A SIMULATED-ANNEALING HEURISTIC FOR SHIFT SCHEDULING USING NON-CONTINUOUSLY AVAILABLE EMPLOYEES

Gary M. Thompson

Cornell University

February 1995

Author Note

Gary M. Thompson is Associated Professor of Operations Management in the School of Hotel Administration at Cornell University. He holds an M.B.A. from the University of Western Ontario and a Ph.D. in Operations Management from The Florida State University. His current research focuses on the effects on customer service of labor staffing and scheduling decisions. His previous research has appeared or is forthcoming in Decision Sciences, Computers & Operations Research, Journal of Operations Management, Management Science, Omega and Operations Research.

Acknowledgement

The author wishes to thank the anonymous referees for their helpful comments, Professors Mike Brusco and Larry Jacobs for their insights regarding simulated annealing, and Dr Samuel Raff for his patience with this article.

SCOPE AND PURPOSE

Labor scheduling, at its most basic level, involves ensuring that enough employees to serve customers are present during all the operating hours of a service facility. Service managers frequently hire service delivery personnel who are available for work only during predefined subsets of the operating day, despite the scheduling difficulties caused by such limited staff availability. In this paper, we develop a simulated-annealing heuristic for shift scheduling using employees having limited availability and, by comparing its performance to that of an efficient optimal integer programming model, demonstrate its effectiveness. Our results also allow us to make observations regarding appropriate search "neighborhoods" in labor-scheduling and other applications of simulated-annealing.

ABSTRACT

This paper presents a simulated-annealing heuristic (SAH) for developing shift schedules. We assume that each employee is available only during an individually-specified portion of the day and has individually-specified limits on the duration of shifts to which he/she can be assigned. We evaluate the effectiveness of the SAH under several "neighborhood" search parameters using a primary set of 144 test problems. These parameters include numerous criteria for adding shifts during schedule construction and for dropping shifts during schedule improvement. Our results allow us to make observations regarding appropriate search "neighborhoods" in labor-scheduling and other applications of simulated-annealing. Using a secondary set of 20 test problems we compare the SAH to an efficient optimal integer-programming model. On average, SAH's schedules are 0.29% more costly than optimal schedules but are obtained in 8.4% of the time required to generate optimal schedules.

INTRODUCTION

Services are the dominant employers in our current economy and their importance is unlikely to decline in the near future. Hence, this efficient use of labor in services is a worthy goal at the macro (country) level as well as at the micro (company) level. Labor efficiency is kept high by avoiding surpluses of on-hand staff. However, to serve customers adequately, enough employees must be present to provide the service without causing undo waiting. These aims-having sufficient but not excess numbers of employees on hand-coupled with customer demand that varies temporally across and within days, make labor scheduling a crucial but complex task facing service managers.

Much of the published labor scheduling research has contained restrictive assumptions defining acceptable shifts. Because the number of valid shifts grows combinatorially, common test conditions have included few eligible periods in which breaks may be taken and planning periods of 0.5 h or longer. Although these restrictions may be legitimate in specific instances, many service environments have considerably more scheduling flexibility. In this paper, scheduling flexibility is understood to be the extent and tightness of restrictions defining acceptable schedules. With a few exceptions (Glover et al. [1], Loucks and Jacobs [2], Love and Hoey [3], Thompson [4]), the labor scheduling literature has either explicitly or implicitly assumed employees are available at any time during the operating day. This is a valid assumption when labour staffing decisions are made (decisions regarding staff size, for example), but it becomes less so once individual employees are hired. In a large number of service environments, many employees are available for work only at limited times, a result of their commitments or desires. Consequently, work schedules must be developed in deference to employee availability. We consider availability as having two dimensions. First, the specific periods during the

operating day that an employee is available limits the shifts to which he/she can be assigned. Second, each employee may have individual limits on the length of shift to which he/she can be assigned, that are independent of the periods in which the employee is available for work.

In this paper we present a simulated-annealing (SAH) for developing shift schedules using homogeneously-skilled employees. We assume that each employee: (1) is available only during an individually-specified portion of the operating day; and (2) has individually-specified limits on the length of shift that he/she may work. Using a primary set of 144 test problems we evaluate alternate neighborhood search patterns in the SAH. We use a secondary set of 20 test problems to compare the performance of the SAH to an efficient integer programming model.

The organization of the remainder of this paper is as follows. We review relevant literature, describe the simulated-annealing heuristic; identify experimental objectives; specify the test environments; and provide and discuss results. We conclude the paper with suggestions for future research.

LITERATURE REVIEW

Dantzig [5] developed the original integer programming (IP) formulation of the shift scheduling problem:

$$minimize Z = \sum_{n \in N} c_n x_n, \quad (1)$$

subject to

$$\sum_{n \in N} a_{np} x_n \geq r_p \text{ for } p \in p, \quad (2)$$

$$x_n \geq 0 \text{ and integer for } n \in N, \quad (3)$$

where

n = index for shifts

N = set of unique shifts

c_n= cost of having an employee work shift n

p = index for periods

P = set of planning periods,

x_n= number of employees assigned to shift n

a_{np}= $\begin{cases} 1, \text{if period } p \text{ is a working (nonbreak) period of shift } n, \\ 0, \text{otherwise,} \end{cases}$

r_p= number of employees needed to provide the desired level of customer service in period p.

We shall call the model defined by (1)-(3) UASSM-the "unlimited-availability shift-scheduling model". UASSM's objective: (1) is to minimize the cost of the schedule, subject to the restrictions that sufficient employees are present in each planning interval (2), and that there are no partial assignments of employees to shifts (3).

Because optimal solutions are difficult to obtain to UASSM when realistic numbers of shift derivatives are used, heuristics have played a major role in scheduling research. Started-from scratch heuristics have been used by Buffa et al. [6], McGinnis et al. [7], Henderson and Berry [8] and Bechtold and Showalter [9]. Henderson and Berry [8], Keith [10], Krajewski et al. [11], Mabert and Watts [12], Morris and Showalter [13], Bailey and Field [14], Holloran and Byrn [15], and Showalter and Mabert [16] derived initial schedules by solving linear programming (LP) relaxations of IP problems. These initial schedules were, in most cases, subsequently heuristically improved. More recently, Bechtold et al. [17] rigorously evaluated a wide range of scheduling heuristics, and found the procedures of Keith [10] and Morris and Showalter [13] to perform best. Brusco and Jacobs [18] developed a simulated-annealing

heuristic for the tour-scheduling problem that yielded better schedules than the Keith [10] and Morris and Showalter [13] heuristics.

