

**12.8.** Consider a hospital with a single operating room. The IC unit has 4 beds. The operating room has to be scheduled from Monday through Friday; the IC unit is staffed 7 days a week. There are two types of elective surgeries that have to be scheduled. Both types of surgeries require half a day of the operating room. However, a patient that undergoes surgery of the first type needs to remain two days in the IC unit, whereas a patient that undergoes surgery of the second type needs four days of care in the IC unit. Each week six surgeries of the first type and four surgeries of the second type have to be scheduled. What is the optimal schedule?

## Comments and References

An enormous amount of research has been done on planning and scheduling in health care. Pierskalla and Brailer (1994) presented an overview of Operations Research applications in health care delivery. Brandeau, Sainfort and Pierskalla (2004) edited a handbook on Operations Research in health care. Several chapters in this book focus on planning and scheduling applications in health care.

A fair amount of research has been done on surgery scheduling, at times also referred to as operating theatre scheduling. It has been clear from the outset that surgery scheduling is a form of stochastic scheduling in which the durations of the operations are random variables drawn from known distributions. May, Strum and Vargas (2000) did a thorough study indicating that the Lognormal distribution does fit operating room data quite nicely. The first analysis of a single operating room with two consecutive surgeries was done by Weiss (1990). Since then, the single operating room scheduling problem has received a significant amount of attention; see, for example, Denton, Viapiano and Vogl (2007). The set packing formulation for the multiple operating room scheduling problem is due to Velasquez and Melo (2006); see also Velasquez, Melo, and Kuefer (2008). The robust surgery loading problem (assuming random durations) is due to Hans, Wullink, van Houdenhoven, and Kazemier (2008).

Conforti, Guerriero, and Guido (2008) developed the integer programming formulation for the radiotherapy treatment problem. A similar problem has also been studied by Fei, Combes, Meskens, and Chu (2006).

The constraint programming approach for the emergency room staffing problem was developed by Rousseau, Gendreau and Pesant (2002); see also Gendreau, Ferland, Gendron, Hail, Jaumard, Lapierre, Pesant and Soriano (2006).

The case study relating the surgery scheduling problem with bed occupancy levels is due to Belien (2006) and Belien and Demeulemeester (2007). As discussed in Section 12.6, the problem underlying this case can be modeled as a flexible flow shop with no buffer storages between the stages. Hsu, de Matta, and Lee (2003) indeed modeled a similar problem as a flexible flow shop and developed a tabu-search heuristic to optimize their environment. Pham and Klinkert (2008) modeled the surgical case scheduling as a flexible job shop and developed a Mixed Integer Programming formulation. Another interesting case study is due to Vissers, Adan, and Bekkers (2005).

## Chapter 13

# Workforce Scheduling

<b>13.1 Introduction</b>	<b>317</b>
<b>13.2 Days-Off Scheduling</b>	<b>318</b>
<b>13.3 Shift Scheduling</b>	<b>324</b>
<b>13.4 The Cyclic Staffing Problem</b>	<b>327</b>
<b>13.5 Applications and Extensions of Cyclic Staffing</b>	<b>329</b>
<b>13.6 Crew Scheduling</b>	<b>331</b>
<b>13.7 Operator Scheduling in a Call Center</b>	<b>335</b>
<b>13.8 Discussion</b>	<b>339</b>

## 13.1 Introduction

Workforce allocation and personnel scheduling deal with the arrangement of work schedules and the assignment of personnel to shifts in order to cover the demand for resources that vary over time. These problems are very important in service industries, e.g., telephone operators, hospital nurses, policemen, transportation personnel (plane crews, bus drivers), and so on. In these environments the operations are often prolonged and irregular and the staff requirements fluctuate over time. The schedules are typically subject to various constraints dictated by equipment requirements, union rules, and so on. The resulting problems tend to be combinatorially hard.

In this chapter we first consider a somewhat elementary personnel scheduling problem for which there is a relatively simple solution. We then describe an integer programming framework that encompasses a large class of personnel scheduling problems. We subsequently consider a special class of these integer programming problems, namely the cyclic staffing problems. This class of problems has many applications in practice and is easy from a combinatorial point of view. We then consider several special cases and extensions of cyclic

staffing. In the sixth section we discuss the crew scheduling problems that occur in the airline industry. In the subsequent section we describe a case that involves the scheduling of operators in a call center.

### 13.2 Days-Off Scheduling

We first consider a fairly elementary personnel assignment problem. Each day of the week a number of employees have to be present. The number may differ from day to day, but the set of requirements remains the same from week to week. There is a total number of employees available and each has to be assigned a sequence of days. However, the assignment of days to any given employee may be different from one week to the next. By a week, we mean seven days that start with a Sunday and end with a Saturday. The problem is to find the minimum number of employees to cover a seven day a week operation such that the following constraints are satisfied.

- (i) The demand per day,  $n_j, j = 1, \dots, 7$ , ( $n_1$  is Sunday and  $n_7$  is Saturday) is met.
- (ii) Each employee is given  $k_1$  out of every  $k_2$  weekends off.
- (iii) Each employee works exactly 5 out of 7 days (from Sunday to Saturday).
- (iv) Each employee works no more than 6 consecutive days.

These constraints can have certain effects on the schedule of an employee. For example, if an employee has one weekend off, then he cannot work six days straight and have his next day off on the following Sunday, because he violates then the third constraint (working exactly 5 days out of seven days that start with a Sunday and end with a Saturday). However, an employee can have a consecutive Saturday, Sunday and Monday off, as long as he works after that for at least 5 days in a row. Actually, he could have a weekend off and take again a single day off after 2 or 3 days of work.

We now describe a method that generates an optimal schedule one week at a time, i.e., after the schedule for week  $i$  has been set, the schedule for week  $i + 1$  is determined, and so on. It turns out that there exists a cyclic optimal schedule that, after a number of weeks, repeats itself.

There are three simple lower bounds on the minimum size of the workforce,  $W$ . First, there is the weekend constraint. The average number of employees available each weekend must be sufficient to meet the maximum weekend demand. In  $k_2$  weeks each employee is available for  $k_2 - k_1$  weekends. So, assuming that (as close as possible) the same number of workers get each of the  $k_2$  weekends off:

$$(k_2 - k_1)W \geq k_2 \max(n_1, n_7)$$

and therefore

$$W \geq \left\lceil \frac{k_2 \max(n_1, n_7)}{k_2 - k_1} \right\rceil.$$

Second, there is the total demand constraint. The total number of employee days per week must be sufficient to meet the total weekly demand. Since each employee works five days per week,

$$5W \geq \sum_{j=1}^7 n_j$$

or

$$W \geq \left\lceil \frac{1}{5} \sum_{j=1}^7 n_j \right\rceil.$$

Third, we have the maximum daily demand constraint

$$W \geq \max(n_1, \dots, n_7).$$

The minimum workforce must be at least as large as the largest of these three lower bounds. In what follows, we present an algorithm that yields a schedule that requires a workforce of a size equal to the largest of these three lower bounds.

