A Rule-Based Expert System for Automated Staff Scheduling

Steven Shaffer
Integrated Systems Consulting Group
Malvern, Pennsylvania

Abstract - **STAFF-IT! is a PC-based system able to schedule up to 100 employees into as many as seven departments based on a user-defined rule set. It incorporates a base scheduling strategy determined to be closest to optimal under a battery of simulation tests. Three types of rules are allowed: (1) DON'T rules, which tell the system to disallow a particular combination of schedule entries, (2) DO rules, which require the system to create a particular schedule entry whenever a certain condition arises, and (3) TRY rules, which are like DO rules but are not mandatory. STAFF-IT! includes an easy to use interface, including pop-up selection boxes and field level help. In tests, STAFF-IT! was able to schedule 30 employees (10 of whom were part time) into three departments based on departmental needs, employee availability, and user-defined scheduling rules in less than a minute with full employee utilization.**

INTRODUCTION

The employee scheduling problem is that of scheduling employees for various duties at various times in some orderly fashion, preferably minimizing some negative (e.g., payroll) or maximizing some positive (e.g., worker satisfaction) value. The scheduling of nurses, police and fire personnel, parcel handlers, mine workers, employees of fast food restaurants, and telephone operators are just a few of the many areas in which scheduling problems abound, and in which the automation of scheduling can be of great use. There are two minimal requirements which must be fulfilled in order for any worthwhile automated scheduling to take place:

1. The workers must be able to be viewed as interchangeable. Although there will always be variations in ability and experience, for purposes of scheduling large groups of people in any effective manner they must be considered homogenous. This also means that it is usually not possible to specifically attempt to minimize payroll costs with a scheduling system; such a goal is approached instead through minimizing overstaffing in general.

2. It must be possible to predict workloads to some extent. Without the ability to define needs, it is not possible to create a useful schedule. This is also true, of course, for creating manual schedules, except that often the workload predictions exist only in the mind of the manager, who will insist that the scheduling can only be done "by experience."

Various aspects of the working environment affect the usefulness of automated scheduling. Each of these can interact with others, making the entire process very complex, and sometimes intractable. This variables include:

a. Total operating hours per schedule period.
b. Days-off policy.
c. "Stretch" limitations.
d. Number and duration of shifts.
e. Seniority rules.
f. "Fairness" heuristics.
g. Pay period.
h. Staff availability.
i. Lunch and rest period issues.

Traditionally, the employee scheduling problem has been handled by line managers who approach the problem in various ways, not all of which are particularly effective. There are many reasons for this, such as the fact that many companies do not offer middle managers any guidance regarding how to approach the problem. Often this is because the company itself has not found a satisfactory solution to the problem; this is caused, in turn, by the fact that the problem is often very complex. Thus, most schedules are created in a very haphazard manner, and often take several hours to several days of the manager's time.

Because of the complexity of the problem and its importance to the operation of a great many organizations, the employee scheduling problem has been studied by mathematicians, operations researchers, and industrial engineers. Many models have been developed utilizing approaches such as linear/integer programming, network flow, graph coloring, and satisfiability. These approaches, while effective in many cases, tend to be ad-hoc and single-purpose. A few (notably Kurzydlowska & Piasecki [2] and

ISSN# 0-7803-0233-8/91 $1.00

Glover & McMillan [1]) have attempted or are attempting to build general employee scheduling systems. This paper adds to the literature the experience of building STAFF-IT!, a forward-chaining rule-based expert system for shift-based staff scheduling[3].

Because of the flexibility required to generate staff schedules for a wide variety of organizations, it was decided that some capacity for user-defined rules must be included in the system. This allows the user to define the parameters within which the schedule will be created, thus eliminating the need for creating customized software for each user. Incorporating this flexibility into the system virtually eliminates the ability to develop truly optimized schedules, however, since the details of the scheduling environment are left for the end-user to define. The goals of the current system are thus (1) to create a schedule (whenever possible) that does not violate any of the user-defined rules, and (2) to create such a schedule that is closer to optimal than a randomly selected one. The fulfillment of the second criterion will is achieved by selecting an heuristic that yields the best results when simulated; this is discussed in more detail below.

