A KNOWLEDGE-BASED FAIR-SHARE SCHEDULER

by

SUSAN M. SAAD

B.S., University of Colorado, 1982

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Approved by:

Major Professor

LD
2668
.R4
CMSC
1989
S23
c.2

A11208 317795

CONTENTS

1. INTRODUCTION.. 1
2. THE SCHEDULING PROBLEM.............................. 5
3. EXPERT SYSTEMS...................................... 8
4. REQUIREMENTS.. 16
5. DESIGN.. 21
6. CONCLUSION AND EXTENSIONS........................... 41
7. BIBLIOGRAPHY 43
8. APPENDIX A - C5 RULES............................... 45
9. APPENDIX B - EXTERNAL SORT FUNCTION................. 68

LIST OF FIGURES

Figure 5-1. Element Classes........................ 22
Figure 5-2. State Diagram.......................... 29
Figure 5-3. Allocating Slots for a Shop............ 33
Figure 5-4. Scheduling Non-Short Selects........... 36
Figure 5-5. Adding a Product to Gypped List......... 37

1. INTRODUCTION

Expert systems are playing a larger role in today's society -- especially in the areas of manufacturing and scheduling. Although expert systems have existed since the early 1960's, only recently have they become commercially viable and have they begun to migrate from the universities to industry [Yusko86]. Knowledge-based systems are being used to diagnose problems with the manufacturing process, to help engineers convert product conceptions into designs, and to train operators on the factory floor. In addition to traditional planning systems which have existed for years, expert systems are now being used to create schedules for manufacturing organizations.

In today's manufacturing environment, many long-term planning systems exist; however, decisions about the sequencing of short-term work is often not supported by computer systems. These schedules are often created by a supervisor who uses rules of thumb that were acquired from years of experience. Often, when creating these schedules, a supervisor will have to consult with other supervisors since the derived schedules can affect many down-line operations and organizations.

Due to more efficient tooling, robotics, and principles of just-in-time manufacturing, the short-term planning and scheduling functions must be performed quicker and more accurately. As setup and throughput times decrease, the lot sizes to be manufactured are decreasing; this creates more work units that must be scheduled during a period. "Just-in-time is having what is needed, when it is needed" [Lubben88]; therefore, accurate schedules are becoming more important. Technological advances and just-in-time concepts require that more work be assigned correctly to schedules in a short time period.

There are two approaches to a scheduling problem--an algorithmic approach and a heuristic approach. Each approach has its advantages and disadvantages. An algorithmic solution often finds an optimal solution at the expense of the time to collect large amounts of data and the time to compute the result. Also, the method to get to the solution can be difficult to understand. On the other hand, a heuristic solution uses rules of thumb which result in good or acceptable solutions; heuristic solutions usually require less data and computing effort, and the solution is easier to understand.

Expert systems use the heuristic solutions to a problem; conventional computer systems use algorithmic solutions. Expert systems usually address small tasks performed by professionals, usually tasks taking from a few minutes to a couple hours; conventional data processing systems process large volumes of data in order to automate time-consuming clerical jobs. An expert system uses an inferential process; a traditional system uses a repetitive process [Bryant88].

There are many numerical and non-numerical problems which are better suited for an algorithmic solution rather than a heuristic solution and visa versa. It can be difficult to determine which of these approaches should be used for a given application. Some scheduling systems are a blend of both approaches incorporating aspects of conventional algorithmic techniques and expert systems where appropriate.

This paper describes an expert system that schedules work for a warehouse that must serve several shops in a factory. It discusses scheduling problems in general, expert systems, and the warehouse's scheduling dilemma. This paper also describes the expert system tool that was selected (C5), how C5 operates, and how it was used to

solve the warehouse's scheduling problem. Last, this paper derives some conclusions regarding expert systems in the scheduling process and provides some recommendations for further work in this area.

2. THE SCHEDULING PROBLEM

There is a gap in the conventional Material Requirements Planning (MRP) packages; they carry long-term planning information and little execution and control information. Therefore, many decisions about the allocation of resources and sequence of jobs is made by a foreman or supervisor [Gilmore87]. The foreman's or supervisor's job is becoming more difficult and important in today's rapidly changing manufacturing environment. The need for timely decisions based on current information is increasing.

Management science solutions (algorithmic solutions) are hindered by: "1. deterministic performance times 2. single project operations 3. splitting activities 4. unconstrained resources" [Kim88]. First, performance times for an activity are never certain; more often, the performance times are variable. Second, much of the past management science work has assumed that a project is isolated from other projects; however, projects often compete for the same resources. Third, the objective of a schedule should be to meet several goals at once rather than to focus on a single goal of one activity. Last, some management science solutions, the critical path

analysis for example, assume unlimited resources are available--a state not experienced by many supervisors. Sang O. Kim reached the conclusion that past research in the scheduling area has lacked generality and practicality [Kim88].

When multiple entities are involved in a scheduling process, scheduling can become more difficult and confusing. This difficulty exists because the various entities may have conflicting goals (Kim88]. Therefore, it can be better to approach the problem on a global level rather than getting tangled in the conflicts that arise between each entity. Hopefully, a schedule produced from a global viewpoint will be an acceptable compromise between the needs of the individual entities.

One of the principles of just-in-time manufacturing is "to continually seek the path of simplicity" [Lubben88]. This principle can apply to scheduling in a factory also. A simple way to view a scheduling problem is to think of the problem as a limited resource allocation problem [Weist67]. A limited amount of work can be performed in each schedule period. A simple rule of thumb is: "If an important piece of work cannot be scheduled during a schedule period, then give that piece of work highest

priority during the next schedule period when resources are available." Other heuristics exist which can simplify the scheduling process.

Stephen F. Smith and Mark S. Fox describe some important factors that affect factory scheduling decisions. First, the schedule must meet the scheduling restrictions; these include the availability of resources and material. Next, the schedule should comply with the scheduling preferences. Scheduling preferences are such things as minimizing work in process, meeting due dates, and stabilizing shop work loads [Smith85]. Although the types of scheduling restrictions and scheduling preferences will vary from case to case, scheduling decisions should be based on these factors.

3. EXPERT SYSTEMS

Frederick Hayes Roth, Donald A. Waterman, and Douglas B. Lenat define an expert system as " a computer system that achieves high levels of performance in task areas that, for human beings, require years of special education and training" [Hayes-Roth83]. Knowledge-based expert systems use human knowledge to solve problems. Expert systems also employ other functions of an expert such as asking relevant questions and explaining its reasoning. However, unlike a human expert, expert systems cannot use common sense reasoning or handle inconsistent knowledge.

An expert system can provide several advantages for a manufacturing company. One main advantage is that an expert system is permanent. An expert's time is often limited, or an expert may leave the company. By creating an expert system, one is making the expert's knowledge a corporate resource [Yusko86]. In addition, the time and cost to produce a new human expert can be very high. Some other benefits of expert systems in the manufacturing arena include that the expertise can be used in many locations, that the expertise is more consistent than a human expert, and that the expert system may fit better in a hostile environment.

The rule-based system consists of three components-- working memory, rule memory, and an inference engine. Working memory contains the data representing the facts and assertions about the problem [Brownston86]. Rule memory, also referred to as production memory, contains the series of rules; each rule consists of a condition portion or left-hand side (if . . .) followed by an action or right-hand side (then . . .). Last, the inference engine is the control mechanism for the system. It determines which rules should be fired next based on the contents of working memory.

There are two types of expert systems -- forward chaining and backward chaining. Forward chaining systems are driven by data. They use the information on the left-hand side of rules to derive the right-hand side [Waterman86]. Backward chaining systems are directed by a goal or a hypothesis. They start with what they want to prove and execute the rules relevant to getting to the goal [Waterman86].

The expert system described in this paper attempts to translate the warehouse and shop supervisor's knowledge into a computerized expert system. This is done after extracting the knowledge from the expert through a

process called knowledge engineering. This process involves watching the expert solve the problem on the job, determining the data and knowledge needed to solve the problem, having the expert describe a typical problem and its solutions, and presenting the expert with a series of problems to solve while recording the reasoning behind each step. The process continues with the expert giving the knowledge engineer a series of problems to solve with the derived rule set. Lastly, the expert should verify the set of rules that were created [Waterman 86]. After this process, one should be close to capturing the expert's knowledge in a series of rules and control constructs.

A task is considered to be a candidate for an expert system if it has the following characteristics. The expert system should be cost effective by increasing revenues or decreasing costs or by boosting the efficiency of an organization by making expertise more readily available [Bryant88]. Also, the application should be of manageable size but should not be a task that is too easy; the nature of the task should involve symbol manipulation and should require heuristic solutions.

Selecting an expert system tool in not an easy task. There are many types of shells available on the market ranging from inexpensive, simplistic systems to more expensive, sophisticated tools. Some tools are very easy to learn and have very user-friendly interfaces. Others are harder to learn but have wider choices of inference strategies and knowledge representations.

"Just as varied problems require different reasoning processes by human experts, they also require different constructs of tools" [Fontana88]. There are different methods of representing knowledge; one application may be better suited for an object-oriented system whereas another better equipped for a rule-based representation. Assuming a rule-based system is selected, there are different inference mechanisms that can be chosen; some problems require forward reasoning or forward chaining, and other problems need backward reasoning or backward chaining as they work toward their goal. One should also consider the cost of the system, the rule or size limitations, speed, interfaces to other software, portability, documentation, training, company support, and most importantly user satisfaction [Gevarter87].

