AN ARTIFICIAL INTELLIGENCE APPROACH TO JOB-SHOP SCHEDULING

by

GEORGE L. CLEMMER, II

B.S. Electrical Engineering (1973),
M.S. Naval Architecture & Marine Eng. (1978),
Massachusetts Institute of Technology

Submitted to the Sloan School of Management
in Partial Fulfillment of
the Requirements of the Degree of
Master of Science in Management

at the

Massachusetts Institute of Technology

September 1984

Signature of the author . . . . . .
Sloan School of Management
June 18, 1984

Certified by . . . . . .
Stephen C. Graves
Thesis Supervisor

Accepted by . . . . . .
Jeffrey A. Barks
Director of the Master's Program

AN ARTIFICIAL INTELLIGENCE APPROACH
TO JOB-SHOP SCHEDULING

by

GEORGE L. CLEMMER, II

Submitted to the Sloan School of Management
on June 18, 1984
in partial fulfillment of the requirements
for the Degree of
Master of Science in Management

ABSTRACT

A classic operations research problem, the job-shop Scheduling problem, is examined from the perspective of artificial intelligence (AI). The operations research literature is reviewed, and recast in the terminology of AI. The job-shop problem is recast as a state-space search problem. A problem representation is developed which employs Gantt charts to represent states, and search operators which preserve schedule feasibility to transit from state to state. This state-space problem formulation is investigated at two levels: First, a LISP program is developed which aids a human scheduler in handling the bookkeeping details of constructing schedules. Second, a heuristic search method is developed which directs the search for good schedules. The performance of the program under human control, and under the heuristic search method are contrasted with results from the operations research literature for 3 benchmark problems. Using the program, the author was able to solve the most complex problem cited in the literature in 2 hours. Using a very simple heuristic, the heuristic search method generates reasonably good schedules. It appears that there is substantial room for improvement in the power of the heuristic.

Thesis Supervisor: Stephen C. Graves

Title: Associate Professor

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank Professor Tom Malone for his early encouragement of my efforts with this topic, and for making computational resources available.

I would like to thank Professor Stephen C. Graves for his open-mindedness in taking on this thesis topic, and for his helpful criticism of the thesis document.

CHAPTER 1 - INTRODUCTION

A SAMPLE PROBLEM

A simplistic job-shop scheduling problem is shown in Figure 1. There are 2 jobs, each consisting of 3 tasks, which pass across three machines in a given sequence. The addition of a scheduling criterion, such as the objective of constructing a schedule of minimum length (makespan), completes the problem statement. (The criterion of minimum length is assumed throughout this paper.)

Using the terminology of Graves (1981), this problem is a deterministic, static job-shop scheduling problem. By deterministic we mean that the processing time, and sequence of each task are known exactly. By static, we mean that all jobs to be scheduled are known at the start of the time period in question, and that no jobs will arrive (or be deleted) from the schedule once it is formulated.

GANTT CHART PROBLEM REPRESENTATION

The sample problem can also be represented by a job Gantt chart, as shown in Figure 2(A). Each column in the chart represents one time unit or slot. A row is printed for each job. Tasks are differentiated within the row by printing a machine identifier (usually a letter) in column(s) representing the number of units of processing time required.

The job Gantt chart in Figure 2(A) is a description of job process routings, but does not represent a feasible schedule (e.g. both jobs compete for the first time slot on machine A). This is indicated by the notation "NOT LOADED" on each job.

Also associated with the problem is a machine Gantt chart showing a row for each machine. This chart is initially empty and is the space within which a schedule for the problem will be constructed.

A feasible (but not necessarily optimum) schedule for the problem can be constructed by sequentially "loading" jobs into the machine Gantt chart. A jobs is "loaded" by reserving spaces in the machine Gantt chart as early as possible, subject to job task ordering (as specified by the job Gant chart) and machine availability (as indicated by the machine Gantt chart).

Figure 1: A Sample Job Shop Scheduling Problem
Described in Tabular Format.

JOB	TASK	MACHINE	PROCESSING TIME
J-1	J-1-1	A	2
	J-1-2	B	1
	J-1-3	C	1
J-2	J-2-1	A	1
	J-2-2	C	1
	J-2-3	B	1

Figure 2: Gantt Chart Representations of the Sample Problem

(A) Problem Statment:

```
Job Gantt Chart:                    Machine Gantt Chart:
      :----+----1                         :----+----1
   J-1:AABC      :NOT LOADED           A:
   J-2:ACB       :NOT LOADED           B:
      :----+----1                      C:
       time-->                          :----+----1
```

(B) Partial schedule generated by loading J-1:

```
Job Gantt Chart:                    Machine Gantt Chart:
      :----+----1                         :----+----1
   J-1:AABC                            A:11
   J-2:ACB       :NOT LOADED           B:  1
      :----+----1                      C:   1
       time-->                          :----+----1
```

(C) Complete, feasible schedule generated by loading J-2 to (B) above. Length (makespan) = 6

```
Job Gantt Chart:                    Machine Gantt Chart:
      :----+----1                         :----+----1
   J-1:AABC                            A:112
   J-2:--A-CB                          B:  1  2
      :----+----1                      C:   12
       time-->                          :----+----1
```

Figure 2(B) shows the "partial" schedule generated by loading Job 1. Figure 2(C) shows the completed, feasible schedule generated by loading Job 2 after loading Job 1. When a job is loaded, the job Gantt chart is updated to reflect the actual time slots in which processing takes place, and a dash is inserted in any time slot in which the job is waiting to be processed. (Note: The Gantt chart scales printed at top and bottom of the charts show time slot number in tens: e.g. 1 indicates time slot 10, etc.)

PRIOR RESEARCH - OPERATIONS RESEARCH FOCUS

Job shop scheduling problems have received the attention of researchers for at least 30 years (see the reference list of Rinnooy Kan (1976)). However, progress with these problems has been disappointing, leading the writers of one reference to assert that "many proficient people have considered this problem, and all have come away essentially empty-handed. Since this frustration is not reported in the literature, the problem continues to attract investigators who just cannot believe that a problem so simply structured can be so difficult until they have tried it." Conway et al. (1967).

In fact, job shop problems have been shown to be "NP-complete" when the number of machines, or the maximum number of tasks in any given job exceeds 3, Lenstra et al. (1977). This puts job shop problems in the same class with such notorious ones as the general 0-1 programming problem and the traveling salesman problem.

A number of approaches to the job shop problem have been suggested in the literature. Broadly these fall into two categories: Optimization and Heuristic. In the following two sections these are reviewed, and recast in the Artificial Intelligence (AI) terminology of search (see Nilsson (1980)).

