CHAPTER 3
VEHICLE AND CREW SCHEDULING PROBLEMS

3.1. INTRODUCTION

Vehicle and crew scheduling problems can be thought of as routing problems with additional constraints having to do with the times when various activities may be carried out. The routing problems reviewed in Chapter 2 give special importance to the spatial characteristics of the activities performed. In particular, the successive moves of a vehicle, generally referred to displacements in space. In scheduling problems, however, a time is associated with each activity. For example, each location may require a delivery at a specified time. Thus, the temporal aspects of vehicle movements now have to be considered explicitly. As a result, the activities are followed in both space *and* time. The feasibility of an activity is also influenced by both space and time characteristics, e.g. a single vehicle could not service two locations with identical delivery times. The sequencing of vehicle activities in both space and time is at the heart of the vehicle scheduling problem.

In crew scheduling, the primary concern is to sequence the movements of a crew in space and time so as to staff the desired vehicle movements. While crew scheduling problems are essentially similar to vehicle scheduling problems, the former generally involve more complicated restrictions, such as requirements for crew lunch breaks, union regulations, etc.

For example, consider the following bus operator schedule:

7:00 AM: report to work at garage
7:03 AM: walk to relief point *A*
7:11 AM: board bus No. 1 at relief point *A* and relieve current operator, then operate bus No. 1 (includes both in-service operations and deadheading)
11:20 AM: get off bus No. 1 at relief point *B* and take lunch break
11:52 AM: travel via transit system from relief point *B* to relief point *C*
12:20 PM: board bus No. 2 at relief point *C* and relieve current operator, then operate bus No. 2
3:30 PM: get off bus No. 2, having driven it to garage, and go off duty.

This example illustrates some of the complexities involved in crew schedules. In particular, crew schedules almost always require rest periods such as the lunch break illustrated. Note that two types of deadheading take place: (1) with the operator driving a vehicle, and (2) with the operator walking or being a passenger on the transit system. Also note that an operator may be associated with more than one vehicle during the day.

In general, vehicle and crew scheduling problems interact with one another: the specification of vehicle schedules will set certain constraints on the crew schedules and *vice versa.* Ideally, therefore, one would want to solve the two problems simultaneously. However, models incorporating both problems into a single optimization problem are generally quite complex.

Consequently, most practitioners have opted for sequential procedures that solve one problem first, and then the other, with some mechanism for taking the interaction between the two into account.

Before we describe our formal representation of scheduling problems let us examine more closely the nature of the temporal constraints. Suppose we wish to deliver a package to Mr. Smith and we know that once we arrive at Mr. Smith's house the delivery will take exactly 10 minutes. If Mr. Smith put no further restrictions on delivery time, we could represent temporal considerations solely as a weight (10 minutes) associated with the delivery task together with a constraint that stated no route could last more than say 9 hours. We would use a routing algorithm to solve this problem, which would have the freedom to assign Mr. Smith's delivery to any time of the day. On the other hand, suppose Mr. Smith specified that the delivery must

take place at exactly 2 : 10 PM. In this case, a start time 2 : 10 PM and an end time 2 : 20 PM would be associated with the delivery task and we would use a scheduling algorithm to solve the problem. Our scheduling algorithm would have to assign Mr. Smith's delivery at 2 : 10 PM.

In general, the input to vehicle and crew scheduling problems is a set of tasks. Each task has a specified start time, end time, start location and end location. The cost function consists of components that might include vehicle operating costs, vehicle capital costs and crew operating costs. The fleet of vehicles and the set of crews may be limited and may be housed at one or more depots. The type of scheduling problem that evolves is a function of the constraints imposed upon the formation of schedules, the types of tasks being serviced and the locations where these tasks must be carried out.