There are specific qualities of an expert system which are desired for a scheduling application. In William B. Gevarter's article "The Nature and Evaluation of Commercial Expert System Tools", some of the important attributes of a planning or scheduling expert system are listed. One attribute is that the tool use forward reasoning or use an an integrated approach combining both forward and backward reasoning. Gevarter also recommended that actions be described in the form of rules or procedures [Gevarter87].

The OPS5 and C5 systems possess these desired qualities. Both systems are forward-chaining, forward-reasoning, rule-based tools.

OPS5 programming language is a popular version of the rule-based language developed at Carnegie Mellon University. It was written by Charles Forgy in Lisp in the 1960's. C5, written by AT&T Bell Laboratories, is another language which is fully compatible with OPS5; however, C5 is written in C language and possesses some additional features and constructs that do not exist in OPS5.

C5 is well integrated with the UNIX operating system and provides much flexibility for the programmer developing a rule-based system on UNIX. The C5 interpreter provides direct access to UNIX system's functions. "The programmer can draw on the strengths of the rule-based methodology that OPS5 provides and the strengths of the procedural methodology that C and UNIX systems provide" [Vesonder88].

The C5 inference engine cycles over three states -- 1.) find matching rules, 2.) select a rule, and 3.) execute the selected rule [Brownston86]. This control cycle is referred to as the recognize-act cycle. The method of selecting a rule when multiple rules match the working memory depends on the recency of each conditional element and on the specificity of the left-hand side of the rule. First, if a rule was previously fired and elements in working memory for that rule were not changed, then the rule will not be fired again. Next, the inference engine orders rules, putting rules referencing the most recently modified working memory elements at the top of a list. If one rule does not dominate the list, then the dominant rules are ordered based on which rules are more specific (have the most conditions). Last, if a single rule still does not dominate, then a rule is selected randomly.

Many production systems spend more than nine tenths of their runtime performing matches [Forgy87]. C5 uses the Rete algorithm to improve its matching performance. The Rete algorithm alters the matching process described above and creates a more efficient process. The match algorithm exploits two properties of production systems -- 1.) working memory changes very little from run to run, and 2.) the left-hand sides of rules contain similar patterns. The algorithm saves information about matches and partial matches from cycle to cycle and updates the information with the changes. Also, common patterns are stored in a Rete network to eliminate duplication of terms and excess search time [Forgy87]. "The Rete algorithm is efficient because it does not match all elements on each cycle, it shares similar tests in different rules, and it recomputes whether or not combinations of matches are consistent only when necessary" [Brownston86]. The efficient Rete match process is a benefit of using OPS5 and C5.

With C5, the control of the system is built in with the Rete algorithm, and knowledge cannot control the order of the execution of rules. The resulting freedom from having to concern oneself with control is considered to be an advantage of expert systems. However, the design

of many expert systems requires that rules be pursued in a particular sequence [Erman84]. These designs are based on solutions obtained from experts who perform expert tasks in a particular order. Using C5, there are two ways to cause rules to fire in a preset sequence. One can order data or arrange conditions of rules to cause the inference engine to select rules in the desired sequence, or one can use control elements to gain some control over the order in which rules will fire.

4. REQUIREMENTS

In the factory, the warehouse is issued many units of work, called selects. A select is a group of parts which are required to build a product type in a shop. A select is uniquely identified by a select number and is specific to a product.

The warehouse attempts to have all components that comprise a select picked, sorted, and prepared in the warehouse prior to a shop requesting delivery of the select material. To prepare a select, the warehouse must pick parts from storage locations and may have to affix components on reels, program integrated circuits, or organize parts into kits. Therefore, due to this lead time, the warehouse tries to keep one step ahead of the shops.

Some shops build many selects in a week, and others build less. Most of the selects are created in the Material Requirements Planning (MRP) computer run over each weekend. The MRP programs use various lot-sizing rules to generate the selects. Therefore, some selects are issued in small quantities causing several selects for a product to be created in a week; others come in large

quantities with just one or two selects being issued per week for a product. The joint goal of the factory shops and warehouse is to complete work on all selects by the end of the week.

In this scheduling problem, there are many diverse objectives and preferences from different shops in the plant. These viewpoints often conflict. Most of the shops would like many of their selects to be top priority on the warehouse's schedule. Each shop feels that its needs are most important.

The warehouse supervisor has the tough task of creating a schedule that will make each of the customer shops satisfied. In order to understand all the needs of each shop, the warehouse supervisor requests each shop to generate a list of their products in priority sequence. If a product usually has many more selects than other products, he asks that the product be included in the list multiple times. The warehouse supervisor uses these priority lists to devise a method to create a schedule.

The devised scheduling process attempts to satisfy the goals of the shops and warehouse. First, the process provides each of the shops with a fair share of the

output. Second, while giving each shop its fair share, the process provides the warehouse with a relatively even workload for each day of the week. The warehouse supervisor identifies the selects belonging to a shop by comparing products on a shop's priority table with the products on the select. Then, a fair share is determined by counting the number of unscheduled selects for a shop and dividing the count by the number of remaining workdays in the week. This calculation is performed for each shop. All calculated schedule positions are summed together scheduling a fairly even number of selects per day for the warehouse to complete.

The scheduling process schedules one shop's selects at a time. To distribute the fair share for a shop, the selects are assigned to the schedule based on the location of a product in a shop's priority table. Starting with the top priority product and working down the table, a select having all parts stored in the warehouse is scheduled for each product on the list. Each select that is scheduled is marked to avoid scheduling the same select multiple times. Since a select that is missing parts cannot be built by a shop, the warehouse defers work on selects having component shortages. If a select cannot be scheduled in today's

schedule due to shortages, then the product is remembered and is given top priority once the shortage clears. Several passes of the shop's priority table may have to be completed until all non-short selects have been scheduled. Short selects are assigned to the schedule after all non-short select have been scheduled.

When enough selects have been scheduled to exhaust a shop's fair share for a day, then the supervisor begins scheduling the next day's work for the shop. Each time a new day's schedule is started, the supervisor starts the scheduling process at the top of the priority table. He also starts at the top of the priority list after completing the scheduling of non-short selects before continuing to scheduling short selects.

When all shops' selects have been scheduled, the supervisor looks for selects having products that were not assigned to a shop's priority list. These products and selects are written on a separate list. The list is later examined with shop supervisors to determine which shop needs to include the product on their priority table.

Once the schedule has been produced, the warehouse supervisor analyzes it. If he notices that an excessive amount of work has been scheduled for a specific shop, he warns the appropriate shop supervisor. He does this in cases where too little work is scheduled for a shop also. Last, the warehouse supervisor may change the schedule of certain selects based on special shop requests or recent deliveries or outages of material.

After a schedule day has passed, the supervisor proceeds through the same process scheduling work for the rest of the week. He uses updated select and priority information. The supervisor does not schedule selects that were scheduled for a previous day again. Prior commitments in previous schedules constrain subsequent scheduling for the week.

5. DESIGN

Seven data structures or element classes were used in the design and implementation of the C5 fair-share scheduling system. These include the shop, priority, gypped, select, slot, stats, and start structures. These element classes, their attributes, and some sample values are shown in figure 5-1. The element classes and how they are used in the scheduling process are described next.

Figure 5-1. Element Classes

SHOP:

shopname	priorityfile	gyppedfile	upperbound	lowerbound
shop-a	/u1/priora	/u1/gypa	10	20

PRIORITY:

product number	level	examine
11	100	yes
22	95	yes
33	90	no
44	85	no
55	80	no

GYPPED:

product number	gypped day
11	1
22	1

SELECT:

select number	product number	short ind	assigned day	tape sequence	program ics	kit parts	status	shop name
1212	11	no	1	no	no	no	part	shop-a
1313	66	no	init	yes	yes	no	init	init
1414	44	no	init	no	no	yes	init	shop-a
1515	55	no	init	no	yes	no	init	shop-a
1616	77	yes	init	yes	yes	no	init	init
1717	88	no	init	no	yes	no	init	init
1818	22	yes	init	yes	yes	yes	init	shop-a
1919	11	no	init	no	no	no	init	shop-a
2020	22	yes	init	yes	yes	yes	init	shop-a
2121	33	no	1	yes	yes	no	part	shop-a

SLOT:

day of week	unscheduled slots
1	3
2	3
3	3
4	3
5	3

STATS:

action	look for shortages	record count	today	work days	last workday	select count	change select number	change schedule	schedule file	current priority
schedule	no	0	2	4	5	15		no	/u1/sch	90

START:

Some element classes are initially loaded with data from external UNIX files whereas others are created during the scheduling process. The shop, priority, gypped, and select classes are primarily populated from files. The attributes in the slot and stats structures are filled with user responses and calculated results acquired at runtime. The start data structure is the first element class to be created but does not contain any attributes.

The shop element class contains information about each shop such as the name of the shop. Two of its attributes are names of external UNIX files containing a shop's priority and gypped data. These names are used to load data into the priority and gypped data structures. Two threshold attributes contain the upper bound and lower bound values for the number of selects that should be scheduled per day. These attributes are used to issue a warning message if the number of selects to be scheduled on a day for a shop does not lie within these limits.

Since one shop is scheduled at a time, only one instance of the shop element class is resident in working memory at a time. After the schedule is completed for a shop, the shop's data item for this element class is removed from working memory, and working memory is populated with

the next shop's information.

