The choice of approach is dependent upon whether choices at a higher level in the top-down approach would prune better partial schedules from the search space. In this case, the top-down approach would generate inadequate solutions which would result in back-tracking. In other words, the stratification does not exhibit *complete dominance*:

Definition: Rated Space A search space where each state (node) has a rational numeric rating assigned to it.

Definition: Stratified, Rated Space A rated search space in which the states are partitioned into levels. States at one level can be related (linked) to states (children) at the immediate level below it. The rating of a state is the sum of its own rating and the ratings of its children.

Definition: Complete Dominance A level in a stratified, rated, search space completely dominates a lower level when the sum of ratings of nodes along any search path in the lower level is less than any rating for any node in the higher level.

Theorem: A rated search space where each level completely dominates the levels below will return the optimal path if searched top-down.

Corollary: A top-down search of a stratified, rated, search space will have the same results as a complete search of the collapsed space if and only if the space is completely dominated.

The first version of ISIS constructed was composed of two levels:

1. lot selection,

2. resource analysis.

Lot selection chose a single lot and handed it to resource analysis. Resource analysis performed a beam search to construct detailed schedules for all resources used in next-operation, choose-machine, and choose-queue-pos. Schedules constructed in this manner were rather poor due to horizon problems[10].

[10] See chapter 6 for a complete analysis.

Based on experience with the first version, a second version of ISIS was constructed which was composed of three levels:

1. lot selection,

2. capacity analysis,

3. resource analysis.

The capacity analysis level generated time bounds for the operations required to build a lot. It ignored much of the information used in resource analysis, and considered only operations, machines, and available capacity. These time bounds were then communicated as constraints to the resource analysis level. The capacity analysis level introduced a hierarchical reasoning capability similar to that suggested by Simon (1962) and implemented by Sacerdoti (1974). While the results of this level were good, i.e., most constraints were satisfied, there were still problems with the satisfaction of the work in process constraint.

A third version of ISIS was then constructed which included a fourth level:

1. lot selection,

2. capacity analysis,

3. resource analysis,

4. reservation selection.

Instead of having resource analysis make reservations for resources such as machines and tools, it generated time bounds on when the resources should be made available to perform an operation. The fourth level, reservation selection, then looked at all the time bounds and made reservations for the resources which reduced the work in process time for the order.

In the rest of this chapter, the architecture of the final version of ISIS is described.

5.2. ISIS Problem-Solving Architecture Overview

The version of ISIS discussed in the rest of this chapter is ISIS-II. ISIS-II employs a hierarchical constraint-directed search to construct schedules. The search space is composed of four dimensions:

1. Scheduling method,

2. Analysis step,

3. Time, and

4. Rating.

The first dimension, scheduling, has four levels:

1: LOT SELECTION

2: CAPACITY ANALYSIS

3: RESOURCE ANALYSIS

4: RESERVATION SELECTION

Lots are scheduled sequentially[11]. The next lot to be scheduled is selected by the level 1 analysis, based on lot status, due date information, and priority class. This lot selection level searches the schedulable lot space, assigning to each lot a start time, finish time, and priority rating. A selected lot is then scheduled at the capacity level based on available machine capacity, and the lot's operation graph. Level 2 generates a graph of states, where each state represents a separate operation with an earliest start time and a latest finish time[12]. The output of the level 2 analysis is a set of constraints which bound the times that each operation for the lot may be performed. The resource analysis level performs an anchored beam search[13]. It generates time bounds on the required resources by using all available constraints, both local to the level and those generated at the capacity level. Level 3's search states represent alternative operations, machines, and operation times for each step in the lot's production process. Level 3 outputs to level 4 a set of time bounds for each resource required for the lot's operations. Level 4 then selects a reservation within the time bounds which reduce work in process time.

The second dimension, analysis, implements the "theory" of constraint-directed search. Each scheduling level is divided into three steps: pre-search analysis, search, and post-search analysis (figure 5-3). The purpose of pre-analysis is to analyze the problem at hand to determine how to bound the search space. If meta-planning module were available, as described in section 5.1, it could be determined which were the "major" constraints which would affect search at a particular level. These major constraints could act as operators, e.g, next-operation constraint, hence bounding the space of possible schedules. Such a module was not constructed. Instead, a rule based was devised and implemented in OPS5 (Forgy, 1979) which represented an expert systems approach to pre-search analysis. For example, there is a rule-base in the pre-analysis of the resource analysis level which selects the search operators, the direction of search, and may add additional constraints to the search.

[11]The intent was to reduce the complexity of the search, enabling the monitoring of a single order as it "weaves" its way through the plant. Parallel scheduling is being investigated in ISIS-III.

[12]The use of alternative operation paths was required in order to utilize the operational flexibility available in the plant.

[13]The investigation of opportunistic search was postponed to ISIS-III. It was anticipated that other "levels of analysis" (e.g., capacity analysis) could detect islands of certainty and guide the resource analysis around or through them.

The second step of analysis performs the search with the operators and other information provided by the pre-analysis step. It uses constraints selected in pre-analysis and found during the search to rate the generated states. These constraints guide the search by pruning and/or reducing the interestingness of states in the search space.

The third step is post analysis. The purpose of this step is to use constraints to determine whether the results of the second step are acceptable. The final states of the search step are rated according to how well their constraints are satisfied. If the ratings are unacceptable, the constraints can be examined to determine which are not satisfied, and corrective action taken. For example, feedback may be provided to pre-analysis to alter the search space. The implementation of this step is also rule based, with a more general constraint-directed diagnosis method suggested.

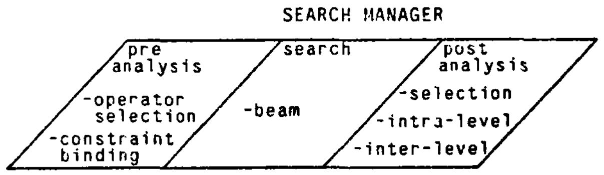


Figure 5-3: Phases of Reasoning at a Level

The third dimension, time, orders each state at each search level, according to the time of its expected implementation.

The fourth dimension, rating, orders each state in the search space according to its utility rating.

It would appear that ISIS-II's approach to scheduling utilizes an architecture which is quickly becoming a standard in AI (e.g., Hearsay-II, SU/x, SU/p, Molgen). What sets ISIS-II's approach apart, is that decisions at a higher level do not restrict decisions at a lower level, but constrain them. The difference is subtle. Decisions at higher levels in top down hierarchical reasoning systems do not allow search paths at lower levels that fall outside of those decided above. That is, they severly restrict "movement" at lower levels. But in ISIS-II, the information communicated between levels is in the form of constraints. Constraints guide (constrain) search but do not necessarily restrict search. And constraints may be relaxed if they prove to be overly constrictive. Hence, levels in ISIS-II may be loosely coupled through the appropriate specification of constraints.

The relaxation of constraints is performed in three ways: generative, analytic, and indirect. Generative relaxation is performed during the performance of search. Each operator embodies a single constraint. The operator generates alternative successor states by generating relaxations of the constraint it embodies. For example, a choose-machine operator embodies a machine choice constraint which specifies that an operation can have one or more machines upon which it is to be performed. The operator generates each of the choices in the constraint as part of its search.

Analytic relaxation is performed in the pre and post-analysis of a level's search. The rules in the analysis phase examine the available information and may decide to choose a particular relaxation of a constraint.

Indirect relaxation is the relaxation of constraints which may not correspond to parameters in the factory which can be directly controlled. Instead, choices of other constraints may indirectly determine the relaxations of these constraints. For example, a due date constraint may be indirectly determined by operation, machine and reservation choices. Hence, ISIS-II does not directly choose the value of the due date constraint, but it may be indirectly chosen through the analytic and generative relaxation of other constraints.

The rest of the chapter elaborates ISIS-II's architecture and provides an example of its use.

5.3. Level 1: Lot Selection

The ISIS-II problem-solving architecture was designed to investigate constraints and their effect on the search process. To simplify the analysis, it was decided to schedule one lot at a time, in priority order[14]. The problem of scheduling all available lots in parallel is being investigated in ISIS-III.

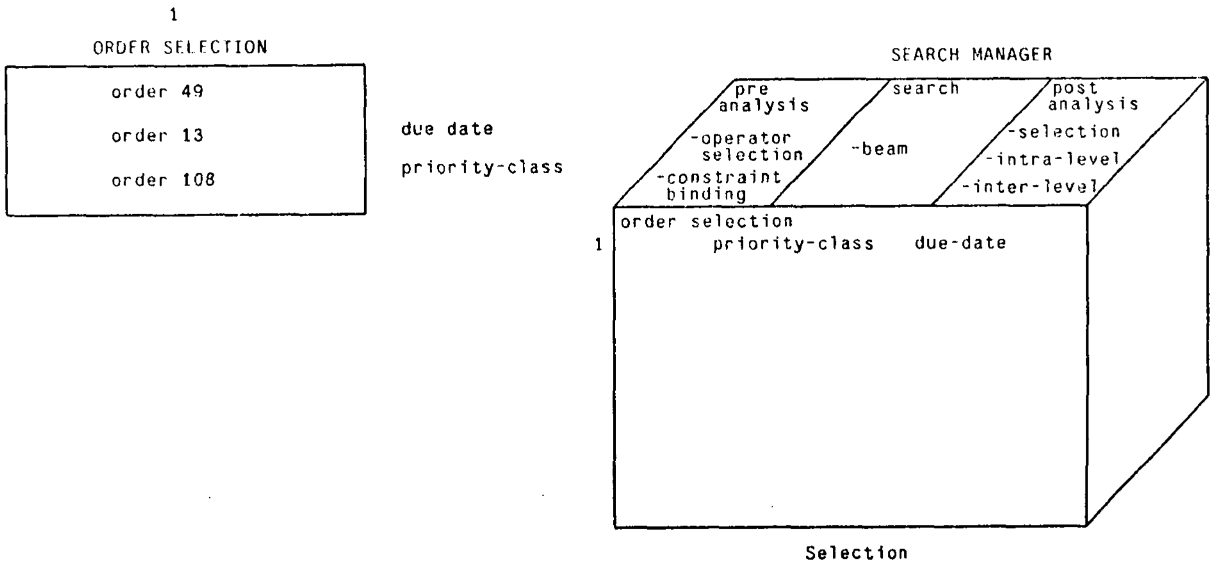


Figure 5-4: ISIS Level 1

The purpose of the lot selection level (figure 5-4) is to select from the set of known lots, which ones are to be scheduled and in what order. ISIS-II first examines the model of the

[14]Lots were manually generated from orders.

factory to determine what lots must be scheduled or re-scheduled. A lot may be posted for scheduling under the following conditions:

1. manual posting by human, (posted)
2. pre-emption of a reservation for a resource by another lot, (pre-empt)
3. manual alterations to an existing schedule resulting in invalid reservations, (invalid)
4. a previous attempt at scheduling failed. (search-failure)

Figure 5-5 defines a lot which has had a reservation pre-empted by another lot.[15]

```
{{ lot-xxxx
    { INSTANCE lot
        PRIORITY-CLASS: forced-outage
        PRIORITY: 20
        STYLE: #7J8924
        ROWS: 12
        DUE-DATE:
            range: {{ INSTANCE due-date-constraint
                                DATE: (450 0 0) }}
        STATUS: pre-empt } }}
```

Figure 5-5: Pre-empted Lot

Once the set of scheduable lots has been determined, they are partitioned by priority class.[16] Each lot within a partition is prioritized according to the nearness of their due date. Two approaches currently exist. The first approach being used, was derived from the human scheduler. It is a simple tardiness measure: the more tardy, the higher priority in its class. The second approach, which remains to be tested, is based on an "apparent urgency" heuristic developed in (Rachamadugu, 1982).

The lot selection level also performs a "cleaning" of the lot to be scheduled. It examines the information provided in the lot schema to assure that it is complete for scheduling. At this time, the system may add its own additional constraints to the lot. For example, if only a simple due date is specified, then the this level will add to it a constraint specifying its relaxations. A due date constraint is chosen by matching the lot's priority-class to one in the set of alternative due date constraints; each due date constraint specifies the priority-class for which it is defined.

[15] *An lot can only pre-empt a reservation for a resource by another lot if the latter lot's priority is lower.*

[16] The priority-class of a lot is equal to the highest priority-class of its orders.

5.4. Level 2: Capacity-Based Analysis

It became apparent during the standalone testing of level 3 that the horizon effect (Berliner, 1973), was increasing a lot's tardiness. Because of the resource analysis level's inability to generate enough alternative states, it was not able to "see" machine bottlenecks near the end of the schedule. Hence, what would appear to be a good partial schedule at the beginning would run into a bottleneck machine and stay queued for many weeks. To correct this form of the horizon effect, a capacity analysis level was created to perform a simple available capacity analysis to detect bottlenecks (figure 5-6). The output of this analysis being bounds on a lot's operation times.

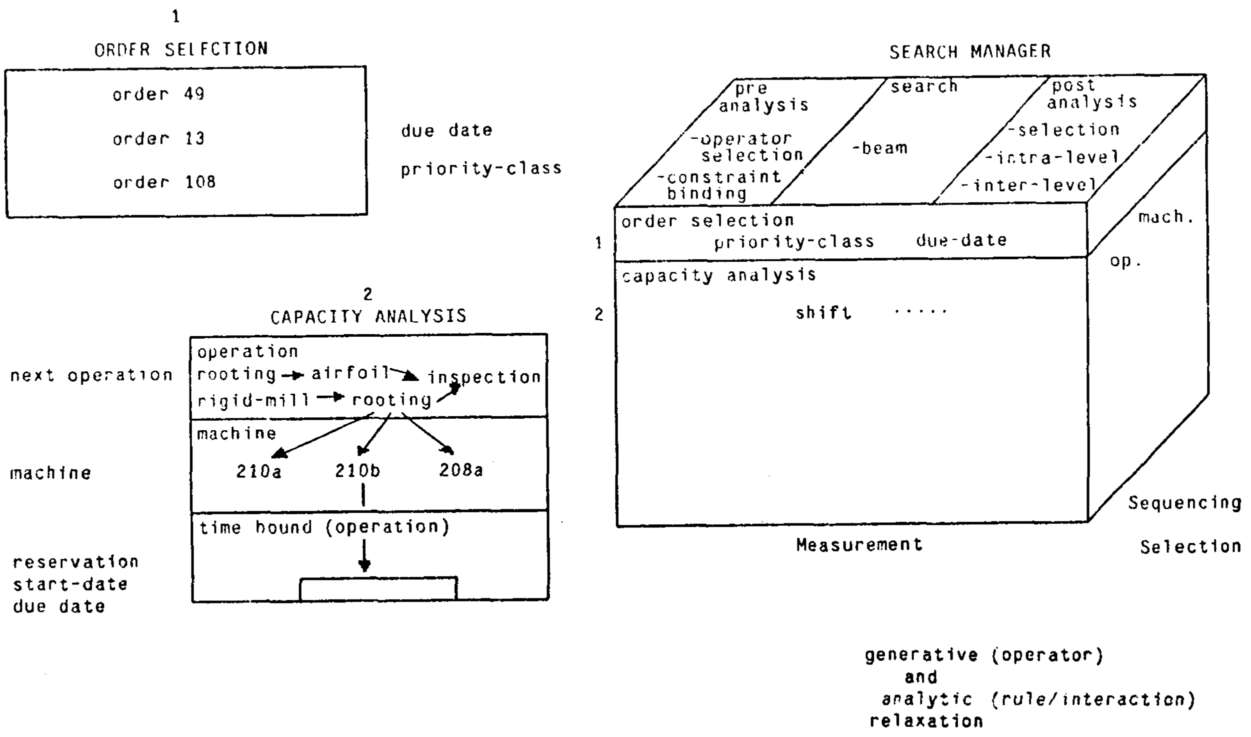


Figure 5-6: ISIS Level 2

The output of level 1 to level 2, is the highest rated lot to be scheduled next. Level 2 performs a capacity analysis of the shop in order to determine the earliest start and latest finish time for each operation in the operations graph of the lot. The capacity analysis bases its times on current reservations (loadings) for machines. Hence, is-able to detect existing bottlenecks in the shop.

This level is composed of two phases: a forward analysis and a backward analysis.

5.4.1. Pre-Search Analysis

At this level, pre-search analysis is simple, in the first phase it creates a set of initial states in the search space: states representing alternative first operations with their earliest start time set to the lot's requested start date. The forward search operator chosen to extend these states is a derivative of a critical path analysis. It performs the following:

- generates a state for each successor operation for the current state's operation,
- sets the start time of the successor state to the earliest time the parent operation may be completed, and its completion time based on available capacity,
- completion time for an operation is calculated based on existing reservations for alternative machines that may perform the operation. The operator looks for the earliest contiguous block of time in which the required operations may be performed. Hence, earliest start times and latest finish times reflect current loading and bottlenecks,
- checks to see if there already exists a state with the same operation in the search space. If one is found, and its finish time is greater than the new state's then the existing state is deleted. Otherwise the new state is deleted, i.e., the earliest finish time for an operation is maintained.

Once the first phase search and post analysis is performed, the system returns to pre-search analysis. In this second phase, a backward search operator is chosen to extend all the last operation states. It is similar to the forward search operator but work's backwards from the finish date to provide a latest finish time for each operation.

5.4.2. Search: Capacity Analysis

The search performed in each phase is a simple breadth first search using the single operator supplied in pre-search analysis. In the first phase, the operations graph is searched in the forward direction from the first operations to generate earliest start times for each operation in the graph. In the second phase, the operations graph is searched in the backward direction from the last operations to provide latest finish times for each operation in the graph.

5.4.3. Post-Search Analysis

In the current implementation of ISIS-II, post-analysis is only performed after the second phase. It simply generates time bounds for each operation based on the earliest start time and latest finish time. These constraints are passed to level 3.

Obviously, a more detailed analysis may be performed. The following are additions under consideration:

- A bottleneck measure for each machine for each month/week will be derived based on the ratio of queue length to available capacity. If the ratio is high, then either more shifts, or more machines will be made available.

- If the utility of the earliest start time for the lot, as determined by the due date constraint, is low, then the capacity of bottleneck machines used by the lot will be selectively increased during the period that the lot is in process.

- Pairwise swapping of reservations may occur if another lot in the queue has more slack.

5.5. Level 3: Resource Analysis

The purpose of level 3 is to perform a detailed search utilizing all known constraints and required resources (figure 5-7). The search space at this level is quite large, hence the method of search was chosen to minimize search time. The goal of this level is to perform a "machine-centered" search, i.e., directed by the operations graph, and to generate reservation time bounds for the machine and resources required by the scheduled operations.

5.5.1. Pre-Search Analysis

ISIS-II does not search the problem space in the manner detailed in section 5.1. The majority of lots exhibit a high enough degree of regularity, to allow the creation of a set of rules which examine an lot and determine its search dimensions. For example, rule **forward-search** (figure 5-8) states that if the lot is of type "shop-order" and its schedule-status is "schedule", then it is to be scheduled in a forward direction from the its start date, and its operators are to search the operation, machine, and queue position dimensions.

```
{{ forward-search
    { INSTANCE rule
       IF:
                 lot.type = shop-order
                 lot.schedule-status = posted
                 lot.start-date NEQ nil

       THEN:
                 lot.direction <-- forward
                 lot.start-states <-- first operations
                 lot.operator <-- choose-operation, choose-machine,
                                choose-queue-pos      }    }}
```

Figure 5-8: **forward-search** Rule

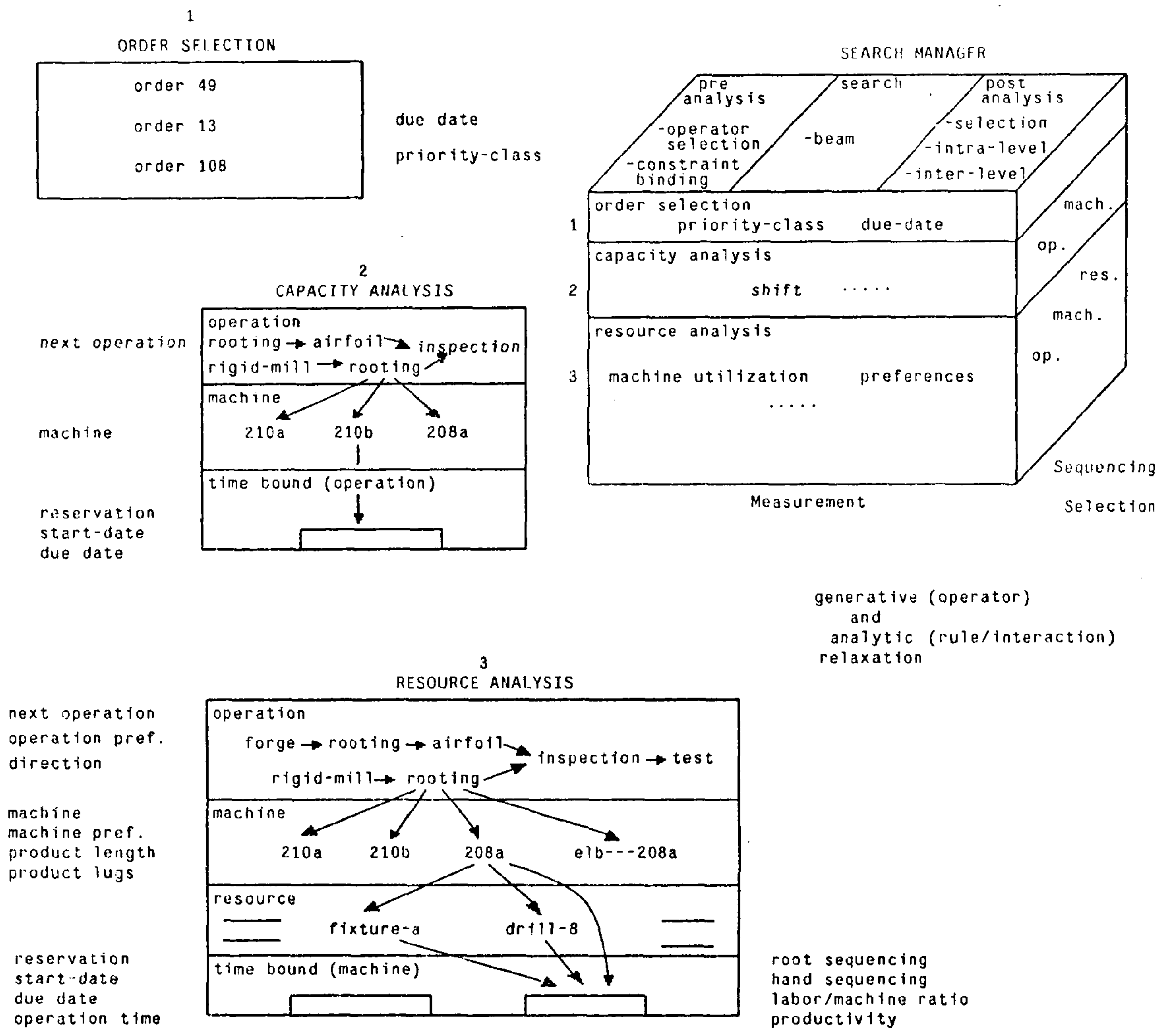


Figure 5-7: ISIS Level 3

Rule (figure 5-9) states that if the lot had a reservation pre-empted or a reservation is invalidated, then scheduling is to take place in the forward direction from the operation previous to the pre-empted operation. And that the operators that define the search space are the operation, machine, and queue elaboration operators mentioned earlier.

```
{{ pre-empt
    { INSTANCE rule
        IF:
            lot.schedule-status = (pre-empt OR invalid)

        THEN:
            lot.direction <-- forward
            lot.start-State <-- (previous-operation of invalid reservation)
            lot.operator <-- choose-operation, choose-machine,
                choose-queue-pos    }    }}
```

Figure 5-9: Pre-empt Rule

All lots, if posted, pre-empted, or invalidated are restricted to searching the operation, machine, and queue dimensions. The search space is expanded only when search in these dimensions have failed.

Consider **lot-xxxx** which was just entered and posted for scheduling. Pre-search analysis would set the lot so that the scheduling direction is backwards, and the initial search states are the last operations that may be performed (figure 5-10). The direction was chosen based on the lot's PRIORITY-CLASS. A "forced-outage" classification implies that the due-date of the lot is the most important constraint, i.e., the due-date is a major anchor in the search space.

```
{{ lot-xxxx
    { INSTANCE lot
        PRIORITY-CLASS: forced-outage
        PRIORITY: 7
        STYLE: #7J8924
        ROWS: 12
        DUE-DATE:  {{ INSTANCE due-date-constraint
                          DATE: (450 0 0) }}
        STATUS: posted
        SCHEDULING-DIRECTION: backward
        INITIAL-SEARCH-STATE: {{ INSTANCE search-state
                                    OPERATION: finish-test }}
        } }}
```

Figure 5-10: Posted Lot

The analysis of lots whose previous attempt at scheduling failed is a little more complex. If the lot of priority class "forced outage", and a backward search from the due-date was attempted but failed, then the system will try again by relaxing the due-date with limits set by the due date constraint. If this in turn fails, then the system must take a more drastic step by increasing the search space. In this case another operator is added which elaborates the available machine shifts dimension only at the point where the lot is being processed.

As mentioned above, the problem space analysis rules have not been generalized to the point that they carry out the general analysis of what the search space operators should be. The current approach is more in common with "expert systems". ISIS-III will incorporate the more general approach.

5.5.2. Search: Constraint-Directed Scheduling

ISIS-II performs a beam search in a collapsed, rated space. A state is rated by the constraints that apply to it. The rated space was collapsed because none of the strata are dominant. A beam search was chosen for two reasons: 1) the space is heavily anchored at either the start-date or the due-date, and 2) opportunistic searches will be investigated in ISIS-III.

5.5.2.1. States

ISIS-II performs a beam search of the state space of partial schedules. It starts by taking the initial states generated in pre-search analysis, and extends these states by the operators designated in that phase. After each application of an operator, the generated states are rated by applying the constraints, and only the best "n" states are kept for the next iteration of operator application (i.e., beam search). A search state is defined by an operation, machine, and queue position choice (figure 5-11).

```
{{ search-state
     LOT:
     PREVIOUS-STATE:
     NEXT-STATE:

     OPERATION:
     MACHINE:
     START-TIME:
     END-TIME:

     CONSTRAINT:
     RATING:
     PRE-EMPTED-LOTS:        }}
```

Figure 5-11: search-state Schema

The **search-state** defines a single point in a proposed process routing for an lot. A complete process routing is determined by following the the PREVIOUS-STATE links to all the states that lead up to the current state. Each state also contains the rating of the state, and the constraints, including importance and utility, that composed the rating.

5.5.2.2. Operators

Operators are the generative interpretation of constraints. An operator embodies one or more constraints, and uses them to create new states by generating relaxations of the constraint(s) they represent. The selection of an operator in pre-analysis is the selection of the dimensions along which relaxations will be investigated via search. The three most common operators chosen by pre-search analysis to generate the search space are choose-operation, choose-machine, and choose-queue-pos. Operators are represented as schemata in ISIS-II, but the contents of the IF and THEN slots are lisp functions. These functions are not derived automatically from constraints, but are written by hand. For simple constraints, it is a simple task to derive the operators automatically, but compound constraints are harder[17].

The first operator searches the operation dimension (figure 5-12). It represents the constraint of which operations may precede or follow the an operation.

```
{{ choose-operation
    { INSTANCE search-operator
        IF: "a state has its operation, machine, and reservation bound"
        THEN: "create a state binding the operation for each operation that follows,
                in the proper direction, the current state's operation" } }}
```

Figure 5-12: operation-operator Schema

In a similar manner, the choose-machine operator represents the constraint of the choice of machines that may implement the operation. It will generate a separate state for each machine that may perform the bound operation.

```
{{ choose-machine
    { INSTANCE search-operator
        IF: "a state has only its operation bound"
        THEN: "create a state binding the machine for each machine associated
                with the state's operation"
```

Figure 5-13: machine-operator Schema

[17] The automatic derivation of operators from constraints has also been postponed till ISIS-III.

The choose-queue-pos operator will generate a separate state for each queue position the lot can feasibly occupy in the machine's queue. This can be viewed as a compound operator in that it considers more than one constraint in making its decision. It examines the reservation constraints on a machine, it examines the reservations for the materials, tools, etc. required for the operation, and it examines the priority of the lots holding reservations for the resources it requires. There exists more than one reservation operator. The choice of operator is currently made by the user, and is specified in the system profile. The choose-queue-pos operator embodies the following constraints:

- Whether to pre-empt the queue position of any lot of lower priority.
- Whether to pre-empt an lot already running on a machine.
- Whether to begin processing of the lot on the next machine before it has finished on the current machine.
- What time in a free time block to reserve.

Ideally the problem space search phase of ISIS-II should have rules which determine which of these constraints should be applied. Currently, the choice of reservation operator is made by the user. Two types of choose-queue-pos operators have been tested in ISIS-II: the "eager-reserver", and the "wait-and-see-reserver".

Eager Reserver. The eager reserver operator was used in the first two versions of ISIS. It generates alternative successors to a state that has both its operation and machine chosen, but does not have a time reservation. Given an arrival time, which is the finish time of its previous operation, it generates blocks of time, from its arrival time on, in which the machine is free or machine reservations exist for lots having lower priority. It also considers the operation resource availability during the block of time. A resource may have its availability constrained in one of two ways. A resource may be possesed by some possessor in what is called a "destroy" manner. That is, if a reservation is pre-empted or split, then the entire reservation is invalid, hence no work during the possession is accomplished by the original possesor. Hence, any resource of possession type "destroy" that is not available during the block, splits the block into two at its point of inavailability. The second type of availability is called a "delay" possession. If the resource is reserved by an possession of type "delay", then the reservation may be split, without any loss to the original reservation. Hence, any resource of type "delay" only extends the time that the lot occupies in the block if it overlaps with the time during which the operation is to be performed. That is, the inavailability of the resource during the time block only delays, instead of destroys. the performance of the operation. For each block of time, which is large enough for the operation to be performed, it creates a state with a start and finish time at the earliest possible time in the block. Hence, the eager reserver jumps to use the machine and its ancillary resource requirements as soon as possible within a time block.

Wait and See Reserver. The wait and see reserver is similar to the eager reserver. It differs in the time it reserves in a block. This operator would like to wait, before choosing its reservation time, in order to see what times are available further on in the schedule; there is no need to reserve a machine too early if it will sit in a queue later in the schedule. The "eager-reserver", in the case of bottlenecks, would result in increased work in process time, i.e., sitting in queues waiting for a machine to become available because it finished an earlier operation too soon. The wait-and-see-reserver solves this problem by "tentatively" reserving the entire time block, and then leaving the choice of actual reservations of resources to level 4. The result is reduced work in process because reservations can be shifted as close together as allowed by the time block.

5.5.2.3. Constraint Resolution

The key part of the search is the application of constraints in rating a search state (partial schedule). The rating of a state can be divided into two parts: resolving what constraints should be applied to the state, and applying the constraints to the state.

Local Resolution. As the search proceeds, states are generated which vary widely in their choice of operations, machines, and queue positions. Not all constraints in the system may be relevant in rating the state (partial schedule) in question. For example, each machine may have zero or more constraints associated with it. These constraints should be considered only when the corresponding machine is part of the schedule being rated.

ISIS-II dynamically resolves the set of applicable constraints at each search state. Constraints may originate from four sources:

Model-Based: Constraints may be embedded in any object or process in the factory model. For example, there may be physical constraints associated with a machine, sequencing constraints associated with an operation, queue ordering constraints associated with certain work centers.

