Comput. & Ops. Res. Vol. 13, No. 5, pp. 563–573, 1986
Printed in Great Britain.

0305-0548/86 $3.00 + 0.00

THE GENERAL EMPLOYEE SCHEDULING PROBLEM: AN INTEGRATION OF MS AND AI

FRED GLOVER* and CLAUDE MCMILLAN†
Center for Applied Artificial Intelligence, Graduate School of Business, University of Colorado, Boulder, CO 80309, U.S.A.

Scope and Purpose—In certain settings the routine scheduling of employees proves to be a complex problem in combinatorics. During the past 12 years most efforts to automate scheduling have relied chiefly on one or another variant of linear programming. We show that those approaches are disappointing in the case of the general employee scheduling problem, since more than 4 million integer variables are involved. On the other hand, by relying on an integration of appropriate techniques from both management science and artificial intelligence we render the general problem thoroughly tractable, permitting the routing scheduling of over 100 employees for one week on an IBM PC in minutes. In this paper we speak of the approach which makes that possible.

Abstract—The general employee scheduling problem extends the standard shift scheduling problem by discarding key limitations such as employee homogeneity and the absence of connections across time period blocks. The resulting increased generality yields a scheduling model that applies to real world problems confronted in a wide variety of areas.

The price of the increased generality is a marked increase in size and complexity over related models reported in the literature. The integer programming formulation for the general employee scheduling problem, arising in typical real world settings, contains from one million to over four million zero–one variables. By contrast, studies of special cases reported over the past decade have focused on problems involving between 100 and 500 variables.

We characterize the relationship between the general employee scheduling problem and related problems, reporting computational results for a procedure that solves these more complex problems within 98–99% optimality and runs on a microcomputer. We view our approach as an integration of management science and artificial intelligence techniques. The benefits of such an integration are suggested by the fact that other zero–one scheduling implementations reported in the literature, including the one awarded the Lancaster Prize in 1984, have obtained comparable approximations of optimality only for problems from two to three orders of magnitude smaller, and then only by the use of large mainframe computers.

INTRODUCTION

Employee scheduling problems arise in a variety of service delivery settings, including the scheduling of nurses in hospitals, check encoders in banks, airline and hotel reservation personnel, telephone operators, patrol officers and others. In their simplest form, these problems involve only the assignment of days-off, as in some of the less complex settings for the scheduling of nurses. A typical problem of this form requires the scheduler to give appropriate days off to each of a number of employees who work standard shifts with differing start times while assuring that the required number of employees are on duty throughout the day and week. Variations of this type were addressed in Brownell and Lawrence [1] and Keith [2].

The shift scheduling problem, as in the scheduling of telephone operators, is more complex. In shift scheduling, the scheduler works with part-time as well as full-time employees, and shift types contrast with each other in the following attributes:

(1) duration (length)
(2) start times
(3) the number of breaks (reliefs)
(4) the placement of the breaks.

The scheduler must determine the shift types (and the number of each type to employ), and in some cases determine which employee should receive which set of shifts. Union rules and company policy

*Fred Glover received the doctorate from Carnegie–Mellon in 1965, and served on the faculty at the University of California, Berkeley, and the University of Texas and the University of Minnesota. He is the author of over 150 published papers in Management Science and related fields.

†Claude McMillan received the doctorate from Ohio State University in 1955, and served on the faculty at Michigan State University until 1965. He is the author or coauthor of a number of papers and four books in the field of management science.

restrictions are handled in a limited fashion. Variations on this problem have been addressed in Refs [2–8].

The objective in the shift scheduling problem generally is to approximate as closely as possible the desired number of employees on duty, either by minimizing the overage or minimizing the "shortage/overage" mix.

The more complex days-off and shift scheduling problem [7] comes a step closer to the general employee scheduling problem, and some variations [3, 4, 8] consider also the assignment of shifts to actual employees, but deal with that assignment of shifts to employees only after the shift types (and number of each) have been determined.

Table 1 classifies shift scheduling problems as they are described in the literature, and indicates the differences in: problem size and complexity, the approach employed, and the extent to which the approach was actually being used in industry.

THE GENERAL EMPLOYEE SCHEDULING PROBLEM

The more complex scheduling problem which we address has wide applicability, especially in the supermarket, reservation office and fast food fields. It differs rather dramatically from the days-off and the shift scheduling problems by including important real world features that resist practical solution by methods of formal analysis. We first describe the problem informally and indicate the features that a practical solution system must have in order to deal with the problem effectively. For comparative reference, we indicate overlaps and contrasts with other employee scheduling problems previously examined. We report the result of applying our approach to problems from real world settings and discuss the implications of our empirical results.