The priority data structure contains data describing the relative priority of each product built by a shop. The product number uniquely identifies a product. The priority level is a numeric value which assigns a relative priority to each product. Last, the priority examine attribute is an attribute used for control; it tells whether a product and its priority has been analyzed for scheduling possibilities. The priority examine attribute keeps track of the current scheduling position in the priority table.

The product number and priority level of each product are loaded from an external file specific to a shop. Therefore, one shop's priority list can be found in working memory at a time. The priority information for a shop can be voluminous and may not change very often; thus, each shop's priority information is stored in an external file.

The gypped data structure contains products for a shop which could not be scheduled on previous days due to shortages of components. The gypped product number describes a product, and the gypped day contains the day

a select for a product would have been scheduled if a non-short select (rather than a short select) could have been found. The gypped day is used to order the list such that those products that could not be scheduled earlier in the week are analyzed first.

Similar to the priority element class, the gypped element class is loaded from an external file for a shop resulting in one shop's gypped list being in working memory at a time. Instances of the gypped element class may be added or deleted from working memory during the schedule process. Instances are removed when non-short selects fulfill past scheduling priorities and are added when a select cannot be added to today's schedule due to shortages. Thus, the changing gypped list is stored in an external file allowing the output of gypped data from one day's schedule to be used as input to the following day's schedule.

The element class select contains several key attributes. The select number uniquely identifies the select, and the product number identifies the product that will be built from the parts in the select. The shortage indicator tells whether or not all components for a select are in stock. The assigned day stores the day the select was

scheduled; if the select has not been scheduled, it will contain the value "init". The shop name describes the shop to which a select is assigned; however, if the select has not been assigned to a shop, this attribute will contain the value "init" also.

The select element class also contains attributes which are not used to schedule. Attributes describing operations performed in the warehouse, such as tape sequence, program ics, and kitting, are carried in this structure. The status attribute describes the amount of work completed on the select thus far. Although these attributes are not used in the scheduling process, these attributes are printed on the final schedule to notify the warehouse of the operations and the amount of work remaining to be performed.

The select element class is loaded from an external file. Unlike the priority and gypped element classes where only data for a specific shop occupy working memory at a time, all selects for all shops are read into working memory at the beginning of the run. This must be done because the selects must be assigned to a shop. One of the tasks of this scheduling expert system is to determine which selects belong to which shops.

The slot element class contains the days of the week remaining to be scheduled and the number of available scheduling positions or slots open to a shop each day of the week. It is initially populated by using the select and priority element classes to calculate the shop's fair share. This calculation will be discussed later.

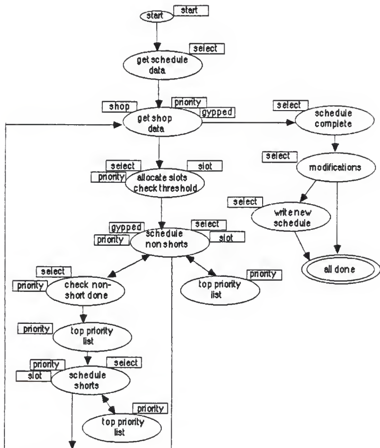
Two control values and many other values are stored in the stats element class. The action attribute designates the state of the scheduling system-- whether it is loading data from files into working memory or assigning selects to the schedule. The look-for-shortage attribute is another control variable denoting when to schedule non-short versus short selects. The user is prompted for values for the today and workdays attributes; they contain the current day of the week and remaining number of days to work respectively. The last-workday attribute holds the calculated value describing the last workday of the week. The select-count field stores the number of selects that must be scheduled for a shop for the rest of the week. The record-count variable is incremented as records are read from external files and is used for record-keeping purposes. The priority level that is currently being analyzed is stored in a current-priority attribute. The change-select-number and change-schedule

attributes are populated after the entire warehouse schedule has been produced and if the user chooses to alter assigned dates of selects; they contain the select number to be rescheduled and a flag denoting if any schedule changes were made respectively. Last, the schedule-file attribute contains the UNIX file name to which the schedule will be written.

The start data structure is used to get the system to begin. It does not contain any attributes and is resident in the system for a brief time.

A state diagram describes the main control used in this scheduling system. Please see figure 5-2. The states are designated with ovals and the element classes required in each state are found in rectangles. The stats element class was omitted from the diagram to avoid clutter since it is used throughout the process. These control points and the processing that may occur between them will be discussed.

Figure 5-2. State Diagram



First the system is started. The first action creates a start element class in working memory. A rule checks for the existence of the start element class, finds it, and removes the start structure. This rule changes the action attribute in the stats class to "get schedule data".

Next, the data required for the schedule is loaded from a file or requested from the user in "get schedule data". This includes reading in the select data and requesting an output file name for the schedule, the current day, and the number of remaining work days from the user. The select data is ordered in working memory based on the sequence of data in the external select file. In order for the scheduler to analyze the non-short selects first, the external file must be sorted with the non-short selects at the top of the file. After selects are identified to a shop, the non-short selects will be the most recent; the inference engine will analyze them first.

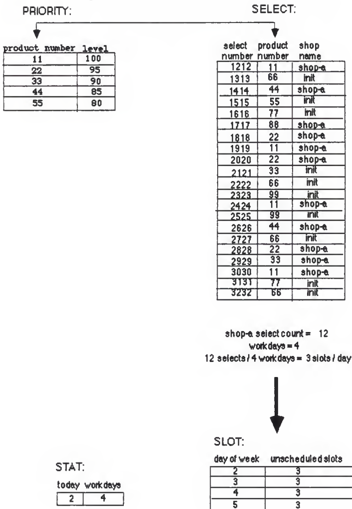
Still referencing figure 5-2, the control now changes to the "get shop data" state. After populating the shop element class with the first instance of a shop, the data relevant to a shop is loaded into the priority and gypped

element classes. The priority data is loaded into working memory prior to loading the gypped information causing members of the gypped element class to be more recent. Therefore, the products in the gypped element class will be considered for scheduling before products in the priority element class. Also, the gypped elements are loaded such that those with earlier gypped days (the day a product could not be scheduled) are loaded last. This causes those products that have been "gypped" the longest to be analyzed for scheduling soonest.

After the shop data has been loaded, the number of available slots for a shop are allocated. This is the fourth circle in figure 5-2. The process is refined further in figure 5-3. First, all unscheduled selects for the current shop are counted. To determine the selects belonging to a shop, a rule finds all occurrences of selects with products having priorities in working memory. When a shop is counted, the shop identifier is populated in the shop name attribute of the select element class, and the select count is incremented. This number is divided by the remaining number of workdays in the week. If the division leaves a remainder, the remainder is distributed over the beginning days of the week. If the number of slots for a

day is not within the bounds of the shop's upper and lower thresholds, then a warning message is printed.

Figure 5-3. Allocating Slots for a Shop



PRIORITY:

product number	level
11	100
22	95
33	90
44	85
55	80

SELECT:

select number	product number	shop name
1212	11	shop-e
1313	66	init
1414	44	shop-e
1515	55	init
1616	77	init
1717	88	shop-e
1818	22	shop-e
1919	11	shop-e
2020	22	shop-e
2121	33	init
2222	66	init
2323	99	init
2424	11	shop-e
2525	99	init
2626	44	shop-e
2727	66	init
2828	22	shop-e
2929	33	shop-e
3030	11	shop-e
3131	77	init
3232	66	init

STAT:

today	workdays
2	4

SLOT:

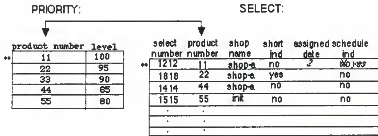
day of week	unscheduled slots
2	3
3	3
4	3
5	3

In order to get C5 to start scheduling with the first day of the week, the instances of the slot element class are created starting with the end of the week and finishing at the beginning of the week. C5's inferencing engine will start with the most recent working memory element (the beginning day of the week) and continue until the least recent slot element is scheduled (the end of the week).

Continuing with figure 5-2, the "schedule non-shorts" state involves the gypped, priority, select, and slot element classes. First, gypped products are assigned to the schedule. In order to schedule a gypped product, several conditions must exist--the day being scheduled must be today, today's schedule must have available slots, and a non-short unscheduled select must be found with the same product number. If these conditions exist, then the instance of the gypped element class is removed from working memory and the assigned date for the select is changed to today. Also, the number of available slots is reduced by one. If the conditions cannot be found, then the gypped product is written out to tomorrow's gypped file and removed from working memory.

Once all gypped elements have been removed from working memory, then scheduling based on the priority list begins. This is illustrated in figure 5-4 with the changes that occur in italics. Starting with the priority entry having the highest priority level, a rule searches for an unscheduled select having the same product number. If a match is found and the select is not short, then the select is assigned a schedule date and a slot is reduced on the day being scheduled. Next, please see figure 5-5. If a select is found but the select is short and the day being scheduled is today, then the product and the current day are added to the gypped list. If neither of these conditions exist, then the next entry in the priority table is examined.

Figure 5-4. Scheduling Non-Short Selects



PRIORITY:

product number	level
11	100
22	95
33	90
44	85
55	80

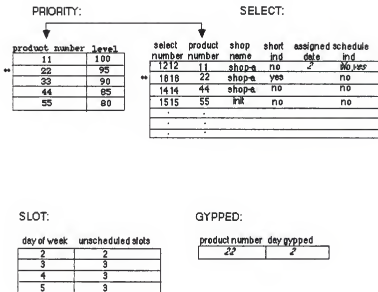
SELECT:

select number	product number	shop name	short ind	assigned date	schedule ind
1212	11	shop-a	no	2	~~no~~ yes
1818	22	shop-a	yes		no
1414	44	shop-a	no		no
1515	55	init	no		no
.	.				
.	.				
.	.				

