OFF-DAY SCHEDULING WITH HIERARCHICAL WORKER CATEGORIES

HAMILTON EMMONS
Case Western Reserve University, Cleveland, Ohio

RICHARD N. BURNS
Queen's University, Kingston, Ontario, Canada

(Received September 1987; revision received November 1989; accepted February 1990)

A work force includes workers of m types. The worker categories are ordered, with type-1 workers the most highly qualified, type-2 the next, and so on. If the need arises, a type-k worker is able to substitute for a worker of any type $j > k$ ($k = 1, \ldots, m-1$). For 7-day-a-week operation, daily requirements are for at least D_k workers of type-k or better, of which at least d_k must be precisely type-k. Formulas are given to find the smallest number and most economical mix of workers, assuming that each worker must have 2 off-days per week and a given fraction of weekends off. Algorithms are presented which generate a feasible schedule, and provide work stretches between 2 and 5 days, and consecutive weekdays off when on duty for 2 weekends in a row, without additional staff.

The problem of scheduling days off for a 7-day-a-week work force has received much attention. In service-oriented operations that function continuously and where demand for staff fluctuates, e.g., toll collectors, nurses, bus drivers, and police, problems of staff requirements and allocation arise. Many manufacturers also operate extra shifts and extended work weeks, at least in times of high demand.

The goal of cyclic manpower scheduling is to construct time-off schedules that meet staffing requirements (assumed to repeat or cycle periodically, say weekly), and satisfy work rule constraints, using the minimal number of workers. Staffing needs are specified in terms of the numbers of workers that must be on duty each day or each shift of a typical week. Work rules, generally defined through labor negotiations, usually include the requirements of 2 days off each week, and a certain fraction, A out of B, of weekends off. In addition, limitations on the length of a work stretch (i.e., a sequence of consecutive work days) are common. A single day at work between 2 days off is considered inefficient and unsatisfying; and 2-day work stretches are discouraged. At the other extreme, work stretches over 5 or 6 days straight are generally associated with fatigue, burnout, and low morale. Finally, union contracts may sometimes specify that, when a worker must go several weeks without a weekend off and therefore gets 2 weekdays off per week in that stretch, consecutive weekdays should be given off when possible. This consolidation of rest and work intervals enhances the quality of both.

Many manpower scheduling algorithms that tackle problems of any complexity are heuristic and/or use relatively ponderous general purpose solution techniques, such as integer programming. Mathematical programming has been applied by Warner (1976) and Miller, Pierskalla and Rath (1976) to nurse scheduling and by Segal (1974) to telephone operators, among many others.

However, real problems of this type usually have a lot of special structure that can be exploited in individually tailored algorithms. Many special cases of surprising complexity have been solved by simple, efficient algorithms which give the minimal work force immediately as a formula and which then lay out a feasible schedule in one pass. We will call this the *combinatorial approach*. For example, Baker, Burns and Carter (1979) make the following assumptions:

1. There is a requirement for n workers per day, 7 days a week.
2. Each worker must be given 2 off-days per week (a week runs from Sunday through Saturday).
3. Each worker must be given A out of B weekends off for any A and B, $A < B$.
4. No worker shall ever be assigned a work stretch of more than 6 days.

A further implicit assumption is that weekends are never split: either both days are off or both are worked.

This model is similar to several that have previously been investigated combinatorially, notably Tibrewala,

Subject classifications: Labor: manpower scheduling. Organizational studies, manpower planning: scheduling. Production/scheduling: manpower scheduling.

0030-364X/91/3903-0000 $01.25

Philippe and Brown (1972), Baker (1974), Brownell and Lowerre (1976), Baker and Magazine (1977), and Burns (1978), which admit varying staff requirements over the week but impose less detailed off-time requirements. Burns and Carter (1985) generalize Assumption 1 to allow varying staff requirements (different for each day of the week), maintaining off-time requirements as listed above. Emmons (1985) discusses the intermediate case of Assumption 1, with the same staff requirements each weekday but different on the weekends. He introduces a generalized cyclical scheduling procedure which simply and systematically produces a *master rotation* schedule that can be used repeatedly, and that:

- satisfies the coverage, off-day and off-weekend requirements using the smallest possible work force without additional staff;
- guarantees work stretches between 2 and 5 days; and
- assures that anyone who must work on consecutive weekends receives adjacent weekdays off in the intervening week.

The general combinatorial procedure that has evolved proceeds as follows. Each constraint i, generally a work rule, generates a lower bound, l_i, on the minimal feasible work force size, W. To show that the smallest value of W satisfying all the constraints, $W = \max_i l_i$, is sufficient, an algorithm is presented that schedules W workers to provide the requisite coverage without violating work rules. This approach, when applicable, has significant advantages over integer programming. It permits quick calculation of work force size, so that a manager, contemplating a change in requirements or work rules, can quickly assess its implications. Furthermore, schedule generation is almost as easy, requiring negligible computer time even for very large problems that would cause the number of integer variables to skyrocket.

Of course, an integer programming formulation can more easily incorporate additional constraints, and be extended to more general models. Combinatorial analysis must be carefully adapted to new assumptions.

All the combinatorial results to date are based on the assumption of a single type of worker. We will consider the case where there are several worker categories or types. Suppose that the work force can be divided into m categories according to job classification, seniority, training, or other qualifications. Also, suppose that the categories are ranked with type-1 workers the most highly qualified (and, presumably, the most highly paid) so that a type-j worker ($j = 1, 2, \ldots, m-1$) is capable of doing the work of, and can be used in place of, a worker of type-k for any $k > j$, but not vice versa. Thus, we have a hierarchy of categories with "downward substitutability."

A notable example of such a work force occurs in hospital nursing. As noted in Warner, a staffing decision must be made every few months that specifies the number of nurses of each skill class (registered nurse, licensed practical nurse, nursing aide, etc.) to be assigned to each nursing unit. Thereafter, at intervals of 4 weeks or so, schedules must be prepared that specify when each nurse is on or off duty, so that sufficient nurses of each type are on duty at all times.

The present combinatorial formulation is a step toward the more general model in which there are several shifts of workers each day (an extension currently under study), and applies in situations where the staffing of each shift is independent, or where there is only one shift, as at military installations and large retail outlets.

With worker substitution allowed, simply stating the work force requirements or formulating an objective function becomes more complicated. We shall define a model for multitype work force scheduling, and extend the single-type techniques to provide a simple solution.