With a few exceptions, which we will discuss, the labor scheduling literature has either explicitly or implicitly assumed employees are available at any time during the operating day. Glover et al. [1] were the first to report on a procedure, a construction/improvement heuristic, for scheduling employees with restricted availability. Love and Hoey [3] developed two network flow models to be used consecutively for scheduling employees having limited availability. The primary model scheduled shifts with the objective of minimizing surplus staffing, while the secondary model assigned employees to the shifts scheduled by the first model using various, differentially-weighted objective components. As the primary model treats all employees as being continuously available, there is no guarantee that schedules it generates are *availability-feasible*, that is, that all scheduled shifts can be uniquely assigned to individual employees. Loucks and Jacobs [2] presented a model and heuristic for scheduling employees having different skills and limited availability to different tasks. A key weakness of their model is its failure to schedule breaks.

Thompson [4] developed a model which, by implicitly matching employees to shifts, enabled a large reduction in the number of variables required to model limited employee availability. This reduction in the number of variables required to model limited employee availability. This reduction in the number of variables in turn resulted in significantly faster solutions to an LP relaxation of the implicit-matching model than to an LP relaxation of an explicit-matching extension of (1)-(4). To facilitate the implicit matching, Thompson [4] defined a *region* to be all those shifts which could be staffed by the same subset of employees. These regions were disjoint, and their union was the set of all shifts. With binary variables defined for

each of the regions an employee could work, the implicit matching of employees to shifts was imposed by first mapping employees onto regions, and then mapping regions onto shifts. Our stimulated-annealing heuristic makes use of an expanded definition of regions that allows individually-specified minimum shift durations, while we use a modified form of the efficient IP model developed by Thompson [4] as a comparison procedure.

Combining and extending the models of Keith [10] and Thompson [4] gives the mathematical programming model of the shift-scheduling problem we consider, which we call IASSM:

$$minimize\ Z = \sum_{e \in E}\ \sum_{n \in S_e}\ c_n x_{en} + \sum_{p \in P}(\sum_{j=1}^{r_p} k_{pj} u_{pj} - \sum_{j=1}^{\infty} b_{pj} 0_{pj}\ ) \quad (4)$$

subject to

$$\sum_{e \in E}\ \sum_{n \in S_e} a_{np} x_{en} + \sum_{j=1}^{r_p} u_{pj} - \sum_{j=1}^{\infty} 0_{pj} = r_p \text{ for } p \in P \quad (5)$$

$$\sum_{n \in S_e} x_{en} \leq 1 \text{ for } e \in E, \quad (6)$$

$$x_{en} \in \{0,1\} \text{ for } e \in E \text{ and } n \in S_e, \quad (7)$$

$$u_{pj} \in \{0,1\} \text{ for } p \in P \text{ and } j = 1, \dots, r_p, \quad (8)$$

$$O_{pj} \in \{0,1\} \text{ for } p \in P \text{ and } j = 1, \dots, \infty, \quad (9)$$

where

$e =$ index for employees

$E =$ set of available employees,

$S_e =$ set of shifts for which employee *e* is available to work (defined by the specific periods in which employee *e* is available for work and by the minimum and maximum lengths of shifts to which employee *e* can be assigned),

$$x_{en} = \begin{cases} 1, if\ employee\ e \text{ is assigned to shift } n, \\ 0, \text{otherwise}, \end{cases}$$

$$u_{pj} = \begin{cases} 1, \text{if period } p \text{ is understaffed by at least } j \text{ employees,} \\ 0, \text{otherwise,} \end{cases}$$

$$o_{pj} = \begin{cases} 1, \text{if period } j \text{ } is \text{ overstaffed by at least } j \text{ employees} \\ 0, \text{otherwise,} \end{cases}$$

$k_{pj} =$ incremental monetary cost of increasing the understaffing in period p from j - 1 to j employees, where $1 \leq k_{p1} \leq k_{p2} \leq \cdots \leq k_{p,r_p,}$

$b_{pj} =$ the incremental monetary benefit of increasing the overstaffing in period p from j- 1 to j employees, where $1 > b_{p1} \geq b_{p2} \geq \cdots$

with p, P, a_{np}, c_n and r_p as earlier defined.

IASSM, or the "individual-availability shift-scheduling model", differs from UASSM in several regards. First, the model explicitly accounts for the availability of individual employees by explicitly matching employees to shifts. A matching of employees to shifts is necessary to ensure the availability-feasibility of a schedule. Second, though the desired staffing levels (ther_p) are identical in UASSM and IASSM, IASSM allows shortfalls in each period. Though typically undesirable, shortfalls of employees may be unavoidable if the limited availability of employees precludes satisfying employee requirements in some periods. Third, IASSM recognizes the well-established queuing phenomenon of diminishing marginal improvement in service with increasing numbers of staff. IASSM uses binary variables to measure both the shortage and surplus of employees, by period. Associated with these binary variables are nonlinear, monotonically decreasing cost or value coefficients. The next section describes the heuristic we developed to solve IASSM.

SIMULATED-ANNEALING HEURISTIC (SAH)

Overview

Figure presents a pictorial representation of our simulated-annealing heuristic (SAH). In developing the heuristic, we drew heavily on the work of Brusco and Jacobs [18). The SAH uses three separate schedules-the best found so far, an incumbent schedule, and a trial schedule. All schedule modifications are made to the trial schedule. The incumbent schedule essentially serves to undo ineffective changes made to the trial schedule. To ensure availability-feasibility, all the routines of SAH maintain an explicit matching of employees to shifts in all three schedules. Six routines comprise the SAH: INITIALIZE, CONSTRUCT, NIXREDUNDANT, FINETUNE, EVALUATE and PERTURB. INITIALIZE initializes the parameters used in the heuristic. CONSTRUCT repetitively adds worthwhile shifts to the schedule. NIXREDUNDANT eliminates shifts that reduce the attainment of the objective (4). FINETUNE seeks to improve the solution by making minor changes in the scheduled shifts, such as in break timing, for example. EVALUATE appraises the solution and, if necessary, replaces the incumbent and best solutions. PERTURB disturbs the solution by dropping selected shifts. CONSTRUCT, NIXREDUNDANT, FINETUNE, EVALUATE and PERTURB are used repetitively until the SAH exceeds its specified executive time. We describe SAH's routines in greater detail below.