SLOT:

day of week	unscheduled slots
2	~~3~~ 2
3	3
4	3
5	3

Figure 5-5. Adding a Product to Gypped List



PRIORITY:

	product number	level
	11	100
**	22	95
	33	90
	44	85
	55	80

SELECT:

	select number	product number	shop name	short ind	assigned date	schedule ind
	1212	11	shop-a	no	2	~~no~~ yes
**	1818	22	shop-a	yes		no
	1414	44	shop-a	no		no
	1515	55	init	no		no
	.	.				
	.	.				
	.	.				

SLOT:

day of week	unscheduled slots
2	2
3	3
4	3
5	3

GYPPED:

product number	day gypped
22	2

As each priority entry is analyzed, the priority examine attribute is changed from the value "no" to "yes". During this process, the scheduler must go to the top of the priority table when a shop's schedule is completed for a day or when all priorities are examined. This is done with the "top priority list" state. Here all priority examine attributes are reset to "no" meaning the priority can be reexamined.

During the scheduling of non-short selects, the process must check if all non-shorts selects have been scheduled. This is done in the "check non-short done" state. Working memory is searched for any unscheduled, non-short selects having products in the priority table. If none can be found, then the next state takes the process to the top of the priority table and to a new state of "schedule shorts". The process to schedule short selects is identical to scheduling non-short selects accept short selects are solely considered and products are not added to the gypped list.

When all selects for the product have been scheduled, the system returns to get data for the next shop. The completion of a shop's schedule is detected when a slot instance having unscheduled slots greater than zero

cannot be found. All instances of the gypped, priority, and slot element classes are removed from working memory prior to loading data for the next shop.

If no more shops exist to be scheduled, the system goes to the "schedule complete" state on the right-hand side of figure 5-2. Here, selects not assigned to a shop are identified. These selects are written to the schedule with a shop name of "no shop" and given an assigned day of "*". Next, an external C routine is called to sort and print the schedule to a file. The file is sorted by assigned date causing all selects scheduled on the same day to be grouped together.

Next, the "modification" state is entered where the supervisor can alter scheduled dates for selects. The user is prompted for the selects to be modified and the revised assigned days. If invalid selects or days are entered, the user is prompted again. The user is not allowed to alter schedules for selects having assigned days less than today.

If modifications are made, the external C routine is called again to resort and print the schedule. The final state "all done" prints a message that the schedule is

complete. At this point, the conflict set is empty and processing ceases.

6. CONCLUSION AND EXTENSIONS

Heuristics and expert systems can be used to create a good schedule. There are experts who perform the scheduling process well. By mimicing their actions, an expert system can do the same scheduling job equally well.

A combination of instruction-driven and data-driven techniques can provide a good solution to the scheduling problem. In the fair-share scheduling system, control attributes are stored in working memory and used to move the system from state to state. On the other hand, many data- sensitive, unordered rules are also incorporated. These rules do not rely on control attributes. Between these extremes, the workings of the inference engine can be exploited to get rules to fire in the desired sequence. There are cases in the fair-share scheduler where data is loaded into working memory in a specific sequence to get rules to fire in a desired order.

The fair share scheduler also uses a traditional sort routine in the rule-based system. Although the sort could have been performed with rules, it is more advantageous to write an external, conventional function.

"The escape mechanism allows production systems to be shorter, more efficient, and more comprehensible" [Brownston].

Further work on the fair-share scheduler is recommended in two areas. First, a friendly user interface screen would enhance the system when the user must enter data; also, rather than dumping the output to a printer or file, a screen to display the resulting schedule is desired. Last, the fair-share scheduler regenerates the schedule each day it is produced; the scheduling process could be enhanced to apply only changes in select and priority information to the schedule each day.

7. BIBLIOGRAPHY

[Brownston85] Brownston, L., Farell, R., Kant, E., and Martin, N., Programming Expert Systems in OPS5, Addison-Wesley, Reading, Massachusettes, 1985.

[Bryant88] Bryant, N., Managing Expert Systems, John Wiley and Sons, New York, 1988.

[Forgy87] Forgy, C.L., and Shepard, S.J., "Rete: A Fast Match Algorithm", AI Expert, January, 1987, pp. 34-40.

[Gevarter87] Gevarter, W. B., "The Nature and Evaluation of Commercial Expert System Building Tools", Computer , May 1987.

[Gross87] Gross, D., "'C' as in Commercial", Computerworld, Vol.21, Isuue N47, Nov 23, 1987.

[Hayes-Roth83] Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., Building Expert Systems, Addison-Wesley, Reading, 1983.

[Kim88] Kim, S.O., "Heuristic Framework for the Resource Constrained Multi-Project Scheduling Problem", Department of Management, College of Business Administration, Kansas State University, 1988.

[Lubben88] Lubben, R. T., Just-in-Time Manufacturing: An Aggressive Manufacturing Strategy", McGraw Hill Book Company, New York, 1988.

[Smith85] Smith S. F, and Fox, M. S., "Constructing and Maintaining Detailed Production Plans, Investigations into the Development for Knowledge-Based Factory Scheduling Systems", Proceedings from 9th International Joint Conference on Artificial Intelligence, Los Angeles, CA, August, 1985.

[Vesonder88] Vesonder, G. T., "Rule-Based Programming in the UNIX (R) System", AT&T Technical Journal, January/February 1988, Volume 67 . Issue 1

[Waterman86] Waterman, D. A., A Guide to Expert Systems, Addison-Wesley, Reading, Massachusettes, 1986.

[Weist67] Weist, J. D. , "A Heuristic Model for Scheduling Large Projects with Limited Resources", Management Science, Vol. 13, No. 6, February 1967.

[Yusko86] Yusko, J. A., "Expert Systems: The Leading Edge", Proceedings of the National Communications Formum, Rosemont, Illinois, October 29, 1986.