FULL-TIME/PART-TIME EMPLOYEES

As in the shift scheduling problem, it is assumed in the general employee scheduling problem that some fraction of the work will be done by full-time employees and the remainder by part-time employees. Full-time employees are those entitled to work a standard number of hours each week (commonly 40) and generally work shifts of standard duration (commonly 8 hours each). The start

Table 1. A classification of the employee scheduling problems addressed by selected papers since 1972

Complexity/size (for papers cited in References)	Approach employed	Time periods	Extent of use
Low:			
[1] U. formula	Analytic	14	None
[4] 100 shifts	LP and heuristics	32	None
[5] 100 shifts	Branch and bound	72	None
[13] U. formula	Analytic	21	None
[14] U. formula	Analytic	Cycles of 7	None
Medium:			
[7] 168 shifts	LP	168	None
[6] 360 shifts	IP and heuristics	49	Banking
[8] 300 to 400	Network and heuristics	48	Phone Co.
[3] < 500 shifts	Heuristics	48	Phone Co.
[2] < 500 shifts	LP and heuristics	96	Phone Co.
High:			
Ours > 1,000,000 shifts	Heuristics—blend of MS/AI	540	Supermkt Fast food

Definitions:

LOW means: no linking constraints between blocks of time periods, a small number of shift types, and homogeneous employees with unlimited availabilities.

MEDIUM means: no linking constraints between blocks of time, a small number of shift types, homogeneous employees with unlimited availabilities, and management rules.

HIGH means: linking constraints between blocks of time periods, a large number of shift types, non-homogeneous employees with limited availabilities, and management rules.

U. formula (uniform formula) means: each shift is characterized by the same rule, such as: every employee works 5 days a week (time periods are in days).

The word "shift" in the foregoing table translates into the word "variable" in an integer programming formulation. (Some references use the word "trick" rather than "shift".)

times and the number and location of breaks may vary. In addition, a full-time employee may or may not be entitled to the same start time each day worked.

In the general problem, as handled by our method, the user is given the ability to specify shift features such as duration, start time, and the number and placement of breaks. The user is also able to specify the number of days off and whether they should be consecutive, and to change those features as the system is routinely used from week to week.

In addition to the above, each employee can specify (or the scheduler can specify for the employee):

(1) Minimum and maximum hours to be worked during the week.
(2) Days and hours of availability during the week.

Item (2) above means that each employee has an availability preference, specifying which days of the week and the hours that employee can work. Since an employee's availability changes (in some settings, about 20% of the work force changes their availabilities each week), the scheduler—human or machine—must be allowed to edit a file of employee availabilities before addressing the composition of a new schedule each week.

Another important feature of the general problem, not treated in the standard scheduling contexts, is that employees are non-homogeneous in ways beyond their availability preferences and thus cannot be treated as interchangeable entities. Employees have differing skill types, skill levels and status attributes which limit the scheduler's freedom in assigning shifts. These include, for example, the training required to work at various work stations. Since these change from time to time, the system must allow the scheduler to edit each employee's profile data.

In the general employee scheduling problem, it may also be necessary to observe seniority rules, such as those specifying that employees with more seniority must get more hours of work and start earlier (an early start may be considered desirable), except when other requirements would be violated.

Union/management rules, in the general problem, can require that a specified minimum amount of time must elapse between the time an employee works one day and begins again the next. In some settings they may also require that certain employees, such as students, may not work beyond a specified hour more than one night during the week.

IMPLICATIONS OF THE NON-HOMOGENEOUS EMPLOYEE POOL

In the simpler shift scheduling problems, it is possible to design shifts (and to determine the desired number of each shift type) without regard to employee availabilities, employee skills and skill levels or employee status. Descriptions of the shift scheduling problem in the literature, on the other hand, sometimes appear to involve a more general solution capability—for example, indicating that shifts are generated to conform to union rules, company policy, etc. What this means, in practice, is that the shifts generated represent categories that are potentially acceptable, but there is no control over whether those selected as a "solution" have an appropriate composition. This is entirely reasonable for settings where restrictions are loose enough that employees can simply "sign-up" for whatever schedule is posted, but in broader settings, such a disregard of individual differences can have dire consequences.

In the general scheduling problem, therefore, the design of the shifts and their assignment to specific employees must be coordinated. Designing shifts without considering whether empolyees can be found to take those shifts will yield either a poor fit (a poor match of employees assigned to employees wanted on duty) or, still worse, an infeasible solution.