Hierarchical Imposition: Constraints generated by other levels of analysis will be incorporated.

Lateral Imposition: Constraints can also be propagated laterally during the search. A decision made earlier in the elaboration of a schedule may result in a constraint being attached to the lot that restricts a choice point further on in the search.

Exogenous Imposition: The user may also create and implant constraints. These constraints can be attached to anywhere in the model, or be globally attached so that it is considered at each search state.

The above constraints are deposited at the appropriate points in the model, i.e., at the lot, operation, machine, etc. Hence, actual resolution is performed by examining each schema (i.e., operation, machine, etc.) in the current state description. The contents of any CONSTRAINT slots, or constraints attached to any slots which enable the schema are added to the local resolution set. After the local constraint set is resolved, ISIS-II filters the set by performing the evaluation described in section 4.6.5.

Global Resolution. Unlike some simple game tree searches, the path which leads to a search state, is as important as the state itself. Each state in the path defines one more operation, machine, and queue bindings for the lot. And a complete schedule is defined by the path from the initial state to the end state in the search space. Local resolution, as defined above, resolves what constraints affect only the state under consideration. But the rating of a state is a rating of the partial schedule up to the current state, and not the single choice represented by the state. Hence, the rating of a state must include not only the local constraints but the constraints applied to all the states along the partial schedule ending at the current state. For example, the blade length constraint locally resolved at state-n is included with the local states when rating state-n + 1.

It would appear that the appropriate action would be to rate a state using all of the constraints associated with all the states in the partial schedule represented by the state. Hence, the constraints associated with a state are those locally resolved, and the local constraints along the path from the current state to the initial state. This turns out to be incorrect. Consider the **due-date-constraint** (figure 4-20). It is a classic evaluation function as defined in heuristic search. Part of the constraint calculates the work-in-process time of the lot to the current state, and the other part *predicts* the remaining work-in-process time to the end state. Each time the constraint is applied, it is a better estimator of the work-in-process time, and should overide applications of the same constraint earlier in the partial schedule. On the other hand, the **queue-stability** constraint is applied at each state which binds a queue position. It rates the state by how much it destabilizes existing queue reservations. The greater the destabilization, the lower the rating. This constraint measures a decision made at that state, and remains invariant over future states, since any future states cannot affect an earlier state.

Constraints are classified into two categories: *invariant* and *transient*. When ISIS-II rates a state, it collects all the invariant constraints along the path from the initial state to the current, and includes them in the rating. ISIS-II also gathers up all the transient constraints, but does not retain duplications. Only the latest instantiation of a transient constraint (closest to the current state) is saved. The union of invariant and transient constraints form the constraint set for the state under consideration.

Relative Resolution. As discussed in the section on constraint differentiation (section 4.4), all constraints are not created equal. Some constraints are more important than others depending upon the lot being scheduled and the status of the factory. Relative resolution differentially interprets the resolved constraints by partitioning the constraint set according to the applicable scheduling goal. As described in section 4.4, a scheduling goal partitions the constraint set and supplies an importance to each partition. The importance is then uniformly divided amongst the constraints in the partition.

5.5.2.4. Constraint Application

The first step in constraint application is deriving the utility of each constraint. The utility reflects the "goodness" of the choice specified by the constraint. Each constraint is procedurally interpreted by applying the contents of its TESTER slot to two parameters: the state, and the name of the constraint schema. Hence, the combination of the state and the constraint provide all the information necessary for the constraint's utility to be derived. That is, the constraint is interpreted in the context of the current state, and looks for the schemata (information) it requires within only the current state.

Now that both the utility of the constraint, and its importance have been derived, ISIS-II computes a rating for the state by computing the importance (weighted) average of each constraint's utility. The state stores each constraint's importance and utility at the state for later use in explaining a schedules search path.

Once all the current states have been rated, ISIS simply throws away all but the top n (beam search) and repeats the search process.

5.5.3. Post-Search Analysis

As the search proceeds, ISIS-II continually tests to see whether the goal state has been generated, and/or the search has died without finding a solution. If either of these has occurred, post-search analysis is entered. Post-search analysis is accomplished by a set of rules that analyze the results in order to determine whether:

- the system found a satisfactory solution.
- the system should continue searching if the solution is unsatisfactory.
- another search strategy should be pursued.

Consider the example of **lot-xxxx** (figure 5-10). It is of priority-class "forced outage". A forced outage blade must be shipped by its due date, if not sooner. ISIS-II first attempts to schedule it backwards from its due date. If a feasible schedule is not found, i.e., it runs out of time before it runs out of operations, it then resets the lots scheduling parameters to schedule forward from the "today". If the results of this scheduling attempt are poorly rated, then it marks the lot as having failed again, and posts it for re-scheduling. This time, the pre-analysis rules add another operator which searches the shift dimension for a machine. As in pre-analysis, these rules represent a portion of "expert" post-scheduling analysis knowledge. The following defines extensions to ISIS-II which are being implemented.

5.5.3.1. Wave Search

The current approach to post-analysis views the search step as a single, indivisible action; once the search is complete, it cannot be re-started from partial state. Yet, post-analysis does have enough information to detect which states in the beam search while rated highly at their point of creation, led to poorer states later on.

One approach that would make use of this information is a *wave search*. A wave search is a derivative of beam search but with the added capability of performing a dependency analysis of states, resulting in a backtracking-like behavior. The following describes the wave search algorithm:

1. Define wave 1 as containing all initial states. Set the current wave number, j, to 1.

2. Create wave j+1 by extending the top n rated states in wave j through the application of all applicable operators. Do not delete any states in wave j.

3. Apply the stop search algorithm (e.g., the top n states in wave j are all goal states). If the search is not complete then set j = j+1 and go to step 2.

Once the search has been completed, post-analysis performs a dependency analysis starting from the last wave. The following describes the analysis.

1. Set j to the last wave generated.

2. For each state in wave j set its *expected actual utility* (EAU) to its rating.

3. If j = 0 then the analysis has failed to detect reasonable retreat.

4. For each state in the top n states in wave j-1, calculate its EAU by the waited average of the EAU of its children states. And for each state not in the top n, set its EAU to its rating.

5. Compare the EAU of the top n states in wave j-1 to the EAU states in the same wave which were not in the top n. If none of the latter states have a "significantly" higher utility, then set j = j-1 and go to step 3.

6. If some of these latter states have a "significantly" higher utility, then replace the lowest states in the top n with these n states and start the search again from this point.

In the search phase, a wave is defined as all possible successors of the top n states of the parent wave. The system will save all the states in a wave for post-analysis. The post-analysis phase of the wave search derives an expected average utility for each of the top n states in a wave by averaging the children EAUs. The resulting EAU is a measure of the expected utility of all schedules that emanate from the state. This is in contrast to the rating assigned to a state. The rating reflects the average utility of all the constraints associated with the partial schedule embodied in the state, and in some cases, a prediction of the utilities of future constraints to be found in the extensions of the state (partial schedule). Hence, the EAU contains "more" perfect information. The result of this analysis is the removal from certain waves the states that had a high "predicted" utility (i.e, rating), but whose resultant EAU was low. The wave may then proceed forward (generating new waves), extending a new set of states which may contain better schedules.

5.5.3.2. Constraint-Directed Diagnosis

A second post-analysis technique (and complementary to wave search) relies on a more theoretical approach to constraint analysis. It supercedes an expert systems approach in that it has "deep" knowledge of what constraints are and how they interact with other constraints. The approach is as follows:

As described in the section 4.5, constraints may be related by a **constrains** relation. These relations define how the selection of a value for a constraint may affect another constraint's selection. Reviewing the form of the **constrains** relation:

```
{{ constrains-
     DOMAIN:
     RANGE:
     DIRECTION:
     SENSITIVITY:    }}
```

Figure 5-14: constrains Relation Schema

This relation defines whether an increase in the utility of the domain constraint will positively or negatively affect the utility of the range constraint. The SENSITIVITY slot defines how sensitive the RANGE's utility is to a change in the DOMAIN's utility.

The method of analysis proposed here is similar to the methods of diagnosis found in systems such as Mycin (Shortliffe, 1976). Prospector (Duda et al., 1978), and PDS (Fox et al., 1983). Constraints and their "constrains" relations form a network. Each constraint can be viewed as having two utilities: a *direct utility* derived from the utility of its relaxation, and a *indirect utility* derived from the constraints constrained by it (i.e., the range schemata for which it is the domain of a **constrains** relation). The indirect utility of a constraint is the sum of the indirect utilities of its range constraints, weighted by the relation's sensitivity and the range constraint's importance.

$$\frac{\sum_{i \text{ member } R} I_i S_i U_i}{\sum_{i \text{ member } R} I_i}$$

where

R = set of constraints in range of constrains relation emanating from constraint being rated.

U_i = indirect utility of constraint i in R

S_i = sensitivity associated with constrains relation linking ith constraint in R

I_i = importance of the constraint i in the R

The procedure for calculating a constraint's indirect utility is:

1. starting with with the leaves of the graph (i.e., constraints which have an empty R set), set their indirect utility to their direct utility.
2. for each constraint which has a non-empty R set, and all members of the set have a non-nil indirect utility, use the above formula to calculate their indirect utility.
3. If all constraints have an indirect utility, then stop, otherwise go to 2.

The resultant indirect utility represents the indirect effect of the constraint's value selection. A decision can now be made as to which constraints to relax, based on the smallness of their indirect utilities.

Selecting a constraint to relax also selects the level in ISIS-II's hierarchy in which to perform the relaxation. Constraints form a hierarchy when structured by **constrains** relations. This hierarchy can be used to derive the levels of scheduling. For example, the following is one such hierarchy. A constraint at one level constrains the values of some or all constraints on the next level lower. Cost constraints affect how many shifts are going to be used. Selection of shifts constrains how well due dates and other time constraints are going to be met.

Level	Constraints
Lot Selection	priority class
Advance Planning	cost quality
Capacity Analysis	shift operation alternatives machine alternatives
Detailed Scheduling	due date work in process start date

Figure 5-15: Constraint Interaction Hierarchy

5.6. Level 4: Reservation Selection

The fourth level (figure 5-16) takes as input time bounds on each of the resources required by the operations in the operation sequence selected at level three. It selects a time reservation for each of the resources which attempts to minimize the work-in-process time of the lot.

5.7. An Example

To show how ISIS-II operates, the construction of a schedule at level 3 (detailed scheduling) will be analyzed. Due to the number of alternative schedules that can be produced, only one path will be traced through the solution space, noting alternatives generated along the way.

Consider the forward scheduling of a **7J8924** blade (figure 5-17).

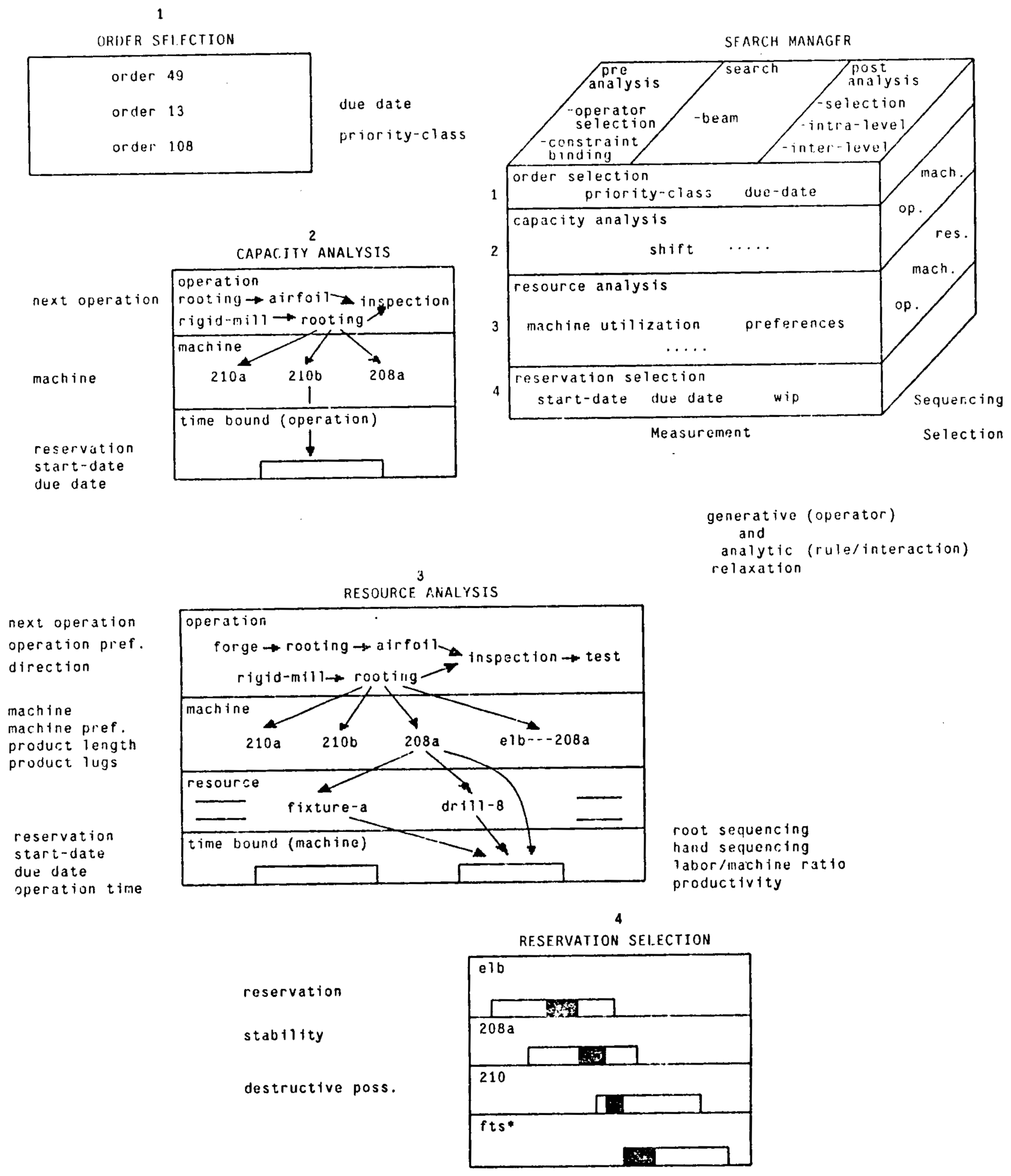


Figure 5-16: ISIS Level 4

```
{{ 7J8924
     { INSTANCE CSE
          ROW-QUANTITY:     120
          MANUFACTURING-QUANTITY:     130
          ROOT-WIDTH:
          FOIL-LENGTH: 31
          LUGS:
          TENON:
          STRIP:     t
          PEEN:      t     }     }}
```

Figure 5-17: Blade 7J8924

The 7J8924 is a CSE blade (figure 5-18). It inherits its operations graph from CSE (figure 5-22).

```
{{ CSE
     { IS-A blade
          FIRST-OPERATION: forge rigid-mill
               Constraint: cse-forge-preference
          LAST-OPERATION: finish-test
          MATERIAL-TYPE: stainless-steel
          ROOT-TYPE: curve   }     }}
```

Figure 5-18: CSE Blade

lot-7J8924 represents the lot for the blade type.

```
{{ lot-7J8924
    { INSTANCE lot
        PRIORITY-CLASS: forced-outage
        PRIORITY: 7
        STYLE: 7J8924
        ROWS: 10
        DUE-DATE:  {{ INSTANCE due-date-constraint
                            DATE: (450 0 0) }}
        STATUS: posted
        SCHEDULING-DIRECTION: forward
        INITIAL-SEARCH-STATE: {{ INSTANCE search-state
                                   OPERATION: forge }}
                             {{ INSTANCE search-state

                                   OPERATION: rigid-mill }}
        } }}
```

Figure 5-19: lot 7J8924 Schema

The presearch analysis decides to schedule the forced-outage in a forward direction because of a lack of sufficient lead time. It creates two initial search states, one for each of two alternative first operations: **forge** (figure 5-20) and **rigid-mill** (figure 5-21)[18]. These first two search states provide alternate bindings of the operation slot. They do not specify either the machine, nor the process times.

[18] The representation of these operations is altered, i.e., explicit specification of enablement is replaced by simply listing the names of the resources required, in order to reduce the length of the example.

```
{{ forge
    { INSTANCE operation
        NEXT-OPERATION: inspection-1
        SETUP-TIME: 6
        PIECE-TIME: 0.2
        MACHINE: cse-forge-line   }    }}

{{ cse-forge-line
    { INSTANCE machine
        CAPACITY: 1 }    }}

{{ cse-forge-preference
    { IS-A operation-preference
        OPERATION: forge
        TRUE-UTILITY: 1.5
        FALSE-UTILITY: 0.8 }    }}
```

Figure 5-20: Forge Operation

The initial constraint set for the forge operation includes:

```
            7J8924-due-date               ; from lot-7J8924
            cse-forge-preference          ; from CSE blade
```

The context of the due-date constraint tests the presence of a start and finish time binding. Since only the operation is bound, the context fails, and the 7J8924-due-date constraint is removed from the constraint set. The cse-forge-preference context evaluates to true, so that it remains in the constraint set. Since there are no earlier, parental states, this comprises the constraint set. The forced-outage goal (figure 4-23) is then applied to prioritize them. There being only one constraint, it receives full priority. Evaluating the cse-forge-preference, it returns a rating of 1.5, which, in turn, becomes the rating of the search state.

```
{{ rigid-mill
    { INSTANCE operation
        NEXT-OPERATION: inspection-1
        SETUP-TIME: 2
        PIECE-TIME: 4
        MACHINE: rigmill
        LATERAL-CONSTRAINT: no-airfoil-op  } }}

{{ rigmill
    { INSTANCE machine
        CAPACITY: 4 }    }}

{{ No-Airfoil-Op
    { IS-A operation-requirement
        OPERATION: (TYPE is-a airfoil-operation)
        TRUE-UTILITY: 0
        FALSE-UTILITY: 1  }    }}
```

Figure 5-21: Rigid Mill Operation

The initial constraint set for the rigid mill includes:

```
7J8924-due-date           ; from the lot-7J8924
cse-forge-preference      ; from CSE blade
```

As before, the context of the 7J8924-due-date fails, leaving only the cse-forge-preference. It evaluates to 0.8 since the bound operation is not the forge. The choice of a rigid-mill operation results in the addition of the no-airfoil-requirement to the LATERAL-CONSTRAINTS slot of the search state. Lateral constraints are inherited by a state's siblings.

Once all the generated states have been rated, the beam search orders, and keeps only the "n" highest rated states. Since n is larger than 2, both states remain in the search space.

In the next iteration of the scheduler the states in the search space are extended through the application of all applicable operators. Since each specify an operation, and no machine or process times, the only operator applicable is choose-machine (figure 5-13). It takes a state and generates new sibling states with the machine bound to a member of the operation's machine set. Extending the state bound to the forge operation, results in the machine being bound to **forge-line** machine.

```
{{ state-3
    { INSTANCE search-state
        PARENT: state-1
        OPERATION: forge
        MACHINE: forge-line }     }}
```

The constraint set for this state does not have any locally specified constraints. Taking the union of the local constraint set, and its parent constraint set, state-3's constraint set is composed only of the cse-forge-preference. Its evaluation maintains the rating of 1.5 for the state.

```
{{ state-3
    { INSTANCE search-state
        PARENT: state-1
        OPERATION: forge
        MACHINE: forge-line
        CONSTRAINT-SET: cse-forge-preference
        RATING: 1.5 }     }}
```

State-3 is extended by applying the choose-queue-pos operator. For simplicity sake, we will assume there does not exist any reservations for the machine. Hence, a single state is generated for the machine with the reservation spanning the time (445 0 0) to (445 ? ?).

```
{{ state-4
    { INSTANCE search-state
        PARENT: state-3
        OPERATION: forge
        MACHINE: forge-line
        START-TIME: (445 0 0)
        END-TIME: (445 ? ?) }     }}
```

The local constraint set contains only the **7J8924-due-date**. Its context evaluates to true because the state contains a start and end time. Combining this with the constraint set of **state-3**, the following constraints are applicable:

```
cse-forge-preference
7J8924-due-date
queue-stability
```

The preference constraint is an example of an invariant constraint. Its rating is incorporated (at a locally determined priority) in every state from the point of its introduction. The due date constraint is a transient constraint. Its rating is included in every state from its point of introduction until introduced again. The new rating at the point of re-introduction overrides the previous rating[19].

The **queue-stability** is also an invariant constraint. It measures the instability in schedules introduced when a lot pre-empts another lots reservation in a queue. This constraint originates in the general **machine** schema.

The due date constraint evaluates a state by computing the expected time to complete the lot, and returning the utility of that date as specified by the utility function. In this case, the expected completion is determined by using average lead-time statistics.

```
{{ state-4
    { INSTANCE search-state
        PARENT: state-3
        OPERATION: forge
        MACHINE: forge-line
        START-TIME: (445 0 0)
        END-TIME: (445 ? ?)
        INVARIANT-CONSTRAINT-SET: cse-forge-preference
                        queue-stability
        TRANSIENT-CONSTRAINT-SET: 7J8924-due-date }    }}
```

The constraint set for **state-4** is partitioned into two sets. A weight of 0.9 for **7J8924-due-date** and 0.05 for each of **cse-forge-preference** and **queue-stability**. If the due date has a rating of 1.2, and the stability a rating of 1 (no pre-emptions), then the rating for the state is (0.9 * 1.5) + (0.05 * 1.5) + (0.05 * 1) = 1.205.

The system continues by extending **state-4** using the choose-operation operator. The next operation is **inspection-1**. Separate states are generated for choosing the machine(s), and queue position(s).

```
{{ inspection-1
    { INSTANCE operation
        PREVIOUS-OPERATION: forge rigid-mill
        NEXT-OPERATION: kt-line rotary-line root-line    } }}
```

After the **inspection-1** operation is fully specified, then three alternative states are generated for the possible next operations: **kt-line**, **rotary-line**, and **root-line**.

[19] In the case of the due date constraint, any re-introduction provides a more certain rating.

```
{{ kt-line
    { INSTANCE operation
        PREVIOUS-OPERATION: inspection-1
        NEXT-OPERATION: root-inspection } }}

{{ rotary-line
    { INSTANCE operation
        PREVIOUS-OPERATION: inspection-1
        NEXT-OPERATION: root-inspection } }}
```

```
{{ root-line
    { INSTANCE operation
        PREVIOUS-OPERATION: inspection-1
        NEXT-OPERATION: root-inspection } }}

{{ root-inspection
    { INSTANCE operation
        PREVIOUS-OPERATION: kt-line rotary-line root-line
        NEXT-OPERATION: xlo-airfoil 910-airfoil } }}
```

At his point, if the **rigid-mill** based schedule was being extended, this operation would not be generated. The lateral constraint attached at **state-2** would force the scheduler to skip over this and the **airfoil-inspection** operations.

```
{{ xlo-airfoil
    { INSTANCE airfoil-operation
        PREVIOUS-OPERATION: root-inspection
        NEXT-OPERATION: airfoil-inspection
        MACHINE: wmf1 wmf2 wmf3 wmf4 wmf5 wmF6 } }}

{{ wmf1
    { IS-A machine
        SHIFTS:          3
        PRODUCTIVITY:     0.81
        LAB-MACH-RATIO:   0.5
        RESERVATION:     lot-22-res
                        lot-48-res
        COST-CENTER:     4608
        CONSTRAINT:      wmf-length-requirement
                        wmf-lug-preference    } }}

{{ 910-airfoil
    { INSTANCE airfoil-operation
        PREVIOUS-OPERATION: root-inspection
        NEXT-OPERATION: airfoil-inspection } }}

{{ airfoil-inspection
    { INSTANCE airfoil-operation
        PREVIOUS-OPERATION: xlo-airfoil 910-airfoil
        NEXT-OPERATION: finish-test } }}
```

The last operation in the search is **finish-test.**

```
{{ finish-test
    { INSTANCE operation
        PREVIOUS-OPERATION: airfoil-inspection } }}
```

Figure 5-22: Example Operation-Graph

5.8. Incremental Scheduling, Interactive Scheduling, and Explanation

ISIS-II incrementally schedules lots. As the status of the plant changes, i.e., new lots are received and scheduled lots have their reservations invalidated, ISIS-II schedules or re-schedules the affected lots, and may pre-empt the reservations of lots of lower priority. Those lots whose reservations are pre-empted are also re-scheduled.

ISIS-II also provides an interactive scheduling capability. By means of a gantt chart interface the user can create reservations, or alter reservations made either manually or automatically. When a reservation is made for resources, ISIS-II checks the attached constraints and provides feedback to the user as to their satisfaction. Interactive scheduling may proceed at any level of abstraction contained in the operations graph. In addition, the interactive system can infer and create any missing reservations based on a standard time analysis, and time constraints provided by the existing reservations (see Smith (1983)).

A simple explanation facility is also provided. Attached to each constraint is an english phrase which describes the constraint. The explanation combines the constraint's description with a simple description of how well the constraint was satisfied. Since constraints represent much of the knowledge in the search, describing a schedule by describing its attached constraints, is quite informative.

5.9. Conclusion

This chapter described ISIS's approach to constraint-directed scheduling. In particular, constraint-directed scheduling is composed of:

- multiple levels of representation which are coupled by constraint passing. The constraints passed between levels represents the knowledge gained from the search passed at each level.

- pre-analysis of each level. The rule base in the pre-analysis of each level examines the search problem and bounds the search space by selecting both operators and constraints to apply.

- constraint-directed search in which constraints are used to rate states in the search space much like a classical evaluation function.

- constraint-directed diagnosis. The rule base in post-analysis uses constraints to detect the acceptability of schedules. Additionally, constraints can be used to detect incorrect decisions and the level at which they occurred, hence enabling the system to back up to the appropriate level of search.

The "theory" of constraint-directed search has only begun to be elaborated. There are pieces of ISIS-II which have yet to be tested. Their are pieces for which alternative mechanisms exist, e.g., using a MYCIN-like combining function for constraint utilities in rating states. and they also have yet to be tested. Though some analysis has been peformed (see the next section), much work remains to be done.

Chapter 6
System Evaluation

Summary

Two series of experiments are performed, all based on a portion of the turbine plant defined by the plant scheduler. Each experiment in a series, tests a different combination of constraints, scheduling direction, search operators, and levels of reasoning. One experiment removes the capacity analysis and reservation selection levels. The resulting schedule displayed high tardiness (65 of 85), and high work in process times with a makespan of 857 days. This is due to horizon effect caused by the coupling of beam search and insufficient machine capacity. The experiment which included the capacity analysis and wait-and-see reservation selection levels displayed fewer tardy lots (17 of 85), and very low work in process time with a makespan of 588.8 days. At this point, machine capacity is the principal limitation affecting tardiness. Experiments with all lots being scheduled backwards from the due date and added capacity were also performed.

6.1. Introduction

As is true in many AI systems (e.g.,Bacon, AM), measuring the success of ISIS can be difficult and controversial. Unless the domain has a clear measure of success, such as: the computer configuration "works" (i.e., R1), the sentence is recognized (i.e, speech understanding), coal was found (i.e., prospector), determining the "success" of the system may be a thesis in itself.

Three approaches have been taken towards measuring ISIS's performance. The first is an analysis of the type of reasoning a job-shop scheduling system requires to successfully consider the multitude of constraints in its environment. The previous chapters describe the approach. In particular they describe how ISIS can represent, reason with, and relax constraints during the scheduling process.

The second approach is to ask the expert how well ISIS performs relative to his/her expectations.

The third approach is to analyze the performance of ISIS on data representative of the plant's orders. Though it appears simple, the experimental design is quite difficult. In particular, there is no single measure of the efficacy of a schedule. Other scheduling systems, because they focus on the optimization of a single criterion such as tardiness, are easy to analyze. But ISIS attempts "optimize" an order's schedule according to its own, peculiar constraints. As stated earlier, each order differs by the constraints applicable and their prioritization. It is the satisfaction of the constraints, in a particular context, that define the efficacy of an order's schedule. Global statistics are not indicative except in a gross sense.

A set of experiments were conducted to ascertain the aggregate performance of the system. A portion of the turbine component plant was modeled and a representative set of orders (85) generated and scheduled over a two year period. The following defines the generation of the test data.

A portion of the turbine component plant was chosen to test ISIS. About twenty turbine blade styles, representative of the types of orders worked on in the area, were chosen by the plant scheduler. These styles were chosen so that their operations would maximize resource contention. A typical lineup is composed of the following sequence of operations and machines:

1. First Straightening (machine: ffs*),

2. Root Forming (xao*, msr*, mr8a),

3. Second Straightening (fss*),

4. Airfoil Forming (wmc1-4, wmf1-4),

5. Peg Milling, Brazing, Grinding (mvt1, sabb, fps*, m3db, xpma, gbbv, jwia-c),

6. Final Straightening (ftq*, fts*),

7. Layout and Pack (ilpa).

Lineups are di-graphs; alternative paths exist to achieve the above sequence. Each node in the di-graph represents an operation which can be implemented on one or more machines. About half the operations have more than one machine or work-center at it can be performed.

The distribution of orders for an average year was specified by the scheduler to be a uniform 300 with a standard deviation of 50. Orders were allocated for delivery in a given month according to a monthly ship distribution determined by the scheduler.

Within each month an order was assigned a ship date on a weekly basis by uniformly dividing the month's orders amongst the weeks. Next, the priority class of an order was determined by the following percentages:

Forced outage (FO)	2%
Critical replacement (CR)	5%
Ship direct (SD)	20%
Service order (SO)	33%
Shop order (SH)	20%
Stock order (ST)	20%

Table 6-1: Percentage of Orders in Each Priority Class

Once the class of order was determined, the lead time for the order (i.e., the time of arrival of the order into the plant), was determined by the following uniform distributions (in weeks):

Forced outage	[0, 12]
Critical replacement	[0, 18]
Ship direct	[0, 62]
Service order	[0, 62]
Shop order	[0, 82]
Stock order	[0, 60]

Table 6-2: Lead Time Distributions by Priority Class

A year's orders were randomly generated according to the above statistics, and the first 85 (ordered by requested start date) were scheduled. Appendix B list the orders used to test ISIS.

A worst-case complexity analysis will shed some light on the difficulty of the problem. Assuming that:

- we are scheduling a *flow* shop containing o operations,
- there are n orders to be processed,
- there does not exist any precedence amongst orders, and
- the size of gaps (i.e., idle time) is ignored.

Then the number of different schedules that exist is:

$$(n!)^{o}$$

Generalizing the problem to include a single alternative machine for an operation, then for each sequence of n orders in front of a machine, there are 2^n ways in which each orders may choose a machine to run on. This results in a worst case complexity of

$$(2^{n}\, n!)^{o}$$

In our test data

o is approx 8 (excluding alternatives), and
n is 85

resulting in a worst case complexity of

$(2^{85}\ 85!)^{10}$

Note that this number does not consider idle time of varying lengths as different schedules, nor does it consider alternative operations, and other resource substitutability.

The testing of ISIS is an ongoing process. The following series of experiments only test of a small number of the possible variations ISIS may take in problem-solving structure and constraint content.