1. MODEL FORMULATION

Consider a facility that must be staffed 7 days a week. Each worker must be given 2 days off every week, including A out of B weekends off. In addition, work stretches should be between 2 and 5 days. There are m-types of workers to be scheduled; their capabilities are ranked hierarchically. The requirements for each worker type are constant over the days of the week. In addition, despite downward substitutability, a minimal number of each type may be required even when extra higher qualified personnel are on duty.

For example, suppose that there are $m = 3$ types of workers, and that we require on duty daily:

- at least two type-1 (the most qualified) workers;
- at least six workers with type-2 or better qualifications. Thus, we might have two type-1 workers and four or more type-2, or three type-1 and at least three type-2, etc., on any day;
- however, at least three genuine type-2 workers, no matter how many extra type-1 workers there are;
- at least nine workers total; that is, three type-3 (the least capable) workers unless there are extra workers of other types to stand in;
- however, at least three type-3 workers, anyway: no substitution here.

It is understood that less qualified workers are less costly, so that we would prefer to employ a work force

with the lowest capabilities consonant with the requirements. We define:

D_k = the minimal number of workers of types 1 through k required each day;
d_k = the minimal number of (genuine) type-k workers required daily.

Observe that $D_1 = d_1$, and without loss of generality assume that $D_k - D_{k-1} \geq d_k \geq 0$. Denoting the vectors $D = (D_1, \ldots, D_m)$ and $d = (d_1, \ldots, d_m)$, we have in the example above $D = (2, 6, 9)$ and $d = (2, 3, 3)$.

Our objective will be to find the minimal total number of workers to hire, and the most economical mix of types, to satisfy the requirements (D, d) and to give each worker 2 days off per week and A out of B weekends off. That is, find

W = the minimal total work force, and
w_k = the minimal number of type-k workers given W and $w_1, \ldots, w_{k-1}$ $(k = 1, \ldots, m)$.

We will also show that this minimal, feasible number of workers is simultaneously the lowest cost mix of workers, assuming only that less qualified individuals cost less.

We then present a simple algorithm for scheduling these workers, and use it to prove that the work force is indeed sufficient to satisfy staffing as well as day-off and weekend-off requirements. A more elaborate algorithm is also given which has the additional properties:

- Work stretches are between 2 and 5 days, with very few as short as 2 days.
- Workers who must be on duty for 2 consecutive weekends are assigned a consecutive pair of weekdays off in the intervening week.

2. COMPUTATION OF WORK FORCE SIZE AND MIX

The total work force size must satisfy the total requirements. If we consider only one worker type, as discussed in Baker, Burns and Carter (1979), then the weekend constraint (each worker must be given A out of B weekends off) leads to a work force $W \geq nB/(B - A)$, where n is the daily worker requirement. Similarly, the off-day constraint (each worker must be given 2 days off each week) implies that $W \geq 7n/5$. It has been shown that the work stretch constraint (no worker is required to work more than 6 days straight) will always be satisfied if these two inequalities hold, so that the minimal work force required to satisfy a daily requirement for n workers is

$$f(n) = \max\{\lceil nB/(B-A)\rceil, \lceil 7n/5\rceil\} \tag{1}$$

where $\lceil x\rceil$ (respectively, $\lfloor x\rfloor$) denotes the upper (lower) integer part of x. Thus, $f(d_k)$ is the necessary number of type-k workers to satisfy requirement d_k in isolation; and $f(D_k)$ is the total number of workers of types 1 through k needed if the cumulative requirement D_k were the only one. Clearly since the total work force must be at least D_m daily

$$W \geq f(D_m). \tag{2}$$

Viewing each type of worker in isolation, we also have

$$W \geq \sum_{k=1}^{m} f(d_k). \tag{3}$$

More generally, for any k, we need enough of types 1 through k to satisfy D_k, plus enough of each remaining type to satisfy minimal requirements separately.

$$W \geq f(D_k) + \sum_{j=k+1}^{m} f(d_j) \quad \text{for } k = 1, \ldots, m. \tag{4}$$

Note that (2) and (3) are special cases ($k = m$ and $k = 1$) of (4), where the summation is taken as zero for $k = m$.

We will show that these necessary conditions are also sufficient, so that

$$W = \max_{k=1,\ldots,m}\left\{f(D_k) + \sum_{j=k+1}^{m} f(d_j)\right\}. \tag{5}$$

To compute $f(n)$ for various values of n, we use (1). For any problem instance, this is simplified by noting that, if

$$R = \max\{B/(B-A), 7/5\} \tag{6}$$

we can compute this parameter once, and then for any n

$$f(n) = \lceil Rn\rceil. \tag{7}$$

In the example, with $m = 3$, $D = (2, 6, 9)$ and $d = (2, 3, 3)$, suppose that $R = 7/5$. Then

$$W = \max\{\lceil 14/5\rceil + \lceil 21/5\rceil + \lceil 21/5\rceil, \lceil 42/5\rceil + \lceil 21/5\rceil, \lceil 63/5\rceil\}$$
$$= \max\{13, 14, 13\} = 14 \text{ workers}.$$

For the breakdown of this complement by worker category, recall that we wish to successively minimize the number in each category, starting with the most

highly qualified. The equations that result are

$$w_1 = f(d_1) \tag{8}$$

$$w_k = \max\left\{ f(d_k), f(D_k) - \sum_{j=1}^{k-1} w_j \right\}, \quad k = 2, \ldots, m. \tag{9}$$

These formulas clearly give lower bounds. They will be shown by construction to be adequate, and hence, optimal.

In the example, using (6) and (7)

$$w_1 = \lceil Rd_1 \rceil = \lceil 14/5 \rceil = 3$$

$$w_2 = \max\{ \lceil 21/5 \rceil , \lceil 42/5 \rceil - 3 \} = 6$$

$$w_3 = \max\{ \lceil 21/5 \rceil , \lceil 63/5 \rceil - 9 \} = 5.$$

Incidentally, although arrived at through separate logic, the formula for the total work force, W, given in (5) agrees with the formulas in (8) and (9) for the separate components w_k. A simple induction (see Appendix A) shows that $W = \sum_{k=1}^{m} w_k$.

If we introduce costs per worker, then if we make only the obvious assumption that less qualified workers cost less to hire (otherwise, eliminate the category), we can show that the formulas obtained give the least costly work force. We will assume for the moment that the given mix of workers is indeed sufficient, as we will show later.