Optimization Approaches

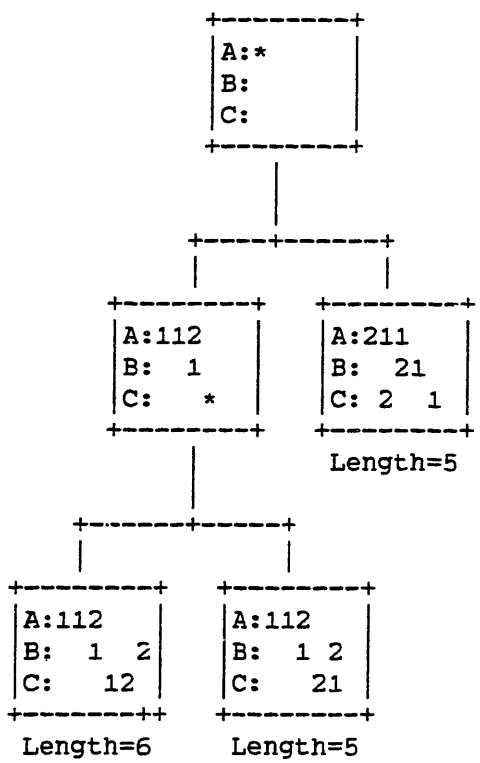
In this category we have the work of researchers who are determined to find the optimum solution to the problem.

Giffler and Thompson (1960) were the first to give an algorithm which would generate an exhaustive enumeration of all possible schedules, perhaps because it seemed a good way to burn up computer time! (Both men were then with IBM.) This algorithm starts with a null schedule, and beginning at time zero marches along tentatively assigning tasks to machines. If a conflict arises, (e.g. two or more tasks overlap on a single machine) it is resolved in every possible way, generating as many new branches in the enumeration space as there are ways of resolving the conflict.

Figure 3 shows the Schedule Enumeration Tree that Giffler and Thompson's approach generates for our sample problem. Each parent node in the tree is associated with a partial schedule containing a conflict situation. (A conflict is indicated with an asterisk (*).) Each tip node in the tree is associated with a feasible, active schedule for the problem. (Active schedules are schedules in which all tasks are shifted left as far as possible given their precedence relationships and queue positions.) The primary appeals of the algorithm are:

1) It generates each schedule only once (non-redundancy), and
2) It generates every possible active schedule (completeness).

Figure 3: A Schedule Enumeration Tree
for the sample problem.



NOTE: * indicates a conflict on the machine.

In AI terminology, we would characterize the exhaustive enumeration approach as a generate and test approach. The generator has the required characteristics of non-redundancy and completeness. Giffler and Thompson proposed a depth-first exploration of the tree, presumably because of the constraints imposed by the hardware (they were using an IBM 704), and because it would generate some schedules even if cut off before the enumeration was complete.

The problem with this approach is that this tree is very bushy! As an example, Giffler et al. (1963) show that a simple problem consisting of 6 jobs on 6 machines, with each job consisting of 5 tasks, has 84,802 active, feasible schedules, or tip nodes.

More recent work with enumeration has used branch-and-bound. (Branch-and-bound algorithms are similar to the A* search algorithms of the AI literature.) The objective here is to prune the enumeration tree in order to reduce the number of schedules generated. At each node in the tree, a lower bound on the objective (e.g. length) is calculated (typically by relaxing the machine capacity constraints on all but one machine). If the lower bound associated with a given parent node is greater than the completion time of the best schedule found so far, the node is pruned from the tree, since the lengths of all tip nodes associated with the parent will be, at best, equal to the lower bound.

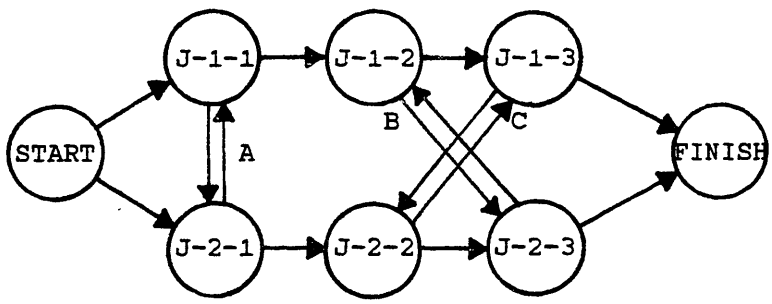
Lageweg, et al. (1977) refer to their work with branch-and-bound as "implicit enumeration", and use a disjunctive graph representation of the problem. Figure 4(A) shows a disjunctive graph representation for

the sample problem. Each job is represented by a set of "conjunctive" arcs indicating the precedence relationships of tasks within the job. Each machine is represented by a set of "disjunctive" arcs indicating the possible processing orders on the machine.

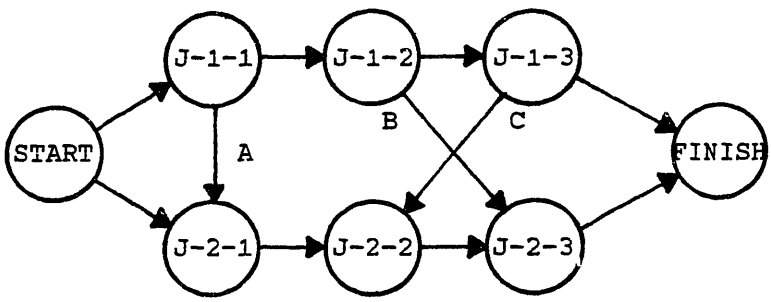
Associated with every feasible schedule is a directed graph, constructed by selecting one arc from each disjunctive pair, with the further restriction that the graph must be acyclic. Figure 4(B) shows a directed graph for the feasible schedule of Figure 2(C).

Figure 4: Disjunctive and Directed Graphs for the sample problem.

(A) Disjunctive Graph for the Sample Problem.



(B) Directed Graph for the Feasible Schedule of Figure 2(C).



The primary appeal of this representation is that it allows an enumeration tree (or perhaps graph) to be generated starting somewhere other than at time zero. The expectation was that this would allow some search strategy to "settle essential conflicts" early, and substantially reduce the number of schedules enumerated. However, the generation of directed graphs from the parent disjunctive graph has proven problematic.

Lageweg, et al (1977) have used branch-and-bound with Giffler and Thompson's active schedule enumeration tree and a "settle essential conflicts" tree, and report that the latter scheme is "clearly worse". Both schemes fail to complete the implicit enumeration of the tree on a 10-job, 10-machine problem in 5 minutes of running time on a Control Data Cyber 73-28.

Heuristic Approaches

Two distinct heuristic approaches have been employed, Monte Carlo and priority dispatching rules. The objective of both approaches is to generate good schedules with a reasonable amount of effort.

Giffler and Thompson (1960) were the first to suggest the Monte Carlo approach. Instead of an exhaustive, depth first search of the enumeration tree, conflicts are resolved by a random choice at each

parent node, until a tip node is encountered. They demonstrate that if this process is repeated enough times, a reasonably good schedule will be generated.

Priority dispatching rules have received attention for two reasons: First, they have the appeal of only requiring local information (e.g. a machine can examine the characteristics of jobs waiting in its queue and make a selection with no knowledge of the status of other jobs or other machines). Secondly, theoretical results for single-machine problems show that local sequencing rules are optimum for certain objective functions, and in the absence of a better solution, their application to complex multi-machine job-shops seems a suitable expedient. Priority dispatching rules eliminate search in the enumeration tree: The application of a given dispatch rule yields a single path in the tree!