6.2. Series 1 Experiments

The first series of experiments were based on a model and set of constraints provided by the scheduling expert. In experiments one through eight, all forced outage (FO), critical replacement (CR) and ship direct (SD) orders are scheduled backwards from their due date, and the rest are scheduled forwards from their order arrival date. In experiments nine through twelve, all orders are scheduled backwards Orders are assigned a priority according to their classification. The ordering from high priority to lowest is FO, CR, SD, SO, ST. The types of constraints included in the series are:

- Alternative operations,
- Alternative machines,
- Pre-specified due date,
- Pre-specified start date,
- Work in Process,
- Queue ordering constraints to reduce setup time,
- Lug and length machine physical constraints,
- Resource availability,
- Shop stability.

Twelve experiments were performed in this series. A detailed a description of the results of these experiments can be found in appendix C. The following analyzes experiments 1, 2, 4, 8 and 12, which I chose as the more interesting. Each experiment was run using the same order/lot data. Experiments 1 and 8 were also run on nine additional sets of test data.

6.2.1. Experiment 1 Analysis

The following are the characteristics of Experiment 1:

- The capacity analysis level was removed,
- the beam search width was set to 9, and
- orders were scheduled in priority order. A higher priority order may pre-empt resources of a lower priority order, and
- the eager reserver version of choose-reservation was used.

The purpose of experiment 1 was to test a simple beam search approach which was found to be useful in both speech (Lowere, 1976) and vision (Rubin, 1978) tasks[20]. A beam search approach reduced the complexity of the problem dramatically. With a beam width of size b, only b schedules are produced. A better measure of the complexity of the task is to examine the number of states generated by the search. With a beam width b, and the number of reservations generated for each state being less than n/2, and no machine alternatives, a reasonable upper bound on the number of states examined to generate a complete shop schedule is

(n/2) b o n

If an alternative machine is specified for each operation, then an upper bound on the number of states generated is:

(n b o n)

In this series of experiments this is approximately 85*9*8*85 = 520,200 states, or 6120 states/order.

The actual number of states generated is much smaller. This is due partially to the existence of alternative paths in the lineups which reduces the number of reservations associated with a machine. Coupled with the time ordered arrival of orders, the number of reservations generated at a machine is well below n/2. The number of states generated is also reduced by having only a single machine for half the operations. Hence if the number of reservations (states) generated for each machine considered is reduced to about 6, and the number of alternative machines set to 1.2, then the number of states generated is approximately (6*1.2*9*10) 648.

The following gantt chart summarizes the machine schedules. Each row is a separate machine. The rows, y axis, have been ordered by time of appearance in a lineup. Machines at the top of the chart are used first, and machines at the bottom are used last. The x axis is time, with each column depicting one week. A row-column position without a character in it does not have any work to perform for that week. A row-column position with an o has 1 to 4 days of the week scheduled, and a position with an @ has 5 to 7 days scheduled.

[20] This was the first version of ISIS constructed (see chapter 5).

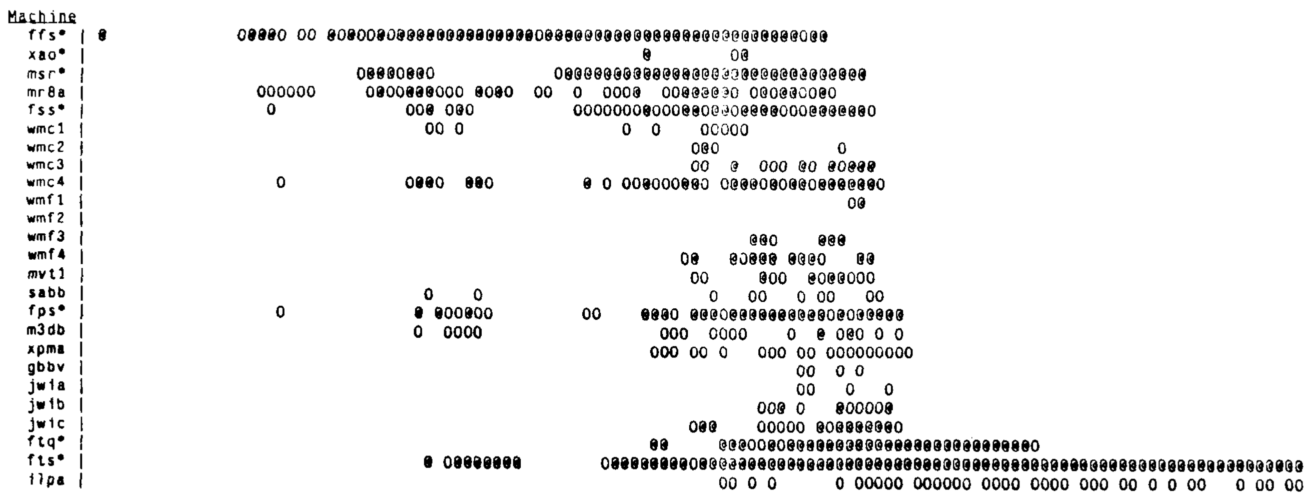


Figure 6-1: Experiment 1 Gantt Chart

The following summarizes the aggregate statistics for orders in the scheduling run, broken down by priority class.

STATISTIC	SD	SO	ST	SH	All
Orders Scheduled	9	19	14	43	85
Tardy lots	0	16	14	35	65
Avg. Tardiness	0	31.36	186.01	122.90	99.82
Avg. Finishing Lateness	-0.97	30.19	186.01	122.50	99.25
Avg. Starting Lateness	263.83	125.88	71.51	118.21	127.56
Avg. WIP	32.31	200.16	324.50	402.34	305.15
Avg. Processing Time	30.86	31.59	28.32	30.28	30.31
Avg. Queue Time	1.45	168.56	296.18	372.06	274.84
Avg. % Processing Time	95.23%	24.98%	9.53%	8.67%	21.62%

Table 6-3: Experiment 1 Statistics

The first glaring problem with Experiment 1 is that 65 of the 85 orders scheduled were tardy! There are two bottlenecks in the shop, the first straightening area, ffs*, and the final straightening area, fts*. Both of these areas do not appear to have enough capacity to handle the orders. This is not actually the case. The average tardiness for the orders is 99.82 days. This is less than the average starting lateness of 127.56 days. The latter statistic states that orders were started about 127.56 days *after* they were known. If these orders were started closer to the time they were received by the plant, then they could be scheduled earlier through the bottlenecks.

Wherein lies the problem? The problem with experiment 1, is that the beam search is unable to spot the second bottleneck in advance (it occurs near the end of the schedule), and chose late rather than early reservations on the preceding machines. When the order finally arrived at the last machine, there was a large queue in place. Hence, orders spent undue amounts of time in queue, while other machines were left idle. This is an example of the horizon effect. The beam search has no provision for "looking into the future" inorder to alter its current scheduling decision. It assumes that by generating enough alternative states that the problem will be overcome. This turned out to be false. There was not enough information early in the schedule to discern problems later on.

The differences amongst classes of orders is dramatic, but expected. Higher priority orders (e.g., SD) which were given pre-emptive access to resources, had little tardiness (0), and a high average percentage processing time (95.23%). The lowest priority orders, (e.g., SH) had an average percentage processing time of 8.67%.

The next table provides another perspective on the performance of this experiment. Five machines were chosen based on their criticality to the majority of the orders in the experiment. For each machine, its percent utilization and average queue time of orders is defined for the makespan period of the version's experiment.

Machine	**% Utilization**	**Avg. Queue Time**
ffs*	41.2	0
fss*	21.2	53.4
fps*	21.9	18.1
fts*	61.6	100.5
ilpa	8.6	35.1

Table 6-4: Machine Utilization for Experiment 1 (makespan: 857.4 days)

There is nothing surprising about these statistics. Basically, all of these machines are under utilized when comparing their processing time to their maximum availability. Never the less, fts* has a high average queue time due to the bunching of orders late in the schedule. Makespan for this experiment is over two years, much of which is due to tardiness of orders.

Another measure of the performance of ISIS is how well the constraints it uses are satisfied. The constraints used in these experiments are summarized in appendix A. The results presented here summarize the utilities of five classes of constraints. Each class contains from one to eight different constraints.

stability: constraints which measure machine queue stability (pre-emption).

wip: constraints on the work in process time of a lot.

due-date: constraints on the due-date of a lot.

attribute-preference: preference constraints which measure machine and product attributes.

q-constraint: constraints which measure preferences for certain orderings of lots in machine queues.

These constraints have a maximum utility ranging from 1.1 to 1.5. For each group of constraints, the number of instances of the constraint found in a priority class and its average utility are listed:

	Class	Number	Average Utility
Priority Class SD			
	Stability	76	1.0
	Wip	9	1.5
	Attribute-Preference	18	1.1
	Q-Constraint	18	1.0
Priority Class SO			
	Stability	160	1.0
	Due-date	19	0.5
	Attribute-Preference	38	1.0
	Q-Constraint	38	1.1
Priority Class ST			
	Stability	118	1.0
	Due-date	14	1.0
	Attribute-Preference	28	1.0
	Q-Constraint	28	1.1
Priority Class SH			
	Stability	366	1.0
	Due date	43	0.5
	Attribute-Preference	86	1.0
	Q-Constraint	86	1.1

Table 6-5: Constraint Satisfaction Summary for Experiment 1

SD's constraints appear acceptable to good, especially wip. ST lots are acceptable. Both SO and SH have poor due date utilities. This confirms what we have already seen in the statistics above.

Experiments 1 and 3 were tested on an additional 9 sets of test orders, randomly generated from the same sales statistics. The following summarizes the performance of experiment 1:

STATISTIC	Min	Max	Mean	Std
Tardy lots	55	72	64.3	4.51
Avg. Tardiness	99.8	159.84	138.44	18.19
Avg. Finishing Lateness	99.25	158.13	137.16	17.7
Avg. Starting Lateness	127.56	219.87	189.55	24.0
Avg. WIP	208.79	305.15	267.4	25.54
Avg. Processing Time	29.26	30.86	29.96	0.52
Avg. Queue Time	179.22	274.84	237.5	25.35
Avg. % Processing Time	21.62%	35.04%	31.12%	3.86

Table 6-6: Experiment 1 Average Statistics

The number of tardy lots is high with the average tardiness exceeded the average starting lateness.

6.2.2. Experiment 2 Analysis

The purpose of experiment 2 was to determine whether the horizon effect could be reduced by increasing the utility of early reservations. This was accomplished by altering the utility of the due date constraint. Where previously, a due date constraint placed a high utility on shipping an order close to its requested due date, it was altered to increase the utility of shipping early. In this way, states with early reservations were rated as good as states with later reservations. Hence, experiment 2 is exactly the same as experiment 1 except for the altered due date constraint.

The same orders were run through experiment 2 as in experiment 1. The gantt chart for experiment 2 is:

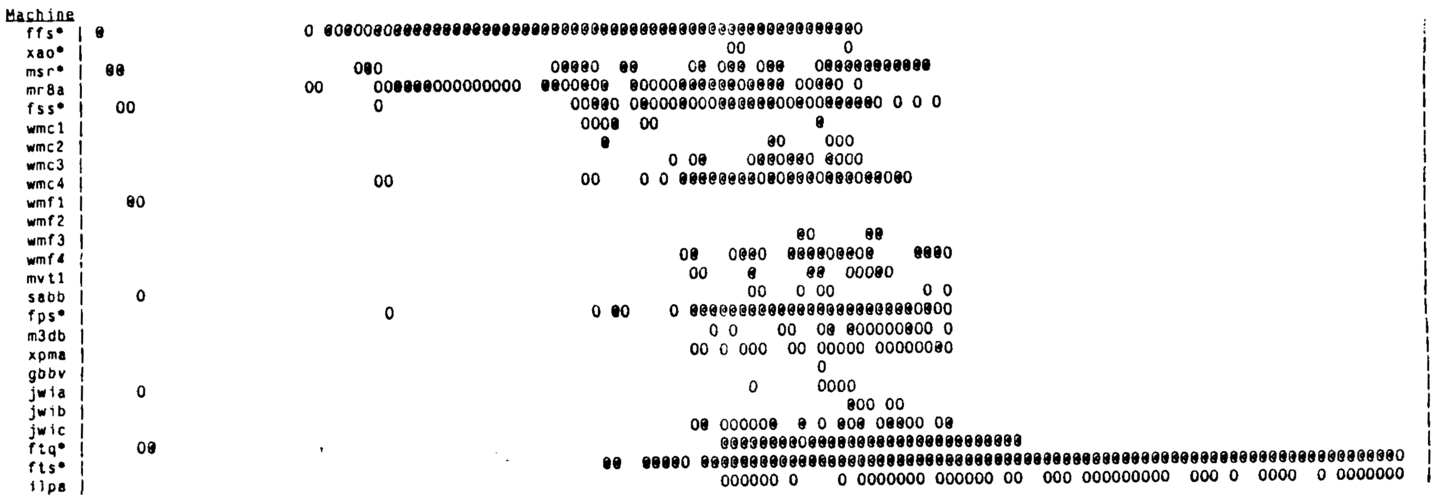


Figure 6-2: Experiment 2 Gantt Chart

The statistics for the orders broken down by priority class are:

STATISTIC	SD	SO	ST	SH	All
Number of orders	9	19	14	43	85
Tardy lots	7	17	14	41	72
Avg. Tardiness	0	32.05	144.74	203.27	133.83
Avg. Lateness (finishing)	-0.97	31.82	144.74	203.16	133.63
Avg. Lateness (starting)	263.83	112.01	74.51	202.51	167.69
Avg. WIP	32.31	215.65	280.23	398.52	299.39
Avg. Processing Time	30.86	29.90	28.32	29.63	29.60
Avg. Queue Time	1.45	185.75	251.91	368.89	269.78
Avg. % Processing Time	95.23%	24.62%	14.73%	8.44%	22.28%

Table 6-7: Experiment 2 Statistics

By changing a single due date constraint, it appears that the resultant schedule is worse than experiment 1 (i.e., tardiness has increased without a significant change in WIP). This appears to be counter intuitive. Actually, the statistics are better than experiment 1. The classes tardiness and WIP for ST orders are decreasing.

CLASS	TARD. V1	TARD. V2	WIP V1	WIP V2
SD	0	0	32.31	32.31
SO	31.36	32.05	200.16	215.65
ST	186.01	144.74	324.5	280.23
SH	122.9	203.27	402.34	398.52

Table 6-8: Experiment 1 & 2 Comparative Statistics

It is only for the SH orders that the statistics degenerate. This can be attributed to the bottleneck in the last operation. Since orders are scheduled in priority order, SD orders were schedule first, followed by SO, ST, then SH. It is only when the SH orders were scheduled that the combination of the bottleneck and the horizon effect caused the last set of orders to have a large increase in tardiness and work in process. The degeneration of SH statistics may be explained by the altered search space. In experiment 2, earlier reservations are chosen, hence the SH orders start earlier but queue in front of the bottleneck never the less. The increase in average tardiness is due to the general inavailability of the final machine fts*. That is, the SD, SO, and ST orders have reserved the fts* earlier than in experiment 1, hence reducing the capacity of the machine for early reservations for SH orders. Hence, SH orders can only find capacity on the "late side" of their due dates.

The following are the machine utilization and queue time statistics:

Machine	% Utilization	Avg. Queue Time
ffs*	38.3	0
fss*	19.7	66.3
fps*	20.3	21.8
fts*	57.2	138.1
ilpa	7.9	11.9

Table 6-9: Machine Utilization for Experiment 2 (makespan: 923.5 days)

Makespan has increased. Utilization remains about the same as in experiment 1 due to equivalent makespans. But average queue times increase at the fss* and fts* machines. The reason for this will be explained in the experiment 4 analysis.

The following is the summary of average constraint utlities:

	Class	Number	Average Utility
Priority Class SD			
	Stability	76	1.0
	Wip	9	1.47
	Attribute-Preference	18	1.08
	Q-Constraint	18	1.0
Priority Class SO			
	Stability	160	1.0
	Due-date	19	0.64
	Attribute-Preference	38	1.09
	Q-Constraint	38	1.0
Priority Class ST			
	Stability	118	1.0
	Due-date	14	0.56
	Attribute-Preference	28	1.1
	Q-Constraint	28	1.0
Priority Class SH			
	Stability	366	1.0
	Due date	43	0.22
	Attribute-Preference	86	1.09
	Q-Constraint	86	1.0

Table 6-10: Constraint Satisfaction Summary for Experiment 2

The SD constraints do not differ from experiment 1. SO lots show a slight improvement in due date, but at the cost of both ST and SH's due date utilities.

6.2.3. Experiment 4 Analysis

To reduce tardiness due to the horizon effect, the capacity analysis level is included (see section 5.4). The new level performs a capacity analysis[21]. It takes a dynamic programming approach to determining the earliest start time and latest finish time for each operation to produce an order. Times are based only on start time, finish time, machine capacity, and existing reservations for each machine. It ignores all the other constraints used by the detailed scheduling level. The result of this analysis is a set of time bounds for an order's

[21] This was the second version of ISIS constructed which is described in chapter 5.

operations. These time bounds are defined as constraints and added to the existing constraints used at the detail scheduling level.

The following is the gantt chart for experiment 4:

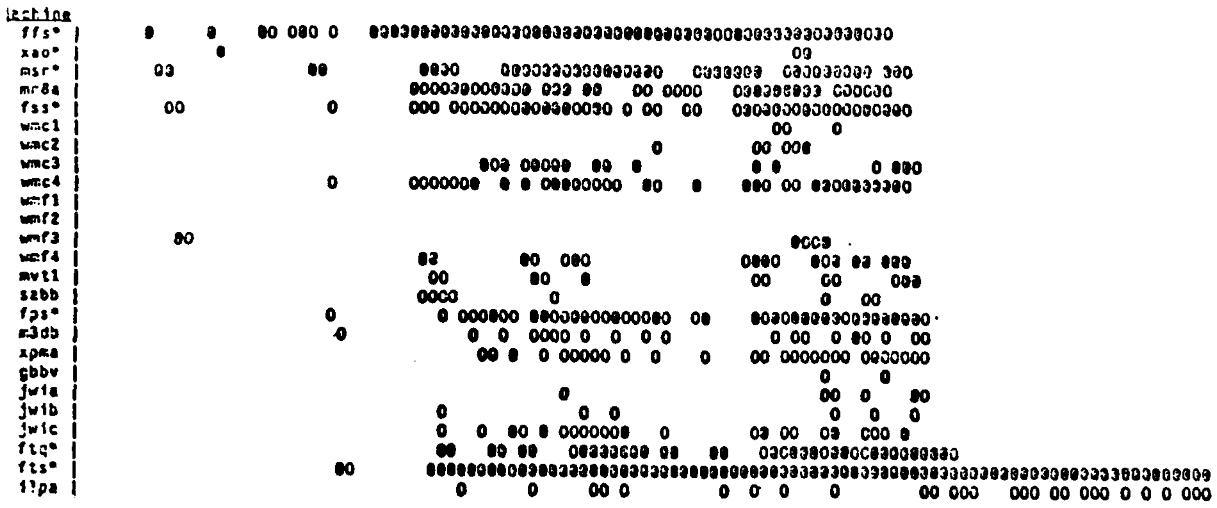


Figure 6-3: Experiment 4 Gantt Chart

The schedule's summary statistics are:

STATISTIC	SD	SO	ST	SH	All
Number of lots	9	19	14	43	85
Tardy lots	0	0	0	13	13
Avg. Tardiness	0	0	0	36.81	18.62
Avg. Lateness (finishing)	-0.97	-30.31	-66.69	19.43	-8.04
Avg. Lateness (starting)	263.83	58.76	24.17	145.51	118.66
Avg. WIP	32.31	106.73	119.14	271.78	206.74
Avg. Processing Time	30.86	29.39	32.25	30.10	30.38
Avg. Queue Time	1.45	177.34	86.89	241.68	176.37
Avg. % Processing Time	95.23	27.94	38.87	13.30	29.46

Table 6-11: Experiment 4 Statistics

The effect of the constraints generated by capacity analysis are dramatic. The number of tardy lots is reduced form the current best of 65 to 13. Work in process, tardiness, and lateness are also reduced across all classes of orders. Again, the bottleneck at the fts* machine manifests itself in the SH orders; the number of tardy orders were not eliminated, but reduced from 35 (experiment 1) to 13 (experiment 4).

The cost of the added capacity analysis is, on the average, linear with the number of operations in an operation graph. A small cost compared to the affect it has on the horizon effect.

The following are the machine utilization and queue time statistics for this experiment:

Machine	% Utilization	Avg. Queue Time
ffs*	47.9	0
fss*	24.6	22.2
fps*	25.4	3.1
fts*	71.6	65.9
ilpa	9.9	52.2

Table 6-12: Machine Utilization for Experiment 4 (makespan: 737.6 days)

Makespan has been reduced by 119.8 in comparison to experiment 1. Better scheduling of orders through the bottlenecks has pushed much of the reservations earlier in time, reducing the "tardiness tail". Utilization of all machines have gone up, though not significantly on the ilpa since it is not a bottleneck. One startling statistic is the dramatic increase in average queue time at the ilpa. A quick look at the gantt chart shows that in any week, the machine is not fully loaded, hence why are the orders waiting? Since ISIS explicitly chooses a time for each operation, orders sit in the queue even though the machine may be idle. Such a decision may be useful if the machine is operated only when enough orders exist to be processed. But such is not the case here. All the orders which have large queue times in front of the ilpa were scheduled in the forward direction from their arrival date. A due date constraint was applied to each of these orders. This constraint biased the system towards chosing reservations as close to the order's due date as possible. Hence, when ISIS generated alternative reservations for the ilpa, earlier reservations were poorly rated when compared to reservations which were closer to the order's due date. This is a symptom of forward scheduling from an order's arrival date which provides more work in process time then ISIS requires.

The following summarizes the utilities of the constraints in experiment 4. Note that the class of operation time bound constraints has been added to reflect the output of capacity analysis to resource analysis.

	Class	Number	Average Utility
Priority Class SD			
	Stability	76	1.0
	op time bound	76	1.1
	Wip	9	1.5
	Attribute-Preference	18	1.08
	Q-Constraint	18	1.0
Priority Class SO			
	Stability	160	1.0
	Op time bound	160	1.1
	Due-date	19	0.85
	Attribute-Preference	38	1.09
	Q-Constraint	38	1.0
Priority Class ST			
	Stability	118	1.0
	Op time bound	118	1.1
	Due-date	14	0.71
	Attribute-Preference	28	1.1
	Q-Constraint	28	1.0
Priority Class SH			
	Stability	366	1.0
	Op time bound	366	0.85
	Due date	43	0.61
	Attribute-Preference	86	1.09
	Q-Constraint	86	1.0

Table 6-13: Constraint Satisfaction Summary for Experiment 4

SD's constraints appear acceptable to good, especially wip. ST lots are acceptable. Both SO and SH have poor due date utilities. This confirms what we have already seen in the earlier statistics.

6.2.4. Experiment 8 Analysis

Experiment 8 was created to deal with a different manifestation of the horizon effect. The eager reserver rule (section 5.5.2.2) generates alternative states for each block of time available, reserving the earliest time in that block. It is not known at the time whether this is a good decision due to the horizon effect. There could be a bottleneck, hence choosing a later time in a block, and arriving later into the bottleneck would reduce its queue time. The wait and see reserver implements this approach. It does not choose a reservation in a block until the search is finished. It then chooses the latest time possible for each operation[22]

The gantt chart for experiment 8, when compared to experiment 4, shows a general compression of reservations, hence, a more even utilization of machines during a smaller time period. Both the ffs* and fts* machines continue to be bottlenecks.

[22] This is the third version of ISIS constructed as described in chapter 5.

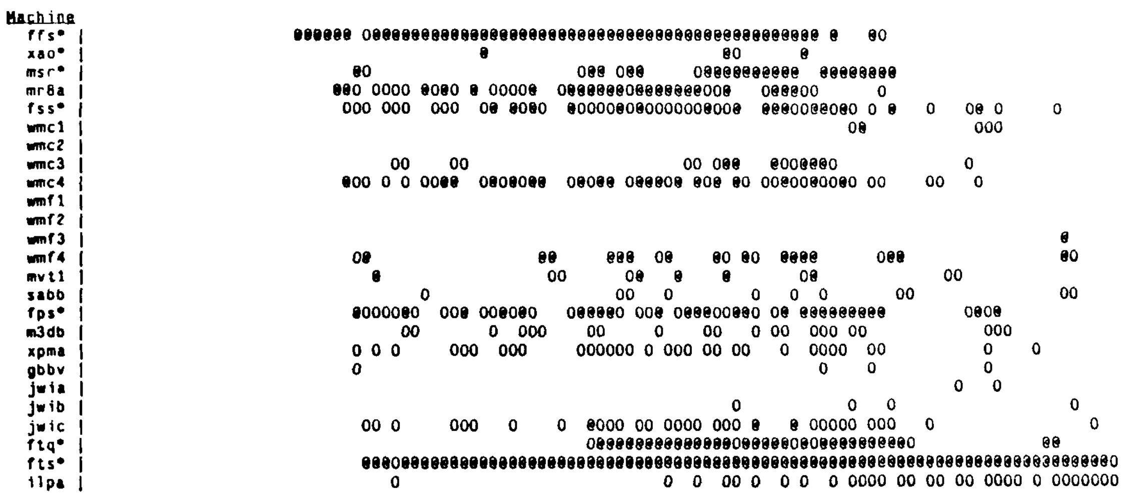


Figure 6-4: Experiment 8 Gantt Chart

The statistics for experiment 8 are:

STATISTIC	SD	SO	ST	SH	All
Number of lots	9	19	14	43	85
Tardy lots	0	1	0	16	17
Avg. Tardiness	0	0.56	0	40.67	20.7
Avg. Lateness (finishing)	-1.13	-19.93	-5.61	27.89	8.61
Avg. Lateness (starting)	263.81	155.94	68.39	149.46	149.66
Avg. WIP	32.18	119.98	136.21	276.29	192.4
Avg. Processing Time	30.86	30.57	28.32	28.93	29.4
Avg. Queue Time	1.31	89.41	107.68	247.36	162.99
Avg. % Processing Time	96.04%	32.93%	24.16%	13.04%	28.11%

Table 6-14: Experiment 8 Statistics

The effect of the wait-and-see reserver is to reduce the number of states generated when in the presence of reservations for orders of lower priority. Instead of creating multiple reservations which bound the existing reservation(s), it creates a single reservation with spans both the available time and the time reserved by lower priority orders. Since, in these tests, orders are scheduled in priority order, there will not be any reservations for orders of lower priority encountered during scheduling. Hence, the primary difference in the search space between experiment 4 and experiment 8 is the compression of reservations to reduce WIP, leaving fewer holes in a machine's queue.

The machine utilization and queue times for experiment 8 are:

Machine	% Utilization	Avg. Queue Time
ffs*	60	0
fss*	30.8	42.2
fps*	31.8	6.9
fts*	89.6	11.2
ilpa	12.4	56.3

Table 6-15: Machine Utilization for Experiment 8 (makespan: 588.9 days)

Makespan has been reduced by about 148 days when compared to experiment 4. This reduction is due to the wait and see reserver pulling reservations closer to an order's due date. With the reduction of makespan, machine utilization also increases. A noticeable deviation is the decrease in average queue time at the fts* (from 65.9 in experiment 4). This reduction is also explained by the wait and see reserver. Since the operator, when operating in the forward direction, chooses as late a reservation as possible, then orders tend to arrive in more of a "just in time" fashion.

The following summarizes the utilities of the constraints in experiment 8:

	Class	Number	Average Utility
Priority Class SD			
	Stability	76	1.0
	Op time bound	76	1.1
	Wip	9	1.47
	Attribute-Preference	18	1.08
	Q-Constraint	18	1.0
Priority Class SO			
	Stability	160	1.0
	Op time bound	160	1.07
	Due-date	19	0.93
	Attribute-Preference	38	1.09
	Q-Constraint	38	1.0
Priority Class ST			
	Stability	118	1.0
	Op time bound	118	1.1
	Due-date	14	1.03
	Attribute-Preference	28	1.1
	Q-Constraint	28	1.0
Priority Class SH			
	Stability	366	1.0
	Op time bound	366	0.81
	Due date	43	0.60
	Attribute-Preference	86	1.09
	Q-Constraint	86	1.0

Table 6-16: Constraint Satisfaction Summary for Experiment 8

SD statistics stay the same. Both SO and ST have improved due date utilities. Work in process constraints were not include in the SO, ST, and SH classes. If they were, they would show a marked improvement in utility due to the reduction in queue time.

Experiments 1 and 8 were tested on an additional 9 sets of test orders, randomly generated based on the same statistics. The following summarizes the performance of experiment 8:

STATISTIC	Min	Max	Mean	Std
Tardy lots	14	16	18.4	4.27
Avg. Tardiness	19.33	59.36	32.8	11.5
Avg. Finishing Lateness	1.55	51.09	23.65	13.7
Avg. Starting Lateness	140.66	194.2	170.27	17.17
Avg. WIP	147.31	186.9	173.25	11.63
Avg. Processing Time	29.32	30.83	30.06	0.52
Avg. Queue Time	117.97	156.54	143.19	11.25
Avg. % Processing Time	32.03%	38.97%	36.51%	1.92

Table 6-17: Experiment 8 Average Statistics

6.2.5. Experiment 12 Analysis

Many of the problems in experiments one through eight stem from the forward scheduling of ST and SH orders. Poor decisions at the beginning of the schedule lead to bottlenecks at the end. The purpose of experiments nine through twelve was to determine the effect of scheduling all orders backward from their due date. The following is the gantt chart for experiment 12:

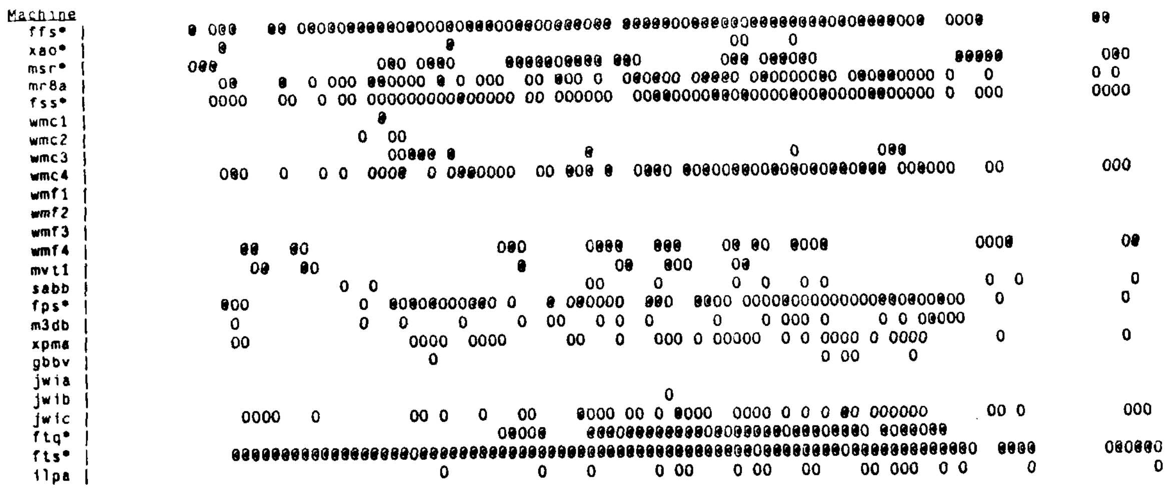


Figure 6-5: Experiment 12 Gantt Chart

The statistics for experiment 12 are:

STATISTIC	SD	SO	ST	SH	All
Number of lots	9	19	14	43	85
Tardy lots	0	0	0	0	0
Avg. Tardiness	0	0	0	0	0
Avg. Lateness (finishing)	-1.13	-30.27	-67.4	-79.44	-58.17
Avg. Lateness (starting)	263.81	218.66	67.24	210.23	194.23
Avg. WIP	32.18	46.92	75.36	108.19	81.04
Avg. Processing Time	30.86	30.63	30.12	29.08	29.79
Avg. Queue Time	1.31	16.29	45.24	79.11	51.25
Avg. % Processing Time	96.04%	79.25%	49.56%	55.31%	64.03%

Table 6-18: Experiment 12 Statistics

Backward scheduling of all lots vastly improves the above measurements, except for SD orders which were scheduled backwards in experiments 1 through 8. The wait and see reserver makes sure all reservations are as close to the delivery date as possible.