INITIALIZE

In INITIALIZE, parameters specific to the SAH are initialized. These parameters are *sahpt*, the temperature, which is initialized to 1.25; *saphl*, the temperature length, which is initialized to 100; *sahpr*, the cooling ratio, which is initialized to 0.92; Z_{best}, the objective value of the best solution found to date, which is initialized to∞; and *L*, the iteration counter, which is initialized to 0.

CONSTRUCT

CONSTRUCT scans all candidate shifts, keeping a list of the five having the highest values on a specific set of criteria. Candidate shifts are those that can be assigned to an employee and that reduce the cost of the schedule. One of the objectives of the research is to evaluate the effectiveness of various rules for selecting shifts. These rules will be discussed later, but, by way of example, one of these rules selects the shift that covers the period having the greatest understaffing, and breaks ties by selecting the shift with the greatest average understaffing for the periods that it covers.

CONSTRUCT randomly selects one of the five best shifts and adds it to the schedule. CONSTRUCT continues to add shifts until all available employees have been assigned shifts or the schedule cost cannot be reduced by adding any shift. For a shift to be added to the schedule, a free (unassigned) employee is necessary. If any of the following conditions hold, then the shift is staff able: (1) if a free employee can work the new shift; (2) if a free employee can be assigned to a second employee's shift and the second employee can work the new shift; or (3) if a free employee can work a second employee's shift, the second employee can work a shift assigned to a third employee, and the third employee can work the new shift. If multiple employees can feasibly be assigned to a new shift, the employee available for the shortest time is assigned to it.

Full-time (FT) employees typically allow for a lower flexibility in constructing schedules than do part-time (PT) employees. Because of this, if the number of PT employees is limited, it can be useful to schedule a minimum number of FT shifts prior to scheduling PT shifts. This minimum is determined by assuming all available PT employees will work their shortest possible shift. Any remaining requirements must then be staffed by FT employees, who are assumed to be

able to satisfy requirements in each of the periods they work. The requirement of a minimum number of FT shifts is removed following completion of the initial CONSTRUCT process.

To reduce both schedule construction and improvement times, the SAH makes use of the concept of regions and the region-definition algorithm presented by Thompson [4]. Regions are particularly useful in checking the availability-feasibility of changes to the schedule. This is because there are typically far fewer regions than shifts, and because all shifts within a region have the same availability-feasibility status.

The initial use of CONSTRUCT iteratively builds a schedule, one shift at a time. The final schedule constructed in this manner is not likely to be optimal since any shift selected affects all later shift choices. The goal of all subsequent actions of the SAH is to make improvements to the initial schedule. We describe these actions in the following subsections.

NIXREDUNDANT

Upon completion of CONSTRUCT, the schedule may contain shifts which may be dropped from the schedule while simultaneously lowering (4). NIXREDUNDANT identifies and eliminates such shifts. It works by scanning all shifts in the schedule, and recording the three shifts that would most reduce the schedule cost, should they be dropped. One of these three shifts is selected at random and dropped from the schedule. The scan-and-drop process repeats until no shift can be dropped without increasing the cost of the schedule.

FINETUNE

The four distinct routines of FINETUNE operate in a single-pass sequence: (1) move breaks (BREAK) and (2) offset (OFFSET), (3) lengthen (LENGTHEN), or (4) shorten shifts (SHORTEN). Three of the FINETUNE routines-OFFSET, LENGTHEN and SHORTEN have both simple and advanced actions.

In BREAK, each shift has its break checked in the full range of allowable positions and the break having the best new position is moved. BREAK concludes when no break can be moved to a position that lowers the cost of the schedule.

In the simple actions of OFFSET, displacements of up to 2 h are considered but break timing is not altered. In the advanced option, displacements of up to 1.5 h are evaluated simultaneously with changes in break timing. OFFSET implements the shift/break displacement yielding the greatest schedule cost reduction and terminates when no shift movement lowers the schedule cost. Both LENGTHEN and SHORTEN make three types of passes searching for an improved schedule. As these routines are very similar, we will only describe LENGTHEN. In the first of three types of passes, LENGTHEN checks all admissible shift extensions without altering breaks. When no such extension reduces the schedule cost, LENGTHEN makes the second type of pass. In this pass, it considers altering the position of a meal break simultaneously with lengthening the shift. After making a single schedule improvement using the second type of extension, LENGTHEN again considers shift changes of the first type. In the third type of pass, LENGTHEN considers altering the position and, if appropriate, incrementing the length of meal breaks simultaneously with lengthening each shift. For all three types of changes, LENGTHEN implements the shift extension and break length and/or position change that yields the greatest schedule cost reduction per unit of length change. After making a single schedule improvement using the third type of extension, LENGTHEN reconsiders shift changes of the first type. It terminates when no extension of any type reduces the schedule cost.

When modifying a shift in the schedule, a free employee is not always necessary to ensure the schedule is availability-feasible. If the shift being modified falls in the same region, or falls in a region that can be staffed by the employee currently assigned to the shift, then the

employee assigned to the shift can continue to staff it. In the event that neither of these conditions hold, the employee currently assigned to the shift is temporarily freed-up, and then the modified shift is treated as if it were a new shift being scheduled. Then, if any of the earlier-listed conditions for new shift applies, the modified shift is staffable.

EVALUATE

In EVALUATE, the trial schedule is evaluated relative to the best solution found to date and to the current incumbent solution. Let Z_{trial}equal the cost of the current trial schedule, $Z_{incumbent}$ equal the cost of the incumbent schedule, and x be a uniform (0, 1) random variate. If $Z_{trial}$is less than Z_{best}, then the current trial solution replaces both the best and incumbent solutions; if Z_{trial} is less than $Z_{incumbent}$ or if *x* is less than exp ($Z_{incumbent} - Z_{trial}$) ÷ sahpt, then the trial solution replaces the incumbent solution; otherwise, the incumbent solution replaces the trial solution.

Next, EVALUATE increments *L* by 1. If *L* equals *sahpl*, the *sahpt* is decreased by the proportion of (1 - *sahpr*) and *L* is reset to zero. Finally, EVALUATE calculates the elapsed time. If the elapsed time is less than the 30 s limit we imposed, SAH begins the PERTURB process; otherwise SAH terminates.

PERTURB

PERTURB attempts to identify shifts which, if dropped from the schedule will, through the addition of other shifts-allow for further improvement in the schedule. PERTURB drops 1/3 of the shifts in the schedule, but at least six, and no more than 10, in any iteration. PERTURB scans all shifts in the schedule, recording the three with the highest values on a criteria set. PERTURB randomly selects one of the three top shifts and drops it from the schedule. After dropping the specified number of shifts, PERTURB passes control to CONSTRUCT.