The system consists of five subsystems: The database manager (DBMAN), the rule editor (RULED), the automated scheduler (SCHEDULER), the interactive schedule update mechanism (ISUM), and the report generator (REPGEN).

RULE STRUCTURE

Three types of rules are defined in the system: DO, DON'T, and TRY. DO and TRY are declarative rule structures, and DON'T is a restrictive rule structure.

A TRY rule is defined as follows:

TRY (I, P1, T1, D1, ET, P2, T2, D2)

And should be interpreted as:

If employee I is of type ET and is scheduled in place P1 at time T1 on day D1, then TRY to schedule employee I at place P2 at time T2 on day D2. Each parameter of TRY, with the exception of I (i.e., P1 ... D2) can be either a specific instance, a variable, or the key word "any." The parameter I may be either a particular instance or the key word "any," but not a variable.

For example, the following rule:

TRY (Susan, Surgery, Shift1, Monday, RN, Surgery, Shift1, Tuesday)

would mean: "If Susan is scheduled for surgery at shift 1 on Monday and she is an RN, then try to schedule her for shift 1 in surgery on Tuesday also." As it would be rather cumbersome to specify this type of rule for every possible employee / dept / shift / day combination, so variables may be used as follows:

TRY (any, @deptx, @shiftx, Monday, any, @deptx, @shiftx, Tuesday)

which says, in effect, "If anyone of any type is scheduled anywhere at any time on Monday, then try to schedule them for the same time and place on Tuesday. A combination of seven such rules, (with Tuesday..Sunday substituting for Monday and Wednesday..Monday substituting for Tuesday) will cause the scheduling module to attempt to give employees some consistency in their schedules - that is, they would tend to have two days off in a row, tend to work in the same department, and tend to work on the same shift throughout the scheduling period. There is only one restriction on the use of variables within the rules: if a variable is used in parameter x1, then it must be used in parameter x2, where "x" stands for P, T, or D. Future enhancements will include the ability for the user to specify, for example, @day+1.

The DO rule is syntactically the same as the TRY rule; the difference between the two is a semantic one, since the results of a DO rule are mandatory: that is, if the left side (antecedent) of the conditional holds, then the right side (consequent) must hold. In practice, this means that the automated scheduling module would not schedule the antecedent instance of the rule unless the consequent instance could also be scheduled. Note that the consequent instance could not be scheduled until all rules for which it was an antecedent were checked recursively.

TRY is a special case of DO in which the return value of subsequent (recursive) scheduling attempts is not checked. For the DO rules, all rules which fire as a result of the execution of a DO rule must execute, or the entire chain of schedules created to that point must be "rolled back" (to borrow a term from relational database systems). This means that it is possible to define a set of DO rules which allow no

one to be scheduled, as in the case of a rule which states "whenever anyone is scheduled at any place / time, schedule them for the same place / time the next day". Clearly, this rule will fail on the seventh recursion, if not before.

The DON'T rule is defined as follows:

DON'T (I, P1, T1, D1, ET, P2, T2, D2)

And should be interpreted as: If employee I is scheduled in place P1 at time T1 on day D1 and is of type ET, then DON'T schedule employee I at place P2 at time T2 on day D2. As with TRY and DO, each parameter of DON'T, with the exception of I (i.e., P1 ... D2) can be either a specific instance, a variable, or the key word "any." The parameter I may be either a particular instance or the key word "any," but not a variable.

The following DON'T rule:

DON'T (any, any, any, Sat, any, any, any, Sun)

would be interpreted as "If anyone is scheduled anywhere at any time on Saturday, then don't schedule them at all on Sunday."