The machine utilization and queue times for experiment 12 are:

Machine	% Utilization	Avg. Queue Time
ffs*	51.4	0
fss*	26.4	0.1
fps*	27.3	0.7
fts*	76.8	1.5
ilpa	10.6	42.9

Table 6-19: Machine Utilization for Experiment 12 (makespan: 687.4 days)

In comparison to experiment 8, the makespan has increased almost 100 days. This implies that backward scheduling is able to meet its due dates by utilizing idle time earlier in the scheduling period. Thus start times are pushed back earlier than what is found in experiment 8.

The following summarizes the utilities of the constraints in experiment 12:

	Class	Number	Average Utility
Priority Class SD			
	Stability	76	1.0
	Op time bound	76	1.1
	Wip	9	1.47
	Attribute-Preference	18	1.08
	Q-Constraint	18	1.0
Priority Class SO			
	Stability	160	1.0
	op time bound	160	1.07
	wip	19	1.22
	Attribute-Preference	38	1.09
	Q-Constraint	38	1.0
Priority Class ST			
	Stability	118	1.0
	Op time bound	118	0.96
	Wip	14	1.07
	Attribute-Preference	28	1.1
	Q-Constraint	28	1.0
Priority Class SH			
	Stability	366	1.0
	Op time bound	366	0.95
	Wip	43	1.18
	Attribute-Preference	86	1.09
	Q-Constraint	86	1.0

Table 6-20: Constraint Satisfaction Summary for Experiment 12

Both SD and SO classes of lots show good performance in satisfying its constraints. ST and SO classes are also good, but the wip utility is below 1.0. This is due to lack of capacity in the bottleneck machines.

6.3. Series 2: Experimental Analysis

A major factor in generating good schedules in the first series of experiments is the ability to "work around" the bottlenecks. In the experiments where forward scheduling and backward scheduling were mixed, the capacity analysis level was required to detect bottlenecks before detailed scheduling is performed. When all scheduling was backward, sufficient lead time was required to spread the load.

In this series of experiments, the bottleneck problem is alleviated by doubling the capacity of the fts* machine. Except for this model alteration, the experiments are equivalent to those in series 1. Only one experiment is analyzed here.

6.3.1. Experiment 8 Analysis

This experiment is equivalent to the series 1 experiment. The gantt chart for experiment 8, when compared to its equivalent in series 1 shows a major compression of reservations, hence, a more even utilization of machines during a smaller time period.

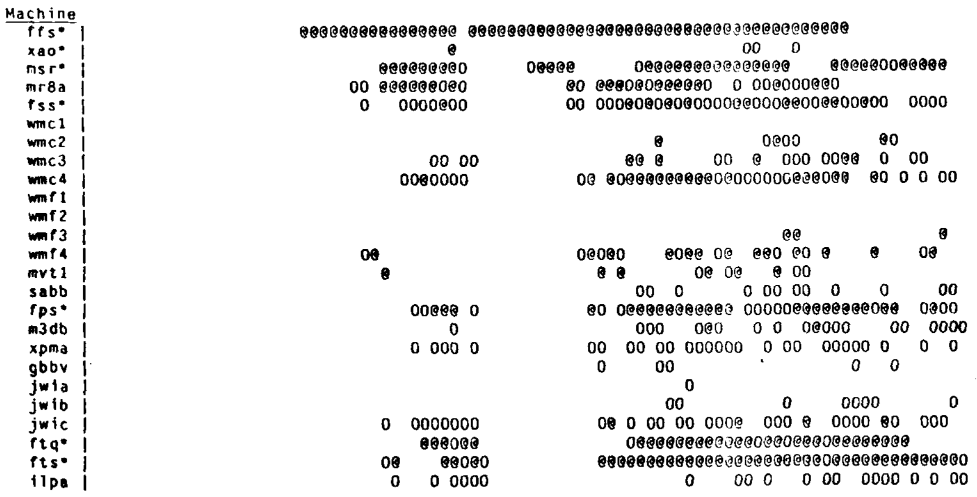


Figure 6-6: Experiment 8 Gantt Chart

The statistics for experiment 8 are:

STATISTIC	SD	SO	ST	SH	All
Number of lots	9	19	14	43	85
Tardy lots	0	0	0	2	2
Avg. Tardiness	0	0.56	0	3.04	1.54
Avg. Lateness (finishing)	-0.69	-28.44	-77.31	-23.54	-31.07
Avg. Lateness (starting)	267.49	192.5	60.76	123.72	143.95
Avg. WIP	28.93	74.91	71.93	250.61	158.43
Avg. Processing Time	28.71	27.16	26.9	27.43	27.41
Avg. Queue Time	0.22	47.75	45.03	223.18	131.02
Avg. % Processing Time	99.3%	46.68%	57.88%	15.78%	38.47%

Table 6-21: Experiment 8 Statistics

Processing time for SH is still quite small compared to its WIP time. This can be alleviated by scheduling SH orders backward.

The machine utilization and queue times for experiment 8 are:

Machine	% Utilization	Avg. Queue Time
ffs*	75	0
fss*	38.5	54.3
fps*	39.8	1.2
fts*	58.6	1.8
ilpa	15.5	26.2

Table 6-22: Machine Utilization for Experiment 8 (makespan: 471.5 days)

Makespan has been reduced by about 120 days when compared to experiment 8 in series 1. The reduction is obviously due to the additional capacity in the main bottleneck machine.

The following summarizes the utilities of the constraints in experiment 12:

	Class	Number	Average Utility
Priority Class SD			
	Stability	76	1.0
	Op time bound	76	1.1
	Wip	9	1.49
	Attribute-Preference	18	1.08
	Q-Constraint	18	1.0
Priority Class SO			
	Stability	160	1.0
	op time bound	160	1.1
	due date	19	0.86
	Attribute-Preference	38	1.09
	Q-Constraint	38	1.0
Priority Class ST			
	Stability	118	1.0
	Op time bound	118	1.1
	Due date	14	1.02
	Attribute-Preference	28	1.1
	Q-Constraint	28	1.0
Priority Class SH			
	Stability	366	1.0
	Op time bound	366	1.06
	Due date	43	0.75
	Attribute-Preference	86	1.09
	Q-Constraint	86	1.0

Table 6-23: Constraint Satisfaction Summary for Experiment 8

The utitilities for these constraints are similar to those found in experierment of series one. The only significant difference is that in the SH class, which contains the majority of lots, the due date utility has increased by about 0.15. Hence, the due date is being better met with the increased capacity.

6.4. Conclusion

This chapter describes the experimental analysis of ISIS as a job-shop scheduling system. The difference in performance between experiments 1 and 8 shows that in very large search spaces where the evaluation function is a poor estimator of future success of the search path, a hierarchical approach is required. The hierarchical approach is a heuristic look ahead procedure which examines a projection of the search space. The output of a level is a set of constraints on the search at the next level. Using this approach, ISIS performed well.

The major difficulty with testing ISIS is that it contains too many variables. The number of tests that should be performed is far beyond that which could be performed. Convergence of ISIS's performance can be enhanced by focusing on the more salient parts such a scheduling of goals and time constraints. But in the end, not all of the utilities, and importance levels will ever fully be examined, and their sensitivity evaluated. Never the less, testing is still continuing in which a single variable, such as a constraint's utility is altered while others are held constant. Another type of testing that needs to be performed, is the iterative effect of post-analysis. Once the implementation of constraint interaction diagnosis is completed, tests will be performed to determine the convergence of schedules through this form of iterative scheduling.

Chapter 7
Observations and Conclusions

Summary

The contributions of this thesis are in three areas: representation, constraint-directed search and job-shop scheduling. In the representation area, a more complete semantics for the modeling of organizations is provided which includes: states, acts, time, causality, multi-level representation, and support for discrete simulation. In the area of search, ISIS introduces a number of new concepts:

- A general representation for constraints with particular attention paid to the representation of relaxations, interactions, and obligations.

- Constraint-based pre- and post-search analysis to bound the solution space before performing search, and diagnose poor constraint decisions at other levels.

- The generation and evaluation of constraint relaxations during the search process.

- The resolution and differentiation among constraints in evaluating states in the search space.

- Hierarchically constrained search. A level in the hierarchy communicates only constraints in order to guide search at the next level.

The contribution to job-shop scheduling made by this thesis is that it provides, for the first time, a system which can represent and consider all the domain constraints during the construction of a schedule. And do so in a reasonable amount of processing time. It also provides incremental scheduling in reaction to changes in the plant's status, and suggests alternative schedules when constraints cannot be satisfied.

7.1. Introduction

In this chapter, some observations of ISIS's performance are made, and following the contributions of this dissertation are summarized. Lastly, a description of future research is provided, much of which is currently being pursued.

7.2. Observations

7.2.1. Tactical vs. Strategic Decisions

Constraints can be used to represent both tactical and strategic information. Tactical information corresponds to constraints that provide specific knowledge of what can or cannot be done on the shop floor. For example, precedence constraints define possible operation sequences. Enabling constraints define what materials must be available before an operation can be performed on a machine. Reservations define possession of objects by other orders. Tactical decisions are implicitly made by the search process. As the search is performed states are generated which represent alternative tactical decisions based on the above constraints.

Different levels of strategic information are also represented by constraints. Organizational goal constraints can be viewed as strategic in nature; they provide an overall rating of a schedule without restricting individual decisions. Scheduling goal schemata also provide strategic information, in that they prioritize the constraints, which in turn affect the choice of schedules.

A problem with experiment 1 is that its tactical decision making is generative; it elaborates the space of alternatives, and chooses the most highly rated. The "generate and test" approach, requires that the number of alternatives be small, so that the search space is of a reasonable size. Better tactics could be used if the operators did less generation and more analysis. For example, if it was determined that a certain machine was a bottleneck, an operator could be added that would generate alternative shifts for that machine or work area only. If the bottleneck information was made available by some hierarchical analysis, or by the post-search analysis, it would be simple enough to add to the pre-search rule-based knowledge to add another operator to the solution space search.

The incorporation of capacity analysis in experiment four and reservation selection in experiment eight succeeded in operationalizing strategic constraints into tactical constraints. Operationalization of constraints is performed by the creation and communication of constraints from one level to the next. General time constraints are strategic in nature. These constraints are used to guide the capacity analysis, whose results are time bounds on individual operations at the detailed scheduling level. Hence, a strategic due date constraint which simply states that due dates should be met, is used to guide and analyze the capacity analysis. This results in the generation of operation time bound constraints which is the operationalization of the due date constraint at a tactical level.

7.2.2. Constraint Representation

The scheduling of a job-shop is defined here as a constraint-directed search process. Hence, the representation of constraints is a focus of this work. The first question to be asked is whether the representation covers the set of possible constraints? By tying the representation of constraint types to the knowledge representation system (SRL) primitives, the representation guarantees that anything that can be represented in SRL, can have a constraint associated with it. Hence, ISIS provides constraint representation at the lowest, structural level. It is then up to the user to define higher level constraints, examples of which can be found in section 4.9.

A second important facet of ISIS's approach to constraint representation is the explicit representation of relaxation. *Any theory of constraint-directed reasoning must include a specification of how to relax constraints.* The specification of relaxation incorporates both predicate and choice set specifications of alternatives with preferences specified as utilities. Hence, ISIS provides the conceptual primitives to represent a variety of constraints, and more importantly their relaxations and associated utility. This approach subsumes the value set approach found in CONSTRAINTS (Steele, 1980), and the predicate bounding approach found in MOLGEN (Stefik, 1981a).

It is interesting to note, that the combination of a schema constraint and its relaxations is equivalent to a schema template which can be used for matching in the knowledge base. The incorporation of utilities in the relaxations provides a simple measure of match performance; bridging the gap between binary matching systems and belief-based reasoning systems.

A third facet is the integration of a constraint's declarative and procedural interpretation. Each constraint contains both its attributive/structural definition and its procedural interpretation. This has proved useful as less of ISIS is hidden in lisp procedures; and even those are placed in the model and subject to the inheritance mechanisms available in SRL.

7.2.3. Obligation

The concept of obligation in ISIS takes many forms. One of its primary forms is the importance of a constraint. The most common form found in AI, being synonymous with the coefficient associated with a term in a heuristic search evaluation function. Much of the generation of evaluation functions is an art, though Samuel (1963) has demonstrated optimizing behavior through machine learning techniques. And Berliner (1979; 1980) has noted that better search behavior is exhibited when coefficients can vary slowly with context. ISIS departs from this paradigm in a number of ways. First there does not exist a single, global *evaluation* function. The function is dynamically determined through the constraint resolution mechanism. Each constraint is a term of the newly defined evaluation function. It can be argued that all of the constraints form the evaluation function, but those not in the resolution set have a zero coefficient. As more constraints are resolved, their coefficients become non zero. But due to scheduling goals, each constraint in an importance partition has its importance reduced as the number of constraints in the partition increases. More interestingly, a scheduling goal results in differing constraint importance across classes of orders. Hence, the evaluation function changes depending on one's perspective (i.e., the order being scheduled). Berliner's results imply that a closer examination is required to further understand how ISIS's search behavior is affected by this "different" form of an evaluation function.

The time and causality primitives extends the notion of obligation from simple attachment of a constraint to a schema to a deeper understanding of duration and enablement. And where these primitives leave off, the simple context checking mechanism of a constraint provides a procedural loop hole for testing arbitrary conditions of applicability.

One of the most interesting concepts of obligation is that arising from inconsistencies. ISIS admits the possibility of inconsistent constraints being created such as two differing due dates from two sources. The removal of such inconsistencies relies upon the explicit representation of authority relations to determine which constraint has precedence.

7.2.4. Modes of Relaxation

ISIS provides two modes of relaxation: *analytic* and *generative*. Analytic relaxation is the examination of the state of the environment to determine which of the existing constraints should be relaxed. The result of this analysis is the explicit relaxation of one or more constraints. For example, the capacity analysis level can detect bottlenecks. As a result, certain constraints such as shifts and alternative routings can be relaxed in two senses: either an operator can be created that generates the relaxation during the solution space search, or the utility of the relaxation, such as an extra shift, may be increased. Analytic relaxation may also be performed in the pre- and post-analysis within a level.

Generative relaxation is built into the operators of the search space. An operator, by generating one or more new states in the schedule space either explicitly or implicitly generates relaxations. The choose-machine operator generates alternative machines for a given operation. The choose-machine operator is the generator associated with the machine discrete choice constraint. But an operator such as choose-reservation is the *composite of more than one constraint*. By choosing alternative queue positions, it may affect work in process constraints, due date constraints, queue ordering preferences, cost preferences, etc.

The issue being dealt with in this two step approach to relaxation is search complexity. The number of constraints, and their relaxations are large, making it impossible to consider all alternatives in reasonable amount of time. Hence, the search space analysis bounds the size of the search space, making the search manageable by methods such as beam search.

The next issue concerns the quality of the analysis. Section 5.1, makes the case that the analysis should be based on search space size, and constraint relevancy. But that the actual implementation is more "simplistic" in that it is a set of rules representing scheduling expertise. A first step towards a deeper analysis is the utilization of constraint interactions to determine which constraints should be relaxed, and at what scheduling level the relaxation should be performed. This is a post-analytic approach which is able to determine how decisions at a lower level have been affected by higher level decisions, and the direction of relaxation that should be performed at the higher level.

7.2.5. Integrating Planning and Scheduling

It would seem that ISIS cleanly separates the task of scheduling from planning. By pre-computing the graph of possible routings through the factory, the planning has been completed before scheduling begins. But this is not quite true. During scheduling, the choice of an operation or machine can be eliminated by a physical constraint such as one that measures the length of the blade and compares it to the size of the machine. Hence, the line separating scheduling from planning is fuzzy. The generation of alternative next operations could have been accomplished in a "planning" sense during the search of the solution space. And could be accomplished by substituting another "choose-operation" operator.

Ultimately, we would like to have ISIS carry out the planning step, resulting in a system that could perform "generative process planning", as it is called in manufacturing engineering. This would require much more knowledge of the part geometries and manufacturers descriptions of what type of processes each machine in the factory could perform. This is analogous to the problem addressed by Stefik (1981a; 1981b).

7.2.6. Search Anchoring

In the previous chapter, the issue of anchors in the search space was discussed. Both end-anchored, and opportunistic anchoring were discussed as search foci. It was decided that ISIS perform either backwards or forward search from the end or beginning anchors. The decision was based on the type of knowledge that ISIS currently receives in the form of constraints. ISIS could reason opportunistically from anywhere, that is, from the islands of certainty designated by constraints. But, currently, it does not have constraints which provide that type of information. Altering ISIS to perform an opportunistic search process would require adding rules to the pre-search analysis which would determine the initial islands of certainty, much like the WOSEQ module generated initial islands for Hearsay-II (Erman et. al., 1980). The beam search would have to be replaced by one similar to Hearsay-II which would extend only the highly rated islands. The transformation to an opportunistic approach will be made in ISIS-III.

In the current version of ISIS, the beam width is held to a constant. The selection of the beam width is another investigation of tradeoffs of time versus quality of schedule. Studies should be performed to determine what is a reasonable beam width; whether there is a relation between beam width and the constraints in the environment; Or whether the width should be dynamically determined by the some analysis of the current states in the beam.

7.2.7. System Efficiency

A major thrust of this research is to discover a theory of constraint-directed reasoning which is able to operate with near realtime performance. This goal has played a major part in deciding the types of search that ISIS is to use. The version of ISIS used in our experiments was written in FRANZ-LISP and run on a DEC VAX 11/780. Moderate effort was made towards increasing the efficiency of the search code, but much remains to be done. Never the less, statistics show that ISIS does not take a great deal of time to schedule a single lot. Table 7-1 depicts the average processing time to schedule a lot, including garbage collection, for each of the experiments in series 1.

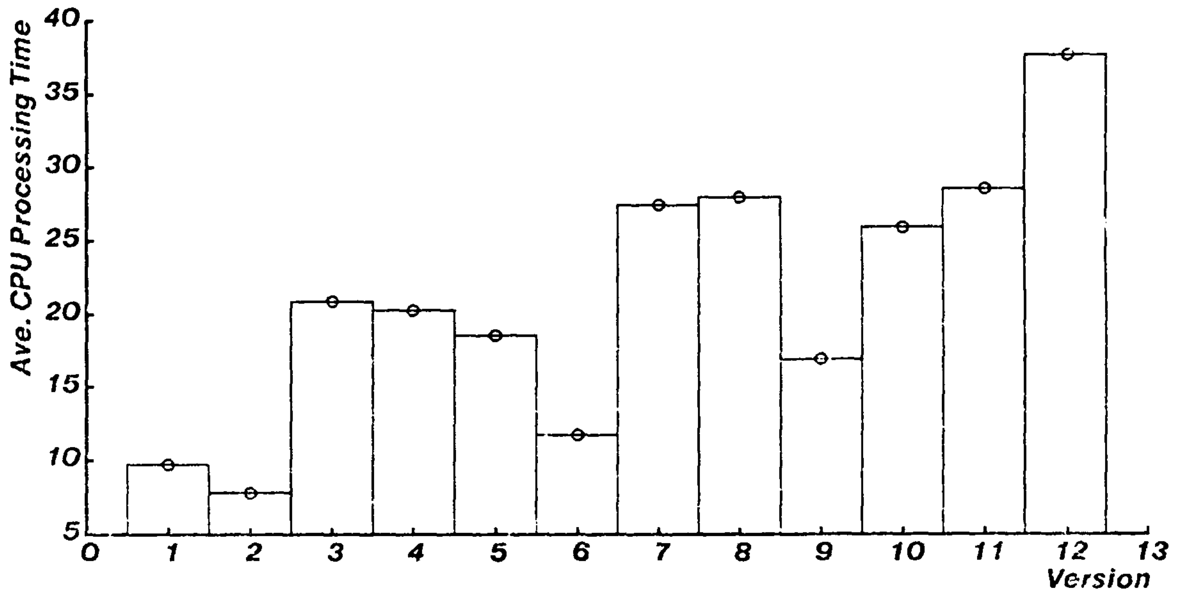


Table 7-1: Average CPU Processing Time (sec)

Much of this time is spent manipulating schemata; as many in the field have discovered, knowledge representations are expensive to use in both time and space.

Table 7-2 depicts the number of states searched, on the average, for each lot scheduled in an experiment in series 1. Compared to the number of states generated in chess games, ISIS generates very few. The combination of better constraint knowledge and the larger number of states generated combine to make experiments 8 and 12 better than experiment 1.

Our expectation is that the combination of more efficient software, and faster hardware will vastly improve ISIS's search efficiency, allowing ISIS to realize even better schedules.

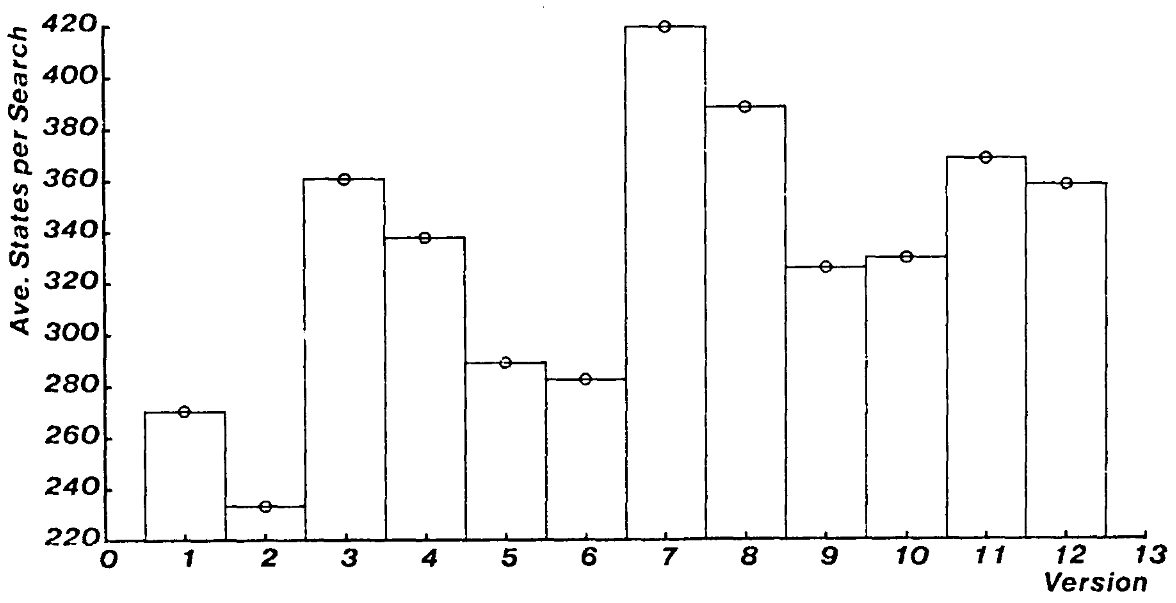


Table 7-2: Average Number States per Search

7.3. Contributions

The focus of this thesis has been the representation of and reasoning with constraints in the job-shop scheduling domain; with the added meta-constraint that it perform in near realtime required for the dynamic control of a factory.

Constraint Categorization. The first contribution is the progression towards a clearer definition of constraints. The categorization of constraints as range, relation, and schema (section 4.2), coupled with predicate and choice relaxation types (section 4.3) explicates what is being constrained, and their alternatives. This representation unifies the somewhat disparate approaches found in other constraint systems, without altering their approaches to reasoning with constraints. But in ISIS-II the definition of constraints extends beyond the above to consider constraint attributes found to be important in "real-world" tasks. Preferences amongst relaxations, and the differentiation (section 4.4) of constraints on a per order basis, guides ISIS-II's decision making. The definition of constraint interactions (section 4.5) enables ISIS-II to perform a better diagnosis of its schedules, and the elaboration of the different types of obligation (section 4.6) enables ISIS-II to determine the applicability of a constraint, and resolve inconsistencies amongst them.

Search Pre-Analysis. The second contribution is the explicit pre-analysis of the search space in order to define its bounds. Because of the innate size of the search space, ISIS-II must dynamically determine, for each order, based on the problem that it is trying to solve. The current analysis of the search space is rule-based. A better solution would be to perform the analysis of the space/time tradeoffs.

Constraint Strengthening and Relaxation. The third contribution is the methods by which ISIS-II can dynamically strengthen or relax constraints. The problem space analysis effectively partitions the space of constraints into those that are strengthened and those that can be relaxed. The analysis is based on attributes of the order, the previous scheduling history of the order, and on the current state of the plant. This analytic approach to constraint strengthening and relaxation admits knowledge from other sources such as a capacity planning system. The generative approach to constraint relaxation entails incorporating into the operators that define the search space, the ability to generate directly or indirectly relaxations of constraints.

Constraint Resolution. The fourth contribution is the approach to constraint resolution. That is, determining the set of constraints that are to be used to rate a state in the solution space. ISIS-II divides constraint resolution into three steps, first determining the set of constraints that are locally defined by local presence in the model, hierarchical imposition, lateral imposition, or exogenous imposition; constraints that are globally defined by presence in the search path; or relatively resolved by prioritizing them according to existing scheduling goals. These methods of resolution introduced the concept of invariant and transiant constraints, and the goal-directed dynamic prioritization of constraints.

Levels of Representation. A fifth contribution is the extension of scheduling to multiple levels of representation. Though reasoning in this manner is well known, the key difference here is the use of constraints as the means by which information is communicated between levels. The allows the degree of coupling of levels to vary according to the constraints' relaxations and their preference.

Constraint-Directed Diagnosis. A sixth contribution is the constraint-directed diagnosis proposed in section 5.5.3.2. A problem with search systems, especially multi-level search systems is the poor ability to diagnosis problem areas in their search space, and relate them to specific decisions at higher levels. While Fahlman's (1974) error analysis technique in robot planning, and dependency-directed backtracking are forms of diagnosis in search, the approach proposed in ISIS-II links diagnosis directly into the representation of constraints, hence formalizing the knowledge: the sensitivity of interaction and the direction that change should take place.

Knowledge Measurement. A seventh contribution is the steps taken towards the measurement of the effects of incremental additions of knowledge both within a search level and at different levels of search. While the current results are minimal, experiments continue to be performed.

Pragmatic Scheduling. And the eighth, and last contribution, is to management science and industry in general. ISIS-II represents the first step forward in the development of a near realtime job-shop scheduling system that can incorporate, reason with and relax a large and expanding set of constraints that define the environment.

7.4. Future Research

Research tends to raise as many questions as it answers. ISIS-II is no different. It would be appropriate to note at this time that the -II on the end of ISIS was deliberately chosen. ISIS-I was a throw a way system, more a vehicle to understand the job-shop scheduling problem than anything else. ISIS-II was designed in parallel with ISIS-III. The purpose of ISIS-II being to investigate the representation and relaxation of constraints with a beam search style search algorithm. The choice of a beam search approach was made with full knowledge that a more opportunistic reasoning approach could possibly be better, but would also increase the complexity of the research. I decided that ISIS-III would explore the opportunistic/distributed reasoning questions, while restricting ISIS-II to the basic questions on representation, resolution, and relaxation. Within this framework, ISIS-II raised the following issues which should serve as future research.

The first problem is that the development of constraints, their utilities, and their priorities is very much an art. Efficacy of a constraint representation cannot be determined other than by continued analysis of system performance. One of the few systems that has attended to this problem has been PROSPECTOR (Gaschnig, 1980). A sensitivity analysis was performed on the rule-base to determine the effects or rules and their beliefs. At least two approaches can be taken towards solving this problem. The first would be to develop a system by which users could state preferences, and the system derive utility functions and priorities. The second would be a machine learning approach. By coupling sensitivity analysis with feedback on its performance, the system could alter its goals and utilities in a more intelligent manner. The former approach is quite difficult, as exemplified by the work in mathematical psychology in dealing with inconsistent preferences and beliefs.

ISIS-II was designed to incorporate a multitude of constraints, but the decision to use a beam search anchored at either the initial or goal state and did not allow for it to take advantage of "islands of certainty" in the solution space. ISIS-III will have the ability to search from any island in an opportunistic manner.

ISIS-II also allows the incorporation of constraints from other, hierarchical analyses such as capacity planning. Though a hierarchical analysis system was constructed, it was incorporated using simple top down control. An opportunistic form of control, such as found in Hearsay-II would require implementation of the previous suggestion. Again, this capability will appear in ISIS-III. Secondly levels of representation should be added, to incorporate other perspectives on the scheduling state.

Currently, ISIS performs complete scheduling of all lots that it is aware of. In reality, only detail schedules need to be developed for lots which are to be delivered in a short time frame. For lots which are not required for many months or years, a simple capacity analysis would suffice. This raises the issue of to what level of detail should ISIS perform its scheduling. Ideally, the level of detail should reflect the level of certainty the system has about the plants future states.

The issue of complexity and the constraint of providing a reasonable response time, resulted in the formulation of the search space analysis step in the scheduling process. This analysis is vital to the system in that it bounds the space of schedules. If the analysis is poor, than the schedules are poor. Currently, ISIS-II performs a rule-based analysis of the problem space. The rules are derived from both experts and myself. Ultimately, the system will have to take a more analytical approach to the analysis of the constraints, their utilities, and their effect on the size of the search space. A first step towards this is the interaction diagnosis proposed in section 5.5.3.2.

Lastly, ISIS-II is really a machine centered scheduling system. Resource requirements are determined by operation/machine routings. There is no explicit attempt to smooth resource utilization, other than noting their inavailability and choosing an alternative schedule. A next step would be to develop a system that would smooth the usage of all resources in parallel. This is a goal of ISIS-III. The approach being investigated there is to create a distributed system. One scheduler for each resource. Each processes schedules its own resource utilizing both its own local constraints and goals, and the constraints communicated to it from other processes. If a machine scheduler requires a tool, it communicates that need as a constraint to the tool scheduler. The system iterates, passing constraints at each iteration. Relaxing some, strengthening others until the group of schedules converge.

Chapter 8
References

Adam N.R., and J. Surkis, (1980), "Priority Update Intervals and Anomalies in Dynamic Ratio Type Job Shop Scheduling Rules", *Management Science*, Vol. 26, No. 12, Dec. 1980, pp. 1227-1237.

Allen J.F., (1981). "Maintaining Knowledge about Temporal Intervals" Dept. of Computer Science, University of Rochester, Technical Report TR-86.

Amarel S., (1968), "On Representations of Problems of Reasoning about Actions", *Machine Intelligence*, Vol. 3, D. Michie (Ed.), Edinburgh: Edinburgh University Press.

Arumugam V., and S. Ramani, (1980), "Evaluation of Value Time Sequencing Rules in a Real World Job Shop", *Journal of the Operational Research Society*, Vol. 31, No. 10., pp. 895-904.

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

Ballard D.H., C.M. Brown, and J.A. Feldman, (1977), "An Approach to Knowledge-Directed Image Analysis", *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge MA.

Bartlett F.C., (1932), *Remembering*, Cambridge: Cambridge University Press.