Proposition. *If c_k is the cost to employ a type-k worker, and if $c_1 > c_2 > \ldots > c_m$, then the work force given in* (8) *and* (9) *is the least cost combination of workers that satisfies staffing requirements.*

Proof. First, we know that for any $j = 1, \ldots, m$, $\sum_{i=1}^{j} w_i$ is the smallest total work force that can satisfy the first j categories of requirements. This is because, as argued before, the formula in (5) gives the smallest possible number of workers for all m categories, and in Appendix A the same logic may be applied to (A1).

Consider any other worker mix, $w'_1, \ldots, w'_m$. Let j be the smallest index of a category in which the two work forces differ. It cannot be true that $w'_j < w_j$ because then $\sum_{i=1}^{j} w'_i$ would be smaller than $\sum_{i=1}^{j} w_i$, which is infeasible. Thus, assume that $w'_j > w_j$.

Consider the cumulative change $\Delta_k = \sum_{i=1}^{k} (w'_i - w_i)$ as the increments in successive categories $k = j, \ldots, m$ are added. Since the successive unit costs are decreasing, to compensate for each increase ($w'_j > w_j$) in the work force of a category, there must be a larger total decrease in one or more later categories. Thus, having started with an increase, we must eventually reach a category K where $\Delta_K < 0$, if the total cost is to be smaller. But this means that $\sum_{i=1}^{K} w'_i < \sum_{i=1}^{K} w_i$, which again is infeasible.

3. ALGORITHM FOR SCHEDULE CREATION

To show that these numbers of workers are sufficient, we give an algorithm to produce a feasible schedule using this work force. The general principles are:

- Weekends off are scheduled first, and then the weekdays off are filled in.
- Since we must give A out of every B weekends off, the master rotation is prepared for B weeks, with the understanding that it can be used repeatedly.
- To provide adequate coverage on each day, the off-days are distributed as uniformly as possible over the days of each week.

3.1. Scheduling Weekends Off

First, it will be our goal to give as many weekends off as possible. From (1), we see that, if

$$B/(B-A) \geq 7/5 \quad \text{or} \quad A/B \geq 2/7$$

then the weekend constraint is tight and the work force will be just sufficient to allow A out of B weekends off. If $A/B < 2/7$, then the off-day constraint determines the work force sizes, and we could give more than A out of B weekends off. Specifically, we have enough workers to give 2 out of 7 weekends off, since clearly, if we use 2 and 7 in place of A and B, the same work force results. Thus, if we are required to give less than $2/7$ of the weekends off, we will automatically adjust this fraction to $2/7$.

Suppose that we have computed the work force size and mix. For any category of worker, if w workers must be given A out of B weekends off, then Aw off-weekends must be distributed over B weekends for an average of Aw/B workers off each weekend. In case this is not an integer, then to keep the time off spread as evenly as possible over all weekends we must give the weekend off to $\lfloor Aw/B \rfloor$ workers on some B weekends, and to one extra, $\lfloor Aw/B \rfloor + 1 = \lceil Aw/B \rceil$ workers, on the rest. The number of extra off-weekends is the excess of Aw over $\lfloor Aw/B \rfloor$ per week for B weeks.

In summary, for w_k workers of type-k, we can assign off-weekends in any B-week cycle to:

$\lceil Aw_k/B \rceil$ workers for $Aw_k - B\lfloor Aw_k/B \rfloor$ weeks.

$\lfloor Aw_k/B \rfloor$ workers for $B - (Aw_k - B\lfloor Aw_k/B \rfloor)$

$= B(\lfloor Aw_k/B \rfloor + 1) - Aw_k$ weeks.

Let

$$x_k = Aw_k/B \text{ the average number of workers off per weekend} \quad (10)$$

and

$$y_k = B(\lfloor Aw_k/B \rfloor + 1) - Aw_k = B(\lfloor x_k \rfloor + 1 - x_k). \quad (11)$$

Then we have $\lfloor x_k \rfloor$ workers off for y_k weekends, and $\lceil x_k \rceil$ workers off for the other $B - y_k$ weekends ($y_k = 1, 2, \ldots, B$). We can now state formally how to assign weekends off, starting with the highest qualified workers and distributing the time off as evenly as possible over the B weeks, both within a worker category and cumulatively over all workers.

Algorithm 1: To Assign Weekends Off

Step 0. Given the parameters m, $d = (d_1, \ldots, d_m)$, $D = (D_1, \ldots, D_m)$, A and B, compute

R using (6)
w_k for $k = 1, \ldots, m$ using (8) and (9)
x_k and y_k for $k = 1, \ldots, m$ using (10) and (11).

Step 1. For worker categories $k = 1, \ldots, m$ sequentially, assign $\lfloor x_k \rfloor$ workers off for y_k weekends, and $\lceil x_k \rceil$ workers off for the remaining $B - y_k$ weekends. The y_k weekends where fewer workers are off should not be consecutive, but should be alternative as far as possible. Specifically, if the B weekends are numbered consecutively, use the sequence $S = (1, 3, 5, \ldots, [B \text{ or } B-1], 2, 4, 6, \ldots, [B-1 \text{ or } B])$, continuing through S from one category to the next, starting over when the end of S is reached.

Step 2. In selecting which workers to give off each weekend, start with the first $\lfloor x_k \rfloor$ workers of type-k, and on successive weekends choose the next group of workers sequentially, wrapping around (the first worker in the category follows the last) when needed.

In our example, where we assume a minimal off-weekend ratio, $A/B = 2/7$, the initial calculations produce the numbers in Table I.

In Table II, the weekends off have been scheduled (a capital X denotes a weekend day off) following Algorithm 1. No type-1 workers are given the first weekend off (the first weekend is taken to be the Sunday at the far left of the table, with its preceding Saturday equivalent to the last Saturday at the right of the table, assuming a repeating B-week cycle), and thereafter one type-1 worker is given each weekend off, cyclically, for the other 6 weeks. Note how this results in precisely 2 out of 7 weekends off for each worker.

Table I
Characteristics of the Sample Problem

k	d_k	D_k	w_k	$\lfloor x_k \rfloor$	y_k
1	2	2	3	0	1
2	3	6	6	1	2
3	3	9	5	1	4

For category 2 with $y_2 = 2$, the 2 weekends with fewer workers off are chosen from the sequence S to be weekends 3 and 5, and for category 3, the next four ($y_3 = 4$) elements of S, namely 7, 2, 4 and 6 are selected. Thus, category 2 has $\lfloor x_2 \rfloor = 1$ workers off on weekends 3 and 5, with two workers off on the others; and similarly, category 3 has $\lfloor x_3 \rfloor = 1$ workers off on weekends 7, 2, 4 and 6, and two workers off on the remaining three weekends.