In Fig. 3.1, a set of vehicle schedules servicing 10 tasks is presented. We assume in this example that each task has the same start and end location (the depot). The start and end times

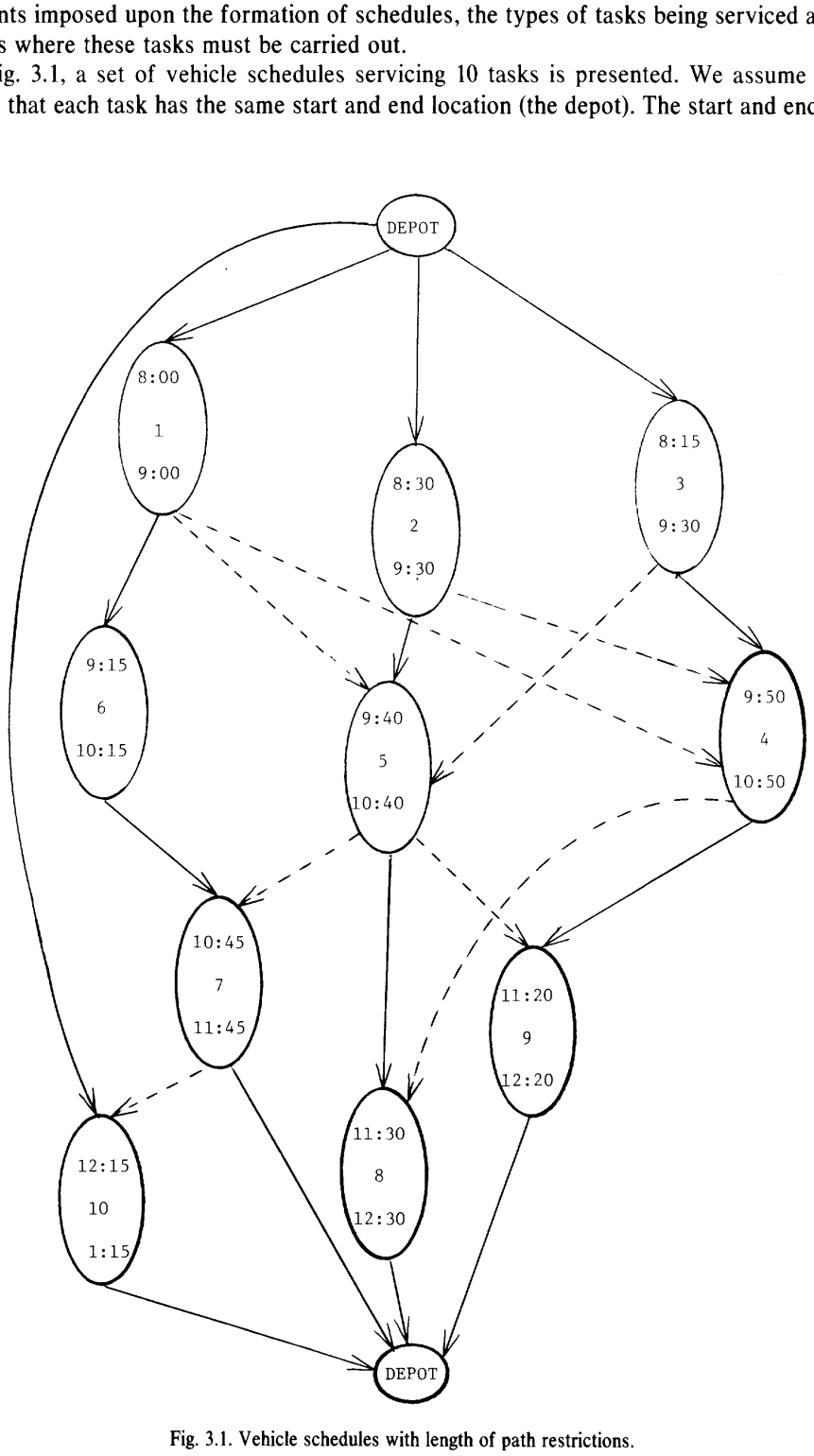


Fig. 3.1. Vehicle schedules with length of path restrictions.

of each task are given within the node representing the task. A solid branch between two nodes indicates that these two nodes are on the same vehicle schedule and that the vehicle schedule will follow the orientation of the branch. The dotted branches indicate feasible connections which are not used in the solution. In this example, a branch is drawn from node i to j if the start time of task j is greater than the end time of task i and if the start time of task j is less than or equal to the end time of task i plus one hour. Thus, in the example, there is no branch from node 6 to node 5 since the start time of node 5 is less than the end time of node 6 and no branch from node 6 to node 8 since the start time of node 8 is greater than the end time of node 6 plus 1 hour. Each vehicle schedule is assumed to be no longer than 5 hours. Note that this restriction prohibits the inclusion of node 10 on the left-most path.