Berliner H.J., (1973), "Some Necessary Conditions for a Master Chess Program", *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pp. 77-85.

Berliner H.J., (1979), "The B* Tree Search Algorithm: A Best First Proof Procedure", *Artificial Intelligence*, Vol. 12.

Berliner H.J., (1979), "On the Construction of Evaluation Functions for Large Domains", *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 53-55.

Berliner H.J., (1980), "Making Judgements", *Proceedings of the First Annual Conference of the American Association for Artificial Intelligence*, pp. 134-137.

Bobrow D., and T. Winograd, (1977), "KRL: Knowledge Representation Language," *Cognitive Science*. Vol 1, No. 1, 1977.

Borning A., (1979), "THINGLAB: A Constraint-Oriented Simulation Laboratory", SSL-79-3, Xerox Palo Alto Research Center, July 1979.

Brachman R.J., (1979), "On the Epistemological Status of Semantic Networks" in*Associative Networks: Representation and Use of Knowledge by Computers*, (N.V. Findler, Ed.), pp. 3-50, New York: Academic Press.

Bruce B.C., (1972). "A Model for Temporal References and Its Application in a Question Answering Program", *Artificial Intelligence*, Vol. 3, pp. 1-25.

Bullers W.I., S.Y. Nof, and A.B. Whinston, (1980), "Artificial Intelligence in Manufacturing Planning and Control", *AIIE Transactions*, December 1980, pp. 351-363.

Carbonell J.G., (1979). "The Counterplanning Process: A Model of Decision Making in Adverse Situations". Technical Report, Computer Science Department, Carnegie-Mellon University, February 1979.

Conway R.W., (1965), "Priority Dispatching And Job Lateness in a Job Shop", *Journal of Industrial Engineering*, Vol. 16, pp. 228-237.

Cook S., (1971), "The Complexity of Theorem-proving Procedures", In *Proceedings of the 3rd Annual ACM Symposium Theory of Computing*, Association of Computing Machinery.

Davis L., (1980), "A Logic Model for Constraint Propagation", Technical Report TR-137, Computer Sciences Dept., University of Texas, Austin Texas.

Davis R., (1976), "Applications of Meta Level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases", Ph.D. Thesis, Computer Science Department, Stanford University, Report No. STAN-CS-76-552.

Davis R. and B. Buchanan, (1977), "Meta-level Knowledge: Overview and Applications," *Fifth International Joint Conference on Artificial Intelligence*, Cambridge MA, August 1977.

Duda R.O., P.E. Hart, P. Barrett, J.G. Gaschnig, K. Konolige, R. Reboh, and J. Slocum, (1978), "Development of the Prospector Consultation System for Mineral Exploration: Final Report", Tech. Rep., SRI International, Menlo Park CA, Oct. 1978.

Engleman C., E. Scarl, and C. Berg, (1980), "Interactive Frame Instantiation", *Proceedings of the American Association for Artificial Intelligence*, pp. 184-186, Stanford University.

Erman L.D., F. Hayes-Roth, V.R. Lesser, and D.R. Reddy, (1980), "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty", *Computing Surveys*, Vol. 12, No. 2, June 1980, pp. 213-253.

Fahlman S., (1974), "A Planning System for Robot Construction Tasks", *Artificial Intelligence*, Vol. 5, pp. 1-50.

Ferguson R.L., and C.H. Jones, (1969), "A Computer Aided Decision System", *Management Science*, Vol. 15, No. 10, pp. 550-561.

Fikes R.E., (1970), "REF-ARF: A System for Solving Problems Stated as Procedures", *Artificial Intelligence*, Vol. 1, pp. 27-120.

Fikes R.E., and N.J. Nilsson, (1971), "Strips: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2,pp 189-208.

Fox M.S., (1979), "On Inheritance in Knowledge Representation", *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo Japan.

Fox M.S., (1981), "The Intelligent Management System: An Overview", Technical Report, Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, August 1981. Also appeared in *Processes and Tools for Decision Support*, H.G. Sol (Ed.), North-Holland Pub. Co.

Fox M.S. and D.J. Mostow, (1977), "Maximal Consistent Interpretations of Errorful Data In Hierarchically Modelled Domains", *Fifth International Joint Conference on Artificial Intelligence*, Cambridge MA, 1977.

Fox M.S., S. Lowenfeld, and P. Kleinosky, (1983), "Techniques for Sensor-Based Diagnosis", *Proceedings of the International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983.

Fukumori K., (1980), "Fundamental Scheme for Train Scheduling", MIT AI Memo No. 596, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge MA.

Godin V.B., (1978), "Interactive Scheduling: Historical Survey and State of the Art", *AIIE Transactions*, Vol. 10, Sept. 1978, pp. 331-337.

Goldstein I.P., (1977), "Bargaining Between Goals", *Proceedings of the Fifth International Conference on Artificial Intelligence*, Cambridge MA, pp. 175-180.

Goldstein I.P., and R.B. Robert, (1977), "NUDGE: A Knowledge-Based Scheduling Program", MIT AI Memo 405, Cambridge MA.

Hax A.C., and J.J. Golovin, (1978), "Hierarchical Production Planning Systems", In *Studies in Operations Management*, (Ed. A.C. Hax), North Holland Elsevier.

Hayes D.G., (1973), "Types of Processes on Cognitive Networks", *International Conference on Computational Linguistics.*

Hayes-Roth B., and F. Hayes-Roth, (1979), "A Cognitive Model of Planning", *Cognitive Science*, Vol. 3, No. 4, pp. 275-310.

Hayes-Roth F., and V.R. Lesser, (1976), "Focus of attention in a distributed logic speech understanding system", *Proceedings of the 1976 IEEE International Conference on Acoustics, Speech and Signal Processing*, Philadelphia, 1976, 416-420.

Kahn K., and G.A. Gorry, (1977), "Mechanizing Temporal Knowledge", *Artificial Intelligence*, Vol. 9, pp. 87-108.

Kosy D. and V. Dhar, (1983), "An Analysis of Long Range Planning", Technical Report, Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA.

Lenstra J.K., and A.H.G. Rinnooy Kan, (1980), "The Scheduling of Several Machines", Dept. of Operational Research, The Mathematical Center, Amsterdam, Aug. 1980.

London P.E., (1978), "Dependency Networks as a Representation for Modelling in General Problem Solvers", (Ph.D. Thesis), Technical Report TR-698, Department of Computer Science, University of Maryland, Sept. 1978.

Lowerre B., (1976), "The HARPY Speech Recognition System", (Ph.D. Thesis), Tech. Rep., Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA.

McCalla G., P. Schneder, R. Cohen, and H. Levesque, (1978), "Investigations into Planning and Executing in an Independent and Continuously Changing Microworld", AI Memo 78-2, Dept. of Computer Science, University of Toronto.

McDermott D., (1982) "A Temporal Logic for Reasoning About Processes and Plans", *Cognitive Science*, Vol. 6, pp. 101-155.

McDermott J., and B. Steele, (1981), "Extending a Knowledge-Based System to Deal with Ad Hoc Constraints", *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver B.C., August 1981.

McDonald G.W., & T.J. Hodgson, (1980), "Interactive Scheduling of a Generalized Flowshop", Research Report No. 80-16, Industrial & Systems Engineering Dept., University of Florida, Gainesville, Fl.

Meehan J.R., (1980), "Everything You Always Wanted to Know About Authority Structures But Were Unable to Represent", *Proceedings of the American Association for Aritificial Intelligence*, pp. 212-214, Stanford University.

Minsky M., (1975), "A Framework for Representing Knowledge", In *The Psychology of Computer Vision*", P. Winston (Ed.), New York: McGraw-Hill.

Newell, A., (1969), "Heuristic programming: ill-structured problems", In J. Aronofsky (Ed.), *Progress in Operations Research* 3. New York: Wiley, 1969, 360-414.

Newell A., and H.A. Simon, (1956), "The Logic Theory Machine: A Complex Information Processing System", *IRE Transactions on Information Theory*, Vol. IT-2, No. 3, pp. 61-79.

Newell A., and H.A. Simon, (1963), "GPS: A Program that Simulates Human Thought", In *Computers and Thought*, E. Feigenbaum and J. Feldman (Eds.), New York: McGraw-Hill Co.

Nilsson N.J., (1971), *Problem-Solving Methods in Artificial Intelligence*, New York, N.Y.: McGraw-Hill.

Petri C.A., (1966), "Communication with Automata", Supplement 1 to Tech. Report RAD C-TR-65-337, Vol. 1, Griffiss Air Force Base, New York.

Rachamadugu R., (1982), "Myopic Heuristics in Job Shop Scheduling", (PhD Thesis), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh PA.

Reddy Y.V., and M.S. Fox, (1982), "KBS: A Flexible Simulation System", Technical Report CMU-TR-82-1, The Robotics Institute, Carnegie-Mellon Univeristy, Pittsburgh PA.

Rieger C., and M. Grinberg, (1977), "The Causal Representation and Simulation of Physical Mechanisms", Dept. of Computer Science, University of Maryland, Technical Report TR-495.

Rubin S., (1978), "The ARGOS Image Understanding System", Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Russell D.M., (1979), "Where Do I Look Now?: Modelling and Inferring Object Locations by Constraints", Technical Report, Computer Science Department, University of Rochester, Rochester NY.

Sacerdoti, E.D., (1974), "Planning in a Hierarchy of Abstraction Spaces", *Artificial Intelligence*, Vol 5, no.2.

Sacerdoti E.D., (1975), "A Structure for Plans as Behavior", (Ph.D. Thesis), Computer Science Dept., Stanford University.

Samuel A., (1963), "Some Studies in Machine Learning using the Game of Checkers," In *Computers and Thought*, E. Feigenbaum and J. Feldman (Eds.), New York: McGraw-Hill Book Co.

Shortliffe E.H., (1976), *Computer-Based Medical Consultations: MYCIN*, New York: American Elsevier.

Simon H.A., (1962), "Scientific Discovery and and the Psychology of Problem Solving", *Minds and Cosmos*, Kolodny (Ed.).

Smith S.F., (1983), "Exploiting Temporal Knowledge to Organize Constraints" Technical Report, Robotics Institute, Carnegie-Mellon University, Pittsburgh PA.

Stanfill C., (1981), "Partitioned Constraint Networks: A Prescripton for Representation Machines", Dept. of Computer Science, University of Maryland, Technical Report TR-1037.

Stefik M., (1979), "An Examination of a Frame-Structured Representation System", *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo, August 1979.

Stefik M., (1981a), "Planning with Constraints (MOLGEN: Part 1)", *Artificial Intelligence*, Vol. 16, pp. 111-140.

Stefik M., (1981b), "Planning and Meta-Planning (MOLGEN: Part 2)", *Artificial Intelligence*, Vol. 16, pp. 141-170.

Steele G.L., (1980), "The Definition and Implementation of a Computer Programming Language Based on Constraints", (PhD Thesis), MIT tech. rep. AI-TR-595, Cambridge MA.

Sussman G., (1975), *A Computational Model of Skill Acquisition*, New York: American Elsevier.

Sutherland I.E., (1963), "Sketchpad: A Man-Machine Graphical Communication System", MIT Lincoln Lab. TR 296.

Vere S., (1981), "Planning in Time: Windows and Durations for Activities and Goals", Technical Report, Jet Propolsion Lab.,

Waltz D., (1975), "Understanding Line Drawings of Scenes with Shadows", in P.H. Winston (Ed.), *The Psychology of Computer Vision*, New York N.Y.: McGraw-Hill.

Wilensky R., (1980), "Meta-Planning", *Proceedings of the First Annual Conference of the American Association for Artificial Intelligence*, Stanford CA, pp. 334-336.

Wirth N., (1971), "Program Development by Stepwise Refinement", *CACM* Vol. 14, No. 4, 221-7.

Woods W. A., et al., (1976), Final report of the BBN speech understanding system research. Cambridge: Bolt, Beranek, Newman, 1976.

Wright J.M., and Fox M.S., (1983), "SRL: Schema Representation Language", Technical Report, Robotics Institute, Carnegie-Mellon University, Pittsburgh PA.

Zucker S.W., (1976), "Relaxation Labelling and the Reduction of Local Ambiguities", In *Pattern Recognition and Artificial Intelligence*, C.H. Chen (Ed.), New York: Academic Press.

Zucker S.W., (1977), "Vertical And Horizontal Processes in Low Level Vision", Report No. 77-4, Electrical Engineering Dept., McGill University, Montreal, Quebec.

Appendix A
Test Constraints

The following is the list of constraints which were used in both series 1 and series 2 experiments.

Preference Constraints

Preference constraints were used which covered preferences on how to order lots in a queue according to physical characteristics, and choice of machine for an operation, again according to physical characteristics.

q-hand prefer to sequence together in a machine's queue turbine blades of similar hand.

true-utility:	1.1
false-utility:	1.0

q-root prefer to sequence together in a machine's queue turbine blades of similar root.

true-utility:	1.1
false-utility:	1.0

length preferences different machines had preferences for the length of a turbine blade they would work on.

true-utility:	1.1
false-utility:	1.0

lug preferences different machines had preferences for the number of lugs on a turbine blade they would work on.

true-utility:	1.1
false-utility:	1.0

Continuous Constraints

Continuous constraints were used to constrain schedules to meet due date, work in process , and queue stability goals. The utility for these constraints are specified as a piece wise linear equation in the form of x-y pairs.

due date Each priority class had its own due date constraint providing a utility which varied with how early or late an order was. The due date constraint calculates a due date for a lot's partial schedule based on the amount of time in the plant so far, and the amount of time, theoretically, till completion. X is the number of days after the due date the order is expected to be shipped.

utility:	Class	X	Y
	FO	0	1.3
		7	0.1
	CR	0	1.3
		7	0.1
	SD	-14	1
		0	1.2
		7	0.1
	SH	-28	0.2
		0	1.2
		14	0.2
	SO-28	0.2	
		0	1.2
		14	0.2
	ST	-7	1
		0	1.1
		7	1

queue stability determined the utility of pre-empting another lots reservation for a machine when the lot is close to setup. X is the number of days left till the pre-empted lot is to be run.

utility:	X	Y
	-1	0
	0	0.1
	7	1

wip determines the acceptability of the amount of time the lot is spending in process. The utility returned is a function of the number of days in process and the theoretical minimum the lot should spend if the shop was empty.

Discrete Choice Constraints

next operation Each operation had a list of next operations to choose from. The utility of each choice was uniform and was not considered in testing.

machine Each operation had a list of machines to choose from. The utility of each choice was uniform and was not considered in testing.

tools Each operation had a list of tools to choose from. The utility of each choice was uniform and was not considered in testing.

Appendix B
Test Data

The following lists the test lots used in version 1 of each experiment in series one and two. They were generated according to the distributions described in section 6.1. (Other test data was generated from the same distributions for the further testing of experiments one and eight in series one.) Each lot has a priority class, a requested start date in the form of a triple: (week day hour), a requested due date, and the number of turbine blades in the lot. The lots were then ordered by start date, i.e., the time the order became known to the plant, and scheduled in that order.

Ship Direct Lots

Lot	Priority	Start date	Due date	blades
tlot-3-3	SD	(58 0 0)	(110 0 0)	186
tlot-4-28	SD	(64 0 0)	(114 0 0)	175
tlot-4-13	SD	(64 0 0)	(115 0 0)	168
tlot-3-4	SD	(64 0 0)	(109 0 0)	204
tlot-3-10	SD	(65 0 0)	(113 0 0)	191
tlot-1-14	SD	(73 0 0)	(104 0 0)	139
tlot-3-28	SD	(75 0 0)	(113 0 0)	167
tlot-5-3	SD	(77 0 0)	(118 0 0)	202
tlot-2-3	SD	(79 0 0)	(105 0 0)	194

Table B-1: SD Lots

Service Order Lots

order	Priority	Start date	Due date	blades
tlot-4-16	SO	(63 0 0)	(114 0 0)	161
tlot-4-29	SO	(66 0 0)	(115 0 0)	163
tlot-4-34	SO	(66 0 0)	(116 0 0)	121
tlot-1-4	SO	(66 0 0)	(102 0 0)	151
tlot-4-3	SO	(67 0 0)	(117 0 0)	123
tlot-3-5	SO	(68 0 0)	(109 0 0)	121
tlot-5-28	SO	(69 0 0)	(119 0 0)	202
tlot-3-17	SO	(69 0 0)	(112 0 0)	140
tlot-3-21	SO	(70 0 0)	(112 0 0)	120
tlot-3-27	SO	(70 0 0)	(113 0 0)	203
tlot-2-19	SO	(70 0 0)	(105 0 0)	134
tlot-1-12	SO	(70 0 0)	(100 0 0)	181
tlot-6-18	SO	(74 0 0)	(123 0 0)	165
tlot-5-24	SO	(74 0 0)	(119 0 0)	198
tlot-4-21	SO	(74 0 0)	(115 0 0)	134
tlot-4-2	SO	(76 0 0)	(116 0 0)	200
tlot-2-15	SO	(77 0 0)	(105 0 0)	164
tlot-6-16	SO	(78 0 0)	(125 0 0)	174
tlot-3-29	SO	(79 0 0)	(112 0 0)	155

Table B-2: SO Lots

Stock Lots

order	Priority	Start date	Due date	blades
tlot-1-5	ST	(71 0 0)	(101 0 0)	155
tlot-1-3	ST	(71 0 0)	(101 0 0)	190
tlot-1-2	ST	(71 0 0)	(101 0 0)	152
tlot-1-11	ST	(72 0 0)	(102 0 0)	197
tlot-1-16	ST	(73 0 0)	(103 0 0)	169
tlot-1-8	ST	(74 0 0)	(104 0 0)	190
tlot-2-17	ST	(77 0 0)	(107 0 0)	168
tlot-2-13	ST	(77 0 0)	(107 0 0)	202
tlot-2-9	ST	(77 0 0)	(107 0 0)	165
tlot-2-14	ST	(78 0 0)	(108 0 0)	176
tlot-2-6	ST	(78 0 0)	(108 0 0)	154
tlot-3-14	ST	(79 0 0)	(109 0 0)	207
tlot-3-2	ST	(79 0 0)	(109 0 0)	160
tlot-3-30	ST	(80 0 0)	(110 0 0)	140

Table B-3: ST Lots

Shop Order Lots

order	Priority	Start date	Due date	blades
tlot-1-17	SH	(41 0 0)	(101 0 0)	164
tlot-5-18	SH	(45 0 0)	(121 0 0)	198
tlot-2-10	SH	(47 0 0)	(108 0 0)	206
tlot-4-10	SH	(49 0 0)	(116 0 0)	200
tlot-1-7	SH	(49 0 0)	(100 0 0)	124
tlot-1-6	SH	(49 0 0)	(101 0 0)	208
tlot-2-12	SH	(50 0 0)	(106 0 0)	147
tlot-7-5	SH	(51 0 0)	(126 0 0)	188
tlot-6-6	SH	(51 0 0)	(123 0 0)	195
tlot-5-12	SH	(51 0 0)	(119 0 0)	125
tlot-3-12	SH	(52 0 0)	(111 0 0)	126
tlot-2-8	SH	(52 0 0)	(106 0 0)	180
tlot-1-15	SH	(53 0 0)	(104 0 0)	151
tlot-5-17	SH	(55 0 0)	(120 0 0)	122
tlot-2-2	SH	(55 0 0)	(108 0 0)	189
tlot-1-9	SH	(55 0 0)	(103 0 0)	166
tlot-7-14	SH	(56 0 0)	(127 0 0)	204
tlot-6-19	SH	(57 0 0)	(124 0 0)	140
tlot-6-5	SH	(57 0 0)	(122 0 0)	144
tlot-4-4	SH	(59 0 0)	(114 0 0)	156
tlot-7-7	SH	(61 0 0)	(128 0 0)	202
tlot-8-17	SH	(62 0 0)	(134 0 0)	150
tlot-2-18	SH	(64 0 0)	(108 0 0)	203
tlot-6-14	SH	(66 0 0)	(123 0 0)	130
tlot-9-30	SH	(68 0 0)	(135 0 0)	157
tlot-6-8	SH	(68 0 0)	(125 0 0)	210
tlot-3-8	SH	(69 0 0)	(113 0 0)	163
tlot-4-35	SH	(71 0 0)	(117 0 0)	133
tlot-4-6	SH	(71 0 0)	(116 0 0)	136
tlot-11-17	SH	(72 0 0)	(145 0 0)	162
tlot-7-13	SH	(72 0 0)	(126 0 0)	146
tlot-11-7	SH	(73 0 0)	(147 0 0)	165
tlot-7-4	SH	(73 0 0)	(129 0 0)	120
tlot-7-6	SH	(74 0 0)	(127 0 0)	122
tlot-4-37	SH	(74 0 0)	(115 0 0)	178
tlot-4-22	SH	(74 0 0)	(116 0 0)	192
tlot-3-31	SH	(74 0 0)	(110 0 0)	193
tlot-4-24	SH	(75 0 0)	(114 0 0)	202
tlot-4-7	SH	(75 0 0)	(117 0 0)	176
tlot-9-31	SH	(77 0 0)	(136 0 0)	196
tlot-4-5	SH	(78 0 0)	(115 0 0)	122
tlot-12-6	SH	(79 0 0)	(149 0 0)	170
tlot-6-17	SH	(79 0 0)	(122 0 0)	164

Table B-4: SH Lots

Appendix C
Series 1 Experiments

This appendix documents the results of the experiments 1 through 12 in series 1. The first series of experiments were based on a model and set of constraints provided by the scheduling expert. In experiments one through eight, all forced outage (FO), critical replacement (CR) and ship direct (SD) orders are scheduled backwards from their due date, and the rest are scheduled forwards from their order arrival date. In experiments nine through twelve, all orders are scheduled backwards Orders are assigned a prioirty according to their classification. The ordering from high priority to lowest is FO, CR, SD, SO, ST. The types of constraints included in the series are:

- Alternative operations.
- Alternative machines.
- Pre-specified due date.
- Pre-specified start date.
- Work in Process.
- Queue ordering constraints to reduce setup time.
- Lug and length mchine physical constraints.
- Resource availability.
- Shop stability.

The following tables and histograms summarize the experiments. Note that normalized statistics are normalized to 100 turbine blades per lot. CPU times are given for a VAX/11-780 running FRANZ LISP under Berekley UNIX.

Version	1	2	3	4
Reservation Mode	Eager	Eager	Eager	Eager
Due Date	Opt.	Pess.	Opt.	Pess.
Capacity Analysis	No	No	Yes	Yes
Scheduling Direction	per Lot	per Lot	per Lot	per Lot
Number of Tardy Lots	65	72	13	13
Ave. Tardiness (only tardy lots)	130.53	158.0	114.98	121.76
Ave. Tardiness (all lots)	99.82	133.83	17.59	18.62
Ave. Delta Start Date	127.56	167.69	107.43	118.66
Ave. Lateness (Delta Due Date)	99.25	133.63	-12.93	-8.04
Ave. WIP	305.15	299.39	213.09	206.74
Ave. Processing Time	30.31	29.6	30.6	30.38
Ave. Percent Processing	21.62	22.28	28.64	29.46
Ave. Queue Time	274.84	269.78	182.49	176.37
Normalized WIP	191.5	188.3	133.8	129.2
Normalized Proc. Time	18.2	17.8	18.4	18.3
Normalized Queue Time	173.3	170.4	115.3	111.0
Normalized Tardy Time	80.6	98.1	70.2	74.2
Makespan	857.4	923.4	729.5	737.6
Ave. Machine Idle Time	754.4	822.8	625.5	634.3
Ave. % Global Machine Utilization	12.0	10.9	14.3	14.0
Ave. Local Duration	242.0	292.8	344.2	336.5
Ave. Local Machine Idle Time	138.9	192.2	240.2	233.2
Ave. % Local Machine Utilization	37.2	33.6	36.3	24.5
Ave. Machine Wait Time	23.2	17.7	14.0	11.4
Ave. Number Machine Reservations	28.8	28.8	28.8	28.8
Ave. Number States per Search	270.40	233.33	360.9	338.01
Ave. CPU Processing Time (sec)	9.73	7.83	20.92	20.31
Ave. CPU Execution Time	7.04	5.35	12.93	12.63
Ave. CPU GC Time	2.69	2.48	7.98	7.69
Ave. %CPU Execution Time	75.96	71.97	66.10	65.93

Table C-1: Series 1 Experiments 1-4 Statistics Comparison

Version	5	6	7	8
Reservation Mode	Wait	Wait	Wait	Wait
Due Date	Opt.	Pess.	Opt.	Pess.
Capacity Analysis	No	No	Yes	Yes
Scheduling Direction	per Lot	per Lot	per Lot	per Lot
Number of Tardy Lots	71	74	14	17
Ave. Tardiness (only tardy lots)	210.35	183.3	117.36	103.5
Ave. Tardiness (all lots)	175.70	159.65	19.33	20.7
Ave. Delta Start Date	232.46	221.63	148.27	149.66
Ave. Lateness (Delta Due Date)	175.42	159.47	1.55	8.61
Ave. WIP	276.4	271.28	186.73	192.4
Ave Processing Time	30.39	30.56	30.19	29.40
Ave. Percent Processing	28.37	29.16	32.03	28.11
Ave. Queue Time	246.01	240.7	156.54	162.99
Normalized WIP	170.5	168.2	117.4	120.0
Normalized Proc. Time	18.2	18.4	18.1	17.7
Normalized Queue Time	152.2	149.8	99.2	102.3
Normalized Tardy Time	128.8	112.8	70.6	62.51
Makespan	755.0	750.0	583.25	588.9
Ave Machine Idle Time	651.7	646.2	480.6	489.0
Ave. % Global Machine Utilization	13.7	13.9	17.6	17.0
Ave. Local Duration	424.8	400.4	362.7	335.9
Ave. Local Machine Idle Time	321.5	296.5	260.1	235.9
Ave. % Local Machine Utilization	26.0	25.1	24.8	27.3
Ave. Machine Wait Time	26.8	26.5	9.6	9.6
Ave. Number Machine Reservations	28.8	28.8	28.8	28.8
Ave. Number States per Search	289.19	282.8	419.59	388.6
Ave. CPU Processing Time (sec)	18.58	11.74	27.41	27.93
Ave. CPU Execution Time	14.71	7.34	16.83	17.39
Ave. CPU GC Time	3.87	4.40	10.59	10.54
Ave. %CPU Execution Time	82.47	68.12	66.57	65.31

Table C-2: Series 1 Experiments 5-8 Statistics Comparison

Version	9	10	11	12
Reservation Mode	Eager	Eager	Wait	Wait
Due Date	N/A	N/A	N/A	N/A
Capacity Analysis	No	Yes	No	Yes
Scheduling Direction	Back	Back	Back	Back
Number of Tardy Lots	0	0	0	0
Ave. Tardiness (only tardy lots)	0	0	0	0
Ave. Tardiness (all lots)	0	0	0	0
Ave. Delta Start Date	193.78	196.44	183.79	194.23
Ave. Lateness (Delta Due Date)	35.86	-35.86	-57.47	-58.17
Ave. WIP	103.81	101.15	92.19	81.04
Ave. Processing Time	30.28	30.13	29.99	29.79
Ave. Percent Processing	57.79	58.0	63.0	64.03
Ave. Queue Time	73.53	71.02	62.2	51.25
Normalized WIP	63.5	61.9	56.5	49.3
Normalized Proc. Time	18.2	18.1	18.0	17.89
Normalized Queue Time	45.3	43.7	38.5	31.4
Normalized Tardy Time	0	0	0	0
Makespan	669.5	669.5	698.0	687.3
Ave. Machine Idle Time	566.5	567.1	596.1	586.1
Ave. % Global Machine Utilization	15.4	15.3	14.6	14.7
Ave. Local Duration	383.7	384.0	414.8	407.3
Ave. Local Machine Idle Time	280.7	281.6	312.8	306.1
Ave. % Local Machine Utilization	26.8	34.7	18.4	26.2
Ave. Machine Wait Time	3.9	4.2	3.2	2.4
Ave. Number Machine Reservations	28.8	28.8	28.8	28.8
Ave. Number States per Search	325.96	329.85	368.59	358.5
Ave. CPU Processing Time (sec)	16.95	25.9	28.51	37.64
Ave. CPU Execution Time	11.07	18.14	19.85	26.83
Ave. CPU GC Time	5.88	7.76	8.67	10.81
Ave. %CPU Execution Time	75.95	74.36	78.46	76.81

Table C-3: Series 1 Experiments 9-12 Statistics Comparison

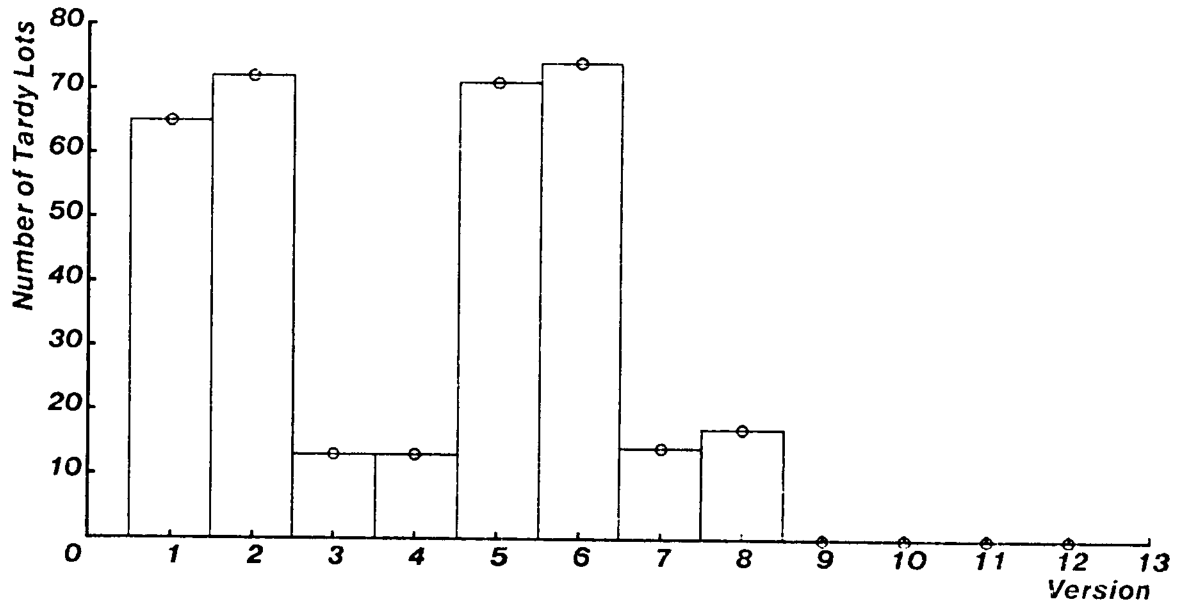


Table C-4: Number of Tardy Lots

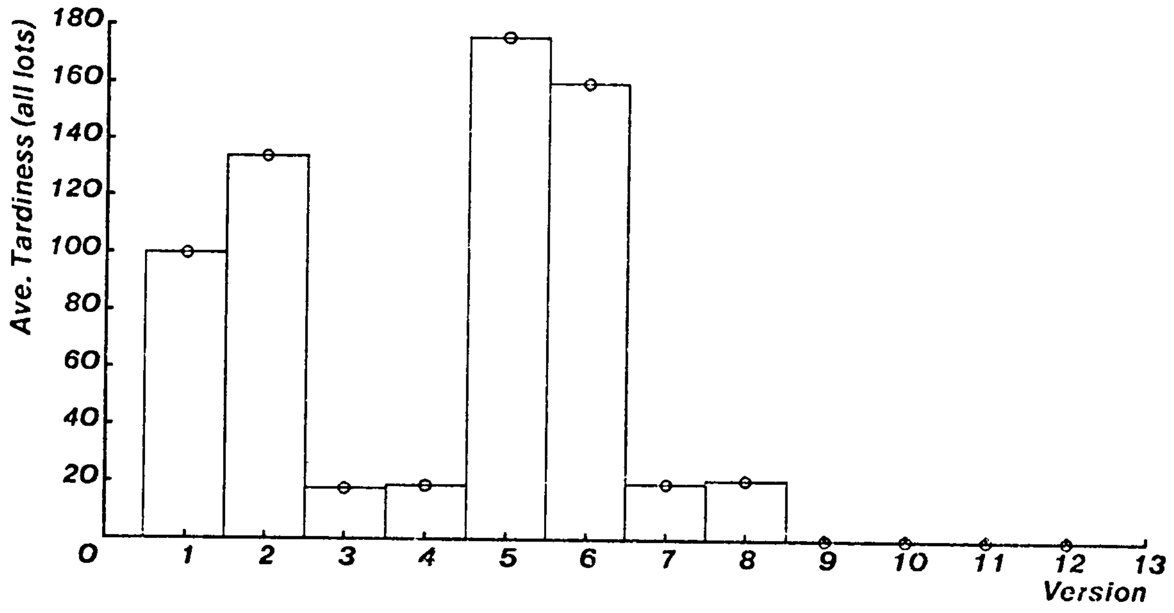


Table C-5: Average Tardiness (all lots)