The algorithm that solves this problem, i.e., that finds a schedule that satisfies all constraints using the smallest possible workforce, is relatively simple. Let  $W$  denote the maximum of the three lower bounds and let  $n$  denote the maximum weekend demand, i.e.,

$$n = \max(n_1, n_7).$$

Let  $u_j = W - n_j$ , for  $j = 2, \dots, 6$ , and  $u_j = n - n_j$ , for  $j = 1$  and  $7$ ; the  $u_j$  is the surplus number of employees with regard to day  $j$ . The second lower bound guarantees that

$$\sum_{j=1}^7 u_j \geq 2n.$$

It is clear that employees should be given days off on those days that have a large surplus of employees. The algorithm that constructs the schedule uses a list of so-called off-day pairs. The pairs in this list are numbered from 1 to  $n$  and the list is created as follows: First, choose day  $k$  such that

$$u_k = \max(u_1, \dots, u_7).$$

Second, choose day  $l$ , ( $l \neq k$ ), such that  $u_l > 0$ ; if  $u_l = 0$  for all  $l \neq k$ , then choose  $l = k$ . Third, add pair  $(k, l)$  to the list and decrease both  $u_k$  and  $u_l$  by 1. Repeat this procedure  $n$  times. At the end of the list pairs of the form  $(k, k)$  may appear; these pairs are called nondistinct pairs.

Now number the employees from 1 to  $W$ . Note that since the maximum demand during a weekend is  $n$ , the remaining  $W - n$  employees can have that weekend off. Assume that the first day to be scheduled falls on a Saturday, and the first and second days are weekend 1.

**Algorithm 13.2.1 (Days-Off Scheduling).**

Step 1. (Schedule the weekends off)

Assign the first weekend off to the first  $W - n$  employees.

Assign the second weekend off to the second  $W - n$  employees.

This process is continued cyclically with employee 1 being treated as the next employee after employee  $W$ .

Step 2. (Categorization of employees in week 1)

In week 1 each employee falls into one of four categories.

Type T1: weekend 1 off; 0 off days needed during week 1; weekend 2 off

Type T2: weekend 1 off; 1 off days needed during week 1; weekend 2 on

Type T3: weekend 1 on; 1 off days needed during week 1; weekend 2 off

Type T4: weekend 1 on; 2 off days needed during week 1; weekend 2 on

Since there are exactly  $n$  people working each weekend,

$$\begin{aligned} |T3| + |T4| &= n \text{ (because of weekend 1);} \\ |T2| + |T4| &= n \text{ (because of weekend 2).} \end{aligned}$$

It follows that  $|T2| = |T3|$ .

Pair each employee of T2 with one employee of T3.

Step 3. (Assigning off-day pairs to employees in week 1)

Assign the  $n$  pairs from the top of the list.

First to employees of T4: each employee of T4 gets both days off.

Second to employees of T3: each employee of T3 gets from his pair the earlier day off and his companion of T2 gets from that same pair the later day off. (So each employee of T3 and T2 gets one day off in week 1, as required.)

Step 4. (Assigning off-day pairs to employees in week  $i$ )

Assume a schedule has been created for weeks  $1, \dots, i - 1$ .

A categorization of employees can be done for week  $i$  in the same way as in Step 2. In order to assign employees to off-day pairs two cases have to be considered.

Case (a): (All off-day pairs in the list are distinct)

Employees of T4 and T3 are associated with the same pairs as those they were associated with in week  $i - 1$ .

A T4 employee gets from his pair both days off.

A T3 employee gets from the pair he is associated with the earlier day off and his companion of T2 gets from that pair the later day off.

Case (b): (Not all off-day pairs in the list are distinct)

Week  $i$  is scheduled in exactly the same way as week 1, independent of week  $i - 1$ .

Set  $i = i + 1$  and return to Step 4.

This algorithm needs some motivation. First, it may not be immediately clear that we never will be confronted with the need to schedule a nondistinct pair of days to a type T4 worker. In that case a worker needs two weekdays off in a week, and we try to give him the same day twice. It can be shown that the number of T4 employees is always smaller than or equal to the number of distinct pairs.

If there are non-distinct pairs, i.e., pairs  $(k, k)$ , in the off-days list, then week  $i$  is independent of week  $i - 1$ . It can be shown that every pair contains day  $k$  and the maximum workstretch from week  $i - 1$  to week  $i$  is 6 days. The only time that the workstretch is greater than 5 days is when there are nondistinct pairs.

In the next example, there is one distinct pair, one nondistinct pair, and  $|T4| = 1$ .

**Example 13.2.2 (Application of Days-Off Scheduling Algorithm).**

Consider the problem with the following daily requirements.

day $j$	1	2	3	4	5	6	7
	Sun.	Mon.	Tues.	Wed.	Thurs.	Fri.	Sat.
Requirement	1	0	3	3	3	3	2

The maximum weekend demand is  $n = 2$  and each person requires 1 out of 3 weekends off, i.e.,  $k_1 = 1$  and  $k_2 = 3$ . So

$$W \geq [(3 \times 2)/(3 - 1)] = 3,$$

$$W \geq \lceil 15/5 \rceil = 3,$$

$$W \geq 3.$$

So the minimum number of employees  $W$  is 3 and  $W - n = 1$ . We assign a weekend off to one employee each week. This results in the following assignment of weekend days off for the three employees.

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X																				X
2							X	X														
3												X	X									

At this point, there is one surplus employee on Sunday and three on Monday.

day $j$	1	2	3	4	5	6	7
$u_j$	1	3	0	0	0	0	0

There are 2 pairs of off days, one distinct and one non-distinct.

Pair 1: Sunday - Monday;

Pair 2: Monday - Monday;

We will use these pairs for each week.

There is one nondistinct pair in the list. The categorization of the employees in the first week results in the following categories: The first pair is assigned to the  $T4$  employee (one each week) and the second pair is split between the remaining two employees (types  $T2$  and  $T3$ ).

Applying the next step of the algorithm yields the following schedule.

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X	X						X	X						X						X
2			X					X	X	X						X	X					
3		X	X							X				X	X	X	X					

The schedule produces a six-day workstretch for one employee each week. This cannot be avoided since the solution is unique.

It can be shown that if all off-day pairs are distinct then the maximum workstretch is 5 days. In the next example all off-day pairs are distinct.

### Example 13.2.3 (Application of Days-Off Scheduling Algorithm).

Consider the problem with the following daily requirements.

day $j$	1	2	3	4	5	6	7
	Sun.	Mon.	Tues.	Wed.	Thurs.	Fri.	Sat.
Requirement	3	5	5	5	7	7	3

The maximum weekend demand  $n = 3$  and each person requires 3 out of 5 weekends off, i.e.,  $k_1 = 3$  and  $k_2 = 5$ . So

$$W \geq \lceil (5 \times 3)/2 \rceil = 8,$$

$$W \geq \lceil 35/5 \rceil = 7,$$

$$W \geq 7.$$

So the minimum number of employees  $W$  is 8 and  $W - n = 5$ . We assign weekends off to 5 employees each week. This results in the following assignment of weekend days off for the eight employees.

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X						X	X													X
2	X	X						X	X													X
3	X	X												X	X							X
4	X	X												X	X							X
5	X	X												X	X							
6								X	X					X	X							
7								X	X					X	X							
8								X	X													X

At this point, there are 8 people available each weekday, so the surplus  $u_j$  is

day $j$	1	2	3	4	5	6	7
$u_j$	0	3	3	3	1	1	0

There are a number of ways in which 3 pairs of off days can be chosen. For example,

Pair 1: Monday - Tuesday;

Pair 2: Tuesday - Wednesday;

Pair 3: Tuesday - Wednesday.