The DON'T rules, which are restrictive and not declarative, have an obvious (but nonetheless dangerous) power to make scheduling impossible by definition. One needs only to create a rule which states "Whenever anyone is scheduled anywhere at any time, don't schedule them anywhere else at any other time," and the system will thus only be able to schedule each person to one shift per scheduling session. Although the possibility of conflicting rules exists, it should be noted that this will not cause the automated scheduling module to fail - it will eventually finish, although no one will be scheduled. The "trace" available from the system also gives the user an ability to track down those rules which are causing the conflicts, so that s/he will be able to adjust them.

Thus, the power inherent in the rules is sufficient to allow the user to define inconsistent or conflicting rules. It must be left as a responsibility of the user not to get into this type of situation. Although some will claim that this is too much of a burden to place on a non-computer professional, there is no known way to give the users sufficient power to perform their scheduling while still protecting them from this responsibility.

INTERFACE

One of the most important aspects of the front-end sub-systems is how well they fit the intuition of the user in reference to how the schedule is viewed. If a user finds the interface confusing or counter-intuitive he is unlikely to use the system productively, if at all. The author has spent a number of years in developing a plausible user interface for the ISUM component of the system. During this time, two potential interfaces were developed and abandoned before the current version was created and accepted.

From an algorithmic standpoint, a schedule entry is represented by a four-tuple: (day, shift, who, where). However, the user sees a schedule entry as a pair of 2-item data points (a "who/where" at a "day/shift"); the difference is subtle. The connecting aspect between the staff member and the department is the day/shift. In order for users to schedule a staff member for a particular department at a particular day/shift, they must have information about both the staff member (availability, employee type, etc.) and the department (staffing needs, required employee type, etc.). What was needed was an interface that captures this relationship, while still giving users access to the information that they need to make decisions.

Figure 1 shows the layout of the ISUM screen. On the left side of the screen is a description of the status of the "place" (department, ward, post, etc.) being considered. On the right is a description of the status of the individual being considered. Both sides of the screen look and act much like a synchronized electronic spreadsheet; the cells on the department half of the screen contain the number of staff still needed for that department in every day/shift. The cells on the staff side show whether or not a staff member is currently occupied for each day/shift. A staff member can be occupied by either being scheduled for a department (in which case the department number is displayed), or s/he can be given the day off by being scheduled for department 8 (which department name is "OFF").

On the bottom part of the staff side of the screen is a five-line display called "INFO," which displays information about the employee that may be important to the person doing the scheduling. The user may view or change department needs, schedule or unschedule an employee, or view a roster of scheduled employees for the day / shifts selected.