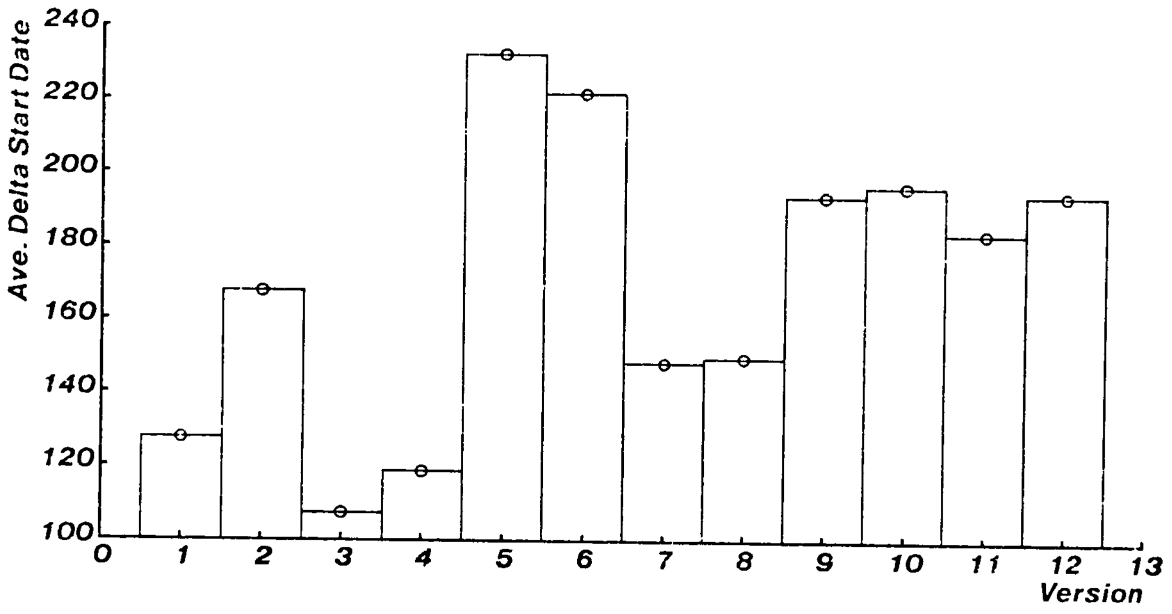


Table C-6: Average Delta Start Date

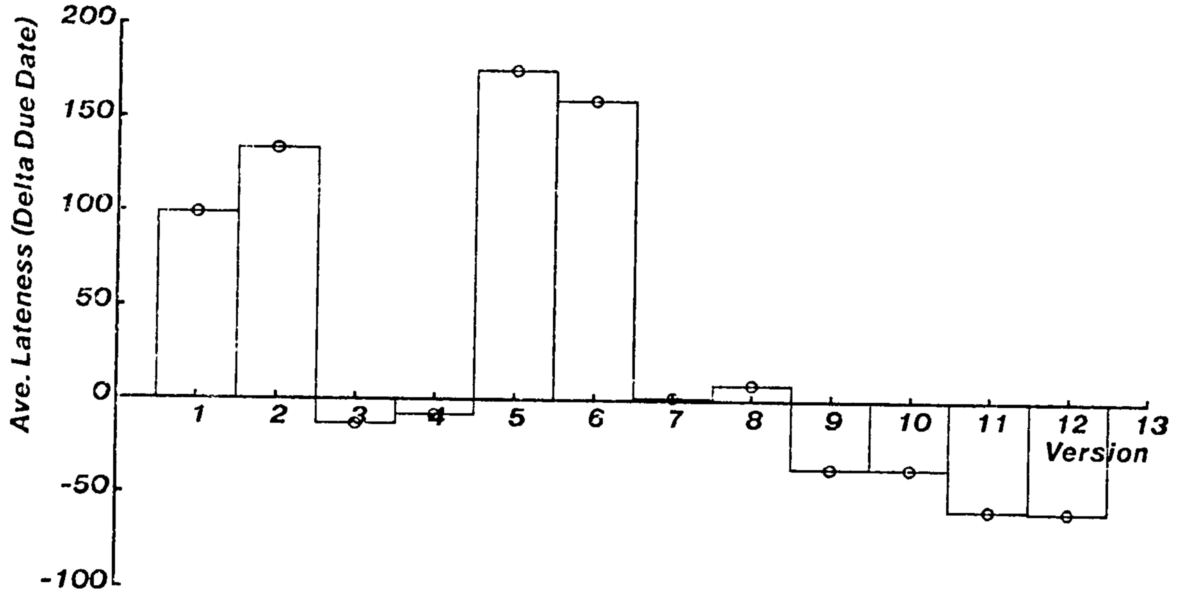


Table C-7: Average Lateness

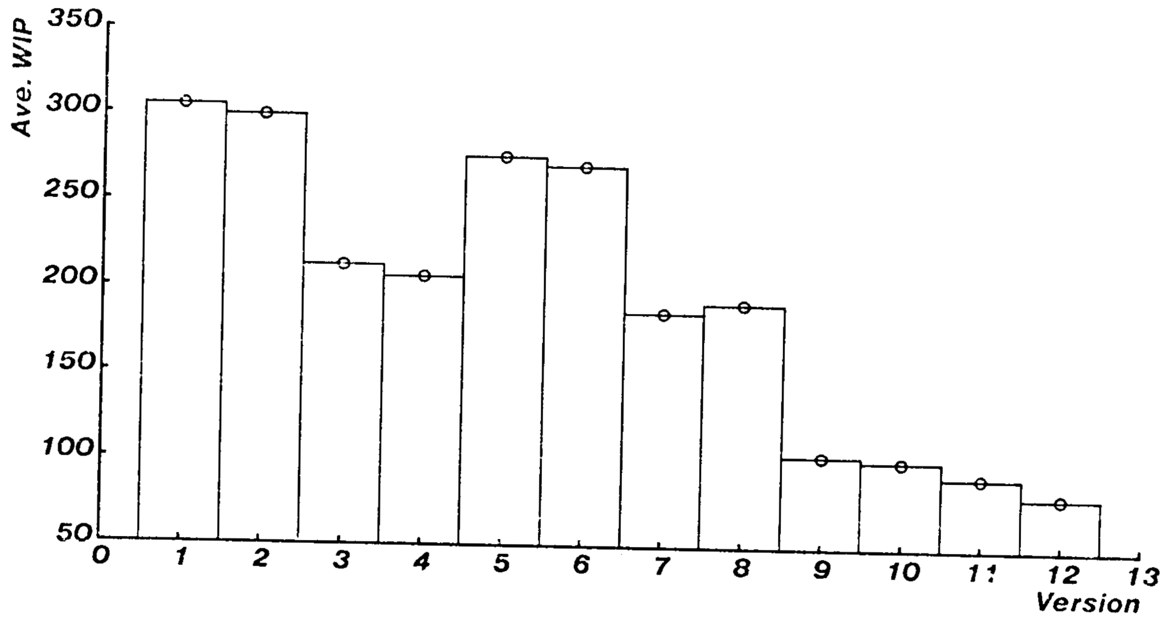


Table C-8: Average WIP

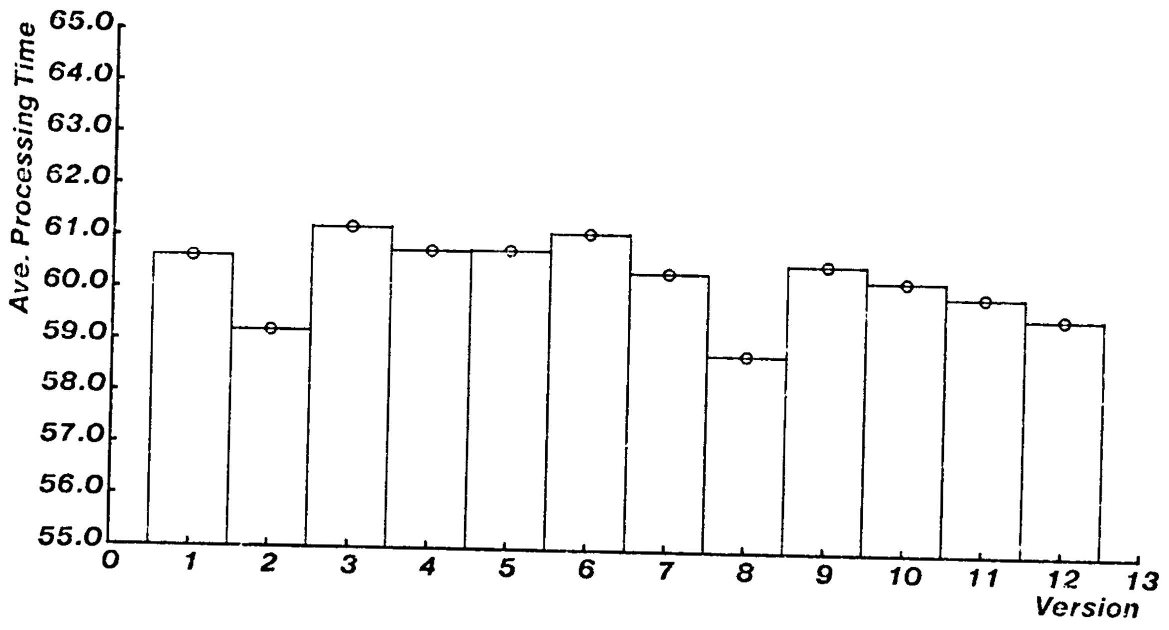


Table C-9: Average Processing Time

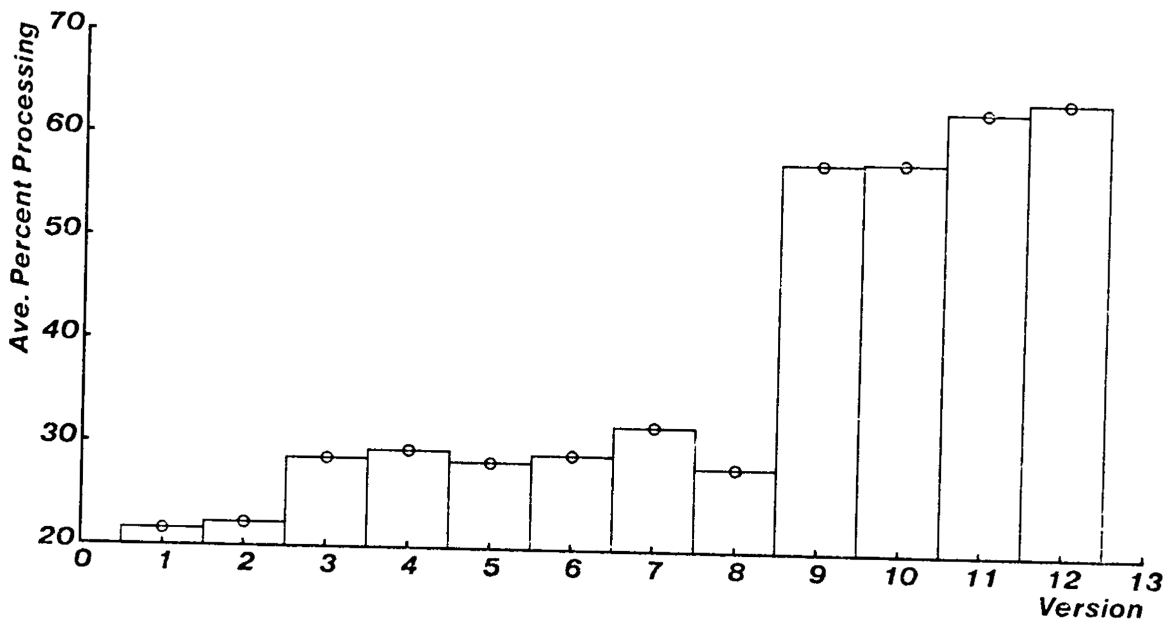


Table C-10: Average Percent Processing Time

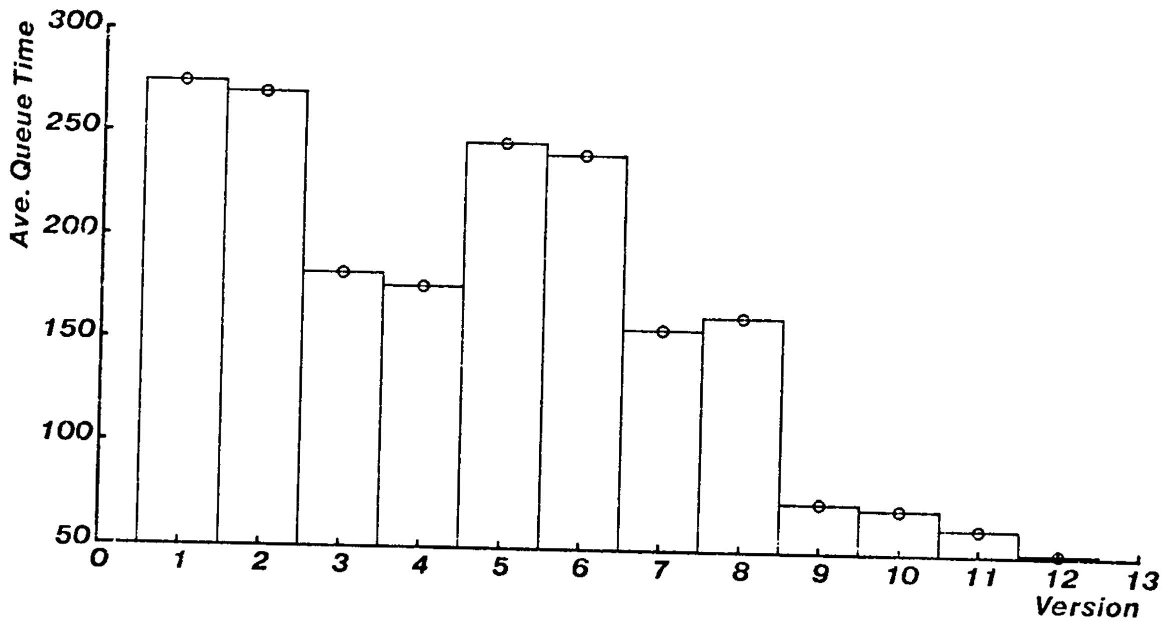


Table C-11: Average Queue Time

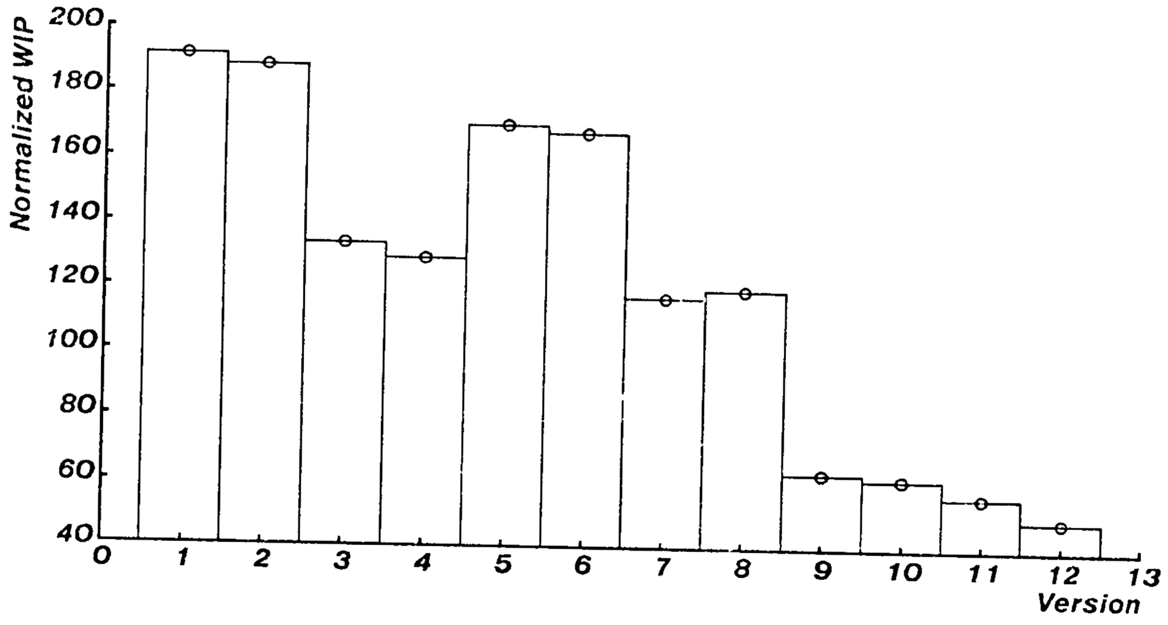


Table C-12: Normalized WIP

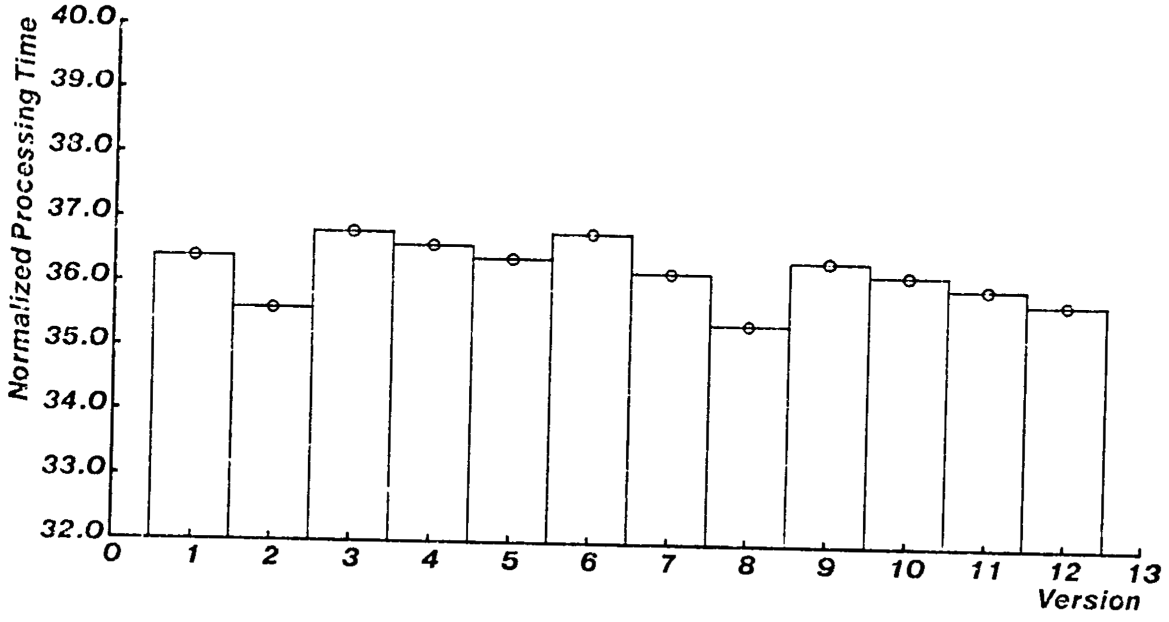


Table C-13: Normalized Processing Time

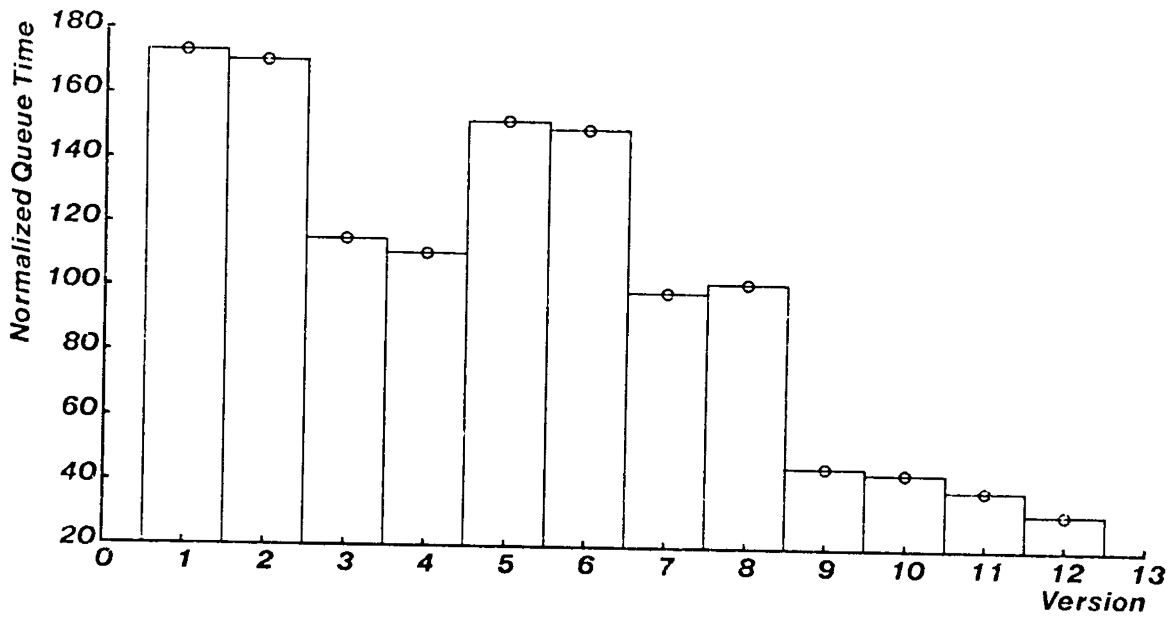


Table C-14: Normalized Queue Time

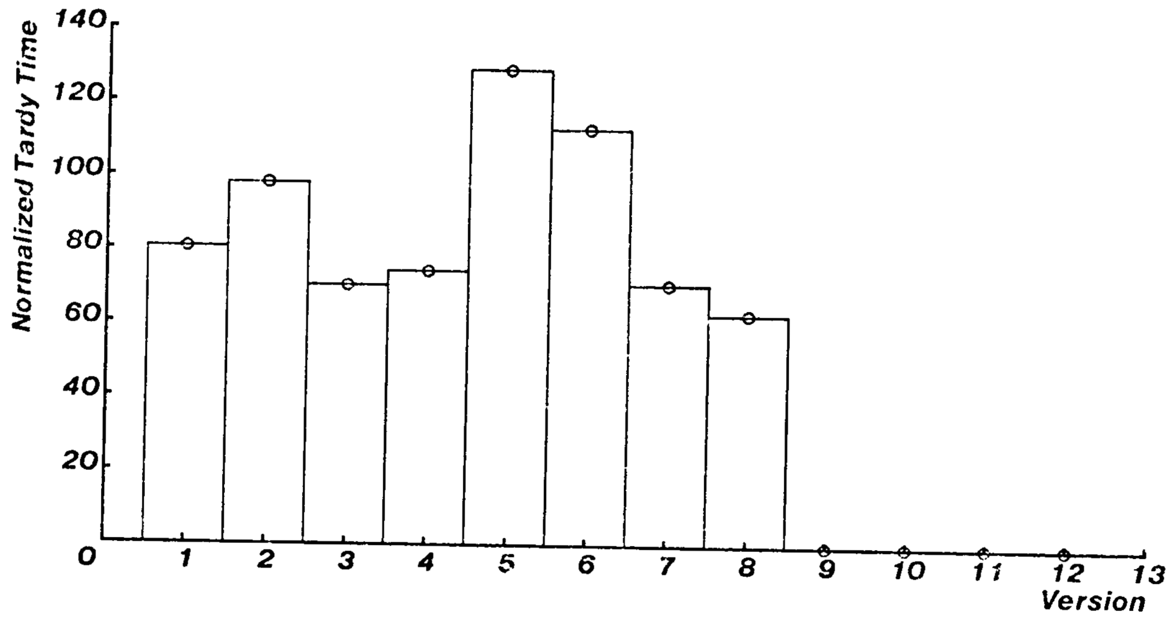


Table C-15: Normalized Tardy Time

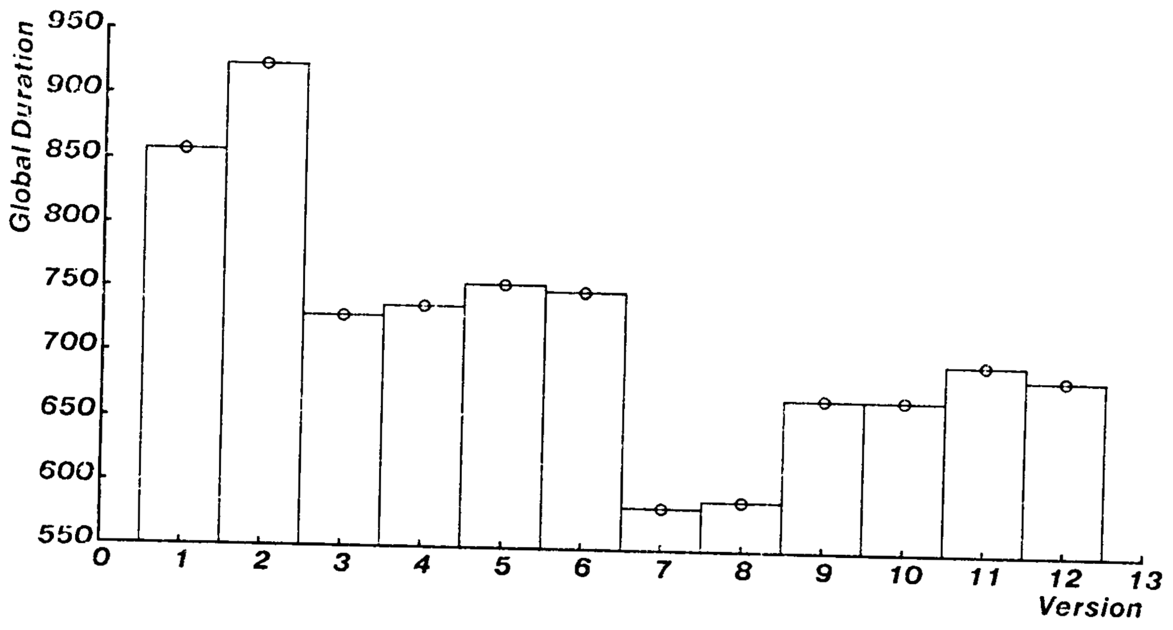


Table C-16: Makespan

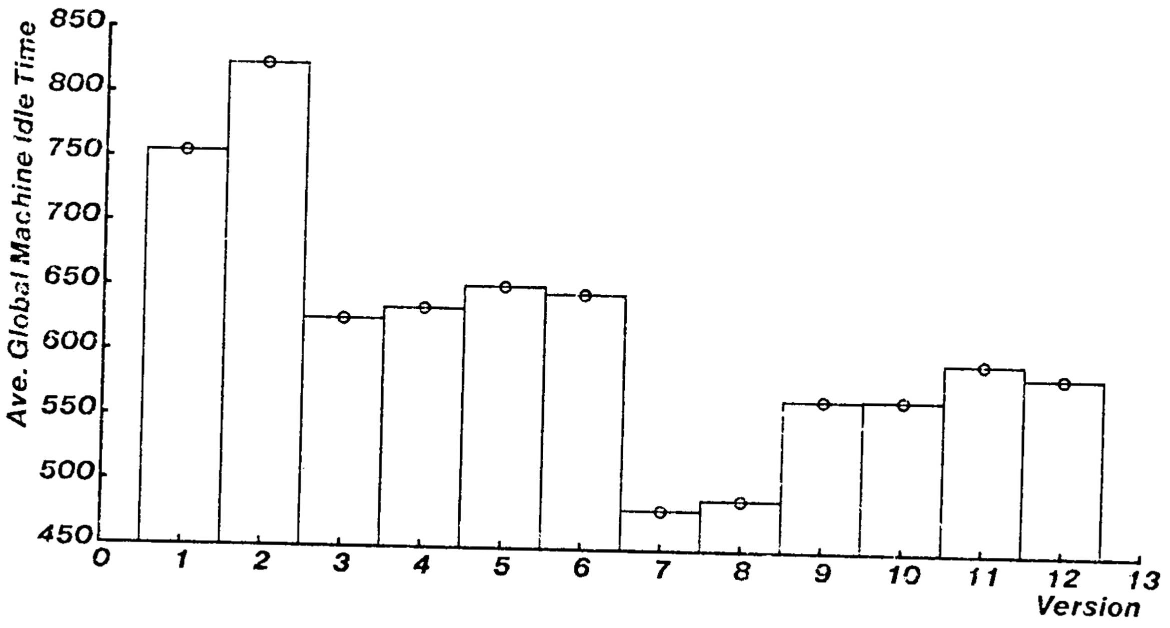


Table C-17: Average Machine idle Time

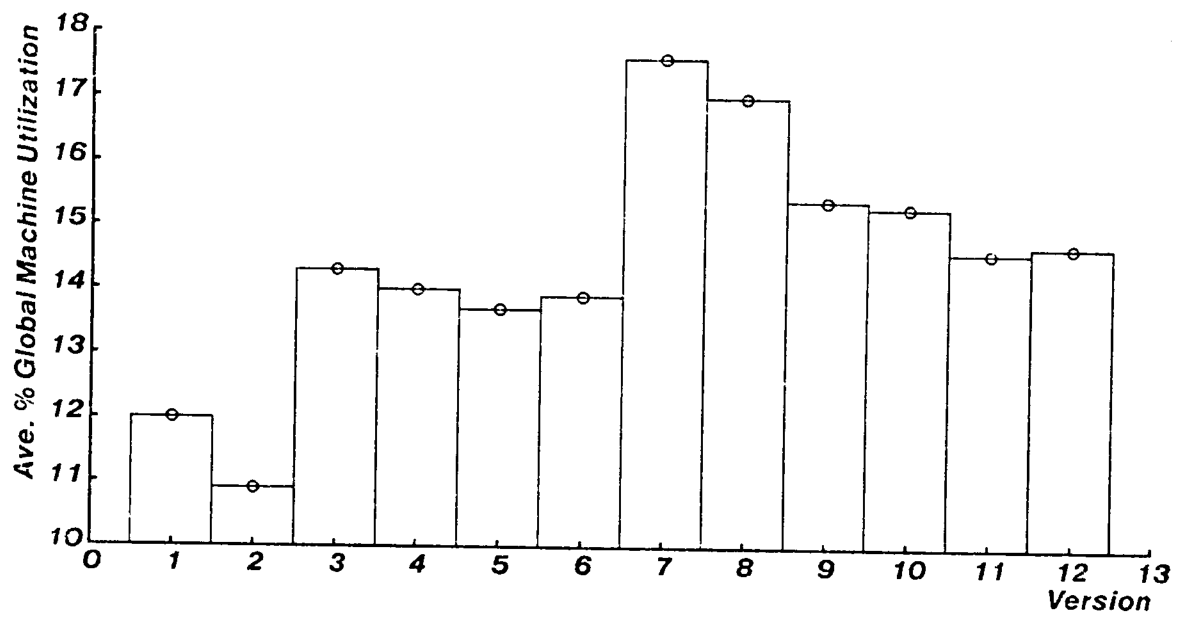


Table C-18: Average % Machine Utilization

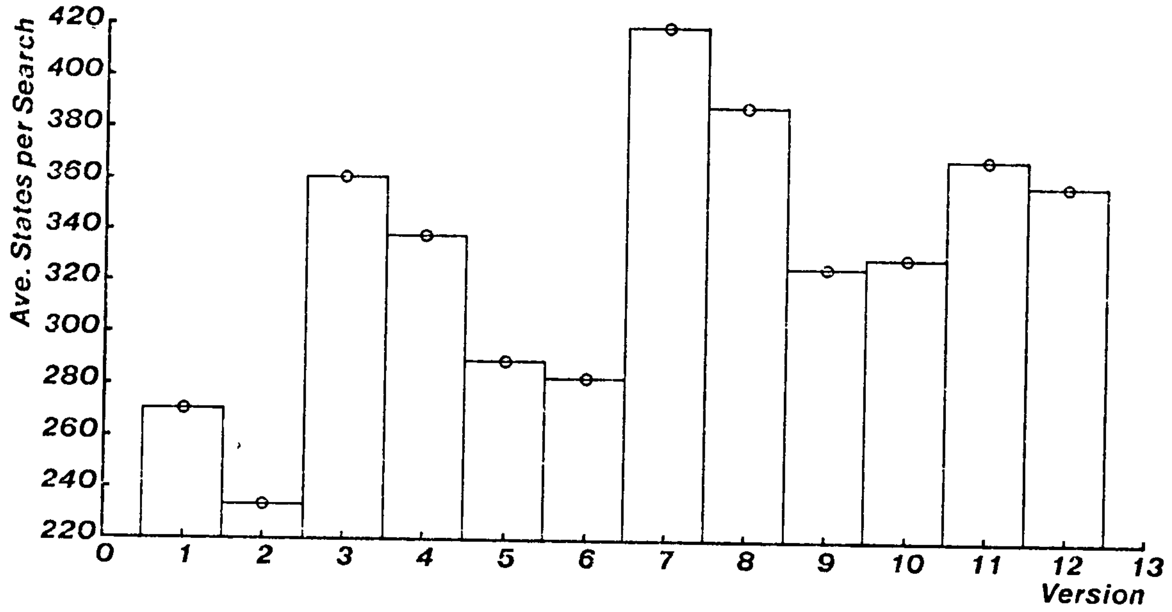


Table C-19: Average Number States per Search

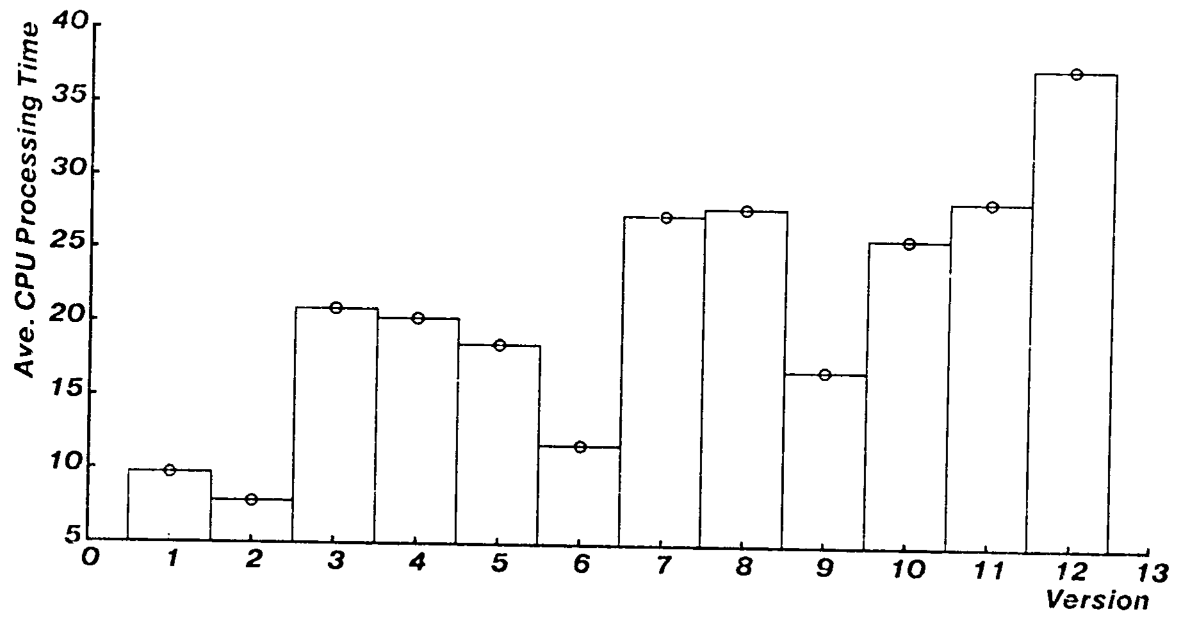


Table C-20: Average CPU Processing Time (sec)

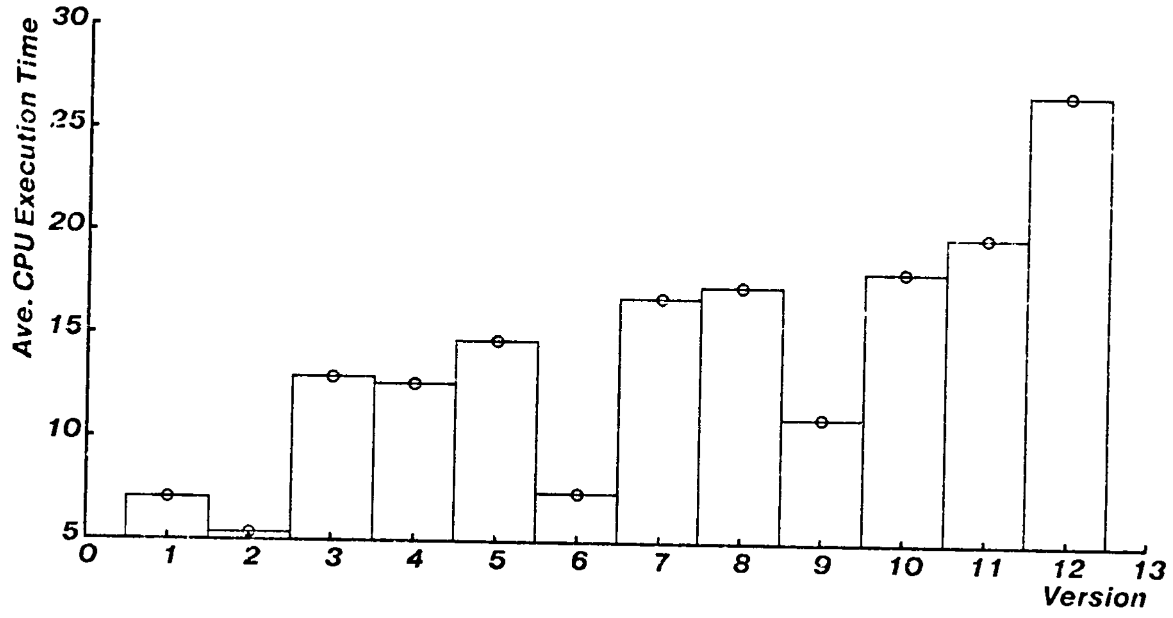


Table C-21: Average CPU Execution Time

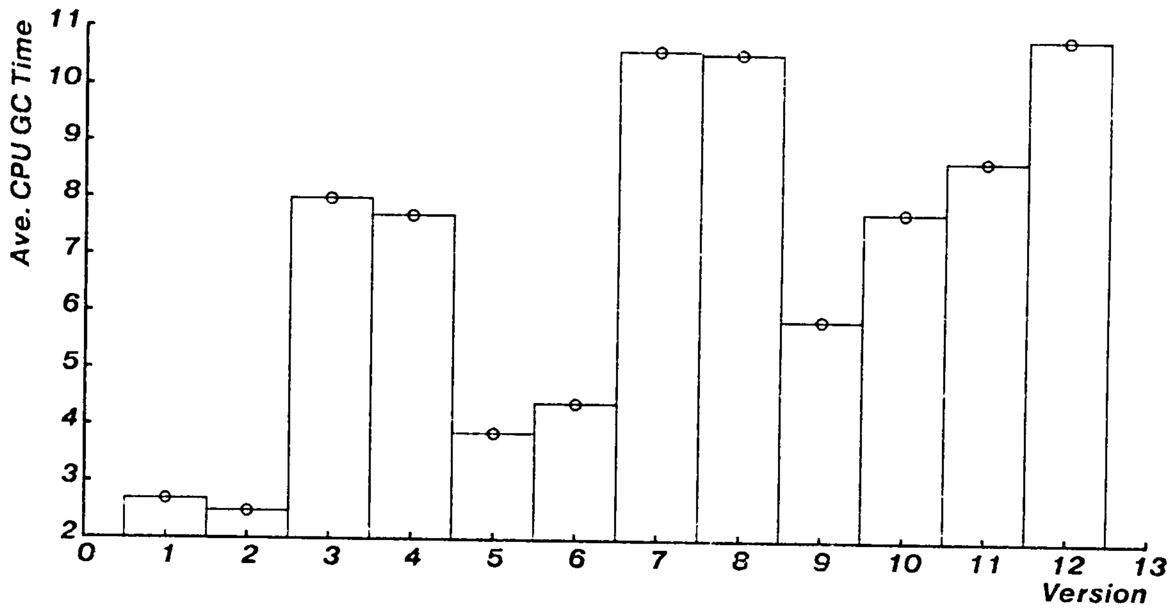


Table C-22: Average CPU GC Time

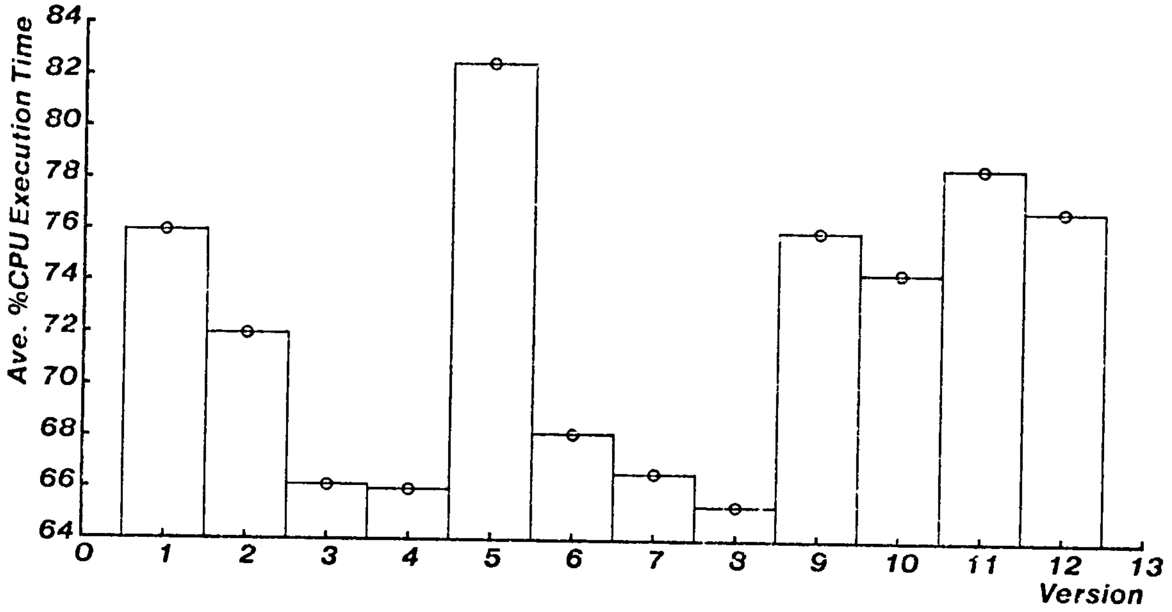


Table C-23: Average %CPU Execution Time

Appendix D
Experiments 1 and 8 Summary Data

Ten versions each of experiments 1 and 8 were generated by running each of the experiments of the same 10 sets of test data. The test data was generated using the same parameters specified in section 6.1. The following tables and graphs summarize the results.

Run		0	1	2	3
Number of Tardy Lots	V1	65	62	55	72
	V2	14	23	26	23
Ave. Tardiness (only tardy lots)	V1	130.53	219.14	234.50	162.60
	V2	117.36	174.36	194.07	128.56
Ave. Tardiness (all lots)	V1	99.82	159.84	151.73	137.74
	V2	19.33	47.18	59.36	34.79
Ave. Delta Start Date	V1	127.56	219.87	189.42	199.64
	V2	148.27	191.38	194.20	171.81
Ave. Lateness (Delta Due Date)	V1	99.25	158.13	148.67	137.55
	V2	1.55	39.29	51.09	28.72
Ave. WIP	V1	305.15	265.20	282.57	251.76
	V2	186.73	174.85	180.20	170.75
Ave. Processing Time	V1	30.31	29.76	29.81	30.04
	V2	30.19	29.91	30.18	30.60
Ave. Percent Processing	V1	21.62	35.04	32.12	30.99
	V2	32.03	37.61	36.45	36.76
Ave. Queue Time	V1	274.84	235.44	252.76	221.72
	V2	156.54	144.93	150.02	140.15
Normalized WIP	V1	191.5	166.0	177.1	149.6
	V2	117.4	108.0	111.5	102.4
Normalized Proc. Time	V1	18.2	18.1	18.0	17.9
	V2	18.2	18.2	18.2	18.2
Normalized Queue Time	V1	173.3	147.9	159.1	131.7
	V2	99.2	89.7	93.3	84.2
Normalized Tardy Time	V1	80.6	135.1	144.6	96.1
	V2	70.6	106.1	118.1	72.4
Makespan	V1	857.41	932.13	927.13	877.08
	V2	583.25	607.96	657.79	579.33
Ave. Machine Idle Time	V1	754.4	831.0	825.8	775.0
	V2	480.6	506.2	555.2	475.3
Ave. % Machine Utilization	V1	12.0	10.9	10.9	11.6
	V2	17.6	16.7	15.6	18.0
Ave. Number States per Search	V1	270.40	306.91	302.19	305.36
	V2	419.59	403.99	386.27	412.64
Ave. CPU Processing Time (sec)	V1	9.73	11.48	9.22	9.02
	V2	27.41	24.54	25.58	24.36
Ave. CPU Execution Time	V1	7.04	9.25	6.21	6.11
	V2	16.83	14.49	16.30	14.07
Ave. CPU GC Time	V1	2.69	2.23	3.01	2.92
	V2	10.59	10.05	9.28	10.29
Ave. %CPU Execution Time	V1	75.96	83.68	72.75	71.86
	V2	66.57	63.92	66.18	61.49

Table D-1: Runs 0, 1, 2, 3; Statistics Comparison

Run		4	5	6	7
Number of Tardy Lots	V1	65	61	62	64
	V2	21	17	17	15
Ave. Tardiness (only tardy lots)	V1	193.62	154.38	213.84	183.28
	V2	129.47	158.18	151.50	138.13
Ave. Tardiness (all lots)	V1	148.06	110.79	155.98	138.00
	V2	31.99	31.64	30.30	24.38
Ave. Delta Start Date	V1	176.58	209.87	199.12	186.07
	V2	165.27	187.41	157.61	140.66
Ave. Lateness (Delta Due Date)	V1	147.07	110.0	154.38	136.41
	V2	25.84	26.06	24.51	9.35
Ave. WIP	V1	283.26	208.79	247.12	268.55
	V2	173.35	147.31	158.76	186.90
Ave. Processing Time	V1	29.54	29.57	29.67	30.86
	V2	29.83	29.34	29.32	30.76
Ave. Percent Processing	V1	31.47	34.54	34.87	31.16
	V2	35.53	38.97	38.41	37.28
Ave. Queue Time	V1	253.72	179.22	217.45	237.68
	V2	143.52	117.97	129.44	156.14
Normalized WIP	V1	176.5	133.1	152.2	163.5
	V2	108.7	96.3	98.4	114.9
Normalized Proc. Time	V1	17.8	18.9	18.2	18.7