The author is aware of only one reference describing AI research directed at the job-shop scheduling problem. Fox (1983) describes a job-shop scheduling system which employs AI concepts. The scheduling domain is represented using a frame based programming language (SRL). Each frame is a collection of slots and values, and may inherit slots and values from other frames. By using slots and values to establish relationships between various frames, a richer problem representation can be achieved, particularly with respect to the many constraints imposed on the problem in the real world (and assumed away in our simple problem statement above).

Constraint frames specify the variable to be constrained, the constraint value, alternate values (relaxations) that the constraint may have, and a utility function which is used to choose between alternative relaxations. (From an OR perspective, this representation is a sort of cross-breed between a constraint and an objective.)

Fox's system develops a schedule using a hierarchical search procedure as summarized below:

> LEVEL 1: ORDER SELECTION. All jobs which are known but not scheduled are kept in a queue at this level. An order is selected for scheduling based on its priority class and due date.
>
> LEVEL 2: CAPACITY BASED SCHEDULING. Given the context of resources currently available in the shop, a critical path approach is used to establish constraints on start and stop times for sub-tasks in the job.

LEVEL 3: BEAM SEARCH FOR A FEASIBLE JOB SCHEDULE. At this level a three stage analysis is performed. First, the search direction (e.g.forward from start date or back from due date) and search operators (e.g.alternative operations or alternative machines) are selected.

Second, a beam search is used to generate alternative schedules for the job. Each node in the search tree is rated based on the "quality" of the partial job schedule leading from the start (or finish) to the node. The rating is determined by first identifying all constraints which are applicable to the partial schedule, and then calculating a utility based on the "acceptability" of the present value of the constrained variables. The search is pushed forward (or backward) until all process steps in the job have been completed.

Third, a post-search analysis evaluates the alternative that have been generated. If the best alternative is acceptable (as determined by a lower bound on the "quality" of the schedule), it is passed to LEVEL 4. If no satisfactory schedule has been found, the schedules are examined to determine an error to pass back to the first stage. Alternative search operators are then selected or the constraints which were passed down from level 2 are modified, and the search repeated.

LEVEL 4: SHOP RESERVATIONS. At this level the job schedule is added to the existing shop schedule by making reservations for the resources required by the job. After LEVEL 4 processing, control returns to LEVEL 1.

Fundamentally, this system works with a partial, feasible shop schedule to which it attempts to add the "most important" job not yet scheduled. Given this top-down orientation, there is little opportunity to discover that jobs should be scheduled in a different order. The job order imposed on the search space at LEVEL 1 may not be optimum, but there is no mechanism to discover this.

The primary features of the system are its ability to capture a more complete set of the real-world constraints (e.g.due dates, tool requirements), and its ability to consider alternative machine routings for each job. (In this paper, and in the OR literature in general, routings are assumed to be fixed).

MOTIVATION FOR THE PRESENT APPROACH

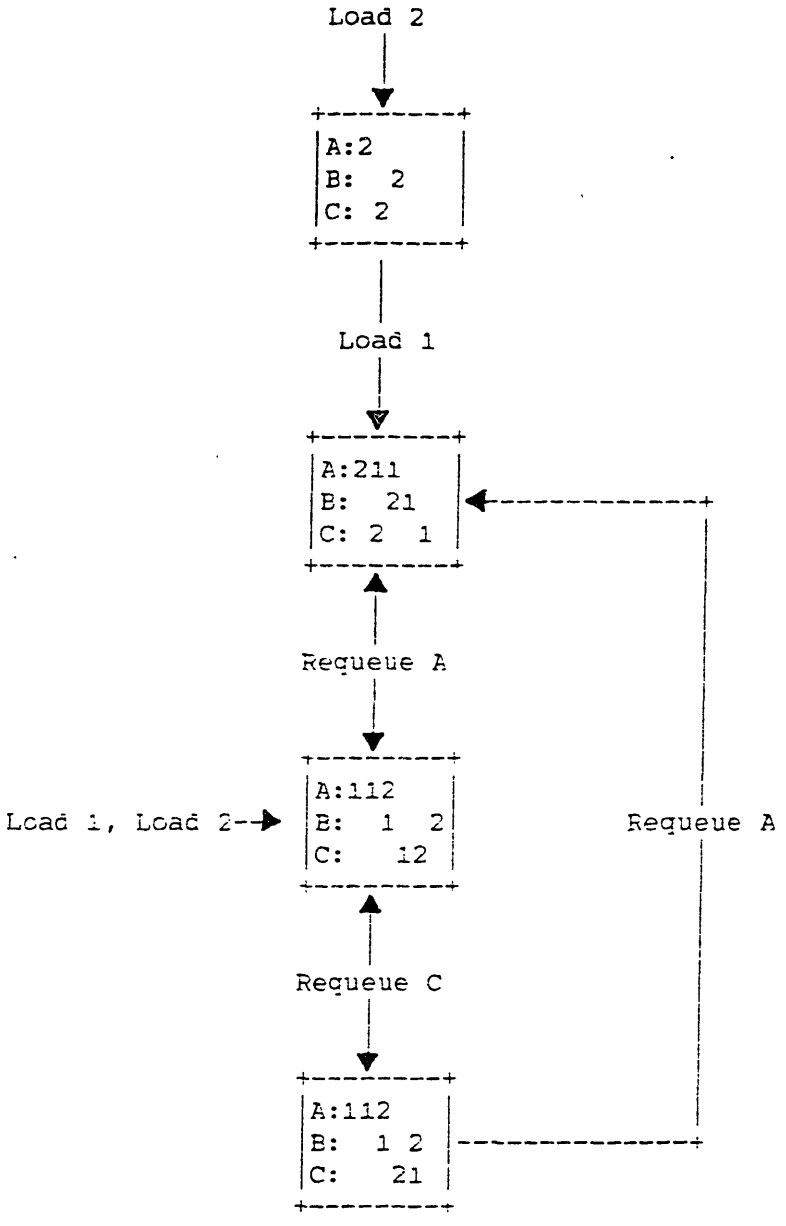
We see that most researchers have adopted the active schedule enumeration tree as the appropriate problem search space. Most of the research effort has then focused on reducing the portion of the tree which must be enumerated to find a solution. Unfortunately, while the enumeration tree is a nice generator for the generate and test approach, it is a representation into which we have little heuristic insight.

Thus the heuristics which have been used have been rather weak. For instance, complete enumeration and Monte Carlo make use only of the length of the best schedule known thus far, throwing away all of the other schedules and any information contained in them. Branch-and-bound makes use of the partial schedule known at a given node and a heuristic choce of lower bound formulation, but apparently the lower bounds devised to date are rather weak. No useful way has been found to use information contained in previously generated schedules to guide the branching and bounding. Finally, dispatching rules are the height of arrogance, assuming away the presence of the tree entirely!

It seems reasonable to assume that some guidance in choosing a new schedule could be gained by examining known, feasible schedules for the problem at hand. This suggests the use of a state-space representation of the problem, where each node represents a feasible schedule, and arcs represent the application of an operator which transforms one feasible schedule into another.