LINKING CONSTRAINTS BETWEEN TIME PERIOD BLOCKS

Another significant aspect of the general employee scheduling problem is the existence of linking constraints between time periods. In typical applications of the general problem, each day may be construed as a block consisting of 96 fifteen-minute periods, and the assignment of employees to duty periods during that block is not independent of assignments to employees in other blocks. In addition to restrictions governing admissible assignments on any given day, these linking constraints imply that there are also restrictions governing the total number of periods throughout the entire week, as

well as governing the selection of certain types of assignments successively, or cumulatively, across days of the week, e.g. limitations on the selection of opening and closing assignments.

In standard shift scheduling problems, by contrast, blocks such as days are construed as essentially independent. Linking conditions either do not exist or are innocuous enough to be ignored while composing a schedule for any given day (where, for example, the ending conditions for one day may be used to give starting conditions for the next, which again is treated independently).

The days-off and shift scheduling problems can be readily treated as special cases of the general problem, allowing many of the more difficult conditions to be relaxed. By assuming, during the shift design phase, a universe of homogeneous employees without linking constraints across blocks of time, the shift design problem is made relatively simple. Thus a system for the general scheduling problem handles these less restrictive problems as a special case.

THE REAL WORLD SETTING

In real world settings, the manual scheduler (generally a supervisor) works in a highly dynamic mode. Each week, and sometimes more frequently, a new schedule must be created to reflect the altered employee availabilities and changes in the forecasted volume of business. To control costs, management frequently requires that the person-hours of work assigned must not call for a dollar expenditure out of proportion to the dollar sales forecasted.

The manual production of a schedule that respects limited and varying employee availabilities, and yet matches the requirements, is very difficult. The consequence is that the manually produced schedule generally calls for overages (too many on duty) at certain times during the day and week, and—to keep labor costs within acceptable boundaries—produces corresponding shortages (too few on duty) in others. This results in poor service: a condition which managers must seek actively to avoid in highly competitive service industries.

Producing a schedule manually also requires a good deal of time. For example, in the supermarket and fast food industries, our investigations indicate that it takes from 8 to 14 hours for a manager to schedule from 70 to 100 employees for one week, depending on the seriousness devoted to the task. The resulting schedule is likely to be substantially less than optimal. Even when all special conditions may be met (which is often not the case for schedules produced manually), shortages and overages often combine to yield less than desirable service or an inflated payroll.

IMPLEMENTING OUR APPROACH

To apply our solution method to the general employee scheduling problem, we design shifts with deference to features specified by the user, and with deference to employee availabilities. When a shift is selected, to augment the growing set of shifts which is generated with the goal of meeting the requirements, the identity of the employee to take that shift is specified. This assures that employees are only assigned shifts they are available for. In case the problem lacks a schedule that is feasible in all respects, our approach generates a schedule that nevertheless comes as close as possible to achieving feasibility, then helps the user identify alternative ways to deal with the limitations that created the infeasible situations.

As we subsequently document, our procedure succeeds in solving problems in the range of 1000 times larger than those related scheduling problems previously studied in the literature. Fundamentally, we view our procedure as an integration of management science (MS) and artificial intelligence (AI). Among the levels of procedural generality to which such an integration is relevant, from the micro level of computer coding to the macro level of global strategies, it is the higher levels that have the greatest impact on solution quality and efficiency, and account for what we believe may be unique to our approach.

Evidently, orders of magnitude of difference in the size of combinatorial problems that are successfully treated cannot be explained by clever computer coding. Intermediate level considerations of specific choice rules are more relevant to achieving such successes. The "structure" on which the primary choice rules are superimposed consists of a procedure for building and amending employee shifts which has its roots in the alternating assignment ideas of Glover [9], and which is characterized more broadly in the context of tabu search in Glover [10]. For this application, we conceptually view each stage of generating a partial (or complete) set of duty

assignments as creating a trial solution, which is modified or elaborated by transition rules: a standard framework. (Those interested in further details of the system at an intermediate level of implementation are invited to contact the authors.) The key ingredient of our approach lies in the macro level strategies, which combine the perspectives of management science and artificial intelligence in ways not commonly done. At this level, our approach consists of three main components.