Having decided the number of workers to be given each weekend off, they are scheduled in order, starting at the first weekend chosen from S (though any weekend could be chosen as the starting point).

3.2. Scheduling Weekdays Off: The Simple Algorithm

Our goal is to assign additional off-days to each worker to bring the total to 2 days off in every Sunday-through-Saturday week. The job will be complicated by the fact that, in addition, we wish to maintain work stretches between 2 and 5 days and to give adjacent pairs of weekdays off to personnel who must work consecutive weekends. These additional requirements will be ignored for the moment to facilitate initial presentation and to prove the sufficiency of the work force. In the next section, the algorithm will be elaborated to incorporate them.

Algorithm 2: The Simple Scheduling Procedure

Step 0. Apply Algorithm 1 to assign weekends off.

Step 1. For each of the B weeks separately, assign weekdays off to each worker, working down through the worker list.

Step 2. Give each worker enough weekdays off (0, 1 or 2) to bring the total days off in that week, Sunday through Saturday, to 2.

Step 3. Choose the particular weekdays to assign as off-days in rotation: Monday, Tuesday, ..., Friday, Monday, ...

Algorithm 2 has been used to complete the schedule in Table II. Note how, reading down any week, the weekdays off cycle repeatedly through the 5 days. This, of course, guarantees that, within a category and cumula-

Table II
Complete Schedule Using the Simple Algorithm

1 m t w t f s	2 m t w t f s	3 m t w t f s	4 m t w t f s	5 m t w t f s	6 m t w t f s	7 m t w t f s
x X	X x	x x	x X	X x	x x	x x
x x	x X	X x	x x	x X	X x	x x
x x	x x	x X	X x	x x	x X	X x
X x	x X	X x	x X	X x	x X	X x
x x	x x	x X	X x	x x	x X	X x
x X	X x	x X	x X	x X	X x	x X
X x	X x	x X	x X	x X	X x	x X
x X	X x	x X	x X	x X	X x	x X
x x	x X	X x	x x	x x	x X	X x
X x	x x	x X	X x	x x	x x	x X
X x	x X	X x	x X	X x	x X	X x
x X	X x	x X	x X	x X	X x	x X
x X	X x	X x	x X	x X	X x	x X
x x	x X	X x	x x	x X	X x	X x
Workers on Duty By Category						
3 2 2 2 2 2 2	2 2 2 2 2 3 2	2 2 2 2 2 3 2	2 2 2 2 2 3 2	2 2 2 2 2 3 2	2 2 2 2 2 3 2	2 2 2 2 2 2 3
4 4 4 4 5 5 4	4 4 4 4 5 5 5	5 4 4 4 5 4 4	4 4 4 4 5 4 5	5 4 4 4 5 4 4	4 4 4 5 5 4 4	4 4 4 4 5 5 4
3 4 4 4 3 3 4	4 4 4 4 3 3 3	3 4 4 4 3 3 4	4 4 4 4 3 3 3	3 4 4 4 3 3 4	4 4 4 3 3 3 4	4 4 4 4 3 3 3

tively over categories, the number of days off on any weekday never differs from the number on any other weekday by more than 1.

We may formally state the principal result of this paper.

Theorem. *In a facility that operates 7 days a week, with m ordered categories of workers, given requirements for*:

- *at least D_k workers of type-k or higher each day*;
- *at least d_k workers of precisely type-k each day*;
- *two days off per week per worker*;
- *A out of B weekends off for each worker.*

Then assuming $A/B \geqslant 2/7$, it follows that:

- *the work force specified by* (8) *and* (9) *is sufficient, and*
- *Algorithm* 2 *always produces a feasible schedule using that work force.*

The proof of this theorem is lengthy, and appears in Appendix B.

3.3. Scheduling Weekdays Off: The Extended Algorithm

Although Algorithm 2 satisfies the requirements for weekdays off and weekends off, it does not consider work stretch and adjacent-weekdays-off constraints. For example, the distribution of work stretch lengths that results from applying Algorithm 2 to the example, as shown in Table II, is given in Table III. They range from one to nine, unacceptable at both ends in most labor situations. In compensation, weekdays off were sometimes given adjacent to weekends, resulting in seven 3-day weekends and three 4-day weekends. While these may be pleasant to receive, management (and the other workers) might object on the grounds of fairness and morale, and they do not compensate for the wide work stretch variance.

With respect to the requirement for adjacent weekdays off when possible, the situation is not so bad because the cyclic assignment procedure tends to give consecutive days. However, about one-fifth of the time Monday and Friday are chosen.

We therefore present a more elaborate, but still quite straightforward, algorithm for scheduling weekdays off that will satisfy all these requirements. It will produce work stretches between three and five, very occasionally two, and adjacent weekdays off in every week with both weekends on. This can be observed in Table IV, where we display the schedule produced by the extended algorithm when applied to the example of Table I. It will be discussed further after we present the algorithm.

We shall refer to a 1-week (Sunday through Saturday) schedule for one worker as a *slot*. Thus, "slot 3 for week 1" refers to the pattern of off-days assigned to worker 3 in the first week. We also use the term *week stretch* to refer to the number of weeks between consecutive weekends off. For example, a worker who regularly gets every other weekend off has "week stretches

Table III
Work Stretch Lengths Using the Simple Algorithm

Work Stretch (Days)	1	2	3	4	5	6	7	8	9
Number Occurring	11	14	23	20	23	26	2	1	2

of two weeks," or "two-week stretches." Of course, each worker gets 2 days off every week; week stretches are determined solely with reference to weekends.

The following principles will guide our algorithm:

- As we did with weekends off, we allocate weekdays off to one worker category at a time, in numerical order.
- For a given worker type, we schedule one week at a time.
- Among the slots in a given week for a given worker type, we first deal with those belonging to the longest week stretches because there are fewer alternative choices of off-days in such slots.
- Our rules for assigning weekdays off will be tailored to the length of the week stretch and the position of the slot in the week stretch.
- As we allocate days off, we must keep track of the number of off-days already given on each day, so that we do not become understaffed at any time.