We will use these pairs for each week.

There are no nondistinct pairs on the list. The categorization of the employees in the first week results in the following 4 categories:

T1: 1, 2

T2: 3, 4, 5

T3: 6, 7, 8

T4: -

Employee 3 is paired with 6, 4 with 7 and 5 with 8. Thus we need three pairs of weekdays to give off.

Categorization of the employees in the second week yields the following 4 categories.

T1: 6, 7

T2: 1, 2, 8

T3: 3, 4, 5

T4: -

Employee 1 is paired with 3, 2 with 4, and 8 with 5.

Categorization of the employees in the third week results in the following 4 categories.

T1: 3, 4

T2: 5, 6, 7

T3: 1, 2, 8

T4: -

Employee 1 is paired with 6, 2 with 5, and 8 with 7.

Applying the next step of the algorithm results in the schedule shown by the following table:

	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	X	X						X	X		X					X						
2	X	X						X	X		X					X						
3	X	X						X	X		X					X						
4	X	X						X	X		X					X						
5	X	X						X	X		X					X						
6			X					X	X		X					X						
7			X					X	X		X					X						
8			X					X	X		X					X						

The arrows illustrate how a pair of off-days is shared by two employees. All pairs are distinct, so the maximum workstretch is 5 days. It can be easily verified that the schedule is cyclic and the cycle is 8 weeks.

It can be shown that schedules generated by the algorithm always satisfy the constraints. Because of the way the off-weekends are distributed over the employees (evenly) and because of the first lower bound, it is assured that each employee is given at least  $k_1$  out of  $k_2$  weekends off. That each employee works exactly 5 days out of the week (from Sunday to Saturday) follows immediately from the algorithm. That no employee works more than 6 days in one stretch is a little harder to see. An employee may have a six day workstretch (but not longer) when there are non-distinct pairs (if all pairs are distinct, then the longest workstretch is 5 days). If there are non-distinct pairs  $(k, k)$ , then day  $k$  has to appear in all pairs. In the worst case, an employee can be associated with pair  $(j, k)$  in week  $i - 1$  and pair  $(k, l)$  in week  $i$ , where  $j \leq k \leq l$ . In this case, he will receive at least day  $k$  off in week  $i - 1$  as well as in week  $i$  which results in a six day workstretch. The stretch is smaller if either  $k < j$  or  $l < k$ , for then he would receive day  $j$  or day  $l$  off.

It can be shown that there exists an optimal schedule that is cyclic and that the algorithm may yield this schedule. The number of weeks in such a cyclic schedule can be computed fairly easily (see Exercise 13.1).

### 13.3 Shift Scheduling

In the scheduling problem discussed in the previous section there are various assignment patterns over the cycle. The cost of assigning an employee to a certain work pattern is the same for each pattern and the objective is to minimize the total number of employees. The fact that each assignment pattern has the same cost is one reason why the problem is relatively easy.

In this section we consider a more general personnel scheduling problem and follow a completely different approach. We consider a cycle that is fixed

in advance. In certain settings the cycle may be a single day, while in others it may be a week or a number of weeks. In contrast to the previous section, each work assignment pattern over a cycle has its own cost and the objective is to minimize the total cost.

The problem can be formulated as follows: The predetermined cycle consists of  $m$  time intervals or periods. The lengths of the periods do not necessarily have to be identical. During period  $i$ ,  $i = 1, \dots, m$ , the presence of  $b_i$  personnel is required. The number  $b_i$  is, of course, an integer. There are  $n$  different shift patterns and each employee is assigned to one and only one pattern. Shift pattern  $j$  is defined by a vector  $(a_{1j}, a_{2j}, \dots, a_{mj})$ . The value  $a_{ij}$  is either 0 or 1; it is a 1 if period  $i$  is a work period and 0 otherwise. Let  $c_j$  denote the cost of assigning a person to shift  $j$  and  $x_j$  the (integer) decision variable representing the number of people assigned to shift  $j$ . The problem of minimizing the total cost of assigning personnel to meet demand can be formulated as the following integer programming problem:

$$\text{minimize } c_1x_1 + c_2x_2 + \dots + c_nx_n$$

subject to

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \geq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \geq b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \geq b_m$$

$$x_j \geq 0 \quad \text{for } j = 1, \dots, n,$$

with  $x_1, \dots, x_n$  integer. In matrix form this integer program is written as follows.

$$\text{minimize } \bar{c}\bar{x}$$

subject to

$$\mathbf{A}\bar{x} \geq \bar{b}$$

$$\bar{x} \geq 0.$$

Such an integer programming problem is known to be strongly NP-hard in general. However, the  $\mathbf{A}$  matrix may often exhibit a special structure. For example, shift  $j$ ,  $(a_{1j}, \dots, a_{mj})$ , may contain a *contiguous* set of 1's (a contiguous set of 1's implies that there are no 0's in between 1's). However, the *number* of 1's may often vary from shift to shift, since it is possible that some shifts have to work longer hours or more days than other shifts.

**Example 13.3.1 (Shift Scheduling at a Retail Store).** Consider a retail store that is open for business from 10 a.m. to 9 p.m. There are five shift patterns.

pattern	Hours of Work	Total Hours	Cost
1	10 a.m. to 6 p.m.	8	\$ 50.00
2	1 p.m. to 9 p.m.	8	\$ 60.00
3	12 p.m. to 6 p.m.	6	\$ 30.00
4	10 a.m. to 1 p.m.	3	\$ 15.00
5	6 p.m. to 9 p.m.	3	\$ 16.00

Staffing requirements at the store vary from hour to hour.

Hour	Staffing Requirement
10 a.m. to 11 a.m.	3
11 a.m. to 12 a.m.	4
12 a.m. to 1 p.m.	6
1 p.m. to 2 p.m.	4
2 p.m. to 3 p.m.	7
3 p.m. to 4 p.m.	8
4 p.m. to 5 p.m.	7
5 p.m. to 6 p.m.	6
6 p.m. to 7 p.m.	4
7 p.m. to 8 p.m.	7
8 p.m. to 9 p.m.	8

The problem can be formulated as an integer program with the following  $\bar{c}$  vector,  $\mathbf{A}$  matrix and  $\bar{b}$  vector:

$$\bar{c} = (50, 60, 30, 15, 16)$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad \bar{b} = \begin{bmatrix} 3 \\ 4 \\ 6 \\ 4 \\ 7 \\ 8 \\ 7 \\ 6 \\ 4 \\ 7 \\ 8 \end{bmatrix}$$

Clearly, an integer solution  $x_1, x_2, x_3, x_4, x_5$  is required. However, solving the linear programming relaxation of this problem yields the solution  $(0, 0, 8, 4, 8)$ . Since this solution is integer, it is clear that it is also optimal for the integer programming formulation.

Even though the integer programming formulation of the general personnel scheduling problem (with an arbitrary  $0-1$   $\mathbf{A}$  matrix) is NP-hard, the special case with each column containing a contiguous set of 1's is easy. It can be shown that the solution of the linear programming relaxation is always integer. There are several other important special cases that are solvable in polynomial time. In the next section we discuss one of them.