First, following an MS based perspective, we develop numerical criteria for evaluating the moves that define possible transitions from one trial solution to another. We do not, however, settle on a single criterion for evaluating a particular type of move. Instead, interlinking criteria and are based on separate evaluation functions are generated which reflect both feasibility and optimality goals. Our use of multiple evaluative criteria presents a new difficulty that MS based heuristics traditionally have not had to face: the fact that a local optimum relative to one criterion may not be a local optimum relative to another. Rather than a stumbling block, we found this difficulty to be a source of fertile opportunity, leading to ways out of blind alleys encountered by other procedures. To exploit this opportunity, we formulated our approach so that each criterion was allowed to "vote" on alternative moves, initially assigning equal weight to the different votes. When a "deadlock" (local optimum) was reached, we increased the weight of those votes that would find a different solution preferable, thereby allowing the procedure to find new trial solutions. The procedure was further endowed with a memory to prevent reversing the direction in which weights were changed, but allowing the memory to decay so that choices would not be unduly influenced by decisions that should be regarded ancient history. This method for managing memory was implemented by the use of tabu lists as described in Glover [10] and Glover *et al.* [11]. Decay factors that "forgot" decisions beyond five to twelve moves earlier all succeeded in avoiding cycling and producing good choices. The combined effect of these features is implicitly to create a tolerance for "bad moves", but only to the extent required to avoid becoming mired down at local optima, resulting in a highly effective strategy.

Our second main component employed on AI based perspective to identify patterns in the ways trial solutions are configured. However, in a departure from standard AI perspective, our goal was not merely to recognize patterns but to create them. In developing this approach, we were guided by the supposition that certain configurations—as manifested in the magnitude and distribution of residual requirements, and available means of meeting them—would ultimately lead to better solutions by our tools of analysis than others. Thus, we adopted the goal of identifying moves that would lead to configurations we judged intuitively promising. This led to creating additional criteria, based on means-end analysis to arrive at exploitable patterns, independent of whether moves to attain these patterns might contribute to the objectives embodied in other criteria. These new criteria were then incorporated into our first component strategy.

The third major component of our approach was to identify significant segments of the problem, and then to subject each segment to its own sequence of solution phases. In this approach, we implicitly perform what might be called a conceptual decomposition. While the problem is an indivisible whole, we nevertheless artifically break it apart. After each round of evaluations and modifications, we put it all back together—a "Humpty Dumpty" process that characteristically yields gains throughout several repetitions. For example, after a global evaluation of an employee's availability versus needs yet to be filled over all periods (relative to the current state of the solution), bias factors are generated for and against scheduling the employee in particular period blocks. Thereupon, the blocks of time periods are treated as though independent for the purpose of generating the next move. The succeeding global evaluation then fulfills the function of restoring the links between different blocks. A similar procedure is applied to meeting union and seniority rules as the solution process evolves, temporarily decoupling these considerations by bias factors and then restoring them by global review.

Viewing our approach in terms of these macro level strategies, it is clear there is nothing to the AI nor MS oriented approaches that would not, in theory, provide support for our undertaking. In practice, however, AI and MS approaches are usually implemented in a narrower fashion. It is our impression that the approaches most often described in the literature either employ rather shallow "AI/human" intuitive techniques, though sometimes in large numbers, or employ slightly deeper MS heuristic techniques, but in very small numbers. In neither instance do we find great interlinking and overlaying of alternative criteria. Certainly, we have not seen widespread implementation of multiple

evaluation functions, integrated and controlled to overcome local optimality and cycling, nor have we seen the active use of pattern creation (in addition to recognition) using notions of exploitability instead of objective gain. Formal mathematical decomposition, although too rigid and inapplicable to discrete problems of the type we examine, has some resemblance to the third component of our approach, if one views the decomposition as susceptible to being carried out in different ways.

THE QUALITY OF THE SCHEDULE

We describe now the results of 10 computer solution tests, producing one schedule for one week, for each of 10 restaurant problems, involving 100 employees, from a real world application in the fast food industry. These tests were conducted on problems selected for benchmark runs by McDonald's Corporation Headquarters, Oak Brook, Illinois. These problems correspond to integer programming problems involving roughly from 1,000,000 to 4,000,000 variables, and from 3400 to 9000 constraints, as noted in the integer programming formulation described subsequently. A summary of the test results is provided in Table 2.

The percent of optimality figures indicated in Table 2 were arrived at as follows. With each run, no shortages were produced. That is, during no 15-minute period during the week was the number of employees assigned to be on duty less than the desired number. In one run, during three 15-minute periods throughout the week there was an excess of one employee, over and above the number desired. This does not mean that fewer employees could be used, since eliminating an employee would then create shortages in all other 15-minute periods the employee worked (assuming the same employee was on duty in these three 15-minute periods during the week). In two other runs, there was an excess of one employee, during eight 15-minute periods during the week, over and above the number desired. For the other seven the average was between three and eight.

In a perfect schedule, the shortages and the overages would all be zero for all periods during the week. In the ten problems tested, periods designated for scheduling equalled 540 (out of 672 15-minute periods for the week). Thus, in the worst case our method achieved zero shortages and zero overages for 98% (532 out of 540) of the total periods under consideration. We did not attempt to obtain the very best solutions possible by our approach (if better solutions did in fact exist), but used an automatic cut-off rule to terminate search, which accounts for the similarity of run times on problems of different sizes.