We will need some preliminary definitions. First, we say that a *slot is of type* (i/j) or is an (i/j) *slot*, if it is week i of a j-week stretch. Thus, the third week of a 4-week stretch has slot type (3/4). To facilitate following the example, each slot in Table IV is labeled with its type, the two digits appearing in the first 2 weekdays of the slot. Next, let

$n_{k,t}$ = the number of type-k workers already given a day off on day-t where $k = 1, \ldots, m$, and t denotes any of the $5B$ weekdays in the B weeks being scheduled.

It is not hard to see that, to permit another type-k

Table IV
Complete Schedule Using the Extended Algorithm

1 m t w t f s	2 m t w t f s	3 m t w t f s	4 m t w t f s	5 m t w t f s	6 m t w t f s	7 m t w t f s
4 4	1 3	2 3	3 3	1 4	2 4	3 4
x X	X x	x x	x X	X x	x x	x x
3 4	4 4	1 3	2 3	3 3	1 4	2 4
x x	x X	X x	x x	x X	X x	x x
2 4	3 4	4 4	1 3	2 3	3 3	1 4
x x	x x	x X	X x	x x	x X	X x
2 3	3 3	1 4	2 4	3 4	4 4	1 3
x x	x X	X x	x x	x x	x X	X x
2 4	3 4	4 4	1 3	2 3	3 3	1 4
x x	x x	x X	X x	x x	x X	X x
1 3	2 3	3 3	1 4	2 4	3 4	4 4
X x	x x	x X	X x	x x	x x	x X
1 4	2 4	3 4	4 4	1 3	2 3	3 3
X x	x x	x x	x X	X x	x x	x X
3 3	1 4	2 4	3 4	4 4	1 3	2 3
x X	X x	x x	x x	x X	X x	x x
3 3	1 4	2 4	3 4	4 4	1 3	2 3
x X	X x	x x	x x	x X	X x	x x
2 3	3 3	1 4	2 4	3 4	4 4	1 3
x x	x X	X x	x x	x x	x X	X x
1 3	2 3	3 3	1 4	2 4	3 4	4 4
X x	x x	x X	X x	x x	x x	x X
1 4	2 4	3 4	4 4	1 3	2 3	3 3
X x	x x	x x	x X	X x	x x	x X
4 4	1 3	2 3	3 3	1 4	2 4	3 4
x X	X x	x x	x X	X x	x x	x x
3 4	4 4	1 3	2 3	3 3	1 4	2 4
x x	x X	X x	x x	x X	X x	x x
Workers on Duty By Category						
3 2 2 2 2 2 2	2 2 3 2 2 2 2	2 2 3 2 2 2 2	2 2 3 2 2 2 2	2 2 3 2 2 2 2	2 2 3 2 2 2 2	2 2 2 2 2 2 3
4 5 5 4 4 4 4	4 5 4 4 4 4 5	5 4 4 5 4 4 4	4 5 4 4 5 4 5	5 4 4 4 4 5 4	4 5 4 4 5 4 4	4 5 5 4 4 4 4
3 4 4 3 4 3 4	4 4 3 3 4 4 3	3 5 3 3 4 4 4	4 4 4 3 3 4 3	3 4 3 4 4 4 4	4 3 4 3 3 4 4	4 3 3 4 4 4 3

worker to have day-t off, we require

$$n_{k,t} < w_k - d_k, \quad \text{and}$$

$$\sum_{j=1}^{k} n_{jt} < \sum_{j=1}^{k} w_j - D_k.$$

We define

$Q_{k,t}$ = the quota of days off for worker type-k on day-t, given that $n_{j,t}$ workers have been given the day off on this day in each of the higher qualified categories, $j = 1, \ldots, k-1$

$$Q_{k,t} = \min\{w_k - d_k, \textstyle\sum_{j=1}^{k} w_j - D_k - \sum_{j=1}^{k-1} n_{j,t}\}.$$

Once $n_{k,t}$ reaches this quota, no more type-k workers can be assigned day-t off. We will say that day-t is *full*; though note that it may no longer be full when we advance to the next category of worker.

It facilitates our task to note that week stretches never exceed four weeks. Indeed, since weekends off are distributed as uniformly as possible over the B-weeks, each worker gets successive off-weekends at intervals that are as nearly equal as possible: either $\lfloor B/A \rfloor$ or $\lceil B/A \rceil$. While this result is intuitively clear, the integer rounding makes it hard to prove, and we do not need it. Since $B/A \leqslant 7/2$, it implies an upper bound of four weeks. Proof of this is outlined in Appendix C.

Algorithm 3: The Extended Scheduling Procedure

Step 0. Apply Algorithm 1 to assign weekends off.

Step 1. Sequentially for each worker type $k = 1, \ldots, m$, and for each k, sequentially for each week $1, \ldots, B$, assign days off according to the following rules.

Step 2. Assign single off-days to first slots in 3- or 4-week stretches, in the order:

Give (1/4) slots Fr or Th.
Give (1/3) slots Fr or Th.

In each case, use the first day listed unless it is already full.

Step 3. Assign off-days to the remaining slots, in the order:

Give (3/3) and (4/4) slots Mo or Tu.
Give (2/4) slots (We, Th), (Th, Fr), (Tu, We) or (Mo, Tu).
Give (3/4) slots (Tu, We), (Mo, Tu), (We, Th) or (Th, Fr).

If there are several slots of types (2/4) and (3/4), rather than scheduling all the (2/4) slots before any (3/4), order them according to Step 5.

Give (2/3) slots (Tu, We) or (We, Th).
Give (1/2) slots Th, Fr or We.
Give (2/2) slots Tu, Mo or We.

Step 4. To choose from the list of alternatives for each slot type, use the following selection rules:

a. Avoid any day that is already full, or that causes a work stretch of less than 2 or more than 5. If ties remain,
b. Keep the off-days as equally distributed as possible over the 5 weekdays (i.e., choose days that have fewer off-days already assigned):
   i. for worker type-k and if ties remain,
   ii. for the total of worker types $1, \ldots, k$. If ties remain,
c. Whenever possible, leave adjacent days available for future assignment. For example, if Tu through Fr are equally available, it is better to use (Tu, We) or (Th, Fr) than (We, Th). If ties remain,
d. For weeks $2, \ldots, B$, make the work stretch with the previous days off as close to 4 as possible, unless:
   the slot starts with a weekend, in which case, skip this rule; or
   the slot ends with a weekend, in which case, balance the work stretches before and after as closely as possible. If ties remain,
e. Use the first alternative listed.