One of the objectives of the research is to evaluate the effectiveness of various rules for selecting shifts to be dropped. These rules will be discussed in the next section, but, by way of example, one of the rules selects the shift that covers the period having the greatest overstaffing, and breaks ties by selecting the shift that yields the greatest improvement in the objective (4).

EXPERIMENTAL OBJECTIVES

We had four goals in testing the SAH:

1. To identify better performing rules for adding shifts during CONSTRUCT.
2. To identify better performing rules for dropping shifts during PERTURB.
3. To evaluate the benefit of including no, simple, or both simple and advanced actions in FINETUNE.
4. To compare SAH's performance to that of an optimal procedure.

We describe these goals below.

Our first goal was to evaluate a wide range of criteria for selecting shifts to add to the schedule in CONSTRUCT. Table 1 presents definitions of and literature references for the criteria used in selecting shifts. The primary criteria are: the maximum understaffing in any working period of a shift; the average understaffing per working period; the reduction in schedule cost that occurs when the shift is added, divided by the number of working periods; the improvement in schedule smoothness that result from the addition of the shift; and a measure of restrictiveness caused by the limited availability of employees.

Table 2 summarizes how the SAH used these criteria in 20 prioritized sets of "add-rules." Each add-rule (AR) consists of a primary criterion with various tie breaking criteria. The tertiary and quaternary tie-breakers are the same for all ARs: choose the shift with the greatest number of

working periods; and alternate between choosing the shift with the earliest or latest starting time, respectively. Working periods are defined to be those periods in a shift when work occurs, i.e. breaks are excluded.

Our second goal was to evaluate rules for selecting shifts to be dropped from the schedule in PERTURB. Table 3 presents definitions of and literature references for the criteria the SAH uses to identify shifts for dropping. The SAH used three criteria sets, or "drop-rules" (DRs). The DRs use a primary criteria: (1) the maximum overstaffing in any working period of a shift; (2) the average overstaffing in the working periods of a shift; and (3) the schedule cost reduction occurring when the shift is dropped, averaged across the number of its working periods. All break ties by dropping the shift yielding the greatest reduction in schedule cost.

The third goal was to evaluate the effectiveness of the SAH implemented without FINETUNE (FOl), with only simple FINETUNE actions (F02), and with both simple and advanced FINETUNE actions (F03). The simple actions of FINETUNE including moving breaks, and offsetting, lengthening, and shortening shifts without moving breaks. The advanced actions of FINETUNE include the simple actions, as well as offsetting, lengthening and shortening shifts while simultaneously moving or altering the length of breaks.

Addressing goals one through three factorially resulted in 180 implementations of the SAH (arising from 20 ARs, three DRs and three FINETUNE options). We tested the 180 implementations of SAH on a set of 144 problems as described in the next section.

Our fourth goal was to evaluate the best implementations of SAH compared to optimal schedules generated using commercially-available software for solving IP models. We conducted this evaluate using a set of 20 test problems, as described in the next section.

TEST ENVIRONMENTS

We utilize two test sets in this paper. The first, Test Data Set 1 (TDSl), was comprised of 144 separate problems and used in Experimental Goals 1-3. The second, Test Data Set 2 (TDS2), was comprised of 20 problems and used in Experimental Goal 4.

A 15 h operating day, comprised of 60 15-min planning intervals, was used in all TDSl problems. Allowable shifts for this problem set were defined by the restrictions given in Table 4. These shift restrictions and the 15 h operating day resulted in a maximum of 4027 unique shifts for all TDS 1 problems, ignoring availability-feasibility.

For TDS2, we randomly selected 20 of the 144 problems in TDSl and converted the problems to 30-min planning intervals from the original 15-min periods. We made the conversion since we were unable to solve the IP model using the original 15-min planning periods. The restrictions defining acceptable shifts were essentially the same as those reported in Table 4, but converted to 30-min planning intervals.

Employee requirements data

A limitation of much of the published scheduling research is the restricted range of staffing requirements (demand) patterns used in testing scheduling procedures. In TDSl, five types of employee requirements patterns were used: uniform, unimodal, bimodal, trimodal and pattern-less (i.e. random). Each of these patterns was generated with mean employee requirements of 5 and 15 employees and, excepting the uniform requirements, with coefficients of variations of approx. 0.12 and 0.30. This resulted in TDSl having a total of 18 distinct staffing requirements patterns representing the range of customer demand occurring in service organizations. Figure 2 illustrates all of the patterns save the uniform.

Employee availability data

Three types of employee availability characteristics were used, each having two levels: the average number of hours of availability for: (1) FT; (2) PT employees; and (3) the requirements coverage ratio. Combining the 18 employee requirements patterns with the eight employee availability combinations yielded the 144 problems in TDSI.

The low and high levels of average FT employee availability were 40 and 55 15-min periods. For PT employees, the comparable values were 25 and 50 15-min periods, respectively. The total number of periods of availability for individual employees was uniformly distributed about the averages.

For each employee, we randomly selected a contiguous subset of periods in which they would be available for work. FT employees all had minimum and maximum acceptable shift lengths of 36 periods. Each PT employee had a randomly selected minimum acceptable shift length of between 12 and 19 periods and a randomly selected maximum acceptable shift length of between 31 and 36 periods.

The low and high levels of the requirements coverage ratio were 1.25 and 2.50. The requirements coverage ratio is given by the maximum hours that could be worked by all employees, divided by the total labor hours required to satisfy customer demand. Coverage values close to one represent a low degree of scheduling flexibility, while higher coverage values represent higher scheduling flexibility.

For all problems, the relative abundance of FT employees was fixed at 0.2. We define this measure to be the ratio of the maximum hours that could be worked by FT employees divided by the maximum labor hours that could be worked by all employees. A level of 0.2 restricts the FT employees to comprising no more than approx 20% of the total hours scheduled.

We define schedule generation time to be the average, across all problems in a test problem set, of the time required on a Gateway P5-90 to perform all activities associated with generating a schedule. The SAH was coded in FORTRAN and compiled using Microsoft FORTRAN PowerStation 1.0 [19]. The IP model was generated using GAMS [20] and solved using OSL [21].