8. APPENDIX A - C5 RULES

```
; This C-5 program creates a work schedule for a warehouse.
; The warehouse must supply selects, sorted groups of
; parts, to several shops on the  factory floor.  The
; schedule tries to satisfy all shops wishes and also
; create a reasonable schedule for the warehouse. The main
; goals of the schedule are as follows:
;    1.  Provide each shop with its fair-share of selects.
;    2.  Provide the warehouse with a level-loaded weekly
;         schedule.
;    3.  Defer work on selects which have component
;         shortages until the end of the week.
;    4.  If a select could not be scheduled on a previous
;         day due to shortages, give the select top
;         priority when the shortage clears. (referred to
;         as gypped selects)
;    5.  Try to schedule a shops high priority selects
;         before its low priority selects.
;
; A file of selects for the week is read into memory
; first.  ; The user is prompted to enter the name of the
; file to hold the schedule and to enter calendar
; information.
;
; Then an instance of the shop file is read into working
; memory.  This shop element class points to the shop's
; gypped file and priority file which are read into memory.
; Next, the slots for the shop are calculated. Then, the
; gypped, non-short, and short selects are scheduled. This
; continues until all shops are scheduled.
;
; Last, all shop schedules are merged and sorted for
; display.  ; The schedule for some selects can then be
; altered prior to having a final schedule.

; This command allows one to undo rules if desired

(back on)

; This command reports the rule and time tags of each
; working element for each instantiation that is fired.

(watch 1)
```

```
; At beginning of schedule process, Select will contain
;  selects for the shops

(literalize Select              ; element class for selects
        shop_name               ; populated as schedule with
                                ;  shop name
        select_number           ; select identifier
        sl_product_number       ; product identifier
        tape_sequence           ; 'yes' if tape sequence
                                ;   involved ,otherwise 'no'
        program_ics             ; 'yes' if programming of ics
                                ;   involved , otherwise 'no'
        kitting                 ; 'yes' if kitting of parts
                                ;   involved , otherwise 'no'
        assigned_day            ; the day select is scheduled
                                ;   to be picked
        shortage_ind            ; 'yes' if select has
                                ;   shortage(s) ;otherwise 'no'
        status                  ; the work done on a select
        schedule_ind            ; flagged 'yes' if scheduled,
                                ;   'no' if not scheduled
)

; Will be refreshed with each new shop that is scheduled.

(literalize Gypped              ; element class containing
                                ;  products that were to be
                                ;  scheduled on past days but
                                ;  could not due to shortages.
        gp_product_number       ; product identifier of gypped
                                ;  product
        gyp_day                 ; day part was "gypped"
)

(literalize Priority            ; element class for shop's
                                ;  priorities
        pr_product_number       ; product identifier
        priority_level          ; integer relative priority of
                                ;  product
        priority_examined       ; flagged 'yes' after priority
                                ;  reviewed, otherwise 'no
                                ;  set to 'no' when start top
                                ;  of table
)


; The following three elements will be intialized or
;  repopulated before each shop is scheduled.
```

```
(literalize Slot                  ; element class with week's
                                  ;  available work
        day_of_week               ; day of week to be scheduled
        unscheduled_slots         ; number of slots to be
                                  ;  scheduled on day of week
)

(literalize Stats                 ; element class for stats on
                                  ;  current schedule
        action                    ; the next state
        reccnt                    ; counts number of records read
        today                     ; the current day of the week
        work_days                 ; number of work days remaining
        last_workdays             ; last work day of week
        look_for_shortages        ; 'yes' if ready to schedule
                                  ;   short selects
        curr_priority             ; the priority being analyzed
        select_count              ; the number of select
                                  ;  remaining to be scheduled
        chg_select_number         ; the select number to have a
                                  ;   a schedule day altered
        chg_schedule_done         ; flags if schedule changes
                                  ;  were made and if schedule
                                  ;  must be resorted ,once
                                  ;  resorted it is changed to
                                  ;  value complete
        sched_file                ; the name of the output
                                  ;   schedule file
)

(literalize Shop                  ; element class for shop
                                  ;  information
        shop_name                 ; shop name
        shop_priority_file        ; filename for shop's priority
                                  ;  file
        shop_gypped_file          ; filename for shop's gypped
                                  ;  file
        shop_lower_bound          ; lower bound of number of
                                  ;  selects/day
        shop_upper_bound          ; upper bound of number of
                                  ;  selects/day
)


(literalize Start                 ; element class to initialize
)
```

```
; Start the production system.  Set the action attribute
;  of Stats to the first state and initialize the record
;  counter to zero.

(p start_schedule
        {<evstart> (Start)}
-->
        (remove <evstart>)
        (write (crlf) |**SELECT SCHEDULER**| (crlf))
        (make Stats  ^action get_schedule_files
                     ^reccnt 0)
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  Get schedule data
;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Open files for schedule production

(p open_schedule_level_files
        {<evstats>(Stats ^action get_schedule_files)}
-->
        (write (crlf)(crlf) |Enter output file name|
            | for schedule : |)
        (bind <sched_file> (accept))
        ; file to contain all shops' schedules
        (openfile sched_out <sched_file> out)
        ; file to display scheduler run statistics
        (openfile run_out /ul/sms/o.run append)
        ; file to containing all selects for the factory
        (openfile select_in /ul/sms/d.select in)
        ; file containing shop information
        (openfile shop_in /ul/sms/d.shop in)
        (modify <evstats> ^action get_schedule_user_input
                          ^sched_file <sched_file>
                          ^chg_select_number none)
        (write run_out |** SELECT SCHEDULER **|
                          (crlf)(crlf))
)

; Ask user for scheduling information

(p get_schedule_run_input
        {<evstats>(Stats ^action get_schedule_user_input)}
-->
        (bind <response> wrong)
        ; verify proper responses to questions
        (while (<response> == wrong)
```

```
            (write (crlf)|1=Mon,2=Tues,3=Wed,4=Thu,|
                 |5=Fri,6=Sat,7=Sun|)
            (write (crlf)(crlf) |Enter number for today: |)
            (bind <vtoday> (accept))
            (case <vtoday>
                 ((1 2 3 4 5 6 7)
                        (bind <response> right))
                 (otherwise
                        (write (crlf) |Invalid response, |
                               |Try again.|))
            )
         )
         (bind <response> wrong)
         (while (<response> == wrong)
            (write (crlf)(crlf) |Including today, enter|
                 | number of remaining work days:|)
            (bind <vworkdays> (accept))
            (bind <total_amount>
               (compute (<vworkdays> + <vtoday> - 1)))
            (case <total_amount>
                 ((1 2 3 4 5 6 7)
                        (bind <response> right))
                 (otherwise
                        (write (crlf) |Invalid response,|
                           | Try again.|))
            )
         )
            (modify <evstats> ^action read_select_input
                              ^work_days <vworkdays>
                              ^today <vtoday>
                              ^last_workday <total_amount>)
)


; Read first record of select input into working memory

(p read_first_select
        {<evstats> (Stats ^action read_select_input
                          ^reccnt <vreccnt>)}
-->
        (make Select (acceptline select_in end_of_file))
        (modify <evstats> ^action continue_select_input
                      ^reccnt (compute (<vreccnt> + 1)))
)

; Continue reading select input into working memory

(p read_select_input
```

```
        {<evstats> (Stats ^action continue_select_input
                          ^reccnt <vreccnt>)}
        (Select ^shop_name {<> end_of_file})
-->
        (make Select (acceptline select_in end_of_file))
        (modify <evstats> ^reccnt
                        (compute (<vreccnt> + 1)))
)

; Stop reading select input once end of file is reached

(p read_select_input_done
        {<evstats> (Stats ^action continue_select_input
                          ^reccnt <vreccnt>)}
        {<evselect>(Select ^shop_name {= end_of_file})}
-->
        (write run_out | Number of selects : |
                       <vreccnt>(crlf))
        (remove <evselect>)
        (modify <evstats> ^action read_shop_file
                          ^reccnt 0)
        (closefile select_in)
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Get shop data
;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Read file containing shop information.  When hit end of
;  file, then done with scheduling process.  Initialize
;  temporary variables for shop to be scheduled.

(p read_shop_input
       {<evstats> (Stats ^action read_shop_file )}
-->
       (make Shop (acceptline shop_in end_of_file))
       (modify <evstats> Stats ^look_for_shortages no
                               ^curr_priority nil
)                              ^action get_shop_input )

; Open files specific to a shop

(p open_shop_files

       {<evstats>(Stats ^action get_shop_input)}
       (Shop ^shop_priority_file <vshop_priority_file>
```

```
              ^shop_gypped_file <vshop_gypped_file>
              ^shop_name {<vshop_name> <> end_of_file})
-->
        (openfile priority_in <vshop_priority_file> in)
        (openfile gypped_in <vshop_gypped_file> in)
        (write run_out |****************| (crlf) (crlf))
        (write run_out |Priority for | <vshop_name>
           |read from: | <vshop_priority_file> (crlf))
        (write run_out | Gypped read from : |
                        <vshop_gypped_file> (crlf))
        (modify <evstats> ^action read_gypped_input)
)


; Read first record of gypped input into working memory

(p read_first_gypped
        {<evstats> (Stats ^action read_gypped_input
                          ^reccnt <vreccnt>)}
-->
           (make Gypped
                 (acceptline gypped_in end_of_file))
           (modify <evstats> ^action continue_gypped_input
                   ^reccnt (compute (<vreccnt> + 1)))
)

; Continue reading gypped input into working memory

(p read_gypped_input
        {<evstats> (Stats ^action continue_gypped_input
                          ^reccnt <vreccnt>)}
        (Gypped ^gp_product_number {<> end_of_file})
-->
        (make Gypped (acceptline gypped_in end_of_file))
        (modify <evstats> ^reccnt
                        (compute (<vreccnt> + 1)))
)

; Stop reading gypped input once end of file is reached

(p read_gypped_input_done
        {<evstats> (Stats ^action continue_gypped_input
                          ^reccnt <vreccnt>)}
        {<evgypped>(Gypped ^gp_product_number
                          {= end_of_file})}
        (Shop ^shop_gypped_file <vshop_gypped_file>)
-->
        (write run_out | Number of gypped parts : |
```

```
                        <vreccnt> (crlf))
        (remove <evgypped>)
        (modify <evstats> ^action read_new_priority
                          ^reccnt 0)
        (closefile gypped_in)
        (openfile gypped_out <vshop_gypped_file> out)
)


; Read first record of priority input into working memory

(p read_first_priority
        {<evstats> (Stats ^action read_new_priority
                          ^reccnt <vreccnt>)}
-->
        (make Priority
                (acceptline priority_in end_of_file))
        if (<evstats> ^action continue_priority_input
                ^reccnt (compute (<vreccnt> + 1)))
)


; Continue reading priority input into working memory

(p read_priority_input
        {<evstats> (Stats ^action continue_priority_input