Figure 5 illustrates a state-space representation for the sample problem, where the arcs respresent a specific requeuing operator. This representation gives up the neat tree structure of the enumeration space in exchange for being able to associate a specific schedule with each node. Perhaps by examining the schedule at a given node we can gain some insight about what operators we might like to apply in order to improve the schedule.

Figure 5: A State-space Graph for the Sample Problem



NOTES:
To Requeue: Unload all tasks in the machine queue, and any tasks subsequent to those in their respective jobs, then reload these partial jobs in the desired queue order.

This is probably the representation a human scheduler uses, and it appears that humans are reasonably successful at dealing with the complexity of scheduling problems: Fischer & Thompson (1963) report that when their now classic 6-job, 6-machine problem was "given as a problem to a production class, a schedule that completes in time 55 was devised, and required about two man-hours to complete." This same 6x6 problem is the most complex problem solved to date by implicit enumeration, Lageweg, et al. (1977).

The fact that humans perform well relative to the implicit enumeration suggests that either humans are very good at this type of problem or that the use of the enumeration tree as the search space is inappropriate. Perhaps the truth lies somewhere between. At any rate, these observations suggest the following directions for exploration:

1) The development of computer based tools which aid the human scheduler with the bookkeeping details might increase the speed with which humans can solve these problems, and enhance the range of problems they can be solved.

2) By modeling the human approach to scheduling, it might be possible to deliver quite acceptable problem solving performance with problems which are too complex for present enumeration approaches.

This thesis reports on preliminary investigations in both of these directions. A computer program is developed which automates the

bookkeeping functions associated with the construction of job-shop schedules. This program, Scheduler's Aid, is described in Chapter 2. Based on experience using the program, a simple heuristic search method is developed to guides search in a state space problem representation. This method is described in Chapter 3 along with results for sample problems. Chapter 4 presents a summary comments.

CHAPTER 2 - SCHEDULER'S AID

Scheduler's Aid is a program which automates the bookkeeping chores associated with the construction of job-shop schedules. It provides facilities for defining jobs, adding jobs and tasks to the schedule, removing jobs and tasks from the schedule, and displaying the schedule.

The program is written in MACLISP, and runs on the OZ PDP-10 system at MIT's Artificial Intelligence Lab. LISP was chosen because the language lends itself well to system prototyping, because its data structure is very flexible, and because it is the language of choice for AI applications.

PROBLEM REPRESENTATION & DATA STRUCTURE

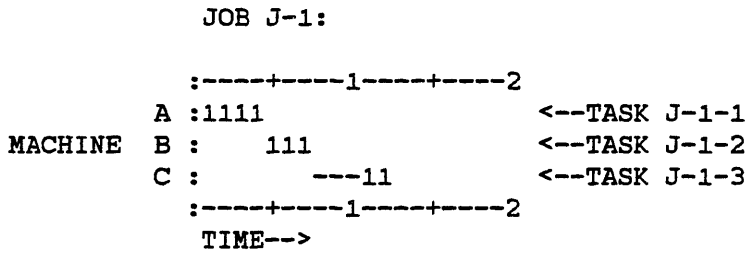
Each job is represented by a collection of atoms. (An atom in LISP is analogous to a variable name in other languages.) Information about the job is carried in the form of property-value pairs associated with these atoms.

Each job has one atom of type JOB which carries information pertaining to the job-at-large, such as starting task, processing time, flow time, etc. For example, in Figure 6, J-1 is a job which starts

with task J-1-1, has total processing time of 9 time units, and a flow time of 12 time units in the present schedule.

Each job is made up of a number of tasks, each of which is represented by an atom of type TASK. These atoms carry information pertaining to the specific task. For example (referring again to figure 6), the second task in job J-1 is task J-1-2. J-1-2 is processed on MACHINE B in 3 units of time (PTIME). The PREVIOUS task in the job is J-1-1, and the NEXT task in the job is J-1-3. J-1-2 is currently LOADED, starting processing in machine slot B-5. Since the previous task is finished processing at the end of time slot 4, J-1-2 spends no time queued (QTIME), and its QUEUED property is NIL. (NIL is LISP's equivalent of the null set.) Task J-1-1 is the first task in the job, thus its PREVIOUS property has a value of START. Task J-1-3 is the last task in the job, thus its NEXT property has a value of FINISH. J-1-3 also spends 3 time units queued (QTIME) at machine C, with the first machine slot in which it is QUEUED being C-11.

Figure 6: JOB Representation



ATOM	PROPERTY	VALUE
J-1	TYPE	JOB
	START-TASK	J-1-1
	PTIME	9
	FTIME	12
J-1-1	TYPE	TASK
	MACHINE	A
	PTIME	4
	PREVIOUS	START
	NEXT	J-1-2
	LOADED	A-1
	QUEUED	NIL
	QTIME	0
J-1-2	TYPE	TASK
	MACHINE	B
	PTIME	3
	PREVIOUS	J-1-1
	NEXT	J-1-3
	LOADED	B-5
	QUEUED	NIL
	QTIME	0
J-1-3	TYPE	TASK
	MACHINE	C
	PTIME	3
	PREVIOUS	J-1-2
	NEXT	FINISH
	LOADED	C-11
	QUEUED	C-8
	QTIME	3

Machines are represented as shown in figure 7. Time is quantized into slots, and each slot is represented by an atom. An atom of type MSLOT is defined for every time slot in which a given machine is loaded with a task, or has a job waiting in its queue. (It is possible for a machine to have a queue when no job is loaded if the job in the queue is being held until after a subsequently available task is processed). Only one task can be loaded into a given machine slot, and the property LOADED carries a value equal to this task. Multiple tasks can be queued at a given MSLOT; a list of these tasks (enclosed in parenthesis) is carried in the QUEUED property of the MSLOT.

Figure 7: MACHINE Representation

```
          :----+----1----+----2
  JOB J-1 :AABB
  JOB J-2 :B-A
          :----+----1----+----2
MACHINE A :1122
MACHINE B :2 1
          :----+----1----+----2
           TIME-->
```

ATOM	PROPERTY	VALUE
A-1	TYPE	MSLOT
	LOADED	J-1-1
	QUEUED	NIL
A-2	TYPE	MSLOT
	LOADED	J-1-1
	QUEUED	(J-2-2)
A-3	TYPE	MSLOT
	LOADED	J-2-2
	QUEUED	NIL
A-4	TYPE	MSLOT
	LOADED	J-2-2
	QUEUED	NIL
B-1	TYPE	MSLOT
	LOADED	J-2-1
	QUEUED	NIL
B-3	TYPE	MSLOT
	LOADED	J-1-2
	QUEUED	NIL

A number of commands are understood by the program. A list and brief description of each is given in Figure 8. These commands are interpreted at LISP's top level, and are actually LISP function calls, which must be enclosed in parenthesis.