Test Data Set 1

Table 5 presents the 20 best implementation of the SAH. AR2-DR3-F03 yielded the best schedules overall, with a mean schedule cost of 964.516. This was followed closely by AR2-DR1-F03, with a mean schedule cost of964.553. It is noteworthy that the best 12 implementations of the SAH all used the advanced FINETUNE option (F03).

In 52 of the 60 different AR/DR combinations, the SAH using F03 yielded a lower average schedule cost than the SAH using F02 and in turn the SAH using F02 yielded a lower average schedule cost than the SAH implemented with FOL We believe the effectiveness of FINETUNE is due to the range of alternate shifts that can be quickly checked. That is, it is more efficient to use a process like FINETUNE to quickly check if a small, incremental change in a shift is beneficial than it is to search for a small improvement by dropping shifts from the schedule and adding other shifts. We discuss the implications of this result in the next section.

Overall, the SAH implemented using AR2 yielded the best schedule, with the nine implementations of the SAH that use AR2 yielding an average schedule cost of 977 .936. This was followed by ARIO at 992.919, ARl 1 at 993.146, ARI at 994.438 and AR9 at 1008.010. AR2 uses the maximum understaffing in any working period as the primary criterion for adding a shift to the schedule.

On average, the 180 implementations of SAH completed 312.95 iterations for the 72 problems having a mean requirement of 5 employees. By comparison, the 180 implementations of SAH completed 95.88 iterations for the 72 problems having a mean requirement of 15 employees. We can thus estimate the number of iterations the SAH can average, per second, as a function of the mean employee requirement *mer* as:

$$5.58 \times mer^{-1.075}$$

The implication of equation (10) is that one should consider increasing the time allocated to the SAH, in a slightly non-linear fashion, as the mean employee requirements increase.

Based on our initial results, we then investigated the reduction in schedule costs that would occur as the best schedule was chosen from multiple schedules generated using various implementations of the SAH. To do this, we sequentially chose the SAH implementation making the biggest successive improvement as the next combination to use in the development of multiple schedules. Table 6 presents the results of doing this on the TDSl problems. With the average cost of schedules generated by AR2-DR3-F03 normalized to 100%, the average schedule cost modestly fell to 99.81, 99.73 and 99.69% as the best schedule was selected from two schedules (generated with AR2- DR3-F03 and AR20-DR3-F03), three schedules (generated with AR2-DR3-F03, AR20-DR3-F03 and AR16-DR1-F03) and four schedules (generated with AR2-DR3-F03, AR20-DR3-F03, AR16-DR1-F03 and AR19-DR2-F03), respectively. Even more modest schedule cost reductions were observed as more than four schedules were generated for each test problem.

Interestingly, four of the five criteria for adding shifts and all three of the criteria for dropping shifts (see Tables 2 and 3) were represented in the four sequentially-best SAH implementations (AR2-DR3-F03, AR20-DR3-F03, AR16-DR1-F03 and AR19-DR2-F03). Thus,

the performance of the SAH when used to generate multiple schedules is likely to be significantly more robust than when only a single schedule is generated for a problem.

Based on the results reported in Table 6, we decided to evaluate five additional implementations of the SAH. The first, F AST2/3/3, was a fast implementation of AR2-DR3-F03. FAST2/3/3 only calculates the parameter values relevant to AR2 and DR3, instead of all parameter values reported in Tables 1 and 3 as for the standard SAH. The second, CMB, was an implementation of the SAH that sequentially uses the top four forms of the SAH (AR2-DR3-F03, AR20-DR3-F03, AR16-DR1-F03 and AR19-DR2-F03), switching between forms at each iteration. The third, SWT, was another implementation using the top four forms, but SWT switches between forms only at multiples of three iterations without an improvement in the schedule. The fourth, EQTIME, uses AR2-DR3-F03, AR20-DR3-F03, and AR16-DR1-F03 for 10s each. The fifth, PRTIME, uses AR2-DR3-F03 for 15 s, AR20-DR3-F03 for 10 sand AR16-DR1-F03 for 5 s. As with the original implementations, the five extra were each limited to 30 s per problem. Table 7 reports the mean schedule costs of the five additional implementations. All five new implementations yielded modestly better schedules than the original implementation of AR2-DR3-F03. PRTIME yielded the best schedules, with a mean cost of 99.84% of that for the original implementation of AR2-DR3-F03.

Test Data Set 2

Table 7 also compares schedules generated with the SAH to those generated with Thompson's [4] efficient IP model on the 20 problems in TDS2. FAST2/3/3's average schedule cost was 0.317% above the optimal schedule cost of 329.164. The best implementation of the SAR was PRTIME, which yielded schedules costing only 0.285% above optimal. Generating the optimal schedules required 357.98 s, on average, while the SAR implementations each used only

30 s, per problem. Thus, the best implementation of the SAR yielded schedules only 0.29% more costly than optimal schedules in 8.38% of the time required to generate optimal schedules.

On TDS2, therefore, SAR's performance must be judged as very good relative to the cost of and time required to generate optimal schedules. SAR's performance/schedule generation time ratio would likely be better with more complex problems, although this remains to be tested (the generation of optimal solutions becomes very difficult with more complex problems like those found in TDSl).

SUGGESTIONS FOR FUTURE RESEARCH

We have described a simulated-annealing heuristic for developing shift schedules using homogeneously-skilled employees available at restricted times. Simulated-annealing heuristics likely have wide practicality for service organizations since: (1) the SAR requires no specialized solution software (such as that required to solved LP or IP models); (2) the SAR develops schedules rapidly on a microcomputer even when a good deal of scheduling flexibility exists; (3) the cost of schedules generated by SAR very nearly equaled the cost of optimal schedules; (4) the SAR generated schedules in much less time than was required to generate optimal schedules; and (5) the robustness of SAH is greater when complementary criteria are used together, with the most beneficial assigned more of the available time.

Given SAH's strong performance, it is useful to consider improvements to it which reduce both schedule generation times and mean schedule costs. It is possible, though unlikely, that a procedure could be developed which would prove more effective than using regions to determine the availability-feasibility of changes to the schedule. Second, the range of add- and drop-rules could be expanded. However, given the close-to-optimal performance of the best SAH

implementations, such an investigation is likely to yield only very small incremental reductions in mean schedule costs.

Although this research has assumed all employees have equivalent skills, in reality skill levels vary. Further, employees may be required to perform multiple tasks. It would seem natural, then, to consider two direct extensions to this work: (1) the scheduling to a single task of non-continuously available, heterogeneously-skilled employees; and (2) the scheduling to multiple tasks of non-continuously available, heterogeneously-skilled employees having heterogeneous qualifications. It is quite possible that the concept of regions could be extended to the multi-task environment to facilitate the checking of availability and task assignment feasibility.