In the papers cited that report on an application in a practical setting, comparisons of solution quality are difficult due to different types of objective functions and a frequent lack of explicit performance data. Partial evidence of the quality of performance is offered in Segal [8] by reporting execution time on an IBM 360/67, which ranged from about 40 seconds to something over a minute (the maximum was not specified) for problems with 300 to 400 variables. The other papers, addressing a problem that comes closest to our general problem structure [2, 3], likewise report reliance on a mainframe computer, but do not offer computation times.

In typical real world settings for fast food stores (supermarkets, banks, reservation offices and the

Table 2. Test results in a ten case study

Case No.	No. of employees	Execution time*	No. of zero–one variables†	No. of constraints†	% Optimality‡
1		24 min	2,640,250	3400	
2		24 min	1,356,772	3400	
3		23 min	1,158,117	3400	
4		24 min	3,251,222	4732	
5		23 min	2,440,605	4732	
6		23 min	4,999,580	4732	
7		23 min	1,366,100	4732	
8		22 min	4,005,202	4732	
9		24 min	2,613,800	9004	
10		22 min	1,215,641	9004	

*All runs were executed on an IBM PC, 128 K memory.
†Determined by program counters, explained in formulation section.
‡Solutions verified to be *at least* the indicated percent of optimality.

like), large mainframes are not common. In fact, it is the increasing prevalence of inexpensive computers that makes automated scheduling in these settings practical.

Our system has accordingly been written in BASIC and implemented on microcomputer systems such as the KAYPRO II TRS 80 Model II, and the IBM PC. Because our system can handle the less general shift specifications and constraining conditions of the standard shift scheduling problem, we executed experimental runs to determine our execution time for problems whose sizes corresponded to those reported in the literature. For problem profiles corresponding to those in Refs [2, 3, 7] our solution times on an IBM PC with 128 K bytes of central memory did not exceed 50 seconds for problems of up to 500 variables. These *microcomputer* times therefore, in fact, compare favorably to the *mainframe* CPU time reported elsewhere.

It should be noted that solution times for our procedure, using the cutoff rule that achieves the indicated percent of optimality figures, do not increase anywhere near linearly with the number of integer variables. Normally the opposite effect would be expected for discrete problems such as these—i.e. an increase far worse than linear. This atypical behavior is a key factor responsible for the advance in the size of real world problems that can be handled successfully.

It is also of some interest to compare the performance of our approach to that achieved for other zero–one scheduling problems and not alone for problems closely related to the class we examined. We are prompted to make such a comparison in view of the 1984 Lanchester Prize award for a study of zero–one problems [12]. The citation for the prize underscored the significance of effectively handling these problems as: "Zero–one linear programming is a very important problem in Operations Research. Efforts to solve large problems of this type have continued for over 20 years. However, success has been elusive, and problems with hundreds of zero–one variables and no special structure could usually not be solved in reasonable computation times".

The award-winning study of Crowder *et al.* [12] represents one of the most effective attempts to solve zero–one problems optimally, and provides a major contribution for its ability to perform well on zero–one problems notably larger than those customarily handled. These problems, ten in number, ranged in size from 33 to 2756 zero–one variables. The problems were solved by reliance on a large mainframe computer (an IBM 370/168), and nearly an hour of CPU time was required to handle the largest problem. The contrasting ability to solve problems with 4,000,000 or more zero–one variables to 98% optimality suggests our approach establishes an appealing trade-off between assured optimality and problem size, and encourages us to believe that gains for solving other zero–one problems may be possible, perhaps by similar integration of management science and artificial intelligence techniques.

CURRENT USAGE

Our system has been used in several settings for over 2 years, producing schedules substantially superior to those an experienced scheduler can produce in any amount of time. Applying our system to problems involving approx. 100 employees, the user is able in one to two hours to update the forecast for the coming week, to modify the employee availabilities, and to make "manual" assignments (using the computer) to selected personnel for whom specific work schedules are wanted (e.g. managers). The computer then completes the process by designing and assigning shifts to approximate the optimal match of employees on duty to employees wanted, while respecting the constraints described above. Figure 1 shows one of the printed outputs which the user may select, indicating by means of a bar graph the shifts assigned to employees on Sunday.

No human intervention is required during the scheduling. Disregarding the improvement in the schedules produced, the process trims 6 to 12 hours off the time required each week for a supervisor to prepare a schedule manually, and this labor savings by itself can pay for the computer in a reasonable period of time.