                          ^reccnt <vreccnt>)}
        (Priority ^pr_product_number {<> end_of_file})
-->
        (make Priority
                (acceptline priority_in end_of_file))
        (modify <evstats> ^reccnt
                (compute (<vreccnt> + 1)))
)


; Stop reading priority input once end of file is reached
;   also initialize the select count to zero

(p read_priority_input_done
        {<evstats> (Stats ^action continue_priority_input
                          ^reccnt <vreccnt>)}
        {<evpriority>(Priority ^pr_product_number
                               {= end_of_file})}
-->
        (write run_out | Number of priority records : |
                        <vreccnt> (crlf))
        (remove <evpriority>)
```

```
        (modify <evstats> ^action determine_slots
                          ^reccnt 0
                          ^select_count 0)
        (closefile priority_in)
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Allocate slots
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Count the selects that need to be scheduled for
; a shop that have not been previously scheduled.  Change
; assigned day to none and populate the shop name after
; count

(p count_shops_selects
        {<evstats>(Stats ^action determine_slots
                         ^select_count <vselect_count>)}
        {<evselect> (Select ^sl_product_number
                                <vsl_product_number>
                            ^assigned_day  init)}
        (Priority ^pr_product_number <vsl_product_number>)
        (Shop ^shop_name <vshop_name>)
-->
        (modify <evstats> ^select_count
                          (compute (<vselect_count> + 1)))
        (modify <evselect> ^assigned_day none
                           ^shop_name <vshop_name>)
)


; Calculate the number of slots per day
; If the number of slots per day does not divide evenly,
;  extra slots are added to the beginning of the week.
; This rule will fire after all the selects have been
;   counted in the previous rule

(p calculate_slots_per_day
        {<evstats> (Stats ^action determine_slots
                          ^today <vtoday>
                          ^select_count <vselect_count>
                          ^work_days <vworkdays>)}
-->
        (bind <vwork_remain> <vworkdays>)
        (bind <vwhole_number>
              (compute (<vselect_count> // <vworkdays>)))
        (bind <vmod_number>
```

```
                (compute (<vselect_count> \ <vworkdays>)))
        (while ( <vwork_remain> <> 0)
                (if (<vwork_remain> >  <vmod_number>)
                   (bind <new_slots> <vwhole_number>)
                else
                   (bind <new_slots>
                        (compute (<vwhole_number> + 1)))
                )
                (make Slot ^day_of_week (compute
                        (<vwork_remain> + <vtoday> - 1))
                          ^unscheduled_slots <new_slots>)
                (bind <vwork_remain>
                        (compute (<vwork_remain> - 1)))
        )
        (modify <evstats> ^action check_lower_threshold)
)

; Check if slots within lower bound of the number of
;  selects/day for a shop

(p check_lower_threshold
        {<evstats> (Stats ^action check_lower_threshold)}
        (Shop ^shop_lower_bound <vshop_lower_bound>
              ^shop_name <vshop_name>)
        (Slot ^unscheduled_slots  <vunscheduled_slots>)
-->
        (if (<vunscheduled_slots> < <vshop_lower_bound>)
             (write (crlf) <vunscheduled_slots>
                           | selects for shop | <vshop_name>
                           | are below the threshold of |
                           <vshop_lower_bound>)
             (write run_out <vunscheduled_slots>
                           | selects for shop | <vshop_name>
                           | are below the threshold of |
                           <vshop_lower_bound> (crlf))
        )
        (modify <evstats> ^action check_upper_threshold)
)


; Check if slots within upper bound of the number of
;  selects/day for a shop

(p check_upper_threshold
        {<evstats> (Stats ^action check_upper_threshold)}
        (Shop ^shop_upper_bound <vshop_upper_bound>
              ^shop_name <vshop_name>)
        (Slot ^unscheduled_slots  <vunscheduled_slots>)
-->
```

```
        (if (<vunscheduled_slots> > <vshop_upper_bound>)
               (write (crlf) <vunscheduled_slots>
                      | selects for shop | <vshop_name>
                      | exceed the threshold of |
                      <vshop_upper_bound>)
               (write run_out <vunscheduled_slots>
                      | selects for shop | <vshop_name>
                      | exceed the threshold of |
                      <vshop_upper_bound> (crlf))
        )
        (modify <evstats> ^action schedule)
)


; The next group of rules affect priority handling

; Find the highest priority that was entered

(p highest_priority
        (Priority ^priority_level <vpriority_level>
                  ^priority_examined no)
        - (Priority ^priority_level > <vpriority_level>
                    ^priority_examined no)
        {<evstats> (Stats ^action schedule
                   ^curr_priority { <> <vpriority_level>})}
-->
        (modify <evstats> ^curr_priority <vpriority_level>)
)

; Delete priority record if no more selects exist for it

(p remove_priority
        (Shop ^shop_name <vshop_name>)
        (Stats ^curr_priority {<vcurr_priority> <> nil})
        (Priority ^pr_product_number <vpr_product_number>
                  ^priority_level <vcurr_priority>)
        -(Select ^sl_product_number <vpr_product_number>
                 ^shop_name <vshop_name> ^schedule_ind no)
-->
        (remove-pattern Priority ^pr_product_number
                                 <vpr_product_number>)
)

; Mark a priority instance if no selects exist having the
;    same shortage indicator.  If looking for shortages
;    when marking and no selects exist, remove priority
```

```
;   from working memory.

(p mark_priority
        (Shop ^shop_name <vshop_name>)
        (Stats ^curr_priority <vcurr_priority>
               ^action schedule
               ^look_for_shortages <vlook_for_shortages> )
        {<evpriority>(Priority ^pr_product_number
                                    <vpr_product_number>
                          ^priority_level <vcurr_priority>
                          ^priority_examined no)}
        - (Select ^sl_product_number <vpr_product_number>
                  ^shop_name <vshop_name> ^schedule_ind no
                  ^shortage_ind <vlook_for_shortages>)
-->
        (if (<vlook_for_shortages> == yes)
                (remove <evpriority>)
        else
                (modify <evpriority>
                        ^priority_examined yes)
        )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Top priority list
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; if all priorities have been examined, reset the examine
;  indicator to no on all priorities

(p reset_priority_examined
        {<evstats> (Stats ^action schedule )}
        -(Priority ^priority_examined no)
-->
        (modify <evstats> ^action reset_examine)
)


;  reset priority examine to no

(p set_examine_no
        (Stats ^action reset_examine)
        {<evpriority> (Priority ^priority_examined yes)}
-->
        (modify <evpriority> ^priority_examined no)
)


; if all priority examines have been set to no, continue
```

```
;   scheduling process

(p all_examine_reset
        {<evstats> (Stats ^action reset_examine)}
        -(Priority ^priority_examined yes)
-->
        (modify <evstats> ^action schedule)
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Schedule non-shorts
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; The next group of rules are for product that should have
;  been scheduled today or on a previous day but could not
;  due to component shortages.


; If the current day being scheduled is equal to today and
;    today's schedule has unscheduled slots and
;    a gypped part exists which has a non-short select
; then
;    schedule the select, remove gyp record, reduce slots

(p gypped_product

        (Gypped ^gp_product_number <vgp_product_number>)
        (Stats ^today  <vtoday> ^action schedule)
        {<evslot> (Slot ^unscheduled_slots
                          {<vunscheduled_slots> > 0}
                        ^day_of_week <vtoday>)}
        {<evselect> (Select ^sl_product_number
                              <vgp_product_number>
                         ^shortage_ind no
                         ^select_number <vselect_number>
                         ^tape_sequence <vtape_sequence>
                         ^program_ics <vprogram_ics>
                         ^kitting <vkitting>
                         ^status <vstatus>
                         ^schedule_ind  no)}
        (Shop ^shop_name <vshop_name>)
-->
        (write (crlf) |Scheduled select| <vselect_number>
                      |of product | <vgp_product_number>
                      |on day: | <vtoday> )
        (write run_out |Scheduled select |
                      |of product | <vgp_product_number>
                       <vselect_number>
```

```
                        |on day: | <vtoday> (crlf))
        (write sched_out <vselect_number>
                         <vgp_product_number>
                         <vtape_sequence> <vprogram_ics>
                         <vkitting> <vtoday> |no|
                         <vstatus> <vshop_name> (crlf))
        (modify <evselect> ^schedule_ind yes
                           ^assigned_day <vtoday>)
        (remove-pattern Gypped ^gp_product_number
                               <vgp_product_number>)
        (modify <evslot> ^unscheduled_slots
           (compute (<vunscheduled_slots> - 1)))
        ; remove slot element if exhausted
        (if (<vunscheduled_slots> == 1)
                (remove-pattern Slot
                               ^day_of_week <vtoday>))
)

; if a non-short select cannot be found for a gypped
;   product, rewrite the gypped element to the gypped
;   file and remove from working memory

(p still_gypped
        {<evgypped> (Gypped ^gp_product_number
                            <vgp_product_number>
                           ^gyp_day <vgyp_day>)}
        (Stats ^action schedule)
        -(Select ^sl_product_number <vgp_product_number>
                 ^shortage_ind no
                 ^schedule_ind no)
-->
        (write gypped_out <vgp_product_number> <vgyp_day>
               (crlf))
        (remove <evgypped>)
)

; Try to schedule project with current priority and no
;   shortages. If cannot schedule a product today due to
;   shortages, add product to gypped list
; If scheduling today's work and
;   find part with priority equal to current priority and
;   that part has a select
;   the day being schedule has open slots
; then
;   if select is not short
;      schedule select,  reduce priority and slots
;   else
```

```
;       check if non-short scheduling is complete
;       if scheduling work for today
;           add product to gypped list

(p sched_non_short

        {<evstats>(Stats ^curr_priority <vcurr_priority>
                         ^action schedule
                         ^look_for_shortages no
                         ^today <vtoday>)}
        {<evpriority>(Priority
                         ^priority_level <vcurr_priority>
                         ^priority_examined no
                         ^pr_product_number
                         <vpr_product_number>)}
        {<evselect> (Select ^sl_product_number
                              <vpr_product_number>
                           ^shop_name <vshop_name>
                        ^select_number <vselect_number>
                           ^shortage_ind <vshortage_ind>
                        ^tape_sequence <vtape_sequence>
                           ^program_ics <vprogram_ics>
                           ^kitting <vkitting>
                           ^status <vstatus>
                           ^schedule_ind no)}
        {<evslot>(Slot  ^unscheduled_slots
                          {<vunscheduled_slots> > 0 }
                       ^day_of_week <vday_of_week> )}
-->
        (modify <evpriority> ^priority_examined yes)
        (if (<vshortage_ind> == no)
           (write (crlf) |Scheduled select|
                               <vselect_number>
                       |of product | <vpr_product_number>
                          |on day: | <vday_of_week> )
           (write run_out |Scheduled select|
                          <vselect_number>
                       |of product | <vpr_product_number>
                          |on day: | <vday_of_week> (crlf))
           (write sched_out <vselect_number>
                            <vpr_product_number>
                            <vtape_sequence>
                            <vprogram_ics>
                            <vkitting> <vday_of_week>
                            <vshortage_ind>
                            <vstatus>
                            <vshop_name>  (crlf))
           (modify <evselect> ^schedule_ind yes
                              ^assigned_day <vday_of_week>)
           (modify <evslot> ^unscheduled_slots
```

```
            (compute (<vunscheduled_slots> - 1)))
          ;  go to top of priority list if day complete
           (if (<vunscheduled_slots> == 1)