Some may argue that tour scheduling is more relevant than shift scheduling. Tour scheduling generates a week-long schedule, commonly assuming that employees work the same shift each day they work. The SAH approach described here may offer possibilities for extension into the tour scheduling environment. In particular, since we assumed that each employee has individually-specified limits on the duration of shift he/she may work, the SAH would be useful in a disaggregation approach to tour scheduling. Thompson [22] provides suggestions for how one might do this. One approach would begin by solving the daily shift scheduling problems independently. It would then use the resultant shifts in an assignment model that includes any relevant across-day restrictions. For example, such restrictions might seek to assign each employee a total work content between specified minimums and maximums. A second approach would be to rank the days from least flexible to most flexible. In developing the rankings, one could use measures similar to the availability index reported in Table 1. Then the daily shift

scheduling problems could be solved, beginning with the least flexible and ending with the most flexible. Any across-day restrictions could be incorporated, as appropriate, in the daily problems.

Our results have several implications for future research using simulated annealing. First, the strong performance of the SAH implementations using the advanced FINETUNE options suggest that similar actions be investigated in other problems. For example, in general set-covering problems, it may prove beneficial to evaluate the "morphing" of a variable in solution into variables that are not in solution. In labor tour scheduling, for example, "morphing" would encapsulate changes in shift duration, but also changes in the days worked. Second, our results suggest that better performance can be achieved if complementary criteria are used to generate the neighborhood in simulated annealing. Third, our findings suggest that one allocate time to various criteria in relation to the benefit of the criteria yield.

REFERENCES

1. F. Glover, C. McMillan and R. Glover, A heuristic programming approach to the employee scheduling problem and some thoughts on 'Managerial robots'. *J. Opers. Mgmt* 4, 113-128 (1984).
2. J. S. Loucks and F. R. Jacobs, Tour scheduling and task assignment of a heterogeneous work force: a heuristic approach. *Decis. Sci*. 22, 719-738 (1991).
3. R.R. Love, Jr and J.M. Hoey, Management science improves fast food operations. *Interfaces* 20, 21-29 (1990).
4. G. M. Thompson, Shift scheduling when employees have limited availability: an L. P. approach. *J. Opers. Mgmt* 9, 352-370 (1990).
5. G. B. Dantzig, A comment on Edie's Traffic delays at toll booths'. *Opers Res* 2, 339-341 (1954).
6. E. S. Buffa, M. J. Cosgrove and B. J. Luce, An integrated work shift scheduling system. *Decis. Sci*. 7, 620-630 (1976).
7. L. F. McGinnis, W. D. Culver and R.H. Deane, One- and two-phase heuristics for workforce scheduling. *Computers Ind. Engng* 2, 7-15 (1978).
8. W. B. Henderson and W. L. Berry, Heuristic methods for telephone operator shift scheduling: an experimental analysis. *Mgmt Sci* 22, 1372-1380 (1976).
9. S. E. Bechtold and M. J. Showalter, A methodology for labor scheduling in a service operating system. *Decis. Sci*. 18, 89-107 (1987).
10. E.G. Keith, Operator scheduling. *AIIE Trans*. 11, 37-41 (1979).
11. L. J. Krajewski, L. P. Ritzman and P. McKensie, Shift scheduling in banking operations: a case application. *Interfaces* 10, 1-8 (1980).

12. V. A. Mabert and C. A. Watts, A simulation analysis of tour-shift construction procedures. *Mgmt Sci.* 28, 520-532 (1982).
13. J. G. Morris and M. J. Showalter, Simple approaches to shift, days-off and tour scheduling problems. *Mgmt Sci*. 29, 942-950 (1983).
14. J. Bailey and J. Field, Personnel scheduling with flexishift models. *J. Opers Mgmt* 5, 327-338 (1985).
15. T. J. Holloran and J. E. Byrn, United Airlines station manpower planning system. *Interfaces* 16, 39-50 (1986).
16. M. J. Showalter and V. A. Mabert, An evaluation of a full-/part-time tour scheduling methodology. *Int. J. Opers Product Mgmt* 8, 54-71 (1989).
17. S. E. Bechtold, M. J. Brusco and M. J. Showalter, A comparative evaluation of labor tour scheduling methods. *Decis. Sci*. 22, 683-699 (1991).
18. M. J. Brusco and L. W. Jacobs, A simulated annealing approach to the cyclic staff-scheduling problem. *Nav. Res. Logist*. 40, 69-84 (1993).
19. Microsoft Corporation, *Microsoft FORTRAN PowerStation 1.0. Microsoft Corporation*, Redmond, WA (1993).
20. A. Brooke, D. Kendrick and A. Meeraus, *GAMS, Release 2.25, A User's Guide*. The Scientific Press, South San Francisco, CA (1992).
21. IBM Corporation. *Optimization Subroutine Library*, Release 2. Kingston, NY (1991).
22. G. M. Thompson, A comparison of techniques for scheduling non-homogeneous employees in a service environment subject to non-cyclical demand. Unpublished Ph.D. Dissertation, The Florida State University (1988).

Table 1. Criteria for selecting shifts to be added to the schedule.

Selection criterion	Word description	Operational definition	Closest earliest reference
mu_n	maximum understaffing occurring during the working periods of shift n	$mu_n = \max_{\{p \mid a_{np}=1\}} (d_p)$	Morris and Showalter [13] (mu)
$au_n = s_n \div l_n$	average understaffing per working period in shift n	$s_n = \sum_{\{p \mid a_{np}=1\}} (g_p d_p)$	Keith [10] (s_n), Showalter and Mabert [16] (au_n)
$acr_n = cr_n \div l_n$	reduction in schedule cost per working period of shift n	$cr_n = \sum_{\{p \mid a_{np}=1\}} (g_p k_{p,d_p} + h_p b_{p,-d_{p+1}}) - c_n$	Buffa *et al.* [6] (cr_j)
smv_n	the improvement in schedule smoothness—the sum across all periods of the absolute change in net staffing levels between adjacent periods—that occurs when shift n is added to the schedule	$smv_n = \begin{pmatrix} \lvert d_{q1} - d_{q1-1} \rvert - \lvert d_{q1} + 1 - d_{q1-1} \rvert \\ + \lvert d_{f1} - d_{f1+1} \rvert - \lvert d_{f1} + 1 - d_{f1+1} \rvert \\ + \lvert d_{q2} - d_{q2-1} \rvert - \lvert d_{q2} + 1 - d_{q2-1} \rvert \\ + \lvert d_{f2} - d_{f2+1} \rvert - \lvert d_{f2} + 1 - d_{f2+1} \rvert \end{pmatrix}$	Glover *et al.* [1] (smoothness as a criterion, but no operational definition)
avl_n	availability index for shift n (undefined for those shifts which cannot be staffed	$avl_n = \left[\sum_{p \in P} \left(a_{np} d_p \div \sum_{e \in E} v_{ep} \right) \right] \div l_n$	N/A