Step 5. When there are several slots of the same type, then having applied rules a, b and c, rules d and e may be used to choose the slot to schedule next. For present purposes, (3/3) and (4/4) slots may be considered the same type. Similarly, (2/4) and (3/4) slots should be considered all together.

For most slot types, a list of alternative off-days (single days or adjacent pairs) is given. The particular selections in each case, and their priority ordering, are motivated by the desire to keep the work stretches close to 4 days. For example, if the successive slots in a 4-week stretch are given their first choices: Fr, (We, Th), (Tu, We) and Mo, then the work stretches in the four weeks are all four days long.

3.4. Illustration of Day-Off Scheduling

We illustrate the algorithm using the schedule in Table IV, where we imagine that only the weekends off have been scheduled. Starting with week 1 for worker type 1, we have three slots to schedule. The first is the last week of a 4-week stretch (recall this is a cyclic schedule, so that week 1 repeats after week 7), hence is a (4/4) slot. We only give one weekday off since Saturday is off

already, and the algorithm indicates Monday. The other two slots are of types (2/4) and (3/4). Step 5 of Algorithm 3 indicates that we consider them together. To choose which of the listed pairs of days to give off, selection rule a eliminates the last choice (Mo, Tu) because it includes Monday, rule b does not apply, rule c eliminates (We, Th), rule d does not apply, and rule e selects (Th, Fr) for the (2/4) slot and (Tu, We) for (3/4). Had there been an overlap, we would have chosen one arbitrarily because all tie-breaking rules have been considered, and then reconsidered the other. As it is, both may be scheduled.

In week 2 of the first worker type, we note that having given Monday and Friday off to slots (4/4) and (1/3), we choose (We, Th) for the (3/4) slot because that creates a 4-day work stretch following the previous Friday off (rule d).

Having completed the 7 weeks for type-1 workers, we note that in closing the loop, with week 1 following week 7, we have not controlled the work stretches that result. In fact, two of the three work stretches are 5 days long, which is on the high side but perfectly acceptable.

Consider week 1 for the second category of worker. First, the (1/4) and (1/3) slots are given Friday off. Since we require six workers with these or better qualifications, Friday is now full. Next, the two (3/3) slots get Monday and Tuesday, respectively. Then (2/4) gets (We, Th) by rule b. Finally, (2/3) gets (We, Th) by rule c.

In week 2, category 2 we start by giving Friday off to both the (1/4) slots, and Friday is full. Then (3/3) gets Tuesday (rule bii). For the (2/4) and (3/4) slots, rule a eliminates (Th, Fr) and (bi) and eliminates all but (We, Th). Rule d indicates that this pair of off-days should go to the (2/4) slot. Now, rule b indicates that Monday be included in the next assignment, so (3/4) gets (Mo, Tu). Finally, (2/3) is assigned (We, Th) by rule bi.

One final illustration will be given. In week 3 for type-2 workers, there is a (3/3) and a (4/4) slot. The first of these to be scheduled (following the (1/4) slot) gets Tuesday off, by rule (bii). Step 5 tells us to make the choice based on rule d. Since work stretches of 4 and 6 days would be formed, we choose the shorter, and give the Tuesday off to (3/3). We then give (4/4) Monday off, bringing the 6-day work stretch down to five.

4. DISCUSSION OF THE EXTENDED ALGORITHM

Algorithm 3 is in some degree heuristic. On the one hand, the proposed work force is necessary and sufficient to give the required weekends off and 2 days off per week. That we can additionally keep work stretches between two and five days, and give consecutive weekdays off as needed, is clear from many examples but not formally proven. Even in instances with $A/B = 2/7$, the most difficult case, there is enough slack capacity that very few work stretches exceed 4 days, let alone 5. For larger values of A/B, week stretches get shorter, so that we have a smaller proportion of double weekdays off to deal with, and greater flexibility to assign off-days.

Table V gives the number and proportion of work stretches of each length that result from the example shown in Table IV and from another illustrative instance. They confirm that most of the work stretches are 3 or 4 days, and all are between 2 and 5. Even the few 2- or 5-day work stretches could mostly be eliminated by a second pass, at the expense of slightly unbalancing the work force over the days of the week (but not, of course, to the point of infeasibility).

One may ask whether it could happen by chance that in one week all the slots are of one type, requiring an unbalanced work force. However, the algorithm that assigns weekends off will always put a varied mixture of slot types into any week by its cyclic nature; the more so as the number of workers in the category increases.

Undoubtedly, the extended algorithm is quite complicated. With practice and patience, one can become quite proficient at hand calculation. However, in a practical application, a computer program can easily be written. The algorithm provides a convenient set of rules that work well.

5. SUMMARY AND CONCLUSIONS

Staffing and scheduling decisions, complicated by varying worker requirements and assorted work rules, must be made frequently in organizations operating seven days a week. While integer programs can be used, it is often possible to determine the necessary work force size directly using simple formulas, and to create a feasible schedule for those workers in one pass.

This paper applies the direct combinatorial approach for the first time to a problem involving workers of different types, with more qualified staff being able to substitute for less qualified. Assuming daily requirements for D_k workers of type-k or better, with at least

Table V
Distribution of Work Stretch Lengths

Instance			Work Stretch (Days)							
			2		3		4		5	
A/B	d	D	#	%	#	%	#	%	#	%
2/7	(2,3,3)	(2,6,9)	1	1	27	21	83	66	15	12
2/5	(1,2,6,2)	(1,3,9,12)	8	5.7	47	33.6	82	58.6	3	2.1

d_k of type-k exactly, and with work rules mandating two days off each week and a specified fraction of weekends off for each worker, the necessary worker mix is specified and a simple scheduling algorithm is presented and proven feasible. The algorithm is elaborated to satisfy additional potential work rules.

Research is continuing to extend the results to cover worker requirements that vary over the week, and to include several shifts per day.

APPENDIX A

Proposition. *$W = \sum_{k=1}^{m} w_k$, where W is given in* (5) *and w_k in* (8) *and* (9).

Proof. Using (8) and (9), we show inductively for $i = 1, \ldots, m$

$$\sum_{k=1}^{i} w_k = \max_{k=1,\ldots,i} \left\{ f(D_k) + \sum_{j=k+1}^{i} f(d_j) \right\} \qquad \text{(A1)}$$

which produces the desired result when $i = m$.