### 13.4 The Cyclic Staffing Problem

A classical personnel scheduling problem is the *cyclic staffing* problem. The objective is to minimize the cost of assigning people to an  $m$  period cyclic schedule so that sufficient workers are present during time period  $i$ , in order to meet requirement  $b_i$ , and each person works a shift of  $k$  consecutive periods and is free the other  $m - k$  periods. Notice that period  $m$  is followed by period 1.

An example is the  $(5, 7)$ -cyclic staffing problem, where the cycle is seven days and any person works 5 consecutive days followed by two days off. As described in the previous section, this problem can be formulated as an integer program. In the integer programming formulation a column vector of the  $\mathbf{A}$  matrix denotes a possible shift assignment that specifies which two consecutive days are off and which 5 days are workdays. There are 7 possible column vectors. Even though the  $\mathbf{A}$  matrix has a very special structure in this case, the columns do not always have a contiguous set of 1's. So this is not a special case of the example described in the previous section.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The cost  $c_j$  of column vector  $j$ , i.e.,  $a_{1j}, \dots, a_{mj}$ , represents the cost of having one person work according to the corresponding schedule. The  $\bar{b}$  vector is again the requirements vector, i.e.,  $b_i$  denotes the number of people that have to be present on day  $i$ . The integer decision variable  $x_j$  represents the number of people that work according to the schedule defined by column vector  $j$ . This results in an integer programming problem with a special structure.

The special structure of this integer programming problem makes it possible to solve it in an efficient manner. Actually, it can be shown that the solution of the linear program relaxation of this problem is very close to the solution of the integer programming problem. Because of this the following algorithm leads to an optimal solution.

**Algorithm 13.4.1 (Minimizing Cost in Cyclic Staffing).**

Step 1.

Solve the linear relaxation of the original problem to obtain  $x'_1, \dots, x'_n$ .

If  $x'_1, \dots, x'_n$  are integer, then it is optimal for the original problem. STOP.

Otherwise go to Step 2.

Step 2.

Form two linear programs  $LP'$  and  $LP''$  from the relaxation of the original problem by adding respectively the constraints

$$x_1 + \dots + x_n = \lfloor x'_1 + \dots + x'_n \rfloor$$

and

$$x_1 + \dots + x_n = \lceil x'_1 + \dots + x'_n \rceil.$$

$LP''$  always will have an optimal solution that is integer.

If  $LP'$  does not have a feasible solution, then the solution of  $LP''$  is an optimal solution to the original problem.

If  $LP'$  has a feasible solution, then it has an optimal solution that is integer and the solution to the original problem is the better one of the solutions to  $LP'$  and  $LP''$ . STOP.

The next example illustrates the use of the algorithm.

**Example 13.4.2 (Minimizing Cost in Cyclic Staffing).** Consider the (3,5)-cyclic staffing problem

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \quad \bar{b} = \begin{bmatrix} 3 \\ 4 \\ 6 \\ 4 \\ 7 \end{bmatrix}$$

The cost vector is

$$\bar{c} = (3.6, 4.8, 5.5, 3.7, 5.2)$$

Applying Algorithm 13.4.1 leads to the following results.

Step 1. Solving the linear programming relaxation yields

$$\bar{x}' = (1.5, 0, 4.5, 0, 2.5).$$

The value of the objective function is 43.15.

Step 2. Adding to the original problem the constraint

$$x_1 + x_2 + x_3 + x_4 + x_5 = 8$$

results in a problem without a feasible solution. Adding to the original problem the constraint

$$x_1 + x_2 + x_3 + x_4 + x_5 = 9$$

yields  $\bar{x} = (2, 0, 4, 1, 2)$  with objective value 43.3. So the optimal solution is  $\bar{x} = (2, 0, 4, 1, 2)$  with objective value 43.3.

## 13.5 Applications and Extensions of Cyclic Staffing

In this section we discuss three applications of cyclic staffing.

(i) *Days-Off scheduling.* Consider the following special case of the problem discussed in Section 13.2. Each employee is guaranteed two days off a week, including every other weekend (a week starts with a Sunday and ends with a Saturday) and is not allowed to work more than 6 days consecutively. This problem can be formulated as an integer program with the following  $A$  matrix.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 1 & 1 & 1 & 1 & \dots \\ - & - & - & - & - & - & \dots & - & - & - & - & - & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 1 & 1 & 0 & 1 & \dots \\ 0 & 0 & 1 & 1 & 1 & 1 & \dots & 0 & 1 & 1 & 0 & 1 & 1 & \dots \\ 1 & 1 & 0 & 0 & 0 & 1 & \dots & 1 & 1 & 0 & 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & 1 & 1 & 0 & \dots & 1 & 1 & 1 & 1 & 1 & 0 & \dots \\ 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & 1 & 1 & 1 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ - & - & - & - & - & - & \dots & - & - & - & - & - & \dots \\ 1 & 0 & 1 & 1 & 0 & 1 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 1 & 0 & 1 & 1 & \dots & 0 & 0 & 1 & 1 & 1 & 1 & \dots \\ 1 & 1 & 0 & 1 & 1 & 1 & \dots & 1 & 1 & 0 & 0 & 0 & 1 & \dots \\ 1 & 1 & 1 & 1 & 1 & 0 & \dots & 1 & 1 & 1 & 1 & 1 & 0 & \dots \\ 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ - & - & - & - & - & - & \dots & - & - & - & - & - & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 1 & 1 & 0 & 1 & \dots \end{bmatrix}$$

The number of possible patterns tends to be somewhat large, since there are many patterns that satisfy the conditions stated. Each row in the matrix represents a day in the week. The first two rows, the middle two rows and the last two rows represent weekends. The first group of columns correspond to the assignments with the first and the third weekend off and the second group of columns to the assignments with the second weekend off. This problem can be solved by the technique discussed in the previous section.

The general model described in Section 13.2 can also be viewed as a cyclic staffing problem. However, it does not fit the framework described in Section

13.4 that well, because of a number of differences. For starters, the cycle length is not fixed a priori. In addition, even if the cycle length were fixed, it would be hard to describe this problem as an integer programming problem. The number of possible columns is very large and not easy to enumerate.

(ii) *Cyclic staffing with overtime.* A basic staffing problem occurs in facilities such as hospitals that operate around the clock. Suppose that there are fixed hourly staff requirements  $b_i$ , and three basic work shifts, each of 8 hours duration: 8 a.m. to 4 p.m., 4 p.m. to midnight and 12 p.m. to 8 a.m. Overtime of up to an additional 8 hours is possible for each shift. A personnel assignment that meets all staffing requirements at minimum cost has to be found. The constraint matrix  $\mathbf{A}$  consists of 9 submatrices, each with 8 rows and 9 columns.

$$\begin{bmatrix} 1 & 0 & 0 \backslash 1 \\ 0 \backslash 1 & 1 & 0 \\ 0 & 0 \backslash 1 & 1 \end{bmatrix}$$

The submatrix  $\mathbf{0}$  is a matrix with all entries 0. The submatrix  $\mathbf{1}$  is a matrix with all entries 1 and the submatrix  $\mathbf{0 \backslash 1}$  is the matrix

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

This problem can be solved by linear programming.