We will consider the following problems in this chapter:

*the single depot vehicle scheduling problem
*the single depot vehicle scheduling problem with length of path restrictions
*the single depot vehicle scheduling problem with multiple vehicle types
*the multiple depot vehicle scheduling problem
*the fixed location worker scheduling problem
*the mass transit crew/vehicle scheduling problem
*the air crew scheduling problem
*the rostering and bid line problems.

In all formulations and algorithms we describe the primary input will be a set of tasks with each task characterized by start and end times and start and end locations.

Sections 3.2 and 3.3 give an overview of the problems discussed in this chapter. The remaining sections discuss algorithms for solving vehicle and crew scheduling problems.

3.2. VEHICLE SCHEDULING

Three real-world constraints commonly determine the complexity of the vehicle scheduling problem. These restrictions are: (a) a constraint on the length of time a vehicle may be in-service before it must return to the depot for servicing or refueling; (b) the restriction that certain tasks can only be serviced by certain vehicle types; and (c) the presence of a variety of depots where vehicles may be housed. Our description of vehicle scheduling problems is classified according to these restrictions: Section 3.2.1 describes the unconstrained case and Sections 3.2.2 through 3.3.4 handle cases (a) through (c) respectively.

3.2.1 *The single depot vehicle scheduling problem (VSP)*

The *single depot vehicle scheduling problem* (VSP) requires the partitioning of the nodes (tasks) in an acyclic network into a set of paths in such a way that a certain cost function is minimized. Each path corresponds to the schedule for a single vehicle. An objective function that minimizes the number of paths effectively minimizes capital costs since the number of required vehicles equals the number of paths. If a weight equal to the corresponding deadhead time is associated with each arc, an objective function that minimizes total arc weight effectively minimizes operating costs since these are proportional to total vehicle travel time. If capital and operating costs can be quantified, then a combined objective can be used to minimize total system costs. Figure 3.1 depicts a set of tasks and a solution to the vehicle scheduling problem with path length constraints; Fig. 3.2 gives a solution to the path minimization VSP without the length of path restrictions.

3.2.2 *Vehicle scheduling problem with length of path restrictions (VSPLPR)*

In the *vehicle scheduling problem with length of path restrictions* (VSPLPR), constraints are placed on the length of time a vehicle may spend away from the depot or the mileage a vehicle may cover without returning to the depot for service. This constraint commonly encountered in practice corresponds to fuel restrictions, maintenance considerations, etc. Whereas the VSP can be solved to optimality using a polynomially-bounded algorithm, the VSPLPR is *NP*-hard[33]. To our knowledge, no optimal vehicle scheduling algorithms have been designed explicitly for this problem. Heuristic algorithms that treat length of path constraints as well as other constraints are described in Section 3.4. In addition, this problem is also solved as a subproblem in an algorithm for generating crew schedules. This algorithm is discussed in