For $i = 1$, $w_1 = f(D_1)$, which checks with (8). Suppose that (A1) is true for $i-1$. Then for i, after substituting for w_i from (9)

$$\sum_{k=1}^{i} w_k = \sum_{k=1}^{i-1} w_k + \max\left\{ f(d_i), f(D_i) - \sum_{k=1}^{i-1} w_k \right\}$$

$$= \max\left\{ f(d_i) + \sum_{k=1}^{i-1} w_k, f(D_i) \right\}.$$

Using the inductive assumption for the summation

$$\sum_{k=1}^{i} w_k = \max\left[\max_{k=1,\ldots,i-1} \left\{ f(D_k) + \sum_{j=k+1}^{i-1} f(d_j) \right\} + f(d_i), f(D_i) \right]$$

$$= \max_{k=1,\ldots,i} \left\{ f(D_k) + \sum_{j=k+1}^{i} f(d_j) \right\}.$$

APPENDIX B

Proposition. *The work force specified by* (8) *and* (9) *is always sufficient to provide adequate daily staffing in all categories and to satisfy off-day and off-weekend requirements when Algorithm 2 is used for scheduling.*

Recall from Section 3.1 that we can (and do) always give at least 2/7 of the weekends off, even if fewer are mandated. Thus, we assume $A/B \geq 2/7$. We will establish this result in a series of lemmas. It will be convenient to use x_k to stand for the average number of type-k workers off each weekend, Aw_k/B, as introduced earlier in (10). It will also be helpful to define

$$W_k = \sum_{j=1}^{k} w_j = \text{the total number of workers of types } 1, \ldots, k$$

and

$$X_k = AW_k/B = \textit{the average number of workers of types } 1, \ldots, k \textit{ off each weekend.}$$

Lemma 1. *Each worker has A out of B weekends off.*

Proof. The algorithm gives a total of Aw_k weekends off to the w_k workers of type-k over B weeks. Since they are assigned cyclically, each worker gets an equal share: A weekends off.

Lemma 2. *Each worker has two days off per week.*

Proof. The weekends that start and end a slot may each contribute an off-day. Additional weekdays are then given off, the number chosen precisely to bring the total to two.

Lemma 3. *At least d_k workers of type-k are on duty each weekend.*

Proof. Using (6), (7), (8) and (9)

$$w_k \geq f(d_k) \geq Bd_k/(B-A) \qquad \text{(B1)}$$

so that

$$(B-A)w_k/B \geq d_k \quad \text{or} \quad w_k - Aw_k/B \geq d_k.$$

Since w_k and d_k are integers, the inequality will remain valid if $x_k = Aw_k/B$ is rounded up:

$$w_k - \lceil x_k \rceil \geq d_k. \qquad \text{(B2)}$$

But this is precisely the desired result because at most $\lceil x_k \rceil$ workers of type-k have any weekend off, so that at least $w_k - \lceil x_k \rceil$ must be on duty.

Lemma 4. *At least D_k workers of types $1, \ldots, k$ are on duty each weekend.*

Proof. Using (6), (7), (8) and (9), and recalling that $W_k = \sum_{j=1}^{k} w_j$

$$w_k \geq f(D_k) - W_{k-1}$$

so that

$$W_k \geq f(D_k) \geq BD_k/(B-A).$$

Since we are careful to keep the cumulative numbers of workers on duty in each weekend as close to equal as possible as we advance from one worker type to the

next, and since a total of AW_k weekends off are given each B weeks to types $1, \ldots, k$ for an average of $X_k = AW_k/B$ each weekend, then the number of workers of types $1, \ldots, k$ who are off on any weekend is either $\lceil X_k \rceil$ or $\lfloor X_k \rfloor$.

The rest of the proof parallels the proof of Lemma 3 with D_k, W_k and X_k replacing d_k, w_k and x_k.

Lemma 5. *At least d_k workers of type-k are on duty each weekday.*

Proof. Since we will focus exclusively on a single worker type, we will suppress the subscript k throughout this proof. In any Sunday-through-Saturday week, for any type of worker, after Algorithm 1 has been applied, a certain number of weekend days off have already been given: either $\lceil x \rceil$ or $\lfloor x \rfloor$ Sundays at the start of the week, and either $\lceil x \rceil$ or $\lfloor x \rfloor$ Saturdays at the end. We will have to consider three cases, depending on the value of $y = B(\lfloor x \rfloor + 1 - x)$. Recall (see 11) that this is the number of weekends where the smaller numbers of workers are off. The more weekend days off there are in a week, the fewer weekdays off need be assigned and the easier it is to maintain d on duty.

Case 1. $y = B$. In this case, x is an integer and every weekend has exactly x workers off. Thus, in any week and for any worker type, there are $2x$ weekend days off and $2w - 2x$ weekdays off. Since days off are distributed over the five weekdays as equally as possible by the algorithm, the largest number of days off on any weekday is $\lceil (2w - 2x)/5 \rceil$. Thus, our mission is to establish that

$$w - \lceil (2w - 2x)/5 \rceil \geq d.$$

As we argued earlier, since w and d are integers it is equivalent to show

$$w - 2(w - x)/5 \geq d \quad \text{or}$$

$$3w/5 \geq d - 2x/5. \tag{B3}$$

Now to show this, using (B2) we know that

$$w \geq d + x. \tag{B4}$$

Also, since $A/B \geq 2/7$, we have $x \geq 2w/7$ or

$$2w/5 \leq 7x/5. \tag{B5}$$

Subtracting (B5) from (B4), we get the desired result.

Case 2. $B/2 < y < B$. With more than half the weekends having the smaller number, $\lfloor x \rfloor$, of workers off, there must be one or more weeks with such a weekend at both ends. This is the worst case, because with fewer days off on the weekends, more days off will have to be given on weekdays to make up the $2w$ off days per week.

In a week of this type, (B3) becomes

$$3w/5 \geq d - 2\lfloor x \rfloor/5. \tag{B6}$$

To establish this, we subtract (B5) from (B2), getting

$$3w/5 \geq d + \lceil x \rceil - 7x/5. \tag{B7}$$

In order for (B7) to imply (B6), we need

$$\lceil x \rceil - 7x/5 \geq -2\lfloor x \rfloor/5.$$

Replacing $\lfloor x \rfloor$ by $\lceil x \rceil - 1$, this becomes

$$\lceil x \rceil - x \geq 2/7 \quad \text{or}$$

$y \geq 2B/7$. Since we are considering the case $y > B/2$, we are done.