(iii) *Cyclic Staffing with Linear Penalties for Understaffing and Overstaffing.* Suppose the demands for each period are not fixed. There is a linear penalty  $c'_i$  for understaffing and a linear penalty  $c''_i$  for overstaffing. The penalty  $c''_i$  may actually be negative, since there may be some benefits in overstaffing. Let  $x'_i$  denote the level of understaffing during period  $i$ . The level of overstaffing during period  $i$  is then

$$b_i - (a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n) - x'_i.$$

The problem can now be formulated as the following linear program.

$$\text{minimize } \bar{c}\bar{x} + \bar{c}'\bar{x}' + \bar{c}''(\bar{b} - \mathbf{A}\bar{x} - \bar{x}')$$

subject to

$$\begin{aligned} \mathbf{A}\bar{x} + I\bar{x}' &\geq \bar{b} \\ \bar{x}, \bar{x}' &\geq 0 \quad \bar{x}, \bar{x}' \text{ integer.} \end{aligned}$$

If  $\mathbf{A}$  is the matrix for the  $(k, m)$  staffing problem (or any other row circular matrix), the problem can be solved by the algorithm described in the previous section, provided that the problem is bounded from below. It turns out that the problem is bounded from below if and only if  $\bar{c} - \bar{c}''\mathbf{A} \geq 0$  and  $\bar{c}' - \bar{c}'' \geq 0$ .

## 13.6 Crew Scheduling

Crew scheduling problems are very important in the transportation industry, especially in the airline industry. The underlying model is different from the models considered in the previous sections and so are the solution techniques.

Consider a set of  $m$  jobs, e.g., flight legs. A flight leg is characterized by a point of departure and a point of arrival, as well as an approximate time interval during which the flight has to take place. There is a set of  $n$  feasible and permissible combinations of flight legs that one crew can handle, e.g., round trips or tours (the number  $n$  usually is very large). A round trip may consist of several flight legs, i.e., a plane may leave city A for city B, then go to city C, before returning to city A. Any given flight leg may be part of many round trips. Round trip  $j$ ,  $j = 1, \dots, n$ , has a cost  $c_j$ . Setting up a crew schedule is equivalent to determining which round trips should be selected and which ones not. The objective is to choose a set of round trips with a minimum total cost in such a way that each flight leg is covered exactly once by one and only one round trip.

In order to formulate this crew scheduling problem as an integer program some notation is required. If flight leg  $i$  is part of round trip  $j$ , then  $a_{ij}$  is 1, otherwise  $a_{ij}$  is 0. Let  $x_j$  denote a 0 – 1 decision variable that assumes the value 1 if round trip  $j$  is selected and 0 otherwise. The crew scheduling problem can be formulated as the following integer program.

$$\text{minimize } c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

subject to

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = 1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = 1$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = 1$$

$$x_j \in \{0, 1\} \quad \text{for } j = 1, \dots, n.$$

Each column in the  $\mathbf{A}$  matrix is a round trip and each row is a flight leg that must be covered exactly once by one round trip. The optimization problem is then to select, at minimum cost, a set of round trips that satisfies the constraints. The constraints in this problem are often called the partitioning

equations and this integer programming problem is referred to as the *Set Partitioning* problem (see Appendix A). For a feasible solution  $(x_1, \dots, x_n)$ , the variables that are equal to 1 are referred to as the *partition*. In what follows we denote a partition  $l$  by  $J^l = \{j \mid x_j^l = 1\}$ .

This problem is known to be NP-hard. Many heuristics as well as enumeration schemes (branch-and-bound) have been proposed for this problem. In many of these approaches the concept of *row prices* is used. The vector  $\bar{\rho}^l = (\rho_1^l, \rho_2^l, \dots, \rho_m^l)$  is a set of feasible row prices corresponding to partition  $J^l$  satisfying

$$\sum_{i=1}^m \rho_i^l a_{ij} = c_j \quad j \in J^l.$$

The price  $\rho_i^l$  may be interpreted as an estimate of the cost of covering job (flight leg)  $i$  using solution  $J^l$ . There are usually many feasible price vectors for any given partition.

The row prices are of crucial importance in computing the change in the value of the objective if a partition  $J^1$  is changed into partition  $J^2$ . If  $Z^1$  ( $Z^2$ ) denotes the value of the objective corresponding to partition 1 (2), then

$$Z^2 = Z^1 - \sum_{j \in J^2} \left( \sum_{i=1}^m \rho_i^1 a_{ij} - c_j \right).$$

The quantity

$$\sigma_j = \sum_{i=1}^m \rho_i^1 a_{ij} - c_j$$

can be interpreted as the *potential savings* with respect to the first partition to be obtained by including column  $j$ . It can be shown that if

$$\sum_{i=1}^m \rho_i^1 a_{ij} \leq c_j \quad j = 1, \dots, n,$$

for any set of feasible row prices  $\bar{\rho}^1$  corresponding to partition  $J^1$ , then solution  $J^1$  is optimal.

Based on the concept of row prices the following simple heuristic can be used for finding better solutions, given a partition  $J^1$  and a corresponding set of feasible row prices  $\bar{\rho}^1$ . The goal is to find a better partition  $J^2$ . In the heuristic the set  $N$  denotes the indices of the columns that are candidates for inclusion in  $J^2$ .

#### Algorithm 13.6.1 (Column Selection in Set Partitioning).

Step 1.

Set  $J^2 = \emptyset$  and  $N = \{1, 2, \dots, n\}$ .

Step 2.

Compute the potential savings

$$\sigma_j = \sum_{i=1}^m \rho_i^1 a_{ij} - c_j \quad j = 1, \dots, n.$$

Find the column  $k$  in  $N$  with the largest potential savings,

$$\sum_{i=1}^m \rho_i^1 a_{ik} - c_k.$$

Step 3.

For  $i = 1, \dots, m$ , if  $a_{ik} = 1$  set  $a_{ij} = 0$  for all  $j \neq k$ .

Step 4.

Let  $J^2 = J^2 \cup \{k\}$  and  $N = N - \{k\}$ .

Delete from  $N$  all  $j$  for which  $a_{ij} = 0$  for all  $i = 1, \dots, m$

Step 5.

If  $N = \emptyset$  STOP, otherwise go to Step 2.

The next example illustrates the heuristic.

**Example 13.6.2 (Crew Scheduling and Truck Routing).** Consider a central depot and 5 clients, see Figure 13.1. From the depot, a single delivery has to be made to each one of the clients. Assume that each truck can serve at most two clients on a single trip. The objective is to determine which truck should go to which client and the routing of the trucks that minimizes the total distance traveled. Each column in the table below represents one possible truck route and the  $c_j$  is equal to the total distance traveled for each trip. For example, column 6 represents a vehicle proceeding from the depot to client 1, then on to client 2, and from there back to the depot. The value of  $c_6$  is 14, which is the total distance traveled during the trip.