Obviously, commands which alter the schedule must properly update the values of various properties in job and machine atoms. For example a call to LOADTASK must find a sufficiently large block of free slots on the target machine. It then loads the task onto the machine, by setting the LOADED property of each machine slot equal to the task name. The LOADED property of the TASK atom is set equal to the name of the first slot that the task is loaded into. Finally, the TASK is added to the list stored at the QUEUED property of each machine slot during which the TASK is waiting to be loaded, and the QUEUED and QTIME property values of the TASK are suitably updated.

Figure 8: Command Summary

COMMAND	DESCRIPTION
(JOBDEF <job id> <tlist>)	Define a job
(LOADJOB <job>)	Load a job into the schedule
(UNLOADJOB <job>)	Unload a job from the schedule
(LOADTASK <task>)	Load a task into the schedule
(UNLOADTASK <task>)	Unload a task from the schedule
(SHUFFLE <task>)	Left-shift task, and all subsequent tasks in the job
(JGANT <job>)	Display a job gantt chart
(JGANTS)	Display job gantt charts for all jobs
(MGANT <machine>)	Display a machine gantt chart
(MGANTS)	Display machine gantt charts for all machines
(GANTS)	Equivalent to (JGANTS),(MGANTS)
(FTIMES)	Display the flow times of all jobs

NOTES:

<jobid>	a single-character job id, e.g. 1
<tlist>	a list of machine, processing time pairs, e.g. '(A 5 B 3 C 6)
<job>	a job name, e.g. 'J-1
<task>	a task name, e.g. 'j-1-1
<machine>	a single-character machine id, e.g. 'A
Left-Shift	Move the task left on the machine Gantt chart into open slots, subject to the constraint that it must not be loaded before the previous task has completed processing

DISPLAYS

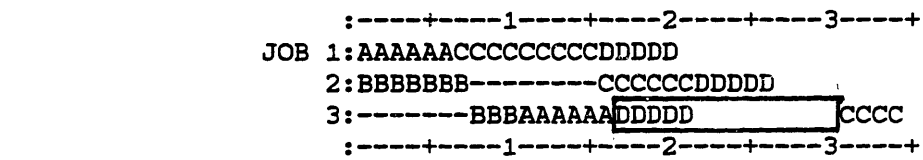
The program displays the current state of the schedule using Gantt charts. A typical display is shown in Figure 9. A column is printed for each time slot occupied by the schedule. Two general types of Gantt chart are produced: Job and Machine.

Job Gantt charts show the sequence of tasks in a job. Tasks are differentiated within job Gantt charts by printing a machine identifier (usually a letter) in the column(s) representing time slots in which the task is loaded. A dash in a column indicates that the task is waiting in a queue. Tasks which are not currently loaded into the schedule are displayed in reverse video (indicated by a box in the example).

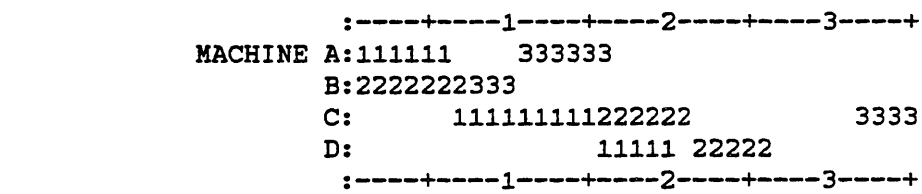
Machine Gantt charts show the sequence of tasks loaded into a machine. Tasks are differentiated by printing a job identifier (usually a number) in the machine slots in which the task is loaded. A dash indicates that no job is loaded, but that jobs are waiting in the queue.

Figure 9: Sample Program Displays

JOB GANTT CHARTS:



MACHINE GANTT CHARTS:



NOTE: A box indicates a task presently not loaded

Using the program, the author was able to solve several benchmark problems from the literature. Figure 10 shows solutions to these problems. (These are assumed to be the solutions since they have length equal to the optimums cited in the literature. Actual schedules are not presented in the literature, presumably because the optimum schedule is typically not unique.) After entering the problem data (typically requiring 5 minutes), the problems were solved in the following times:

A) A 4x4 problem (from Rinnooy Kan (1976) page 164) required about 4 minutes.

B) A 5x4 problem (also from Rinnooy Kan) required about 7 minutes.

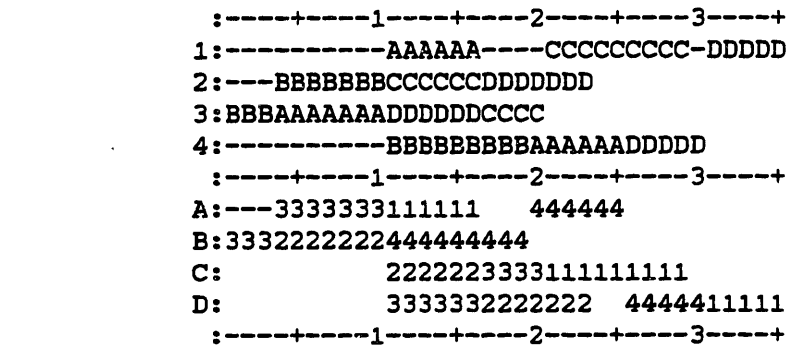
C) A 6x6 problem (from Fisher and Thompson (1963) page 236) required about 2 hours.

The author may have had an unfair advantage since he was aware of he minimum lengths of the problems, and thus knew when to keep working on the problem, and when to stop and have a beer.

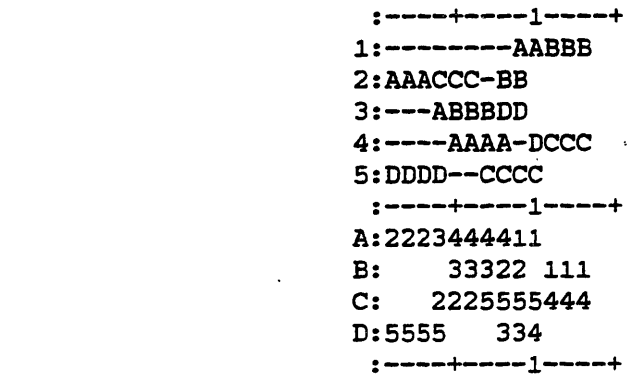
The 6x6 problem is the largest problem solved to date by Lageweg et al. (1977). As noted earlier, it has been solved manually before, however it required the efforts of a production class. We might tentatively conclude that Scheduler's Aid provides some improvement in a human scheduler's performance.