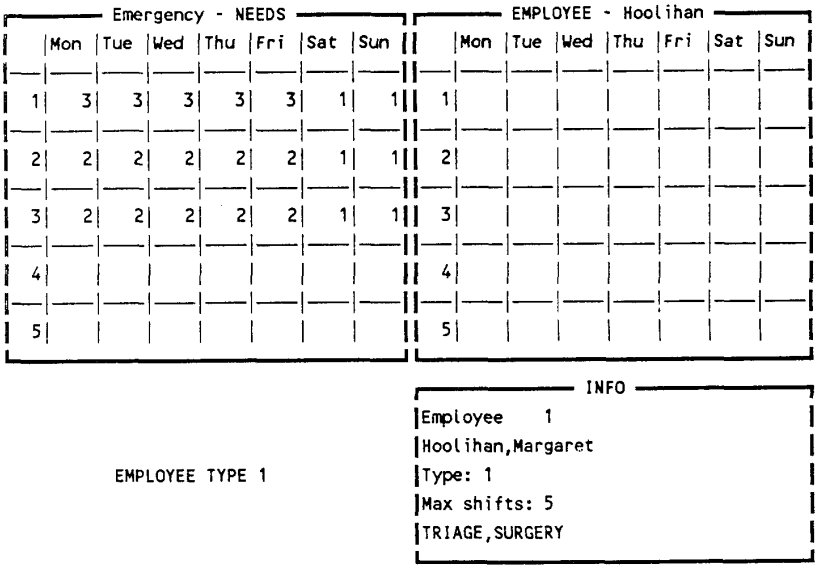

FIGURE 1

IMPLEMENTATION

As mentioned, the author does not believe that the concept of a rule base and the concept of real optimization of schedules can efficiently co-exist in a software system. Thus, it is suggested that an overall effective scheduling strategy in conjunction with a rule base will achieve the most plausible results for the task at hand. It was, therefore, important that various general strategies be tested against one another for overall effectiveness.

Rather than decide a priori which is the best approach, it was decided that a simulation of each potential strategy on a series of potential scheduling problems would be performed. In order to decide which is the best, three measures of effectiveness were defined, as follows:

Fulfillment: $\frac{\text{scheduled employees per shift}}{\text{available employees per shift}}$

Utilization: $\frac{\text{employee scheduled hours}}{\text{employee available hours}}$

Average Scheduling time (in seconds)

The overall "best" scheduling strategy was selected by filtering the measures as follows: select the strategies with the best fulfillment; of those, select the strategies with the best utilization; of those, select the fastest algorithm.

Monte Carlo simulations were run which simulated potential strategies and calculated the fulfillment, utilization, and speed of each approach under a variety of conditions. These statistics were compared, and the "best" strategy was determined to be the selection of the most needy department and the most available staff member to fill it. (It is somewhat ironic that over one thousand simulations were run to determine that the most intuitive answer is, in fact, the best.) Once the candidate department and employee are selected, the system tries to find a time and place which matches both the department's needs and the employee's availability. If such a match is found, the schedule is updated; if not, nothing happens, and the process continues.

The "most needy" department and "most available" employee are determined via a strict "greater than" operation - thus, the order in which the candidates (departments or employees) are chosen for comparison could create a strong bias in the system. For example, if the employees were always polled for

availability ordered by their employee numbers, the lower employee numbers might tend toward getting the better schedules from week to week. This would be perceived as "unfair," and is therefore undesirable. Thus, the first candidate (department or employee) tested is selected at random from the list; testing for variances then proceeds in numeric order from that point and continues through the list until the first employee or department is again encountered. The selection of the candidate days and shifts are performed in the same manner. The seed value for the random number generator is based on the system clock and thus the schedules are not reproducible. This is an important feature of the system - every attempt was made to remove biases from the base algorithm. All biases, if required, are built into the rule set for the individual installation.

Another important feature of the algorithm utilized is that of a decaying availability measure for the employees. If a candidate employee is refused a potential scheduling assignment due to a mismatch in availability or a rule restriction, that candidate is assigned a penalty, which will lower his availability for the next round of candidate-choosing. As soon as an employee is scheduled, his/her penalty level is reset to zero. This removes the threat of "thrashing", and helps to guarantee that the process will end. In the current implementation, the decay factor is linear (one misschedule, one penalty point), but in later implementations it may be altered to incorporate a more robust (e.g., exponential) method.

Every effort has been made to make STAFF-IT! simple to use. Field-level help, consistent function key mappings, and pop-up selection boxes are some of the features which make the system straightforward and easy to use. Every screen, most notably the ISUM screen, was designed to present the data in a clear, consistent and intuitive manner so that anyone familiar with computer usage will be able to utilize the system almost immediately. Several additional user aids are available, such as a pop-up electronic "note pad" for making scheduling notes, a simple "backup to floppy" facility, and a continuously displayed clock.

EVALUATION

Two types of evaluation were performed on the system: (1) a test case of moderate complexity and (2) an analysis of the relationship of system parameters to execution times. Each is discussed in turn below.