	V2	16.0	18.7	18.0	18.6
Normalized Queue Time	V1	158.7	114.2	134.0	144.8
	V2	90.7	77.6	80.4	96.3
Normalized Tardy Time	V1	118.8	99.8	131.3	112.9
	V2	77.9	100.4	93.1	87.1
Makespan	V1	883.41	703.16	864.54	821.00
	V2	682.46	527.63	565.00	648.38
Ave. Machine Idle Time	V1	783.0	602.6	763.7	716.1
	V2	481.1	427.9	465.3	543.8
Ave. % Machine Utilization	V1	11.4	14.3	11.7	12.8
	V2	17.4	18.9	17.6	16.1
Ave. Number States per Search	V1	310.28	317.21	343.18	339.52
	V2	416.28	418.02	449.33	438.39
Ave. CPU Processing Time (sec)	V1	11.57	11.93	12.63	12.18
	V2	30.15	25.47	39.21	26.57
Ave. CPU Execution Time	V1	9.20	9.52	9.96	9.68
	V2	21.97	16.31	27.19	16.80
Ave. CPU GC Time	V1	2.37	2.40	2.67	2.50
	V2	8.18	9.16	12.02	9.77
Ave. %CPU Execution Time	V1	83.51	82.95	83.05	83.76
	V2	75.85	70.71	74.09	71.19

Table D-2: Runs 4, 5, 6, 7; Statistics Comparison

Run		8	9
Number of Tardy Lots	V1	68	69
	V2	14	14
Ave. Tardiness (only tardy lots)	V1	175.23	175.28
	V2	131.43	168.20
Ave. Tardiness (all lots)	V1	140.18	142.28
	V2	21.65	27.70
Ave. Delta Start Date	V1	183.12	204.31
	V2	166.99	179.08
Ave. Lateness (Delta Due Date)	V1	139.51	140.66
	V2	13.77	16.39
Ave. WIP	V1	289.59	272.76
	V2	179.98	173.72
Ave. Processing Time	V1	30.86	29.26
	V2	30.83	29.64
Ave. Percent Processing	V1	27.14	32.26
	V2	34.62	37.50
Ave. Queue Time	V1	258.72	243.50
	V2	149.15	144.08
Normalized WIP	V1	177.1	166.4
	V2	109.4	105.5
Normalized Proc. Time	V1	18.5	17.8
	V2	18.5	18.0
Normalized Queue Time	V1	158.5	148.6
	V2	90.9	87.4
Normalized Tardy Time	V1	108.5	106.1
	V2	83.3	102.9
Makespan	V1	803.04	836.79
	V2	553.92	602.42
Ave. Machine Idle Time	V1	698.1	737.3
	V2	449.1	501.6
Ave. % Machine Utilization	V1	13.1	11.9
	V2	18.9	16.7
Ave. Number States per Search	V1	287.04	309.33
	V2	419.29	435.58
Ave. CPU Processing Time (sec)	V1	11.07	11.79
	V2	25.98	27.79
Ave. CPU Execution Time	V1	8.96	9.53
	V2	17.40	18.66
Ave. CPU GC Time	V1	2.11	2.26
	V2	8.57	9.14
Ave. %CPU Execution Time	V1	83.90	83.79
	V2	72.27	71.96

Table D-3: Runs 8, 9; Statistics Comparison

		Min	Max	Mean	Std.	Var.
Number of Tardy Lots	V1	55.0	72.0	64.3	4.51	20.4
	V2	14.0	26.0	18.4	4.27	18.04
Ave. Tardiness (only tardy lots)	V1	130.53	234.5	184.24	30.19	911.52
	V2	117.36	194.07	149.126	23.15	536.06
Ave. Tardiness (all lots)	V1	99.82	159.84	138.442	18.19	330.92
	V2	19.33	59.36	32.832	11.51	132.7
Ave. Delta Start Date	V1	127.56	219.87	189.55	24.0	580.79
	V2	140.66	194.2	170.27	17.17	294.88
Ave. Lateness (Delta Due Date)	V1	99.25	158.13	137.16	17.7	316.02
	V2	1.55	51.09	23.65	13.7	187.78
Ave. WIP	V1	208.79	305.15	267.4	25.54	652.49
	V2	147.31	186.9	173.25	11.63	135.35
Ave. Processing Time	V1	29.26	30.86	29.96	0.52	0.27
	V2	29.32	30.83	30.06	0.52	0.27
Ave. Percent Processing	V1	21.62	35.04	31.121	3.86	14.96
	V2	32.03	38.97	36.51	1.92	3.71
Ave. Queue Time	V1	179.22	274.84	237.5	25.35	642.8
	V2	117.97	156.54	143.19	11.25	126.63
Normalized WIP	V1	133.1	191.5	165.3	15.96	255.02
	V2	96.3	117.4	107.25	6.41	41.17
Normalized Proc. Time	V1	17.8	18.9	18.21	0.359	0.12
	V2	18.0	18.7	18.26	0.24	0.05
Normalized Queue Time	V1	114.2	173.3	147.08	16.10	259.43
	V2	77.6	99.2	88.97	6.42	41.23
Normalized Tardy Time	V1	80.6	144.6	113.38	18.52	343.15
	V2	70.6	118.1	91.19	14.84	220.46
Makespan	V1	703.16	932.13	850.59	62.92	3958.97
	V2	527.63	657.79	590.81	38.08	1450.69
Ave. Machine Idle Time	V1	602.6	831.0	748.7	63.19	3994.16
	V2	427.9	555.2	488.61	37.47	1404.6
Ave. % Machine Utilization	V1	10.9	14.3	12.06	1.0	1.01
	V2	15.6	18.9	17.35	1.0	1.08
Ave. Number States per Search	V1	270.4	343.18	309.14	20.50	420.48
	V2	386.27	449.33	419.93	17.05	290.77
Ave. CPU Processing Time (sec)	V1	9.02	12.63	11.06	1.21	1.47
	V2	24.36	39.21	27.7	4.16	17.34
Ave. CPU Execution Time	V1	6.11	9.96	8.54	1.41	1.99
	V2	14.07	27.19	18.0	3.70	13.73
Ave. CPU GC Time	V1	2.11	3.01	2.51	0.28	0.08
	V2	8.18	12.02	9.70	1.04	1.10
Ave. %CPU Execution Time	V1	71.86	83.9	80.52	4.69	22.0
	V2	61.49	75.85	69.43	4.40	19.42

Table D-4: Final Statistics Comparison

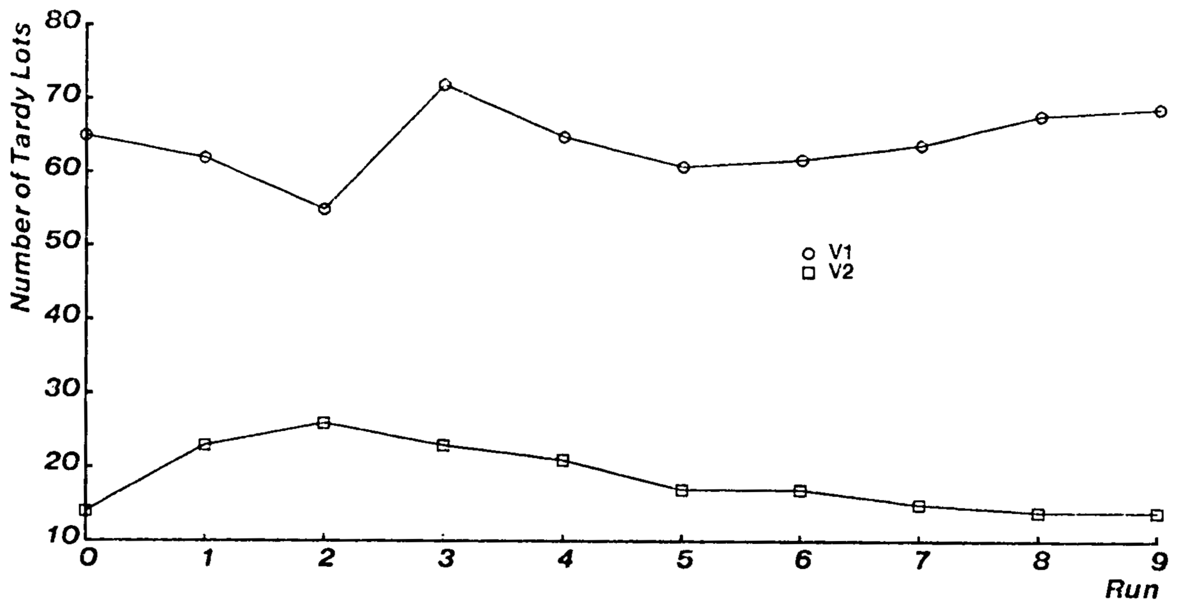


Table D-5: Number of Tardy Lots

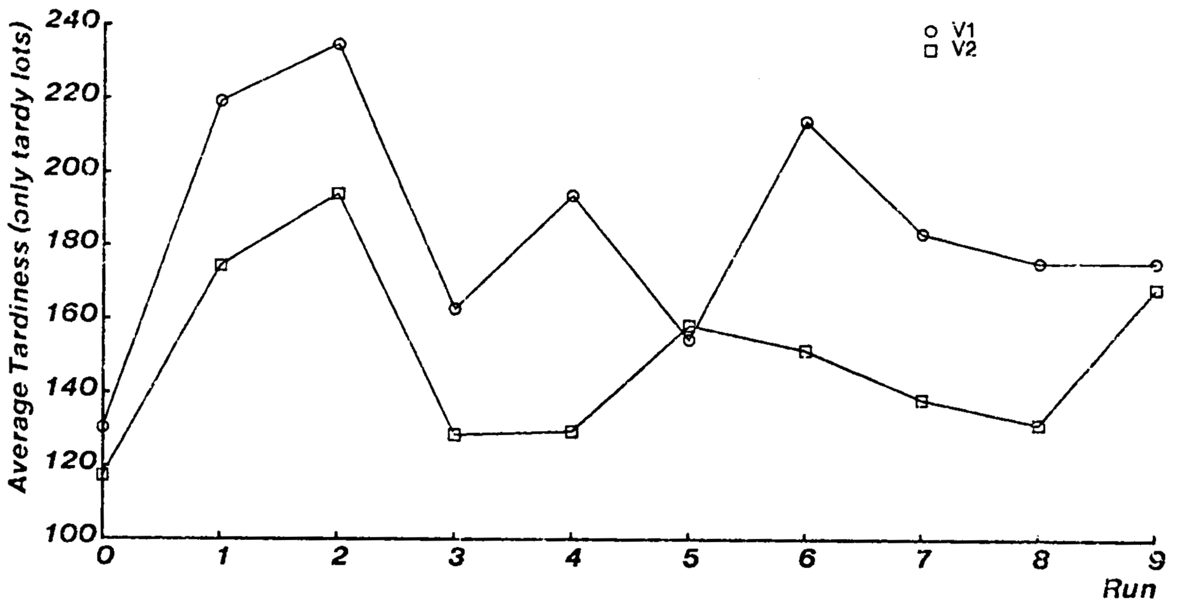


Table D-6: Average Tardiness (only tardy lots)

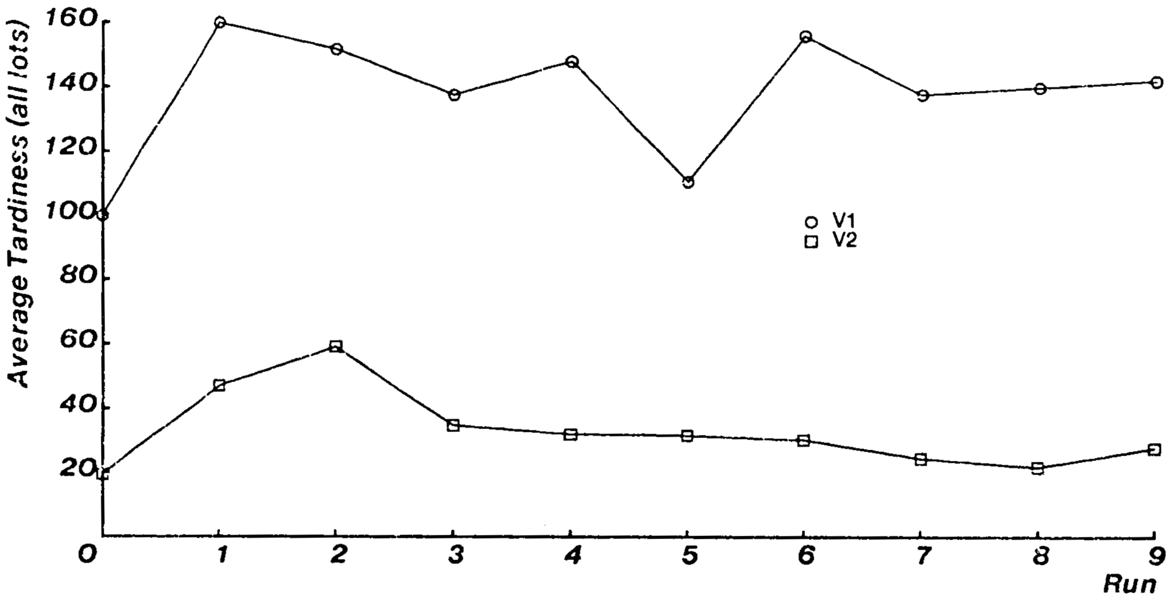


Table D-7: Average Tardiness (all lots)