Figure 10: Solutions to three benchmark problems:

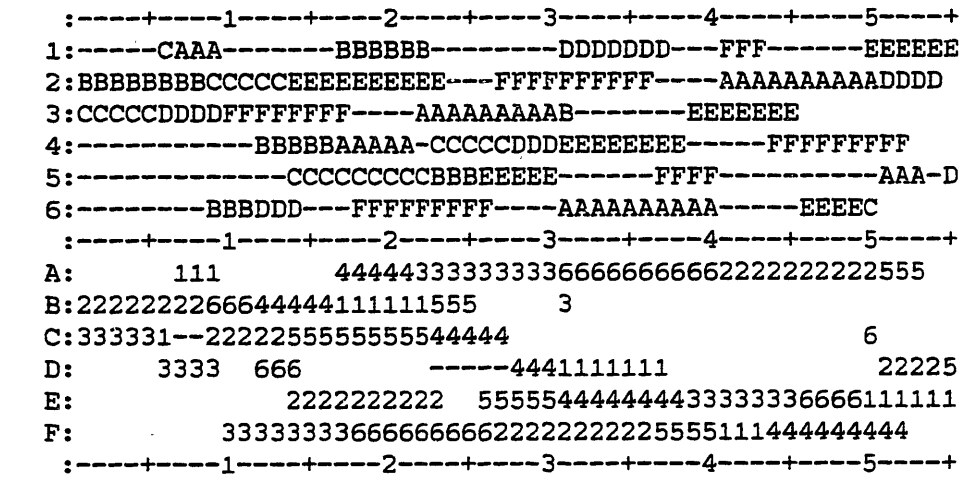
A) 4-job, 4-machine problem:



B) 5-job, 4-machine problem:



C) 6-job, 6-machine problem:



CHAPTER 3 - A SIMPLE HEURISTIC SEARCH APPROACH

In this section a simple heuristic search method is developed. Results are presented for two sample problems.

SEARCH OPERATOR

The objective is to develop a search (or successor) operator which, when given a schedule, will generate one or more successor schedules. Ideally, the operator would generate only schedules which are improvements on the given schedule. More reallisticly, the operator should generate only a limited number of new schedules, each of which is reasonably likely to be better than the given schedule.

In general, it is not possible to tell when an optimum schedule has been encountered; thus when presented with an optimum schedule, the operator can only be expected to generate schedules which are similar, or worse than the given schedule.

The simple operator presented here is derived from observing the first order behavior of human schedulers solving schedules with the objective of minimizing length. It is clearly simplistic, but serves to illustrate the approach.

Given a schedule, we select the job with the longest flow time. (In the event that more than one job has this flow time, a job is chosen from this set at random.) If any tasks in this job are waiting in queues, then one or more load options are generated. (If no tasks are waiting in queues then the optimum schedule has been found, an unlikely situation.) A load option consists of a target task, which will be one of the queued tasks in the job, and a target slot determined as described below:

For each queued (target) task, target slots (machine slots in which the task might be loaded) are determined by examining the machine slots in which the task is queued. A target slot is identified as follows:

1) The first slot in which the task is queued.
2) Any slot in which the task loaded is different than the task loaded in the previous slot.
3) Any machine slot which is empty, and which follows a loaded slot.

Figure 11 shows the load options for a given schedule.

Associated with each of the load options are (possibly) other tasks which are "in the way". These are tasks from other jobs which will overlap the target task if it is loaded at the target machine slot. These are also illustrated in Figure 11.

Figure 11: Load Options and Tasks In-the-way for a Sample Schedule.

```
SAMPLE SCHEDULE:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAACCCCCCCCCDDDDD
2:BBBBBBB--------CCCCCCDDDDDDD
3:-------BBBAAAAAAA-----------DDDDDDCCCC
4:----------BBBBBBBBBAAAAAA---------DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111    3333333  444444
B:2222222333444444444
C:      111111111222222              3333
D:               11111-222222233333344444
 :----+----1----+----2----+----3----+----4----+


------LOAD OPTION--------
Target Task   Target Slot          TASKS IN-THE-WAY
-------------------------------------------------
   J-4-1         B-1               J-2-1, J-3-1
   J-4-1         B-8               J-3-1
   J-4-3         D-26              J-2-3, J-3-3
   J-4-3         D-29              J-3-3
```

Each time the search operator is invoked on a given schedule (parent node), it generates successor schedules (successor nodes) for all of the load options. The generation of a successor schedule from a given load option proceeds as follows:

1) Determine what tasks are in the way.
2) For each in the way task, UNLOAD that task, and all NEXT tasks in its job.
3) SHUFFLE the target task into the target slot
4) LOAD each task that was in the way, and all NEXT tasks in its job.
5) Examine the machine Gantt chart to see if there are any machine slots which are QUEUED but not LOADED. If so, attempt to left-shift any of the QUEUED tasks into these empty slots.

Figure 12 illustrates the generation of a successor schedule for a given parent schedule and load option.

Figure 12: The Generation of a Successor schedule from The Parent Schedule shown in Figure 11.

```
LOAD OPTION: (J-4-1 B-1)
TASKS IN-THE-WAY: J-2-1, j-3-1

UNLOAD EACH IN-THE-WAY TASK, AND NEXT TASKS:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAACCCCCCCCCDDDDD
2:BBBBBBBCCCCCCDDDDDDD
3:BBBAAAAAAADDDDDCCCC
4:----------BBBBBBBBBAAAAAA---------DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111              444444
B:----------444444444
C:      111111111
D:                11111--------------44444
 :----+----1----+----2----+----3----+----4----+

SHUFFLE J-4-1:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAACCCCCCCCCDDDDD
2:BBBBBBBCCCCCCDDDDDDD
3:BBBAAAAAAADDDDDCCCC
4:BBBBBBBBBAAAAAA-----DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111   444444
B:444444444
C:      111111111
D:               1111144444
 :----+----1----+----2----+----3----+----4----+

LOAD EACH IN-THE-WAY TASK, AND NEXT TASKS:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAACCCCCCCCCDDDDD
2:---------BBBBBBBCCCCCC---DDDDDDD
3:----------------BBBAAAAAAA------DDDDDDCCCC
4:BBBBBBBBBAAAAAA-----DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111   444444    3333333
B:4444444442222222333
C:      111111111 222222                 3333
D:               11111444442222222333333
 :----+----1----+----2----+----3----+----4----+
```