where:
l_n = number of work periods in shift $n = \sum_{p \in P} a_{np}$;
$w_p = \sum_{e \in E} \sum_{n \in S_e} a_{np} x_{en}$;
$d_p = r_p - w_p$;

$$g_p = \begin{cases} 1, & \text{if } r_p > w_p, \\ 0, & \text{otherwise;} \end{cases}$$

$$h_p = \begin{cases} 1 \text{ if } r_p \leqslant w_p, \\ 0, & \text{otherwise;} \end{cases}$$

$$v_{ep} = \begin{cases} 1, & \text{if employee } e \text{ is available for work in period } p \text{ but not currently assigned to a shift,} \\ 0, & \text{otherwise;} \end{cases}$$

$f1$ = last worked period in the initial work stretch of shift i;
$f2$ = last work period in the final work stretch of shift i (only for those shifts which require meal breaks);
$q1$ = first work period of the initial work stretch of shift i;
$q2$ = first work period in the final work stretch of shift i (only for those shifts which require meal breaks).

Table 2. Add-rule shift selection criteria priority ranking.

	Add-rule																			
Criterion	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
mu_n	1	1	1	1	2				2				2				2			
au_n	2				1	1	1	1		2				2				2		
acr_n		2				2			1	1	1	1			2				2	
smv_n			2				2				2		1	1	1	1				2
avl_n				2				2				2				2	1	1	1	1

Table 3. Selection criteria for dropping shifts.

Table 3. Selection criteria for dropping shifts

Selection criterion	Word description	Operational definition	Closest earliest literature reference
mo_n	maximum overstaffing occurring during the working periods of shift n	$mo_n = \max_{\{p \mid a_{np}=1\}}(-h_p d_p)$	Morris and Showalter [13][a]
$ao_n = t_n \div l_n$	average overstaffing per working period of shift n	$t_n = \sum_{\{p \mid a_{np}=1\}} (-h_p d_p)$	Showalter and Mabert [16] (ao_n)
$yl_n = y_n \div l_n$	reduction in schedule cost per working period of shift n	$y_n = c_n - \sum_{\{p \mid a_{np}=1\}} (g_p k_{p,d_p-1} + h_p b_{p_i-d_p})$	Keith [10][b]

[a] Morris and Showalter [13] allowed no understaffing and used a related rule: drop the shift with the maximum *minimum* overstaffing.
[b] Keith [10] uses a comparable rule: drop the shift which least increases the understaffing. This rule would be virtually equivalent to the one used here if all shifts contained the same number of working periods.

Table 4. Restrictions defining allowable shifts for Test Data Set 1.

[1]	Shifts start in any quarter hour period.
[2]	Shift length can vary in quarter hour increments.
[3]	Shifts must contain at least three but no more than eight working hours.
[4]	Shifts with less than 5 h of working time do not receive meal breaks.
[5]	Shifts with five or more, but less than 6 h of working time require a half-hour meal break.
[6]	Shifts with six or more working hours require an hour-long meal break.
[7]	Split shifts (shifts with meal breaks longer than 1 h) are not considered.
[8]	Rest breaks (breaks of less than30 min) are not scheduled.
[9]	A meal break must be both preceded and followed by at least eight quarter-hour periods of work.
[10]	A meal break must be neither preceded nor followed by more than 19 quarter-hour periods of work.
[11]	Full-time employees work only 8 h shifts for which they are available.
[12]	Part-time employees can work any shift for which they are available.

Table 5. Average final schedule costs on Test Data Set I for the 20 best implementations of the SAH[a]

Rank[b]	AR	DR	FO	MSC	GPD	NTB	Rank	AR	DR	FO	MSC	GPD	NTB
1	2	3	3	964.516	3.502	5	11	1	2	3	965.407	4.446	1
2	2	1	3	964.553	4.636	3	12	1	3	3	965.453	3.541	2
3	2	2	3	964.942	3.812	2	13	2	2	2	969.868	6.028	0
4	1	1	3	965.007	3.867	0	14	6	3	3	969.880	18.866	2
5	11	2	3	965.024	5.264	5	15	2	3	2	970.094	9.445	0
6	12	2	3	965.104	4.741	1	16	2	1	2	970.506	10.748	0
7	10	1	3	965.105	4.892	0	17	10	2	2	970.592	7.471	0
8	10	3	3	965.291	4.254	0	18	12	2	2	970.625	8.660	0
9	11	1	3	965.305	6.199	3	19	10	1	2	970.828	9.610	0
10	9	1	3	965.310	5.088	4	20	10	3	2	970.854	9.099	0

[a]AR = add-rule. DR = drop-rule. FO = FINETUNE option (1 = no FINETUNE, 2 = simple FINETUNE, 3 = advanced FINETUNE). MSC = mean schedule cost. GPD = greatest percentage difference, across all 144 problems, between the SAH implementation's solution for a problem and the best solution found for that problem. NTB = number of problems for which the SAH implementation yielded the best solution.
[b]Rank out of 180 implementations of the SAH.

Table 6. The impact of generating multiple schedules with the SAH using different add- and drop-rules and selecting the best.

Number of schedules generated per test problem	1	2	3	4	5	6	7	8	9	10
AR–DR–FO[a]	2–3–3	20–3–3	16–1–3	19–2–3	20–2–3	18–1–3	19–3–3	3–1–3	13–3–3	12–1–3
Mean schedule cost[b]	964.516	962.65	961.938	961.536	961.318	961.196	961.104	961.031	960.964	960.906
Relative schedule cost[c]	100.000	99.807	99.733	99.691	99.668	99.656	99.646	99.639	99.632	99.626

[a]Multiple schedules are generated in the order given using the specified add-rules (AR), drop-rules (DR) and FINETUNE options (FO). For example,if the best schedule for each test problem is to be selected from three schedules, then AR2–DR3–FO3, AR20–DR3–FO3, and AR16–DR1–FO3 would be the criteria used in generating the schedules.
[b]The average, across the 144 problems in TDS1, of the lowest cost of the schedules generated using the multiple implementations of the SAH.
[c]Measured as a percentage of the mean schedule cost for AR2–DR3–FO3.