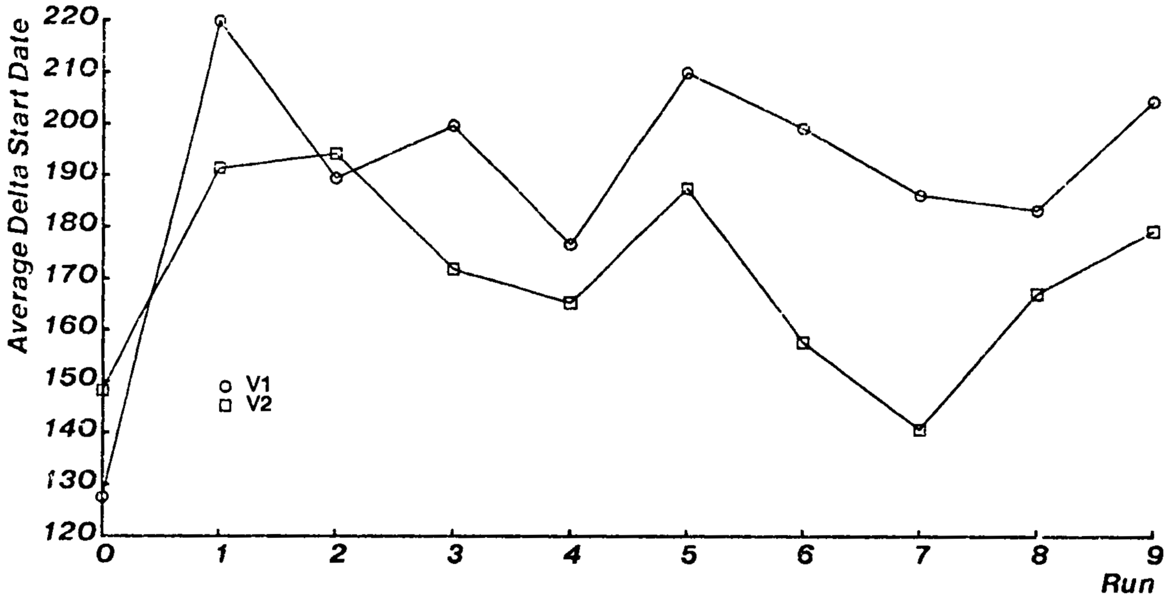


Table D-8: Average Delta Start Date

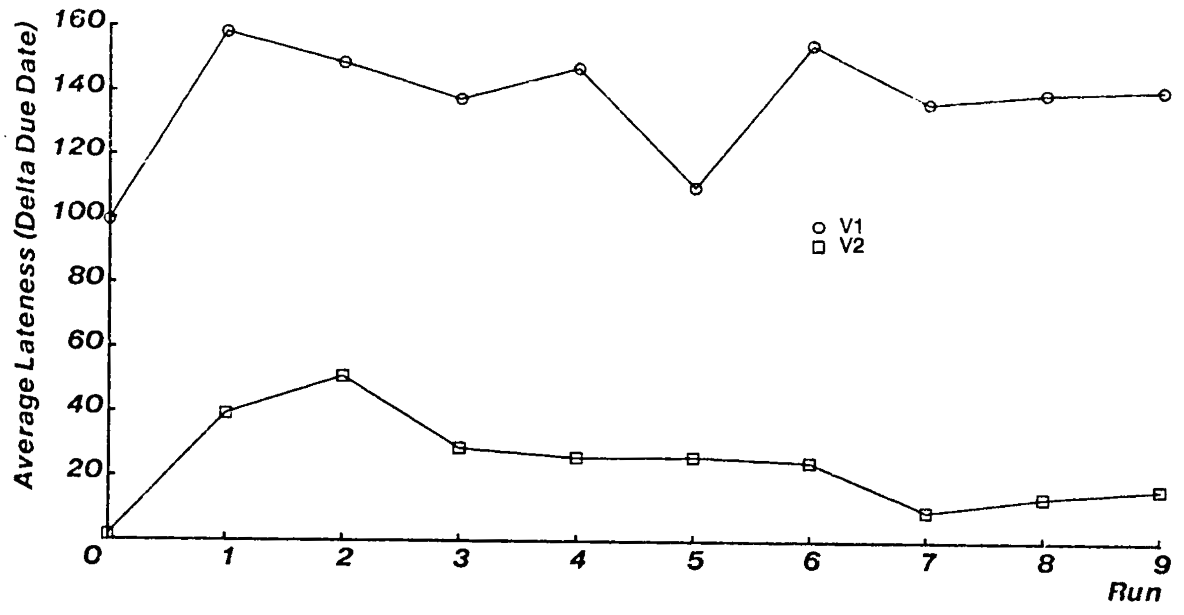


Table D-9: Average Lateness

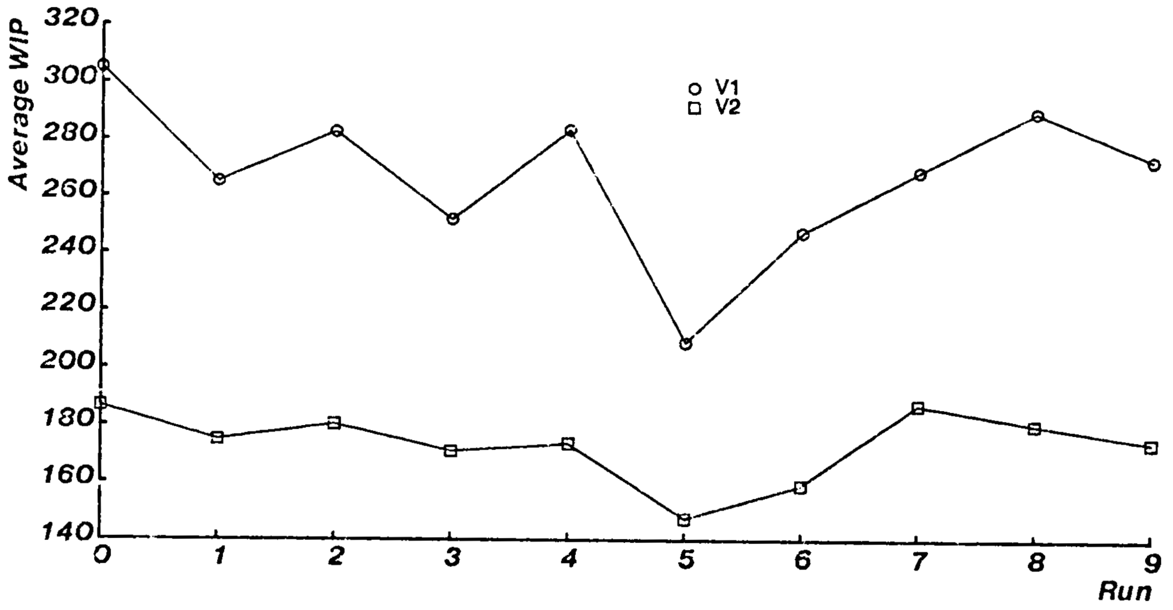


Table D-10: Average WIP

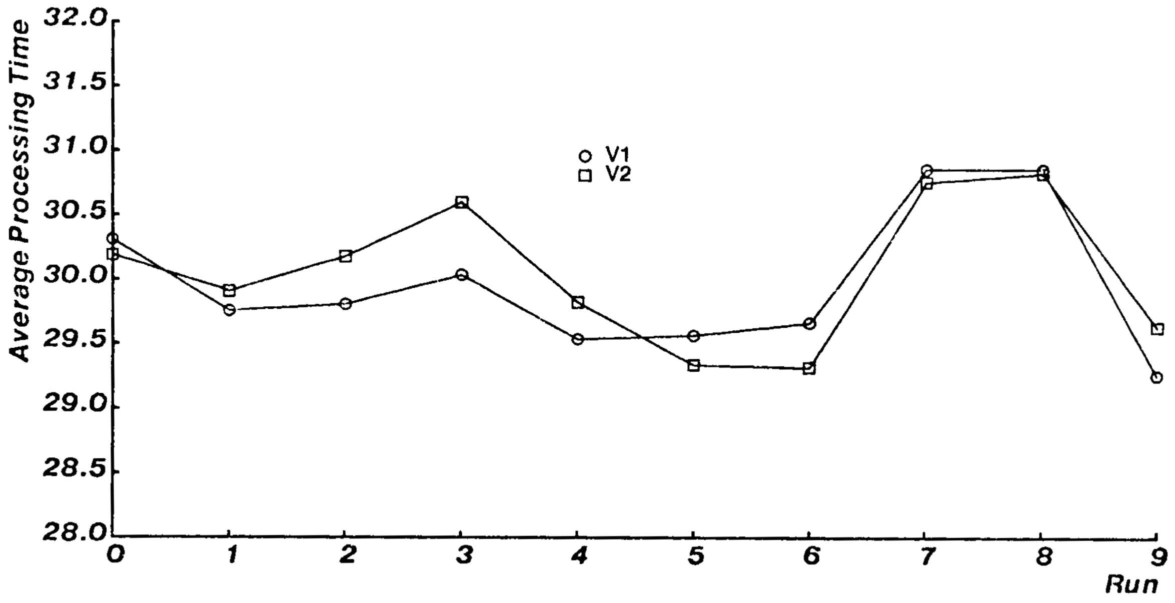


Table D-11: Average Processing Time

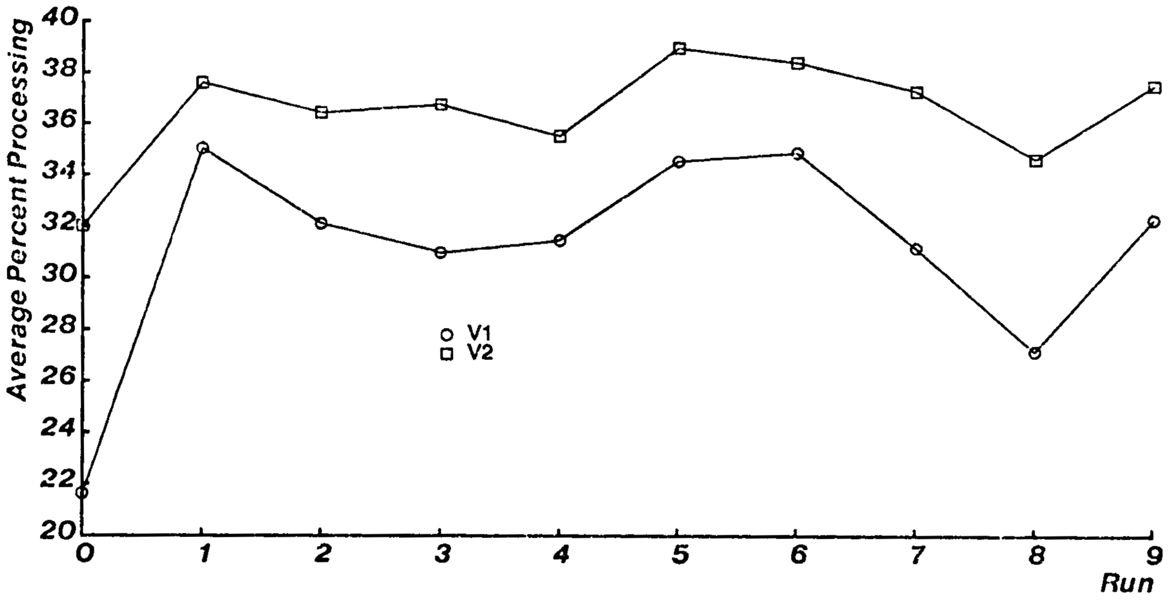


Table D-12: Average Percent Processing Time

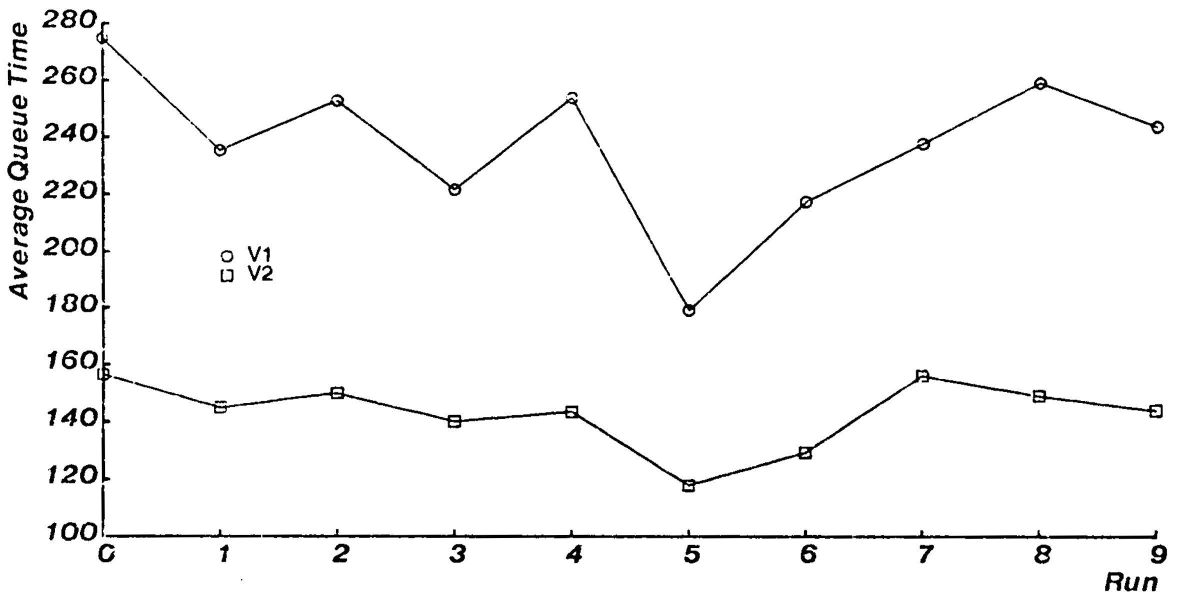


Table D-13: Average Queue Time

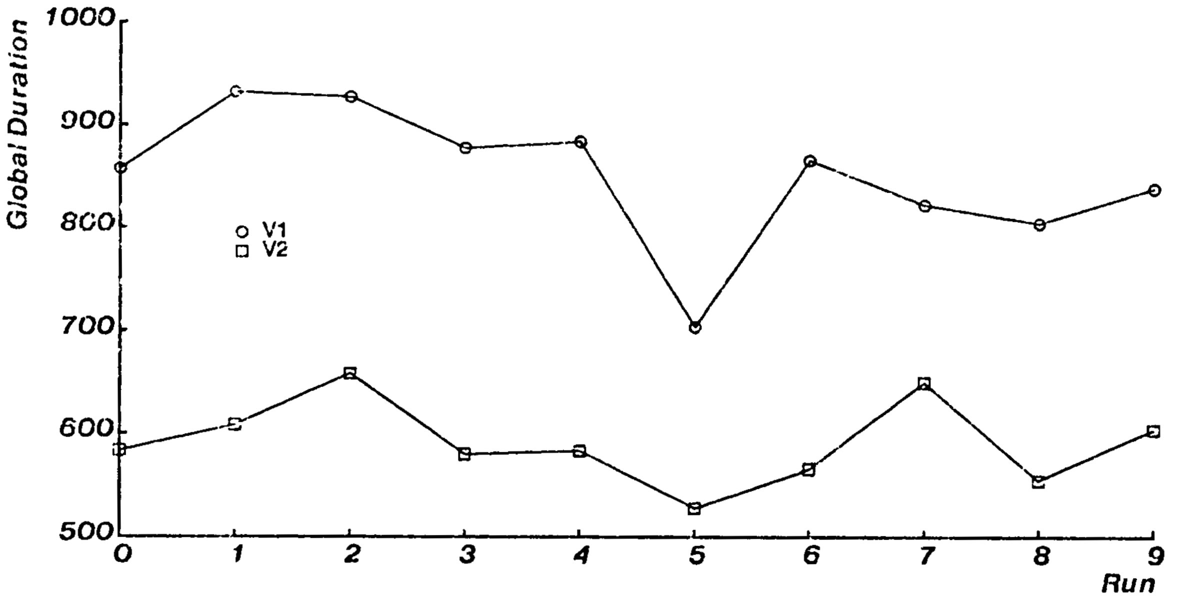


Table D-14: Makespan

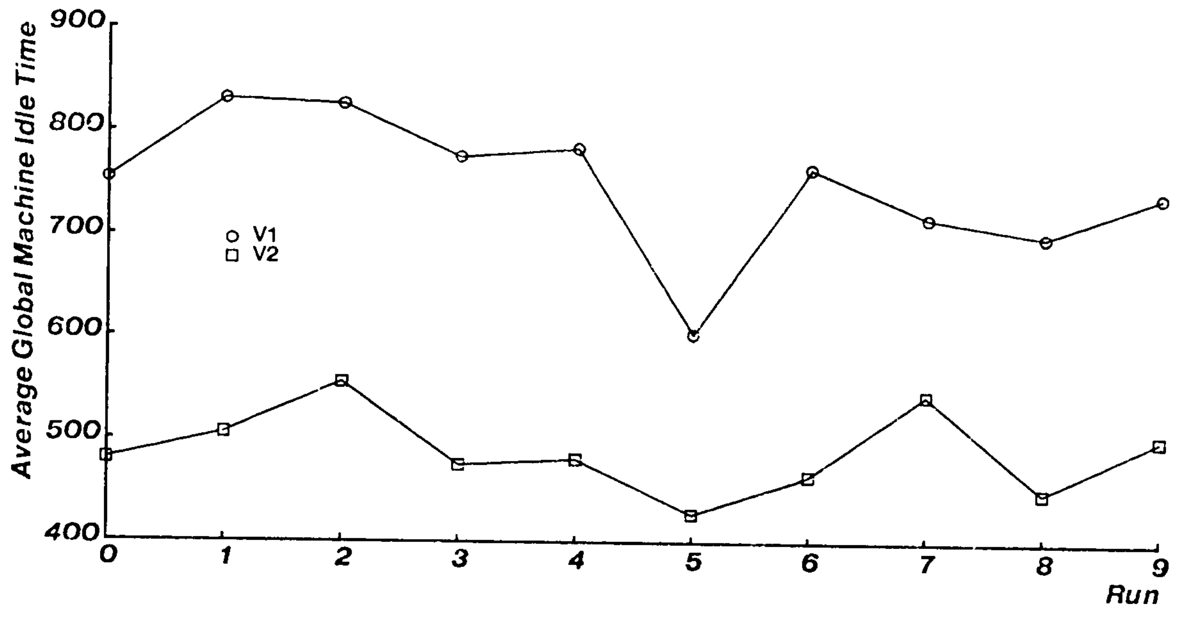


Table D-15: Average Machine idle Time

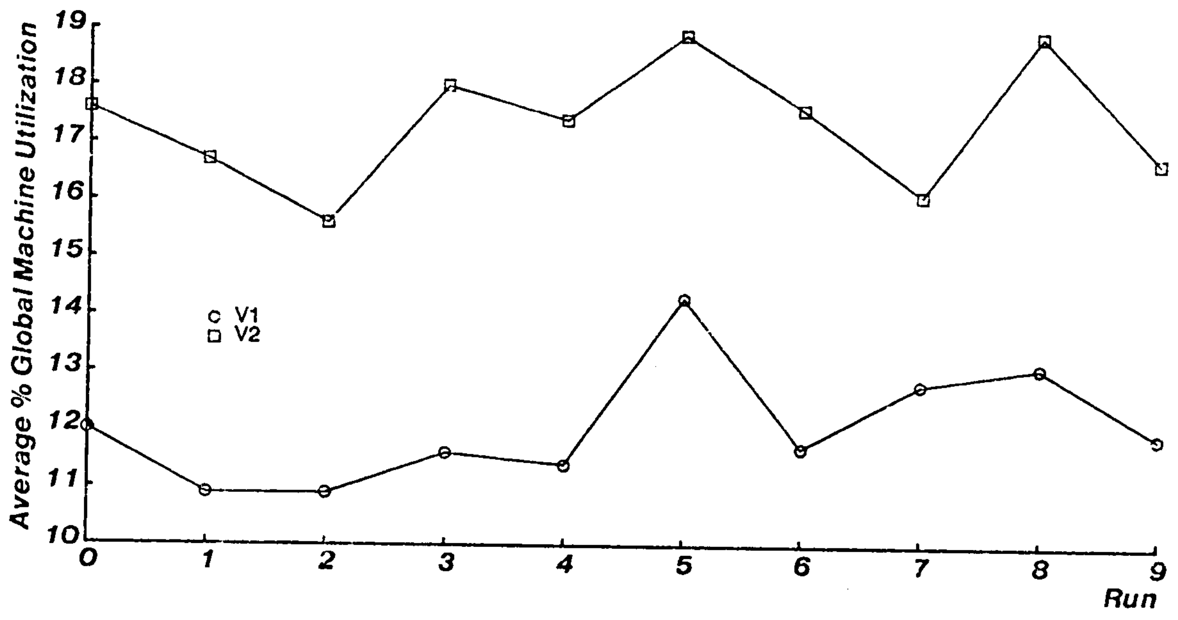


Table D-16: Average % Machine Utilization

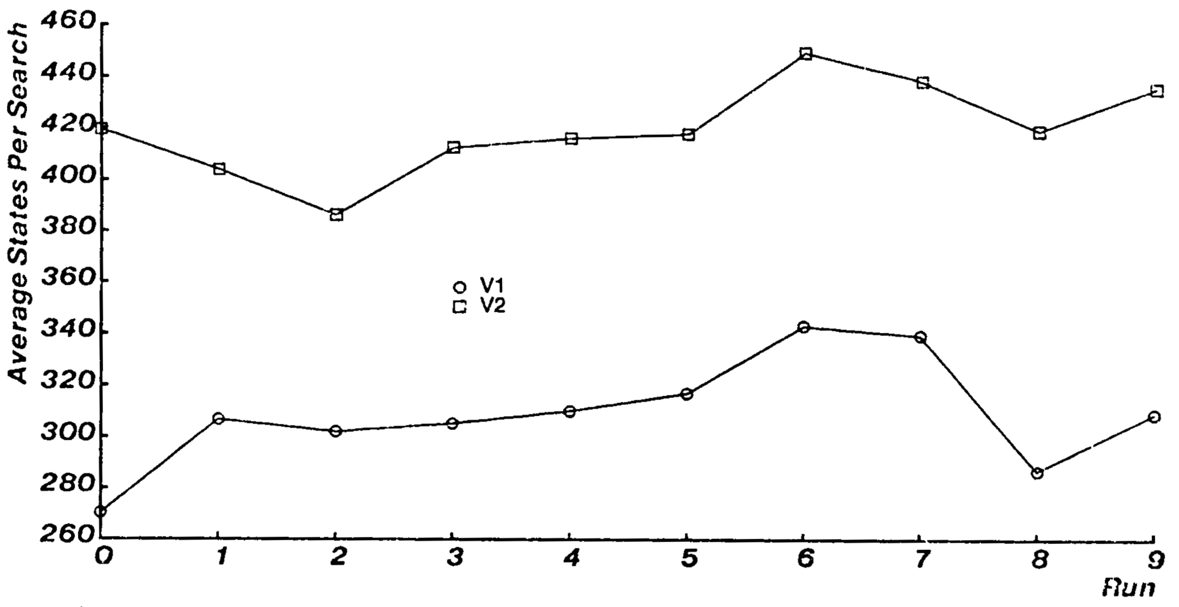


Table D-17: Average Number States per Search

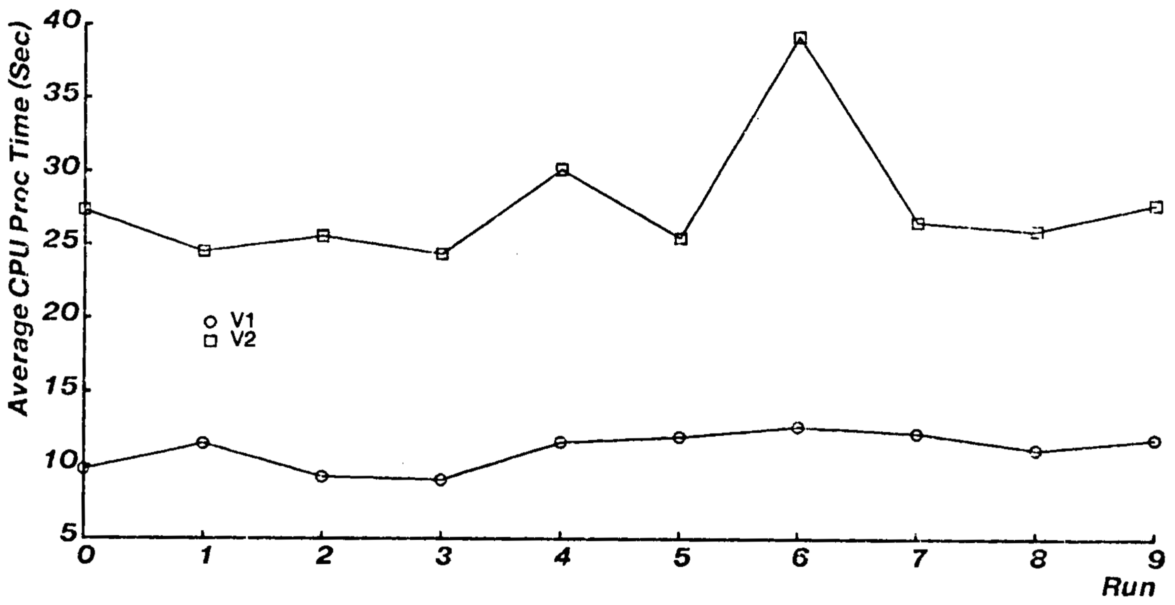


Table D-18: Average CPU Processing Time (sec)

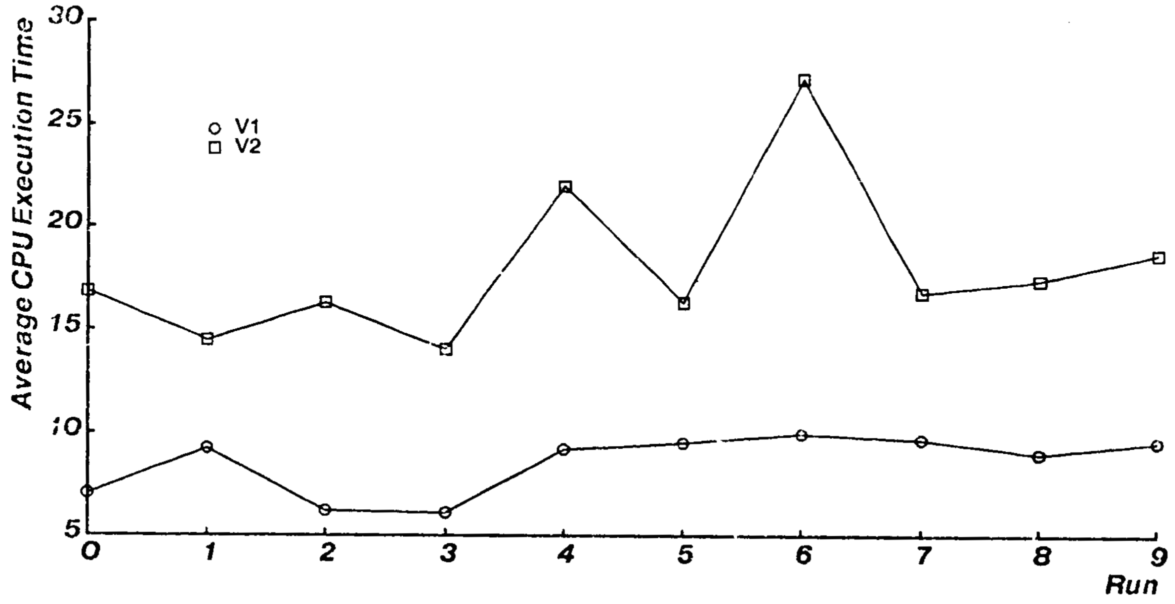


Table D-19: Average CPU Execution Time

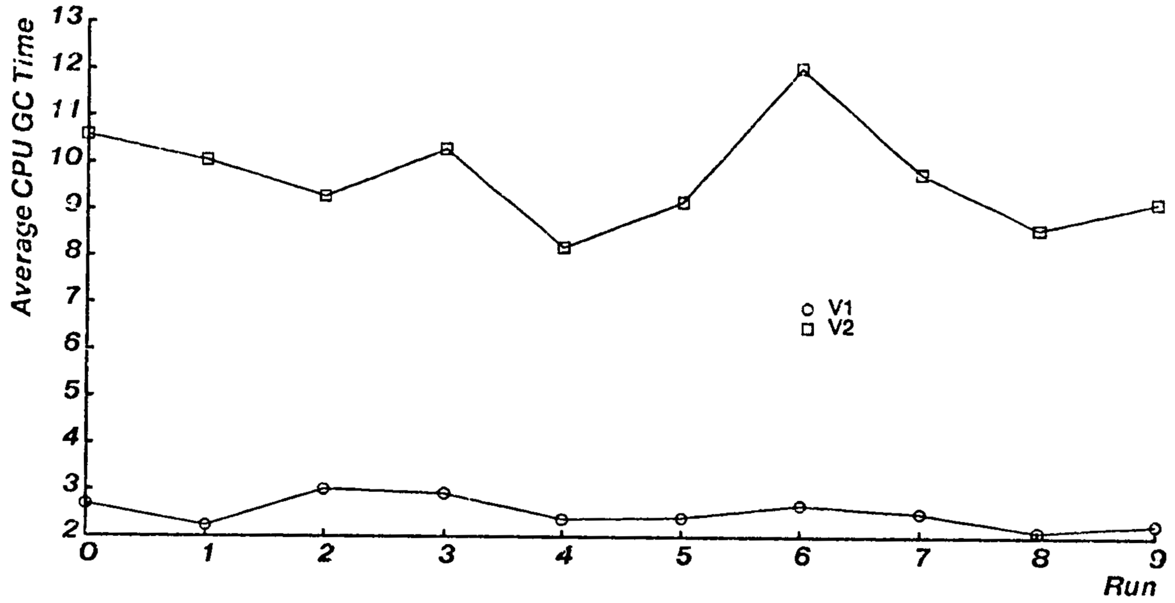


Table D-20: Average CPU GC Time

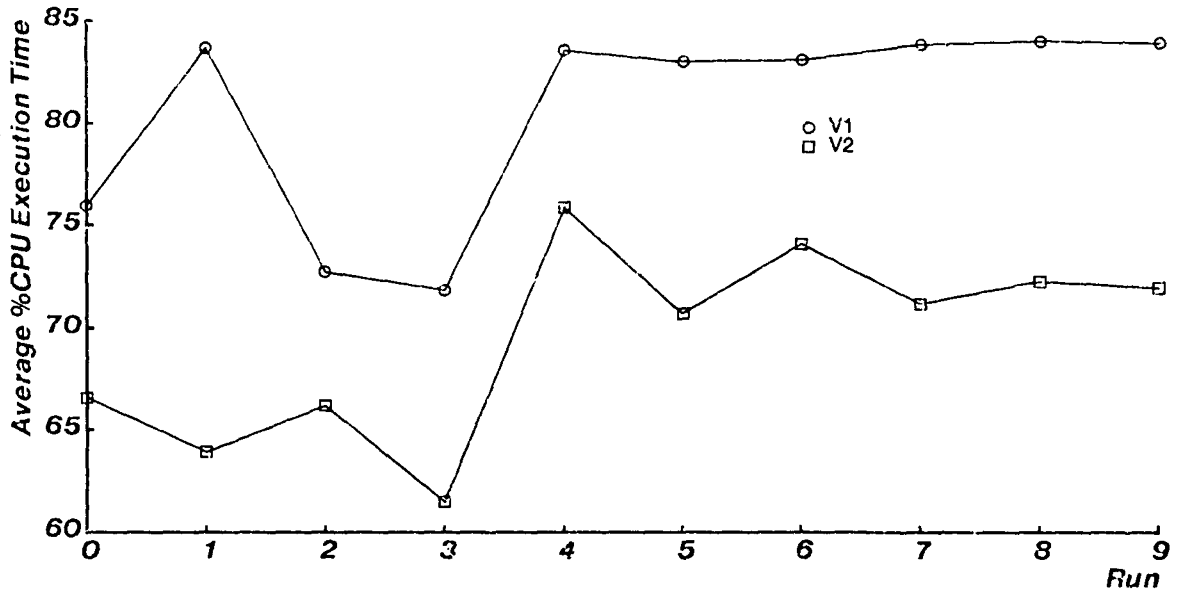


Table D-21: Average %CPU Execution Time