SEARCH CONTROL STRATEGY

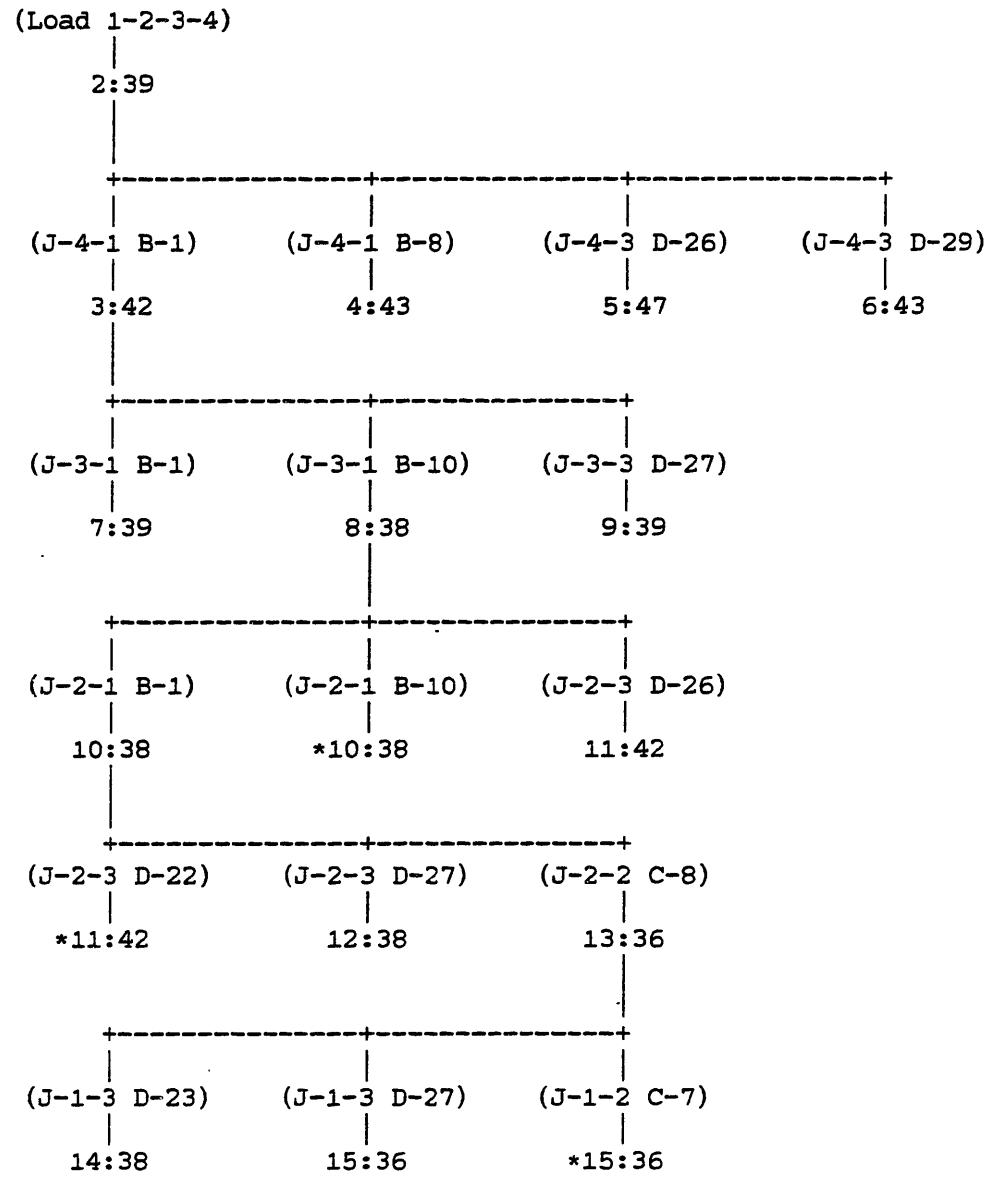
The search control strategy employed is hill-climbing with backtracking. Each time the search operator is applied to a schedule (node), all successor nodes are generated, forming an "expansion" of the given node. If any node in this expansion has been encountered before, it is ignored. Nodes in the expansion are evaluated by observing the length of the associated schedule. The best node in the expansion (with minimum length) is selected and the search operator is applied to this node. In the event that the search operator fails to generate any unique nodes, the search backtracks to next best node in the prior expansion. The depth of search is controlled by specifying the number of nodes to be expanded.

RESULTS

This heuristic search method has been applied to the 4x4 and 6x6 benchmark problems of Chapter 2. Figure 13 shows the search tree generated by the method for the 4x4 problem. The search was started at a schedule generated by loading the jobs in the order 1-2-3-4. Figure 14 shows the starting schedule, and the best schedule found in each expansion. Since the performance of the method is likely to be a function of the starting schedule, it was applied to several starting

schedules for each problem. Figure 15 shows the results of applying the method to four starting schedules for the 4x4 problem. For each starting schedule the following are shown: the order in which the jobs were loaded to generate the starting schedule, the length of the starting schedule, and the length of the schedules in each expansion. Since no backtracking occurs, the shortest schedule in each expansion is the parent of the following expansion. With each of 4 starting schedules, the method found at least one schedule of length 36. As already cited, the optimum for this problem is 35.

Figure 13: The Search Tree generated by the method for the 4x4 benchmark problem.



Notation:

2:39	indicates schedule number 2: length 39.
(J-4-1 B-1)	indicates a load option with target task J-4-1, target slot B-1.
*10:38	indicates that the schedule has been encountered earlier in the search.

Figure 14: The Starting Schedule, and the Best Schedule Found in Each Expansion of Figure 13

```
Schedule 2:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAACCCCCCCCCDDDDD
2:BBBBBBB--------CCCCCCDDDDDDD
3:-------BBBAAAAAAA----------DDDDDDCCCC
4:----------BBBBBBBBBAAAAAA---------DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111    3333333  444444
B:222222333444444444
C:      111111111222222             3333
D:               11111-222222233333344444
 :----+----1----+----2----+----3----+----4----+


Schedule 3:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAACCCCCCCCCDDDDD
2:---------BBBBBBBCCCCCC---DDDDDDD
3:----------------BBBAAAAAAA------DDDDDDCCCC
4:BBBBBBBBBAAAAAA-----DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111   444444    3333333
B:4444444442222222333
C:      111111111 222222                  3333
D:               1111144444222222333333
 :----+----1----+----2----+----3----+----4----+


Schedule 8:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAACCCCCCCCCDDDDD
2:-----------BBBBBBBCCCCCC------DDDDDDD
3:---------BBB---AAAAAAA---DDDDDDCCCC
4:BBBBBBBBBAAAAAA-----DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111   4444443333333
B:4444444443332222222
C:      111111111    222222      3333
D:               11111444443333332222222
 :----+----1----+----2----+----3----+----4----+
```

Figure 14: (Continued)

```
Schedule 10:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAACCCCCCCCCDDDDD
2:BBBBBBB--------CCCCCC----------DDDDDDD
3:-------BBBAAAAAAA---DDDDDDCCCC
4:----------BBBBBBBBBAAAAAA-DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111    3333333  44444
B:2222222333444444444
C:      111111111222222     3333
D:               11111333333444442222222
 :----+----1----+----2----+----3----+----4----+


Schedule 13:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAA-------CCCCCCCCC---------DDDDD
2:BBBBBBBCCCCCCDDDDDDD
3:-------BBBAAAAAAA---DDDDDDCCCC
4:----------BBBBBBBBBAAAAAA-DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111    3333333  444444
B:2222222333444444444
C:      -222222111111111    3333
D:             22222223333334444411111
 :----+----1----+----2----+----3----+----4----+


Schedule 15:

 :----+----1----+----2----+----3----+----4----+
1:AAAAAA-------CCCCCCCCC----DDDDD
2:BBBBBBBCCCCCCDDDDDDD
3:-------BBBAAAAAAA---DDDDDDCCCC
4:----------BBBBBBBBBAAAAAA------DDDDD
 :----+----1----+----2----+----3----+----4----+
A:111111    3333333  444444
B:2222222333444444444
C:      -222222111111111    3333
D:             222222233333311111144444
 :----+----1----+----2----+----3----+----4----+
```

Figure 15: Results for the 4x4 Problem

---STARTING NODE---		----------EXPANSION----------	
Load Order	Length	Number	Lengths of Successors
1-2-3-4	39	1	42,43,43,47
		2	38,39,39
		3	38,38,42
		4	36,38,42
		5	36,38,38
4-3-2-1	40	1	40,40,42
		2	38,42,42,42
		3	36,38,39
		4	36,36,38
		5	36,41,41,42
1-4-2-3	42	1	38,39,39
		2	38,38,42
		3	36,38,39
		4	36,36,38
		5	41,41,42,42
4-1-2-3	38	1	38,42,42
		2	36,38,39
		3	36,36,38
		4	36,41,41,41,42,43,45

NOTE: The best schedule from an expansion is the parent of the following expansion.