Route	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$c_j$	8	10	4	4	2	14	10	8	8	10	11	12	6	6	5
	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	0	1	0	0	0	1	0	0	0	1	1	1	0	0	0
	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0
	0	0	0	1	0	0	0	1	0	0	1	0	1	0	1
	0	0	0	0	1	0	0	0	1	0	0	1	0	1	1

Suppose we select  $J^1 = 1, 2, 3, 4, 5$  as our first partition. A set of feasible row prices is  $\bar{\rho}^1 = (8, 10, 4, 4, 2)$ . The corresponding potential savings are

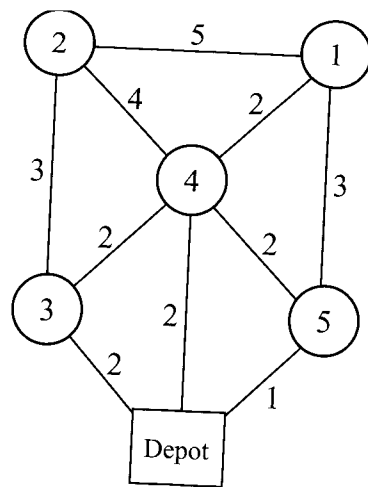


Fig. 13.1. Truck Routing Network

presented in the table below. Applying the heuristic and breaking ties by selecting the column with the lowest index yields the new partition  $J^2 = \{6, 13, 5\}$ . This new partition has a cost of  $Z^2 = 22$  as compared to a cost of  $Z_1 = 28$  for the first partition.

Route	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
$c_j$	8	10	4	4	2	14	10	8	8	10	11	12	6	6	5	
$\sigma_j$	0	0	0	0	0	4	2	4	2	4	3	0	2	0	1	$x_6 = 1$
			0	0	0	-6	-4	-6	-6	-7	-10	2	0	1		$x_{13} = 1$
					0			-6			-10	-4	-3			$x_5 = 1$

Many sets of row prices are feasible. The new row prices can be determined using the following procedure. It is clear that

$$\rho_1^2 + \rho_2^2 = 14.$$

Choose

$$\rho_1^2 = \frac{\rho_1^1}{\rho_1^1 + \rho_2^1} \times c_6 = \frac{8}{8 + 10} \times 14 = 6.222$$

and

$$\rho_2^2 = \frac{\rho_2^1}{\rho_1^1 + \rho_2^1} \times c_6 = \frac{10}{8 + 10} \times 14 = 7.777.$$

Using the row prices  $\bar{\rho}^2 = (6.2, 7.8, 3, 3, 2)$  the potential savings can be computed and the heuristic yields the new partition  $J^3 = \{8, 10, 5\}$  with a cost  $Z_3 = 20$ .

Route	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
$c_j$	8	10	4	4	2	14	10	8	8	10	11	12	6	6	5	
$\sigma_j$	-1.8	-2.2	-1	-1	0	0	-0.8	1.2	0.2	0.8	-0.2	-2.2	0	-1	0	$x_8 = 1$
		-2.2	-1	0	-6.2	-7	-6	0.8	-3.2	-2.2	-3	-1	-3			$x_{10} = 1$
				0				-6		-10	-4	-3				$x_5 = 1$

Using the row prices  $\bar{\rho}^3 = (5.3, 7.1, 2.9, 2.7, 2)$  we find that all potential savings are negative, so the partition  $J^3$  is optimal.

Route	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
$c_j$	8	10	4	4	2	14	10	8	8	10	11	12	6	6	5	
$\sigma_j$	-2.7	-2.9	-1.1	-1.3	0	-1.6	-1.8	0	-0.7	0	-1.2	-2.9	-0.4	-1.1	-0.3	

When the problems become very large, it is necessary to adopt more sophisticated approaches, namely branch-and-bound, branch-and-price, and branch-cut-and-price. The bounding techniques in branch-and-bound are often based on a technique called Lagrangean Relaxation. Branch-cut-and-price combines branching with so-called cutting planes techniques and has been used to solve real world problems arising in the airlines industry with considerable success.

## 13.7 Operator Scheduling in a Call Center

In every large organization that has a help line or a toll free number, there is a concern with regard to the quality of service. This is measured by the time it takes a caller to get an operator on the line, or, in the case of a conversant system that responds to digital input, by the time it takes the caller to get an appropriate person on the line.

Companies often have detailed statistics with regard to the frequencies of past requests for service. These statistics typically are broken down according to the time of day, the day of the week, and the type of service requested. Figure 13.2 depicts statistics with regard to calls for operators in a call center.

In this section we describe the design and development of an operator scheduling system for a call center. The objective is to determine a set of shifts and assignment of tasks within the shifts, so that the requirement curves are matched as closely as possible by the availability of operators. The shift configurations that are allowed have a certain structure. These structures are often referred to as *tour templates* and are characterized by their start times, finish times, and the number, length and timing of non-work periods. The time unit is 15 minutes (a coffee break is in this case one time unit and a lunch break is anywhere from two to four units). Different templates  $j$  and  $k$  can be of the same *type*  $i$ . A type is characterized by the time between start

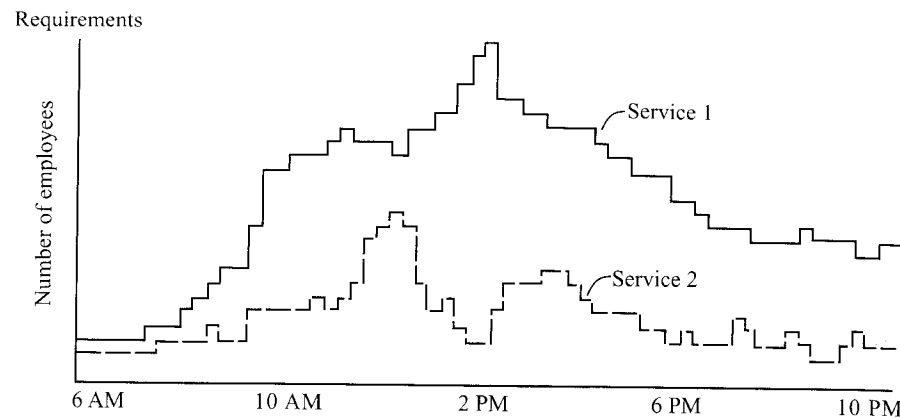


Fig. 13.2. Statistics with Regard to Calls for Operators at a Telephone Company

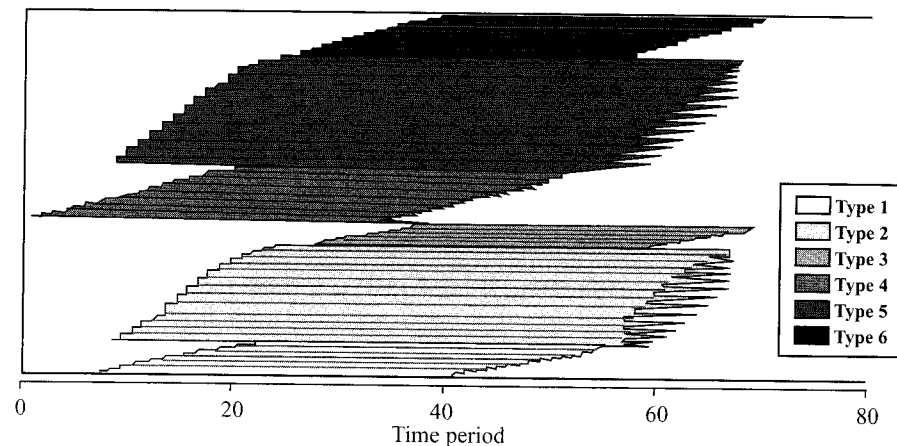


Fig. 13.3. Space of Tour Template Types

and finish, but not by the timing of the non-work periods. A sample of the tour template types is depicted in Figure 13.3.