First, a two-part test case was executed using the system. The test case utilized a hospital nurse scheduling model in which three departments (Surgery, Emergency, and Maternity) compete for the use of nurses from a hospital pool of 20 full-time, 5 high part-time (3 shifts a week) and 5 low part-time (2 shifts a week). Each employee, along with their corresponding maximum shifts per week, was entered into the system. The hospital was assumed to run on a three shift, 24-hour schedule and the following metarules were implemented (shown here in colloquial English):

(1) Whenever anyone is scheduled for any department on any shift on day N, try to scheduled him/her for the same department and shift for day N+1.

(2) If anyone is scheduled anytime on Saturday, then don't schedule him/her on Sunday and vice-versa.

Metarule (1) actually incorporates 7 rules, while metarule (2) requires 2 rules.

The execution of the automatic scheduler (running on a 12 Mhz 80286 MS-DOS computer) took 109 seconds to execute. Table 1 shows a summary of the results. The automated scheduler was then run on a second test case, nearly identical to the first, except for the exclusion of the rules which implement metarule 2. The execution of the system under these conditions took only 37 seconds, and yielded much better results, as shown in Table 2.

Average employee utilization: 88%
Average occurrence of "same shift": 61%
Average occurrence of "same department": 62%
65% of full time employees received 2 consecutive days off

Department requests fulfillment:

Department	% Fulfillment
Surgery	80%
Emergency	78%
Maternity	78%

TABLE 1

Average employee utilization: 100%
Average occurrence of "same shift": 90%
Average occurrence of "same department": 90%
97% of full time employees received 2 consecutive days off

Department requests fulfillment:

Department	% Fulfillment
Surgery	90%
Emergency	89%
Maternity	90%

TABLE 2

Thus, we can again see the danger in utilizing the rules in too restrictive a manner. Because of the power and the flexibility of the rules, it is practically impossible to predict or define a relationship between the number of rules and the execution time of the system. Many non-restrictive TRY rules will allow much faster execution than one, very restrictive, DO or DON'T rule. However, analysis determined that the execution time has a tight linear correlation to the number of employees being scheduled (a factor of 3.6 on the machine used for the testing).

CONCLUSIONS

It is believed that the current work was successful in determining a need and in taking the initial steps toward fulfilling that need. Once the concept of true optimality was dropped in favor of overall usefulness, the plausibility of creating a generalized system increased. The addition of user-defined rules as a substitute for ad hoc recoding adds elegance and commercial viability to the system.

A significant achievement of this effort is the development of an intuitive user interface for making adjustments to the schedule. This development is important for two reasons: (1) The user will almost certainly want or need to make adjustments to make allowances for absences, etc., and thus the system must allow the user to do this with a minimum of difficulty; (2) The interface gives the user a clear "view" of the scheduling process and thus adds to the overall understanding of the system by the user.

Another significant achievement of this effort is the development of a rule syntax which, while small, is very powerful. Many types of scheduling procedures can be incorporated into the system via the language as it stands, although planned extensions will almost certainly make it even more flexible.

Future work on this product will include a longer scheduling horizon, an enhanced rule syntax, and a data import/export utility.

The author believes that the system developed here has uses beyond staff scheduling; in fact, the current system has been slightly modified and been used to schedule children for activities at camp and students for classes at a junior college. Further work should be done in these and other areas of scheduling to see if the concepts defined here could be used to aid users in other scheduling tasks.

ACKNOWLEDGMENTS

The author would like to thank Robert Beck for his help with this project.

References

1. Glover, Fred, and McMillan, Claude, The General Employee Scheduling Problem: An Integration of MS and AI, COMPUTERS AND OPERATIONS RESEARCH, 13, 5, (1986), 563.

2. Kurzydlowska, Anna, and Piasecki, Stanislaw, Formalization and Computer Representation of Organizing Problems for Purposes of Computer-Aided Problem Resolution, COMPUTERS IN INDUSTRY, 10, (1988), 21.

3. Shaffer, Steven, A Rule-Based Expert System for Automated Staff Scheduling, MASTER'S THESIS, Villanova University, 1990.