Case 3. $1 \leq y \leq B/2$. Now, the fewest workers off on the two weekends of any week is $\lfloor x \rfloor$ on one and $\lceil x \rceil$ on the other. The version of (B3) that must be established is

$$3w/5 \geq d - \lfloor x \rfloor/5 - \lceil x \rceil/5. \tag{B8}$$

From (1) and (9), $w \geq \lceil 7d/5 \rceil$, and since $A/B \geq 2/7$, $x \geq 2w/7 \geq 2\lceil 7d/5 \rceil/7$. Thus, it will suffice to show that

$$3\lceil 7d/5 \rceil + \lfloor 2\lceil 7d/5 \rceil/7 \rfloor + \lceil 2\lceil 7d/5 \rceil/7 \rceil \geq 5d. \tag{B9}$$

To handle the rounding up of $7d/5$, we write d as a multiple of 5 plus a remainder: $d = 5q + r$ for some integer q and $r = 0, 1, 2, 3$ or 4. Then

$$\lceil 7d/5 \rceil = 7q + h \quad \text{where } h = \lceil 7r/5 \rceil.$$

Now (B9) becomes

$$21q + 3h + 2q + \lfloor 2h/7 \rfloor + 2q + \lceil 2h/7 \rceil \geq 25q + 5r$$

or

$$3h + \lfloor 2h/7 \rfloor + \lceil 2h/7 \rceil \geq 5r. \tag{B10}$$

To verify this, since r has just five values, we consider each case:

r	h	(B10)
0	0	$0 + 0 + 0 \geq 0$
1	2	$6 + 0 + 1 \geq 5$
2	3	$9 + 0 + 1 \geq 10$
3	5	$15 + 1 + 2 \geq 15$
4	6	$18 + 1 + 2 \geq 20$

This establishes (B10), and hence (B8) is proven.

Lemma 6. *At least D_k workers of types $1, \ldots, k$ are on duty each weekday.*

Proof. Since, for any week, weekdays off are scheduled continuously through the successive worker types; and since the total weekends off for types $1, \ldots, k$ have the same properties as for a single type of worker: the number of workers off on any weekend is either $\lceil X_k \rceil$ or $\lfloor X_k \rfloor$, with the two numbers alternating as far as possible; and since the defining inequalities for w_k, such as (B2) and (B5) have their exact analog for W_k, it follows that the proof of this lemma exactly parallels the proof of Lemma 5, with D_k, W_k and X_k replacing d_k, w_k and x_k.

APPENDIX C

Proposition. *For any A and B with $A/B \geqslant 2/7$, no week stretch exceeds 4 weeks.*

Proof. A week stretch is the time between successive weekends off for a worker. The worker who is last on the list to get a certain weekend off will get another off-weekend when all w workers in the same category have been given one. We will show that this must happen in four weeks or less. Since we are focusing on one category of worker throughout this appendix, we will suppress the subscript k.

Suppose that $x = Aw/B$ is an integer. Then we have exactly x workers off each week, and our objective is to show that $4x \geqslant w$, or

$$4Aw/B \geqslant w, \quad A/B \geqslant 1/4.$$

But this we know because $A/B \geqslant 2/7$.

In general, we know that at least $\lfloor x \rfloor$ workers are off on any weekend. In the worst case, four consecutive weeks will all have $\lfloor x \rfloor$ workers off. Thus, it is clearly sufficient to show $4\lfloor x \rfloor \geqslant w$, or since $x = Aw/B \geqslant 2w/7$, it is enough to prove

$$4\lfloor 2w/7 \rfloor \geqslant w.$$

By defining $w = 7q + r$ for some integer $q \geqslant 0$ and $r = 0, 1, \ldots, 6$, we can rewrite this as

$$q + 4\lfloor 2r/7 \rfloor \geqslant r$$

and consider the seven cases. It turns out that there are several values of w for which this inequality does not hold: $w = 1$, 2, 3, 5, 6, 9, 10, 13 and 17. Perhaps surprisingly, this crude test suffices to establish the result for all other values.

The remaining nine values of w have been checked, one at a time, with $A/B = 2/7$. Note that $w = 1$ is a special, but trivial, case that should be handled separately. For all the rest, Algorithm 1 gives week stretches of three or four weeks only. Finally, for larger values of A/B, weekends off are more frequent and so week stretches get shorter.

ACKNOWLEDGMENT

We would like to thank Rudy Hung for helpful suggestions.

REFERENCES

BAKER, K. R. 1974. Scheduling a Full-Time Workforce to Meet Cyclic Staffing Requirements. *Mgmt. Sci.* **20,** 1561–1568.

BAKER, K. R., R. N. BURNS AND M. W. CARTER. 1979. Staff Scheduling With Day-Off and Workstretch Constraints. *AIIE Trans.* **11,** 286–292.

BAKER, K. R., AND M. J. MAGAZINE. 1977. Workforce Scheduling With Cyclic Demands and Day-Off Constraints. *Mgmt. Sci.* **24,** 161–167.

BROWNWELL, W. S., AND J. W. LOWERRE. 1976. Scheduling of Work Forces Required in Continuous Operations Under Alternative Labor Policies. *Mgmt. Sci.* **22,** 597–605.

BURNS, R. N. 1978. Manpower Scheduling With Variable Demands and Alternate Weekends Off. *INFOR* **16,** 101–111.

BURNS, R. N., AND M. W. CARTER. 1985. Work Force Size and Single Shift Schedules With Variable Demands. *Mgmt. Sci.* **31,** 599–607.

EMMONS, H. 1985. Workforce Scheduling With Cyclic Requirements and Constraints on Days Off, Weekends Off, and Workstretch. *IIE Trans.* **17,** 8–16.

MILLER, H. E., W. P. PIERSKALLA, AND G. J. RATH. 1976. Nurse Scheduling Using Mathematical Programming. *Opns. Res.* **24,** 857–870.

SEGAL, M. 1974. The Operator Scheduling Problem: A Network Flow Approach. *Opns. Res.* **4,** 808–823.

TIBREWALA, R., D. PHILIPPE AND J. BROWNE. 1972. Optimal Scheduling of Two Consecutive Idle Periods. *Mgmt. Sci.* **19,** 71–75.

WARNER, M. D. 1976. Scheduling Nursing Personnel According to Nursing Preference: A Mathematical Programming Approach. *Opns. Res.* **24,** 842–870.