The problem is simplified by assuming that none of the variables (e.g., the number of operators available, the call frequencies, and so on) change within any given time unit. So for each 15 minute period the demand as well as the number of operators are fixed. Because of this simplification, each term in any one of the performance measures is a sum rather than an integral.

To describe the objectives formally, we need the following notation. Let  $x_j$  denote the number of tours of template  $j$  and  $c_j$  the cost of a tour of template  $j$ . Let  $a_j$  and  $b_j$  denote the starting time and the finishing time of template  $j$ . Let  $m_i$  denote the number of tours of type  $i$ . Let  $y(k, t)$  be 1 if tour  $k$  is at

work at time  $t$  and 0 otherwise and  $s(t)$  the number of operators available at time  $t$ , i.e., the exact supply at time  $t$ . Let  $e(t)$  denote the difference between the supply and the demand,  $e^-(t)$  the negative part (the shortage), i.e.,

$$e^-(t) = \max(0, -e(t)),$$

and  $e^+(t)$  the positive part (the surplus), i.e.,

$$e^+(t) = \max(0, e(t)).$$

Based on this, a number of precise performance measures can be defined that take everything into account including coffee breaks. Let  $\mathcal{F}$  denote the fitness measure defined as

$$\mathcal{F} = \psi^- \sum_{t=1}^H e^-(t) + \psi^+ \sum_{t=1}^H e^+(t),$$

where  $\psi^-$  and  $\psi^+$  denote the respective penalty costs and  $H$  denotes the number of time units. Let  $\mathcal{C}$  denote the cost measure

$$\mathcal{C} = \mathcal{F} + \sum_j c_j x_j$$

and  $\mathcal{L}$  the smoothness measure

$$\mathcal{L} = \sum_{t=1}^H e(t)^2.$$

The overall framework of the approach adopted in the system is depicted in Figure 13.4. The optimization process is divided into a number of modules. The solid tour selection module is based on a mathematical program. In the solid tour selection the breaks in the tours are not taken into consideration. If there are no side constraints then this mathematical program is equivalent to a network flow problem.

The break placement module attempts to minimize  $\mathcal{L}$  subject to all break placement rules. This procedure operates in a sequential manner. It finds and places a break in a tour with a maximum decrease in  $\mathcal{L}$ . This is repeated for all tours and breaks and for all break hierarchies. The target demand modification module attempts to minimize the fitness measure.

A number of side constraints have to be satisfied. There are bounds on tour types and on the total number of tours. Moreover, there are constraints on the tightness of the fit during certain critical periods. With these side constraints, the optimization problem in the solid tour selection module is no longer a simple network flow problem. One way of dealing with this more complicated problem is the following. The original network flow formulation can be replaced by a conventional linear programming formulation and the

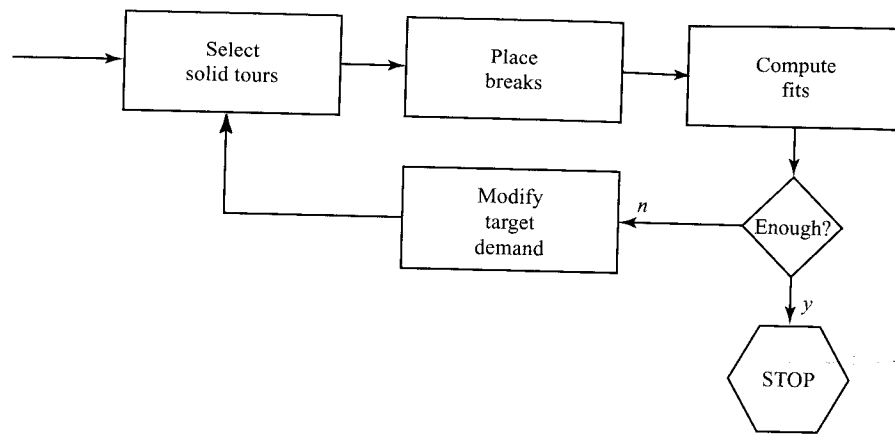


Fig. 13.4. Framework of the Approach in Operator Scheduling

side constraints can be incorporated as additional constraints. In most cases the linear programming formulation of the solid tour selection yields an integer solution. When it does not, a simple rounding heuristic can restore integrality.

To describe the mathematical programming formulation of the solid tour selection module in more detail additional notation is needed (since in this module the breaks are not taken into account). Let  $D(t)$  denote the target demand for solid tours during time  $t$ ; this target demand for solid tours has to be somewhat higher than the actual demand for operators since at any point in time not all of them may be working. Let  $S(t)$  denote the supply of tours during period  $t$  (not taking into account that an operator may be on a break). So

$$S(t) = \sum_{j: t \in [a_j, b_j]} x_j$$

and

$$S(t) > s(t).$$

Let

$$\begin{aligned} E(t) &= S(t) - D(t), \\ E^+(t) &= \max(E(t), 0), \\ E^-(t) &= \max(-E(t), 0). \end{aligned}$$

So  $E(t)$  denotes the amount of surplus at time  $t$  and  $E^+(t)$  and  $E^-(t)$  denote the positive and the negative part of this surplus. Let  $\Psi^+(t)$  and  $\Psi^-(t)$  denote the respective penalty costs at time  $t$ . The optimization problem can now be formulated as follows.

$$\min_{x, E^+, E^-} \sum_{t=1}^H \left( \Psi^-(t) E^-(t) + \Psi^+(t) E^+(t) \right) + \sum_{j=1}^J c_j x_j$$

subject to

$$\begin{aligned} E^+(t) - E^-(t) &= \sum_{j: t \in [a_j, b_j]} x_j - D(t), \quad t = 1, \dots, H \\ \sum_{j=1}^N x_j &\leq U \end{aligned}$$

In addition, most of the variables have upper and lower bounds, i.e.,

$$\begin{aligned} x_j^{\min} &\leq x_j \leq x_j^{\max} & j &= 1, \dots, N \\ 0 &\leq E^-(t) \leq E^-(t)^{\max} & t &= 1, \dots, H \\ 0 &\leq E^+(t) \leq E^+(t)^{\max} & t &= 1, \dots, H \\ m_i^{\min} &\leq m_i \leq m_i^{\max} & i &= 1, \dots \end{aligned}$$

The approach used here is much faster than implicit tour/break representation approaches, and can handle complex break placement rules.

The operator scheduling problem described here is a more complicated version of the problem described in Section 13.3. The real life version is even harder because of other issues that have to be taken into account. For example, a company may have operators who speak only English and others who speak both English and Spanish. So, some calls can be handled by either type of operator and others by only one type of operator. In the terminology of Chapter 2, the calls are the jobs and the operators are the machines; the  $M_j$  sets of the jobs are nested. The problems are further complicated by consideration of labor agreements and personnel policies. An example of such a policy is the FIFO rule, that is, if person  $A$  starts his shift earlier than person  $B$ , then  $A$ 's first break cannot start later than  $B$ 's first break.

## 13.8 Discussion

It is interesting to compare the models in this chapter with the workforce constrained scheduling and time-tabling models described in Section 9.4. The jobs have to be scheduled in such a way that a certain objective, e.g., the makespan, is minimized and at any point in time the demand for people remains within the limit. So, there is a flexibility in the scheduling of the jobs. In this chapter, the models are somewhat different. There is no flexibility in the requirements, since these are given. However, the size of the workforce and the number of people in each shift are the variables.