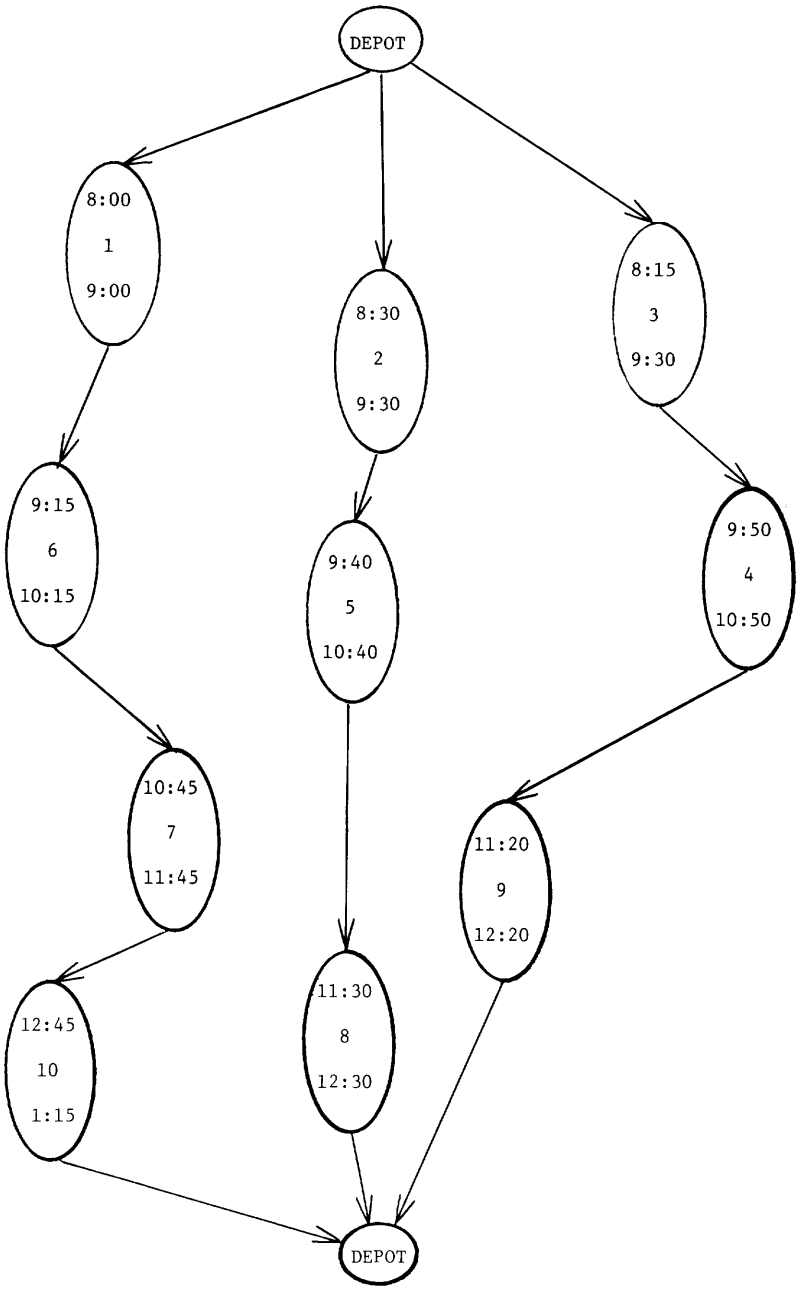


Fig. 3.2. Vehicle schedules with no length of path restrictions.

Section 3.5. A solution to this problem is illustrated in Fig. 3.1 where the length of path restriction is 5 hours.

3.2.3 *Vehicle scheduling problem with multiple vehicle types (VSPMVT)*

The *vehicle scheduling problem with multiple vehicle types* (VSPMVT) allows the possibility that vehicles with different characteristics are available to service the tasks. In most cases, the characteristic is vehicle capacity. For example, in transit systems, mini-buses can service the lines with low demand, regular buses can service the lines with high demand and either vehicle can service lines with medium demand. For each task the set of vehicles that may service it is specified. Consider the example in Fig. 3.3 where nodes 2, 6, and 7 can be serviced by type 1 vehicles, nodes 1, 3, 4, and 5 by type 2 vehicles, and nodes 8, 9, and 10, by either vehicle. There is no length of path restriction. A minimum path solution uses 4 vehicles rather than the 3 vehicles given in the single vehicle case (Fig. 3.2).