THE INTEGER PROGRAMMING FORMULATION

A more detailed description of the conditions which were respected in the solutions cited above (the schedules produced using our approach) is as follows:

(1) The number of employees on duty (and not taking a lunch or quarter-hour break) in each 15-

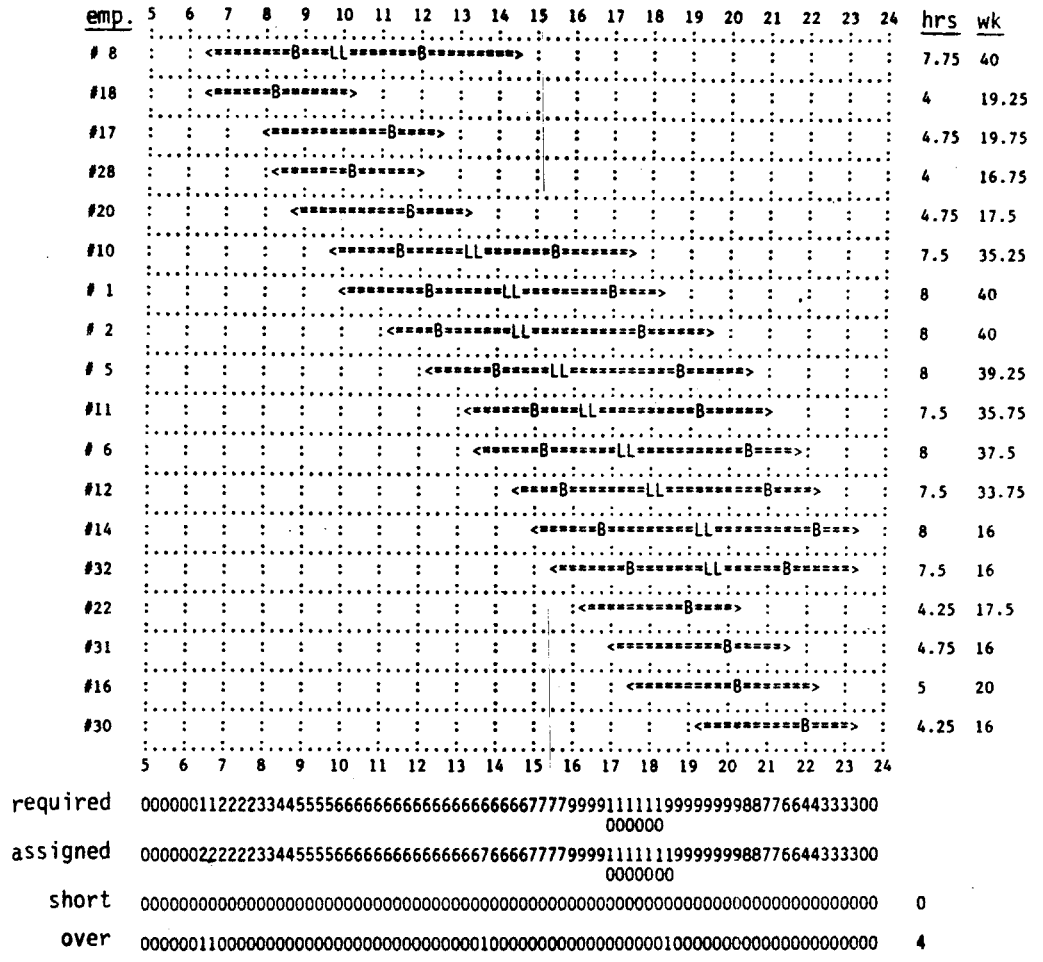

Fig. 1. Schedule for Sunday. In this example, the problem size is substantially smaller than in typical applications, both in terms of number of employees and period requirements. At 18:30 it is apparent 10 employees are assigned (No. 2, 5, 6, 11, 12, 14, 16, 22, 31 and 32) while 9 are required, yielding an average of one person for that 15 minute period. B means 15 minute break, LL means 30 minute lunch.

minute period must come as close as possible to satisfying the demand for employees in that period.

(2) All shifts assigned must be members of a feasible set specified by management rules: from 2 to 8 hours duration; with 0 to 2 quarter-hour breaks, and 0 or 1 half-hour break—depending on the duration of the shift; and with the placement of the breaks specified by reference to "windows" within which they may be moved, as specified by management.

(3) No employee can be assigned to more than one shift on any day.

(4) No employee can work less than his or her minimum required number of hours during the week, nor more than the maximum.

(5) No employee will work less than his or her minimum desired (as opposed to required) number of hours during the week unless there is not enough work to go around. In this latter case, the desired minimum is a goal where employees with greater seniorities have their goals respected first.

(6) No employee can work over 6 shifts during the week.