In practice, personnel scheduling problems tend to be intertwined with other factory scheduling problems. For example, when it is evident that committed shipping dates cannot be met, extra shifts have to be put in, or overtime has to be scheduled.

In the literature, these more aggregate problems (integrating machine scheduling and personnel scheduling) have not yet been considered. However, a number of scheduling systems, that are currently available on the market, offer machine scheduling features together with shift scheduling features.

## Exercises

**13.1.** Consider the model described in Section 13.2.

- Explain how algorithm 13.2.1 may result in an optimal schedule that is cyclic.
- Develop a method to compute the number of weeks in the cycle of an optimal cyclic schedule.
- Are all optimal schedules cyclic?

**13.2.** Consider Example 13.2.3. Note that the schedule has the disadvantage that there is a 1-day workstretch (e.g., employee 3 works in week 1 on Monday, while he is off on Sunday and Tuesday). Consider the following list of paired days:

Pair 1: Monday - Wednesday;  
 Pair 2: Tuesday - Thursday;  
 Pair 3: Tuesday - Wednesday.

Work out the new schedule and observe that the schedule has minimum 2-day and maximum 5-day workstretches. However, the surplus is less evenly distributed.

**13.3.** Consider the days-off scheduling model of Section 13.2 and the instance with the following daily requirements.

<i>day j</i>	1	2	3	4	5	6	7
	Sun.	Mon.	Tues.	Wed.	Thurs.	Fri.	Sat.
<i>Requirement</i>	3	5	7	5	5	7	3

Each person requires 3 out of the 5 weekends off. Apply Algorithm 13.2.1 to this instance.

**13.4.** Consider the days-off scheduling model of Section 13.2 and the instance with the following daily requirements.

<i>day j</i>	1	2	3	4	5	6	7
	Sun.	Mon.	Tues.	Wed.	Thurs.	Fri.	Sat.
<i>Requirement</i>	1	7	7	7	10	11	3

Each person requires at least 1 out of every 2 weekends off. Apply Algorithm 13.2.1 to this instance.

**13.5.** Consider a retail store that is open for business from 10 a.m. to 8 p.m. There are five shift patterns.

<i>pattern</i>	<i>Hours of Work</i>	<i>Total Hours</i>	<i>Cost</i>
1	10 a.m. to 6 p.m.	8	\$ 50.00
2	1 p.m. to 9 p.m.	8	\$ 60.00
3	12 p.m. to 6 p.m.	6	\$ 30.00
4	10 a.m. to 1 p.m.	3	\$ 15.00
5	6 p.m. to 8 p.m.	3	\$ 16.00

Staffing requirements at the store varies from hour to hour.

<i>Hour</i>	<i>Staffing Requirement</i>
10 a.m. to 12 a.m.	3
12 a.m. to 2 p.m.	6
2 p.m. to 4 p.m.	7
4 p.m. to 6 p.m.	7
6 p.m. to 8 p.m.	4

- Formulate this problem as an integer program.
- Solve the linear programming relaxation of this problem.
- Is the solution obtained under b) optimal for the original problem?

**13.6.** Consider the instance described in Exercise 13.4. All the assumptions are still in force with the exception of one. Assume now that each person must work every other weekend and must have every other weekend off. (In Exercise 13.4 it was possible for a person to have every weekend off.)

- Formulate this instance as an integer program.
- Solve the linear program relaxation of the integer program formulated. Round off the answer to the nearest integers.
- Compare the solution obtained under b) with the solution obtained in Exercise 13.4.

**13.7.** Consider the (5,7)-cyclic staffing problem with the **A** matrix as depicted in Section 13.4. The  $\bar{b}$  vector is (4,9,8,8,8,9,4). The first entry corresponds to a Sunday and the last entry corresponds to a Saturday. The cost vector  $(c_1, \dots, c_7)$  is (6,5,6,7,7,7,7), i.e., the least expensive shift is the one that has both Saturday and Sunday off.

Apply Algorithm 13.4.1. to find the optimal solution.

**13.8.** Consider Application (i) in Section 13.5.

- Compute the number of columns in the matrix.
- Compute the number of columns if one-day workstretches are not allowed (a one day workstretch is a working day that is preceded and followed by days-off).
- Compute the number of columns if one and two day workstretches are not allowed.

**13.9.** Consider application (ii) in Section 13.5. Assume that the  $\bar{c}$  vector is

(1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3,  
1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5,  
2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, 3.75, 4)

The requirements vector  $\bar{b}$  is

10, 10, 10, 10, 10, 10, 10, 10, 8, 8, 8, 8, 8, 8, 8, 5, 5, 5, 5, 5, 5

Apply Algorithm 12.4.1 to this instance. (You will need a linear programming code to do this, e.g., LINDO.)

**13.10.** Consider Example 13.6.2. In the second iteration the row prices (6.2, 7.8, 3, 3, 2) are used. However, this is not the only set of feasible row prices. Consider the set (7, 7, 3, 3, 2), which is also feasible. Perform the next iteration using this set of prices.

**13.11.** Consider a central depot and 5 clients. From the depot a single delivery has to be made to each one of the clients. The routes that are allowed are shown in the table below. The objective is to determine which truck should go to each client and the routing that minimizes the total distance traveled. Each column in the table represents a possible truck route and the  $c_j$  represents the total distance of the route.

Route	1	2	3	4	5	6	7	8	9	10	11	12	13
$c_j$	8	10	4	4	2	14	10	8	11	12	6	6	5
	1	0	0	0	0	1	1	1	0	0	0	0	0
	0	1	0	0	0	1	0	0	1	1	0	0	0
	0	0	1	0	0	0	1	0	0	0	1	1	0
	0	0	0	1	0	0	0	0	1	0	1	0	1
	0	0	0	0	1	0	0	1	0	1	0	1	1

Apply Algorithm 13.6.1 to this instance.

## Comments and References

The elementary textbook by Nanda and Browne (1992) covers some (but not all) of the models discussed in this chapter.

Section 12.2 is taken from Burns and Carter (1985) and is based on the seven days per week, one shift per day model. Burns and Koop (1987) extend this work and look at the seven days per week, multiple shifts per day model. Emmons (1985), Emmons and Burns (1991) and Hung and Emmons (1993) consider related models.

The general integer programming formulation considered in Section 13.3 appears in many handbooks and survey papers; see, for example, the survey papers by Tien and Kamiyama (1982) and Burgess and Busby (1992).

The material presented in Sections 13.4 and 13.5 is primarily based on the paper by Bartholdi, Orlin and Ratliff (1980).

The crew scheduling heuristic presented in Section 13.6 comes from the paper by Cullen, Jarvis and Ratliff (1981). Many papers have focused on crew scheduling problems; see, for example, Marsten and Shepardson (1981), Bodin, Golden, Assad and Ball (1983), and Stojkovic, Soumis, and Desrosiers (1998). A branch-and-cut method applied to crew scheduling is described in Hoffman and Padberg (1993). The airline crew recovery problem is described in Lettovsky, Johnson, and Nemhauser (2000).

A description of the operator scheduling system designed for a long-distance telephone company was presented at a national meeting of the INFORMS society in Washington, D.C., see Gawande (1996).