Figure 16 shows the results of applying the method to the 6x6 problem. Five expansions of 2 starting schedules were generated. In both cases the method found at least one schedule of length 59 in less than five expansions and less than thirty schedules. It is interesting to contrast these results with results from the literature: The implicit enumeration method found an optimum value of 55 after expanding between 62 and 411 nodes in the enumeration tree, depending on the choice of enumeration algorithm and lower bound formulation, Lageweg et al. (1977). A Monte Carlo sampling of 500 active schedules yielded a minimum length of 60, the LRT (Longest Remaining Time) dispatching rule yielded 61, and the SIO (Shortest Imminent Operation) rule yielded 67, Fisher and Thompson (1960). Given the simplicity of the heuristic used to generate load options, and the arbitrary choice of search control strategy, the heuristic search method appears to perform reasonably well.

The heuristic search method can most likely be improved. It is the author's experience in solving these problems that a point is usually reached where examination of the longest job fails to give useful ideas as to load options. At this point attention usually shifts to the machine Gantt chart to determine which machines are bottlenecks. Often a different loading order on other machines can get the bottleneck machine started earlier resulting in a shorter schedule. Presumably the heuristic could be expanded to take account of machine loading patterns, and its performance improved.

Figure 16: Results for the 6x6 Problem

---STARTING NODE---		----------EXPANSION----------	
Load Order	Length	Number	Lengths of Successors
1-2-3-4-5-6	71	1	70,71,73
		2	61,62,62,67,71,71,91,91
		3	59,60,64,64,82,90
		4	59,59,59,68,70
		5	64,69,70,70
6-5-4-3-2-1	86	1	67,69,71,71,73,86,86
		2	64,64,67,80,89
		3	62,63,63,70,93
		4	59,60,62,62,85,93
		5	64,69,69,70

NOTE: The best schedule from an expansion is the parent of the following expansion.

CHAPTER 4 - SUMMARY

A BRIEF REVIEW

This thesis has taken a look at an old problem from a new perspective. While the idea of taking a heuristic approach to the job-shop scheduling problem is not new, we have seen that previous heuristics have accepted the scheduling enumeration tree as the appropriate search space. This tree is appealing because of its non-redundancy and completeness, but it is hopelessly bushy for problems of interesting complexity.

It has been the thesis of this research that an alternative representation and search space might yield more powerful heuristics at the cost of the ability to state an algorithm which guarantees an optimum solution. We have shown that the use of Gantt charts in conjunction with an appropriate choice of operators yields a state-space problem representation. With this choice of representation, the focus of research shifts logically to the development of heuristic search methods of sufficient power to give good schedules. We have shown that even a rather crude heuristic search method gives surprisingly good schedules. Certainly there is room for improvement in the heuristic described in this paper.

The Gantt chart problem representation essentially transforms the job-shop scheduling problem into a puzzle. The entries in the job and machine Gantt charts can be thought of as the pieces of the puzzle. The object of the puzzle is to arrange the pieces on the two Gantt charts in such a way that the precedence relationships of the tasks in each job are observed, no more than one piece occupies any one time slot, and the Gantt charts are of minimum length.

Various examples of puzzles are found in the AI literature; popular ones include the 8-Puzzle and the Tower of Hanoi. The job-shop puzzle is similar to these in that we can visualize the doing of the puzzle as the moving of pieces, and the objective of our heuristic should be to tell us which pieces to move next. The AI puzzles that the author is aware of involve a goal state which is clearly defined, typically in terms of a specific arrangement of the pieces. However, in the case of the job-shop puzzle the goal state is not known at the outset: We can't state the desired positions of the pieces, in fact we can't even state the length of the schedule we are seeking!

On the other hand, the job-shop puzzle has a well defined objective function, a characteristic of classic optimization problems. This is in contrast with AI puzzles, where the formulation of an evaluation function is often problematic. Thus the job-shop problem is a sort of

hybrid; it has an interesting mix of the features of both puzzles and optimization problems.

THE REPRESENTATION AS AN INTERPFACE

Finally, we note that a major advantage of the Gantt chart problem representation is its convenience as a debugging tool and a user interface. It seems reasonable to assume that no matter how advanced scheduling systems become, users will need to interact with them, (e.g. to account for some constraint that isn't in the model but which suddenly becomes important). A system build around a Gantt chart representation lends itself easily to the construction of the user interface. As Graves (1981) has pointed out, "a frequent comment heard in many scheduling shops is that there is no scheduling problem but rather a rescheduling problem." This suggests the need for a friendly user interface since scheduling in practice will likely be a highly interactive activity; given the dynamics of real environments and the fact that models of the environment are seldom perfect.

REFERENCES

Conway, R.W., W.L. Maxwell, L.W. Miller (1967)
Theory of Scheduling.
Addison-Wesley, Reading, MA,
as cited by Lageweg et al.

Fisher, H., G.L. Thompson (1963)
Probabilistic Learning Combinations of Local Job-Schop Scheduling Rules.
in Industrial Scheduling, edited by J.F. Muth and G.L. Thompson, Prentice-Hall, Inc, Englewood Clifts, NJ, pp 225-251

Fox, M.S. (1983)
Constraint-Directed Search: A Case Stude of Job-Shop Scheduling.
PhD. Thesis, Computer Science Department, Carnegie-Mellon University, Pittsburg, PA

Giffler, B., G.L. Thompson (1960)
Algorithms for Solving Production-Scheduling Problems.
in Operations Research, Vol 8, p 487-503

Giffler, B., G.L. Thompson, V. Van Ness (1963)
Numerical Experience with the Linear and Monte Carlo Algorithms for Solving Production Scheduling Problems.
in Industrial Scheduling, edited by J.F. Muth and G.L. Thompson, Prentice-Hall, Inc, Englewood Clifts, NJ, pp 21-38

Graves, S.C. (1981)
A Review of Production Scheduling.
in Operations Research, Vol 29, No 4, July-August 1981

Lageweg, B.J., J.K. Lenstra, A.H.G. Rinnooy Kan (1977)
Job-shop Scheduling by Implicit Enumeration.
in Management Science, Vol 24 Num 4, December, pp 441-450

Lenstra, J.K., A.H.G. Rinnooy Kan, P. Brucker (1977)
Complexity of Machine Scheduling Problems.
in Ann. Discrete Math., Vol 7, pp 343-362,
as cited by Lageweg et al.

Nilsson, N.J. (1980)
Principles of Artificial Intelligence.
Tioga Publishing Co., Palo Alto, CA

Rinnooy Kan, A.H.G (1976)
Machine Scheduling Problems.
Martinus Nijhoff, The Hague