(7) Closing and opening rules:
   (a) No employee can "close" more than 2 nights in a week, and never two nights in succession (closing means working later than a specified hour).
   (b) No employee can close one night and "open" the next day (opening means working earlier than a specified hour).
   (c) No student can close more than once during the days Sunday through Thursday.

(8) Other things being equal, employees with more seniority must be assured more work and an earlier start.
(9) During certain periods of the day those on duty must represent a specified minimal set of skills and skill levels.

The constraints

We define:

S = the set of acceptable shift types
S_{ed} = the set of shifts which employee e is available to take on day d
P_s = the number of quarter-hour periods in shift s
$x_{esd} = 1$ if employee e is assigned shift s on day d; 0 otherwise (decision variable)
$A_{ps} = 1$ if period p is a duty period for a schedule s; 0 otherwise
D_{pd} = demand, in projected numbers of employees required to be on duty in period p of day d
u_{pd} = goal programming deviation variable for falling short of the projected demand
v_{pd} = goal programming deviation variable for exceeding the projected demand
U_e = maximum quarter-hour periods of work for employee e during the week
L_e = minimum quarter-hour periods of work for employee e during the week
G_e = the desired (goal) minimum quarter-hour periods of work for employee e during the week
y_e = a goal programming deviation variable that allows employee e to work less than G_e periods if there is not enough work to go round.

Then the preceding conditions can be modeled by the following constraints, which are numbered to provide a direct correspondence.

$$\sum_{e}\sum_{s\in S_{ed}} A_{ps}x_{esd} + u_{pd} - v_{pd} = D_{pd} \text{ for all } p \text{ and } d. \tag{1}$$

The composition of S_{ed} assures that condition (2) above is satisfied, independently of the other constraints. Therefore, we do not include a corresponding constraint (2) in our formulation.

$$\sum_{s\in S_{ed}} x_{esd} \leqslant 1 \qquad \text{for all } e \text{ and } d \tag{3}$$

$$L_e \leqslant \sum_{d}\sum_{s\in S_{ed}} P_s x_{esd} \leqslant U_e \qquad \text{for all } e \tag{4}$$

$$G_e \leqslant \sum_{d}\sum_{s\in S_{ed}} P_s x_{esd} + y_e \qquad \text{for all } e \tag{5}$$

$$\sum_{d}\sum_{s\in S_{ed}} x_{esd} \leqslant 6 \qquad \text{for all } e. \tag{6}$$

Note that conditions (7), (8) and (9) are not modeled in the preceding constraints. Modeling them is straightforward but tedius, requiring the use of additional notation without adding to the basic understanding of the requirements conveyed by their verbal description.

The objective function

A perfect schedule (as distinguished from an optimal schedule) occurs when the number of employees assigned, by quarter-hour periods throughout the week, equals the number wanted, and no employee works less than his or her minimum desired number of hours during the week, while respecting the various constraints specified. Given the non-homogeneous character of employees, the fluctuating requirements, and the attributes of the various shift types, a perfect schedule is frequently impossible.

An optimal schedule is one that minimizes some weighted function of the shortages and overages,

by quarter-hour periods throughout the week, plus a weighted function of each employee's hours below the desired minimum.

We define:

WU = a weight to penalize falling below forecasted requirements (yielding a shortage)
WV = a weight to penalize exceeding the forecasted requirements (yielding an overage)
W_e = a weight to penalize falling below the desired (as opposed to required hours for employee e to work during the week).

We seek, therefore, to:

$$\text{minimize} \quad WU = \sum_{p,d} u_{pd} + WV \sum_{p,d} v_{pd} + \sum_{e} W_e Y_e.$$

The preceding formulation is not the only or most general our procedure can be made to handle, but represents the model applied to the 10 real world problems whose solution statistics we have reported. In this application, shortages were penalized more than overages by a ratio of WU to WV of roughly 4 to 1. Minimum desired employee hours were satisfied automatically by setting W_e to an internally computed parameter based on WU, WV and the user-specified seniority factor for employee e.

The number of variables in the integer programming formulation is of course independent of such parameter choices, and is potentially immense. If all possible placements of lunch and quarter-hour breaks were included in the schedules, the total number would amount to hundreds of millions of variables (see the Appendix). However, the admissibility of all such possibilities, or even uniformly structured subsets of all possibilities such as those resulting from the standard assumption of homogeneous employees, would permit variables to be aggregated and thus effectively shrink their number to a minute fraction of those otherwise required. The simplifying homogeneity feature of Buffa [3] and [15], for example, makes it possible in these instances to deal with only about 300 to 500 variables in total.