               (modify <evstats> ^action reset_examine)
               (remove-pattern Slot
                            ^day_of_week <vtoday>))
      else
              (modify <evstats> ^action check_short)
               (if (<vtoday> == <vday_of_week>)
                       (write crlf |Product was gypped|
                                 <vpr_product_number>)
                       (write run_out |Product|
                                 <vpr_product_number>
                                 |gypped on|
                                 <vday_of_week> (crlf))
                       (write gypped_out
                                <vpr_product_number>
                                 <vday_of_week> (crlf))
              )
      )
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Schedule shorts
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Try to schedule project with current priority and
;   shortages
;

(p sched_short

       {<evstats>(Stats ^curr_priority
                              <vcurr_priority>
                        ^look_for_shortages yes
                        ^action schedule)}
       {<evpriority>(Priority
                       ^priority_level <vcurr_priority>
                       ^priority_examined no
                       ^pr_product_number
                       <vpr_product_number>)}
       {<evselect> (Select ^sl_product_number
                             <vpr_product_number>
                        ^shop_name <vshop_name>
                        ^select_number <vselect_number>
                        ^shortage_ind yes
                        ^tape_sequence <vtape_sequence>
                        ^program_ics <vprogram_ics>
                        ^kitting <vkitting>
                        ^status <vstatus>
```

```
                              ^schedule_ind no)}
        {<evslot> (Slot ^unscheduled_slots
                              {<vunscheduled_slots> > 0}
                          ^day_of_week <vday_of_week>)}
 -->
           (write (crlf) |Scheduled select|
                          <vselect_number>
                        |of product | <vpr_product_number>
                           |on day: | <vday_of_week> )
           (write run_out |Scheduled select|
                          <vselect_number>
                        |of product | <vpr_product_number>
                           |on day: | <vday_of_week> (crlf))
           (write sched_out <vselect_number>
                            <vpr_product_number>
                            <vtape_sequence> <vprogram_ics>
                            <vkitting> <vday_of_week> |yes|
                            <vstatus> <vshop_name> (crlf))
           (modify <evselect> ^schedule_ind yes
                              ^assigned_day <vday_of_week>)
           (modify <evpriority> ^priority_examined yes)
           (modify <evslot> ^unscheduled_slots
                   (compute (<vunscheduled_slots> - 1)))
           ;  to to top of priority list if day complete
            (if (<vunscheduled_slots> == 1)
                (modify <evstats> ^action reset_examine)
                (remove-pattern Slot ^day_of_week
                                     <vday_of_week>))
 )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Check non-short done
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; If non-short selects do not exist for the product line,
;   then change Stats looking for shortages to 'yes'

(p look_for_shorts

        (Shop ^shop_name <vshop_name>)
        {<evstats>(Stats ^look_for_shortages no
                         ^action check_short) }
        - (Select ^shortage_ind no ^shop_name <vshop_name>
                  ^schedule_ind no)
-->
        (modify <evstats> ^look_for_shortages yes
                          ^action reset_examine)
)
```

```
(p found_no_shorts

        (Shop ^shop_name <vshop_name>)
        {<evstats>(Stats ^look_for_shortages no
                        ^action check_short) }
        (Select ^shortage_ind no ^shop_name <vshop_name>
                ^schedule_ind no)
-->
        (modify <evstats> ^action schedule)
)


; Next rule is for frozen part of schedule-- the part of
;  schedule where assigned days are less than today
;   These selects were scheduled on days gone by.

(p write_frozen_sched
        (Priority ^pr_product_number <vpr_product_number>)
        {<evselect>(Select ^select_number<vselect_number>
                           ^sl_product_number
                               <vpr_product_number>
                           ^assigned_day <vassigned_day>
                           ^shortage_ind <shortage_ind>
                         ^tape_sequence <vtape_sequence>
                           ^program_ics <vprogram_ics>
                           ^kitting <vkitting>
                           ^status <vstatus>
                           ^schedule_ind no)}
        (Stats ^today > <vassigned_day>)
        (Shop ^shop_name <vshop_name>)
-->
        (write (crlf) |Select | <vselect_number>
               was scheduled on day: | <vassigned_day>)
        (write run_out |Select | <vselect_number>
                | was scheduled on day: |
                <vassigned_day> <vshop_name> (crlf))
        (write sched_out <vselect_number>
                         <vpr_product_number>
                         <vtape_sequence> <vprogram_ics>
                         <vkitting> <vassigned_day>
                         <shortage_ind>
                         <vstatus> <vshop_name> (crlf))
        (modify <evselect> ^schedule_ind yes
                           ^shop_name <vshop_name>)
)


; Check if scheduling complete for product
```

```
(p schedule_complete
        {<evstats> (Stats ^action schedule)}
        - (Slot ^unscheduled_slots  > 0)
        {<evshop> (Shop ^shop_name <vshop_name>)}
-->
        (write (crlf) | ***Schedule complete for shop|
                <vshop_name>)
        (remove-pattern ^unscheduled_slots 0)
        (remove <evshop>)
        (modify <evstats> ^action read_shop_file)
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Schedule complete
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; When hit end of file on shop input, then scheduling
;   for each shop is complete.

(p no_more_shops
        {<evstats>(Stats ^action get_shop_input)}
        (Shop ^shop_name {<vshop_name> = end_of_file})
-->
        (closefile shop_in)
        (modify <evstats> ^action scheduling_complete)
)

; Find selects that were not assigned to a shop and write
;  warning

(p find_no_priorities
        {<evstats> (Stats ^action scheduling_complete)}
        {<evselect> (Select ^shop_name init
                    ^select_number <vselect_number>
                        ^sl_product_number
                            <vsl_product_number>
                    ^tape_sequence  <vtape_sequence>
                        ^program_ics <vprogram_ics>
                        ^kitting <vkitting>
                        ^shortage_ind <vshortage_ind>
                        ^status <vstatus>)}
-->
        (write (crlf) |Select | <vselect_number>
                | has no shop. It was not scheduled|(crlf))
        (write sched_out <vselect_number>
                    <vsl_product_number>
```

```
                         <vtape_sequence> <vprogram_ics>
                         <vkitting> * <vshortage_ind>
                         <vstatus> no_shop (crlf))
        (write run_out |Select | <vselect_number>
                |has no shop.  It was not scheduled|
                (crlf))
        (modify <evselect> ^shop_name no_shop
                          ^assigned_day *)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Sort and print
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(p sort_schedule
        {<evstats> (Stats ^action scheduling_complete
                         ^sched_file <vsched_file>)}
-->
        (closefile sched_out)
        (write (crlf) (crlf) (crlf) (crlf) (crlf) (crlf))
        (call c5_sched_sort_cmd <vsched_file>)
        (modify <evstats> ^action modifications)
)

; See if modifications to the schedule are required.  If
;  so obtain the select number to change

(p make_modifications
        {<evstats>(Stats ^action modifications
                         ^chg_select_number none )}
-->
        (bind <response> wrong)
        (while (<response> == wrong)
                (write (crlf) |Would you like to change |
                   |the schedule? (yes or no) |)
                (bind <answer> (accept))
                (case <answer>
                   ((y yes)
                      (write (crlf) |Enter select number|
                              | to change: |)
                         (bind <chg_select> (accept))
                         (modify <evstats>
                                 ^chg_select_number
                                 <chg_select>)
                         (bind <response> right))
                    ((n no)
                     (modify <evstats> ^action all_done)
                        (bind <response> right))
                   (otherwise
```

```
                    (write (crlf) |Invalid response. |
                                  | Try again|))
            )
        )
)

; Get day  for the select to be scheduled if the select
;  number that is to be changed was found with an assigned
;  day greater than or equal to today;  The assigned day
;  must be greater than or equal  to today to prevent past
;  schedules from being altered.

(p modify_assigned_day
        {<evstats> (Stats ^chg_select_number
                    {<vchg_select_number> <> none}
                          ^action modifications
                          ^today <vtoday>
                          ^last_workday <vlast_workday>)}
        {<evselect> (Select ^select_number
                                <vchg_select_number>
                            ^assigned_day
                            {<vassigned_day> >= <vtoday>})}
-->
        (bind <response> wrong)
        (while (<response> == wrong)
            (write (crlf) |Enter revised schedule day: |)
            (bind <chg_day> (accept))
            (if ((<chg_day> >= <vtoday>) && (<chg_day> <=
                    <vlast_workday>))
                (bind <response> right)
                (modify <evselect> ^assigned_day <chg_day>)
                (modify <evstats> ^chg_select_number none
                                  ^chg_schedule_done yes)
                (write (crlf) |Select |
                        <vchg_select_number>
                        | schedule change to day |
                        <chg_day>)
            else
                (write (crlf) |Invalid schedule day. |
                              | Please try again|)
            )
        )
)


; If the select number to be changed does not exist or has
;  an assigned day less than today , reprompt user
```

```
(p invalid_modify_select
        {<evstats> (Stats ^chg_select_number
                          {<vchg_select_number> <> none}
                           ^today <vtoday>
                           ^action modifications)}
        -(Select ^select_number <vchg_select_number>
                 ^assigned_day
                {<vassigned_day> >= <vtoday>})
-->
        (write (crlf) |Invalid select number. Please try |
                        | again. |)
        (modify <evstats> ^chg_select_number none)
)

; Write and sort the schedule

(p reopen_schedule_file
        {<evstats>(Stats ^action all_done
                         ^chg_schedule_done yes
                         ^sched_file <vsched_file>)}
-->
        (openfile sched_out <vsched_file> out)
        (modify <evstats> ^action write_out
                          ^chg_schedule_done complete)
)


; If selects were changed,  write selects back to file
;  after select is written, remove from working memory to
;  keep track of which selects were written

(p write_schedule
        (Stats ^action write_out)
        {<evselect>(Select ^select_number<vselect_number>
                            ^sl_product_number
                                 <vpr_product_number>
                            ^assigned_day <vassigned_day>
                            ^shortage_ind <shortage_ind>
                          ^tape_sequence <vtape_sequence>
                            ^program_ics <vprogram_ics>
                            ^kitting <vkitting>
                            ^shop_name <vshop_name>
                            ^status <vstatus>)}
-->
       (write sched_out <vselect_number>
                        <vpr_product_number>
                        <vtape_sequence> <vprogram_ics>
                        <vkitting> <vassigned_day>
                        <shortage_ind>
```

```
                        <vstatus> <vshop_name> (crlf))
        (remove <evselect>)
)


; After all selects have been written to file, stop

(p write_complete
        {<evstats> (Stats ^action write_out
                          ^sched_file <vsched_file>)}
-->
        (closefile sched_out)
        (write (crlf) (crlf) (crlf) (crlf) (crlf) (crlf))
        (call c5_sched_sort_cmd <vsched_file>)
        (modify <evstats> ^action all_done)
)


; close run file and stop processing

(p call_it_quits
        (Stats ^action all_done)
-->
        (write (crlf) |Schedule is complete|)
        (write run_out |Schedule is complete|(crlf))
        (closefile run_out)
)

; Get system started by following make command

(make Start)
```