Table 7. A comparison of various implementations of the SAH on TOSI and TDS2[a]

SAH implementation	Mean schedule cost	Percentage of reference	Solution procedure	Mean schedule cost	Percentage of reference
REG2/3/3	964.516	100.000	Optimal	329.164	100.000
FAST2/3/3	964.498	99.998	FAST2/3/3	330.208	100.317
CMB	964.037	99.950	CMB	330.168	100.305
SWT	963.271	99.871	SWT	330.151	100.300
EQTIME	963.515	99.896	EQTIME	330.425	100.383
PRTIME	962.982	99.841	PRTIME	330.101	100.285

[a]REG2/3/3 is the original implementation of AR2–DR3–FO3. FAST2/3/3 only calculated criteria used in AR2, DR3 or FO3. CMB changes between AR2–DR3–FO3, AR20–DR3–FO3, AR16–DR1–FO3 and AR19–DR2–FO3 at each iteration. SWT switches between AR2–DR3–FO3, AR20–DR3–FO3, AR16–DR1–FO3 and AR19–DR2–FO3 every three iterations without an improvement in the schedule. EQTIME uses AR2–DR3–FO3, AR20–DR3–FO3, and AR16–DR1–FO3 for 10 s each. PRTIME uses AR2–DR3–FO3 for 15 s, AR20–DR3–FO3 for10 s and AR16–DR1–FO3 for 5 s.

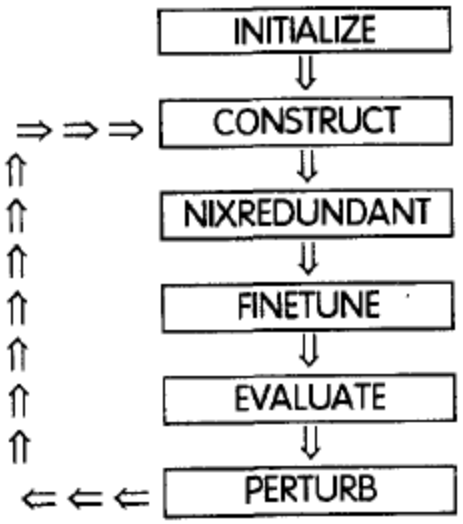

Figure 1. A pictorial overview of the simulated-annealing heuristic.

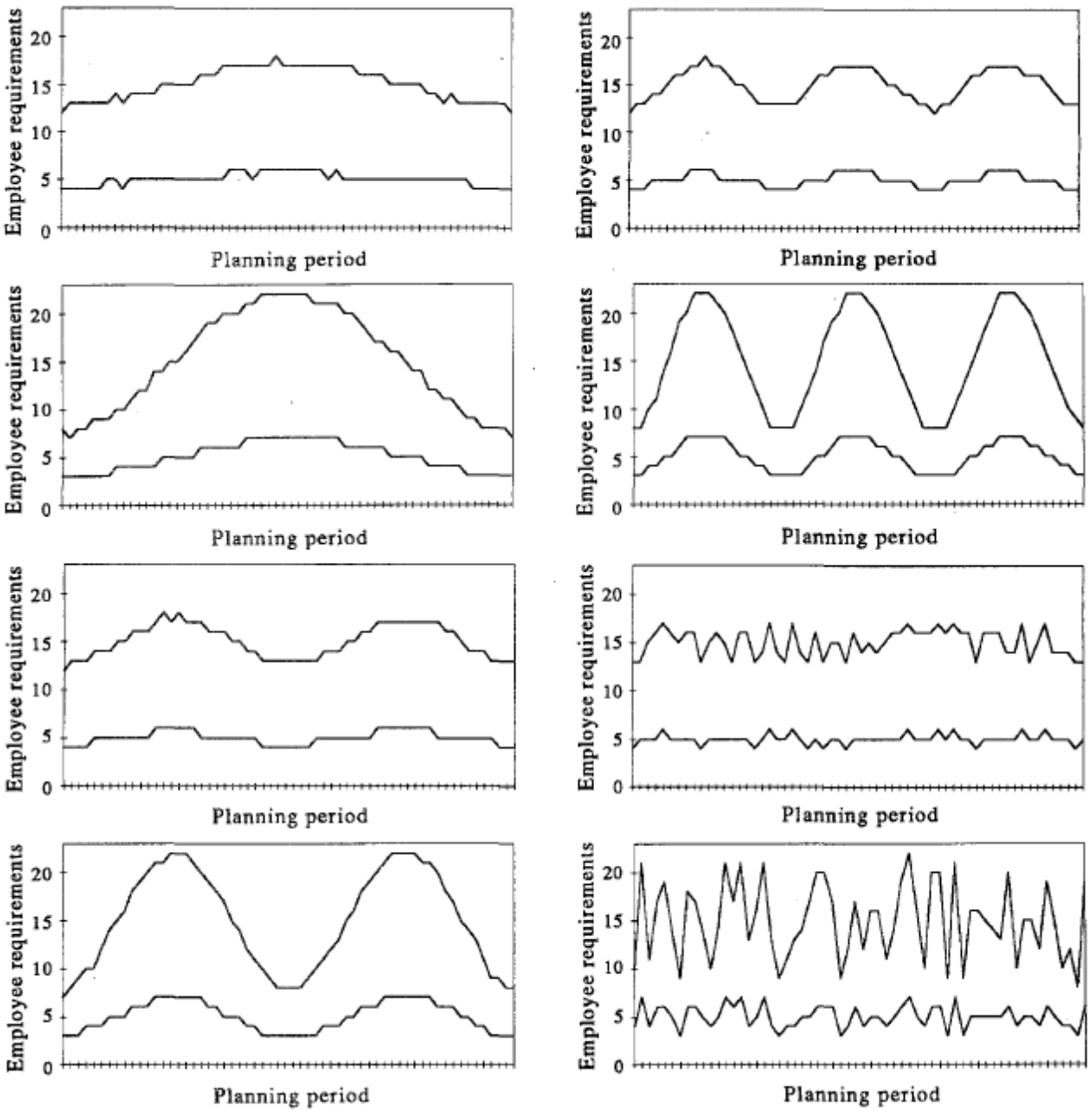

Figure 2. Employee requirements patterns in TDSI.