In the setting of the general employee scheduling problem, where such implicit aggregation is not possible, we have created a component for our solution method that counts the total number of variables precisely for the real world examples we reported in Table 2. These same counting rules were applied to data of the other papers to ensure that all reported and inferred numbers of variables were derived by the same means.

CONCLUSIONS

Given the best of today's state-of-the-art codes for solving very large scale integer programming problems, the general employee scheduling problem clearly does not lend itself to practical solution by standard procedures.

It is encouraging both in view of the practical significance of the general problem, and its combinatorial complexity, that a method which combines elements of management science and artificial intelligence techniques can generate solutions of exceedingly high quality in very modest amounts of time. It is also noteworthy that such results have been achieved on a microcomputer.

The "leap" in size by comparison to previous comparable scheduling applications reported in the literature demonstrates that the effective solution of truly large-scale scheduling applications is possible, contrary to widespread belief. It is inviting to believe that similar gains may be possible for other combinatorial zero–one applications, perhaps by similar strategies.

REFERENCES

1. W. S. Brownell and J. M. Lawerre, Scheduling of workforce required in continuous operations under alternative labor policies. *Mgmt Sci.* **22**, 597–605 (1976).
2. E. G. Keith, Operator scheduling. *AIEE Trans.* **11**, 37–41 (1979).
3. E. S. Buffa, M. J. Cosgrove and B. J. Luce, An integrated work shift scheduling system. *Decis. Sci.* **7**, 620–630 (1976).
4. W. B. Henderson and W. L. Berry, Heuristic methods for telephone operator shift scheduling: an experimental analysis. *Mgmt Sci.* **22**, 1372–1380 (1976).

5. W. B. Henderson and W. L. Berry, Determining optimal shift schedules for telephone traffic exchange operators. *Decis. Sci.* **8,** 37–41 (1977).
6. V. A. Mabert and A. Raedels, The detail scheduling of a part-time work force: a case study of teller staffing. *Decis. Sci.* **8,** 109–120 (1977).
7. J. G. Morris and M. J. Showalter, Simple approaches to shift, days-off and tour scheduling problems. *Mgmt Sci.* **29,** 942–950 (1983).
8. M. Segal, The operator-scheduling problem: a network-flow approach. *Opns Res.* **22,** 808–823 (1974).
9. F. Glover, Heuristics for integer programming using surrogate constraints. *Decis. Sci.* **8,** 156–166 (1977).
10. F. Glover, Future paths for integer programming and links to artificial intelligence. *Comput. Opns Res.* **13,** 533–549 (1986).
11. F. Glover, C. McMillan and B. Novick, Interactive decision software and computer graphics for architectural and space planning. *Annls Opns Res. Algorithms Software Optim.* In press.
12. H. Crowder, E. Johnson and M. Padberg, Solving large scale 0–1 linear programming problems. *Opns Res.* **31,** 803–834 (1983).
13. J. M. Lowerre, Work stretch properties for the scheduling of continuous operations under alternative labor policies. *Mgmt Sci.* **23,** 963–971 (1977).
14. K. R. Baker and M. J. Magazine, Workforce scheduling with cyclic demands and days-off constraints. *Mgmt Sci.* **24,** 161–167 (1977).

APPENDIX

Shift Definitions Used in the Runs Cited

Shifts are from 2 to 7.5 hours in length (not counting lunch) as follows:

(1) Shifts longer than 5 hours have one lunch;
(2) Shifts from 2 to 5 hours have no lunch;
(3) All work sessions are at least 1.5 hours in length;
(4) Fifteen minute breaks are not scheduled.

In runs cited, schedules were created covering 76 periods each day Sunday to Thursday and 80 periods each day Friday and Saturday.

This shift description results in 158 different shifts that can be assigned to begin at any period. Over 76 periods and with 30-minute breaks, 7914 different shifts could be considered for an employee with unlimited availability (the number is less than 76 *158 because no shifts can begin in the last 7 periods, only one shift can begin in the eighth period from the end, and so forth). For the entire week 56,662 different shifts are theoretically possible for such an employee, yielding a potential of 5,666,200 shifts for 100 employees.

The non-homogeneity of employees, which restricted available work days and available periods within days differently for each, caused the actual number of shift alternatives to be less than this quantity, though the total number of employees sometimes exceeded 100. The counters placed in our program indicated the actual number of shifts ranged from a low of 1,158,117 to a high of 4,999,580 shifts, as noted in Table 2.

In other applications where it is desired to schedule 15-minute breaks, the number of variables involved soars. A typical 7.5 hour shift with a lunch and two 15-minute breaks has 375 variations compared with 19 variations for the 7.5 hour shift used in the above restaurant runs. We are currently experimenting with another version of our procedure which is handling over 26 million IP variables.