9. APPENDIX B - EXTERNAL SORT FUNCTION

```
#include <stdio.h>
#include <string.h>
#include "/u1/rgb/c5/c5.h"

#define PRD_LEN         4
#define SEL_LEN         6
#define DAY_LEN         1
#define AFFIRM_LEN      3
#define STATS_LEN       5
#define SHOP_LEN        7
#define MAX_SCHED       1000
#define LINE_LEN        256

typedef struct {
        char    select[ SEL_LEN + 1 ];
        char    product[ PRD_LEN + 1 ];
        char    seq[ AFFIRM_LEN + 1 ];
        char    pic[ AFFIRM_LEN + 1 ];
        char    kit[ AFFIRM_LEN + 1 ];
        char    day[ DAY_LEN + 1 ];
        char    shortage[ AFFIRM_LEN + 1 ];
        char    status[ STATS_LEN + 1 ];
        char    shop[ SHOP_LEN + 1 ];
} SCHED;

main()
{
        /* initialize c5 */
        c5_init() ;

        /* load schedf and run until no rules match */
        c5_tl_cmd ("(load schedf) (run)") ;
/*      exit(0) ;*/
}

/* this function tells c5 about user defined functions */
c5_usrfuncs()
{
        void c5_sched_sort_cmd();

        c5_define_function
        ("c5_sched_sort_cmd",c5_sched_sort_cmd);
}
```

```
/* this is a user defined function that will sort the */
/*  scheduled by the schedule day */
void
c5_sched_sort_cmd()
{
        char    *filename;
        FILE    *fp;
        SCHED   schedule[ MAX_SCHED ];
        SCHED   temp;
        char    cur_line[ LINE_LEN ];
        C5_VALUE c5_parameter() ;
        C5_VALUE value ;
        char    *c5_string_value();
        int     tot_lns;
        int     i,j;
        int argc ;
        int c5_parametercount() ;

        argc = c5_parametercount() ;
        if (argc != 1)
        {
                fprintf( stderr,"usage: (call
                c5_sched_sort_cmd filename)") ;
                return ;
        }

        /* Copy name in filename. Find value in slot 1 */
        /* of the result element using c5_parameter(1). */
        /* Extract the value of slot as a string with */
        /* c5_string_value.Last, must copy values into */
        /* local storage. */

        strcpy(filename,c5_string_value(c5_parameter(1)));

        /* Open file and read in data */
        if( ( fp = fopen( filename, "r" ) ) == NULL )
        {
                fprintf(stderr,"ERROR:c5_sched_sort_cmd():
                Can't open file: %s for read.0,
                filename );
                return ;
        }
        else
        {
                tot_lns = 0;
                while
                 (fgets( cur_line, LINE_LEN, fp ) != NULL)
                {
                        sscanf
```

```
         (cur_line, "%s%s%s%s%s%s%s%s%s",
        schedule[tot_lns].select,
        schedule[tot_lns].product,
        schedule[tot_lns].seq,
        schedule[tot_lns].pic,
        schedule[tot_lns].kit,
        schedule[tot_lns].day,
        schedule[tot_lns].shortage,
        schedule[tot_lns].status,
        schedule[tot_lns].shop );
        tot_lns++;
}

fclose( fp );

/* Compare values in array and sort */
for( i=0; i < tot_lns; i++ )
{
        for( j=i; j < tot_lns; j++ )
        {
            if( strcmp( schedule[i].day,
               schedule[j].day ) > 0 )
            {
               temp = schedule[i];
               schedule[i] = schedule[j];
               schedule[j] = temp;
            }
        }
}

/* Write sorted array back to file */
if
 ((fp = fopen( filename, "w" )) == NULL)
{
        fprintf( stderr,"ERROR:
        c5_sched_sort_cmd(): Can't open
        file: %s for write.0, filename );
        exit( 1 );
}
else
{

   fprintf (fp,"%31s%-16s"," ",
   "SELECT SCHEDULE0) ;
   fprintf (stderr,"%31s%-16s"," ",
   "SELECT SCHEDULE0) ;
   fprintf (fp,"%27s%-22s"," ",
   "SCHEDULE DAY SEQUENCE0);
   fprintf (stderr,"%27s%-22s"," ",
```

```
"SCHEDULE DAY SEQUENCE0);
fprintf (fp,"%s","0) ;
fprintf (stderr,"%s","0) ;
fprintf (fp,"%4sSELECT%2sPRODUCT%2s
SCHEDULE%2sTAPE%3sPROGRAM%2sKIT%3s
SHORT %2sSELECT%3sSHOP0," "," ",
" "," "," "," "," "," "," "," ");
fprintf (stderr,"%4sSELECT%2sPRODUCT
%2s SCHEDULE%2sTAPE%3sPROGRAM%2sKIT
%3sSHORT%2sSELECT%3sSHOP0," ",
" "," "," "," "," "," "," "," "," ");
fprintf (fp,"%4sNUMBER%2sNUMBER%5sDAY
%3sSEQUENCE%3sIC%4sPARTS%1sSTATUS%2s
STATUS0," "," "," "," "," "," ",
" "," ") ;
fprintf (stderr,"%4sNUMBER%2sNUMBER
%5sDAY%3sSEQUENCE%3sIC%4sPARTS
%1sSTATUS%2sSTATUS0," "," "," ",
" "," "," "," ") ;
fprintf (fp,"%s","0) ;
fprintf (stderr,"%s","0) ;
for( i=0; i < tot_lns; i++ )
    {
        fprintf( fp,"%5s%4s%6s%2s
        %8s%s%6s%3s%6s%3s%4s%3s%4s
        %3s%5s%4s%4s%s0," ",
        schedule[i].select,
        " ",schedule[i].product," ",
        schedule[i].day," ",
        schedule[i].seq," ",
        schedule[i].pic," ",
        schedule[i].kit," ",
        schedule[i].shortage," ",
        schedule[i].status," ",
        schedule[i].shop );
        fprintf( stderr,"%5s%4s%6s%2s
        %8s%s%6s%3s%6s%3s%4s%3s%4s%3s
        %5s%4s%4s%s0," ",
        schedule[i].select,
        " ",schedule[i].product," ",
        schedule[i].day," ",
        schedule[i].seq," ",
        schedule[i].pic," ",
        schedule[i].kit," ",
        schedule[i].shortage," ",
        schedule[i].status," ",
        schedule[i].shop );
    }
    fclose( fp );
```

```
                }
        }
/* End c5_sched_sort_cmd () */
```

A KNOWLEDGE-BASED FAIR-SHARE SCHEDULER

by

SUSAN M. SAAD

B.S., University of Colorado, 1982

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Today, many long-term planning and scheduling computer systems support factories. The day-to-day scheduling tasks are most often performed by a factory foreman or supervisor. However, recent focus on more efficient manufacturing processes and just-in-time philosophies require that efficient, short-term planning and scheduling systems be developed.

The day-to-day scheduling of work can be arduous. The scheduling problem becomes more difficult when multiple entities are competing for resources from the same supplier. Many traditional management science solutions employ algorithms that are limited to special scheduling cases that can make invalid assumptions. A broader, global view of scheduling problems using heuristics can also provide solutions. This can be done by embedding the foreman's or supervisor's expertise in a knowledge base.

This paper describes an expert system that creates a daily schedule for a warehouse which must serve many customers--many shops in a factory. The rules try to satisfy the needs of each of the shop and also provide the warehouse with a doable schedule.