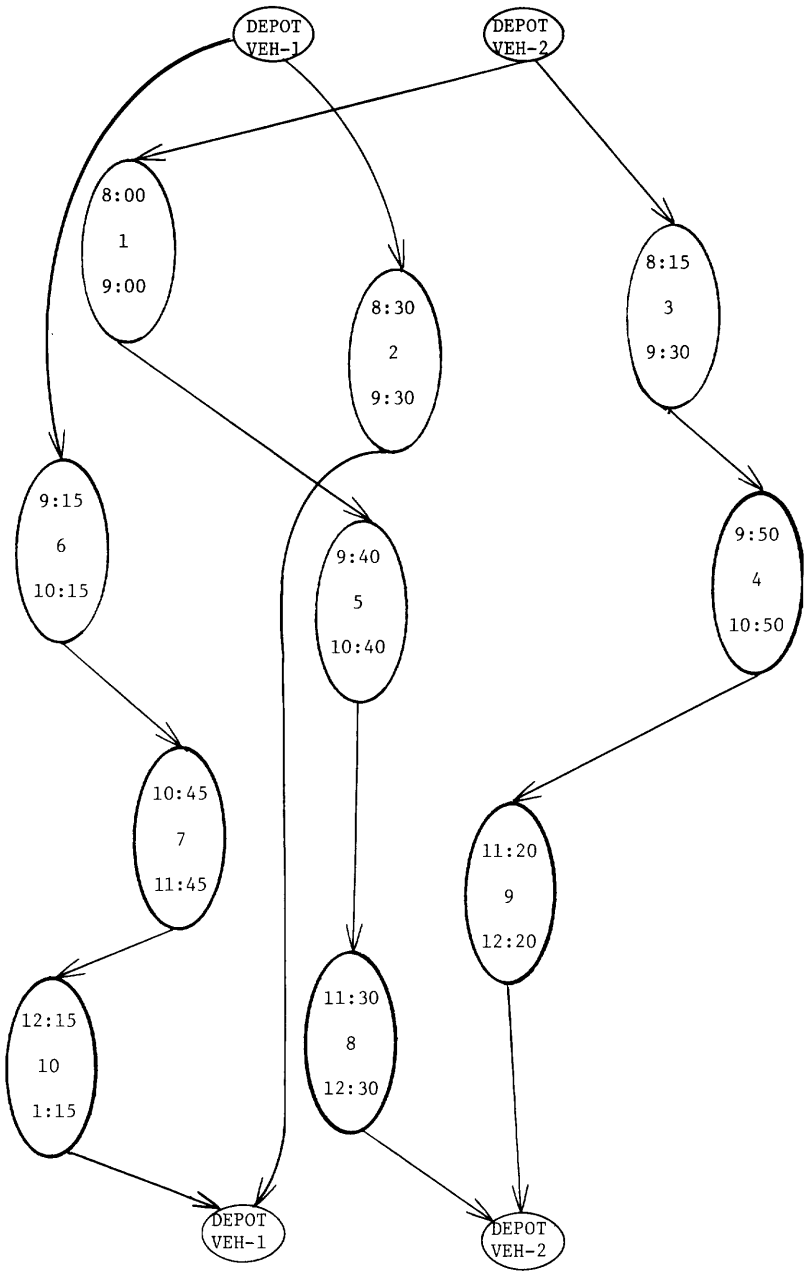


Fig. 3.3. Vehicle schedules with 2 vehicle types.

3.2.4 *Vehicle scheduling problem with multiple depots (VSPMD)*

In the *vehicle scheduling problem with multiple depots* (VSPMD), tasks may be serviced out of more than one depot. As with the vehicle routing problem, each vehicle must leave and return to the same depot and the fleet size at each depot must range between a specified minimum and maximum. An illustration of a feasible solution to the VSPMD is given in Fig. 3.4. In the solution given, nodes 1, 5, 6, 7, 8 and 10 are serviced by vehicles out of depot 1, nodes 2, 3, 4 and 9 are serviced by vehicles out of depot 2. This solution might be optimal in a situation where the objective function included operating costs and the start locations of tasks 1 and 6 and the end locations of tasks 8 and 10 were close to depot 1 and the start locations of tasks 2 and 3 and the end locations of tasks 3 and 9 were close to depot 2.

3.3. CREW SCHEDULING

Before describing crew scheduling problems it is worthwhile to examine more closely the relationship between crew and vehicle scheduling. Figure 3.5 illustrates a set of vehicle

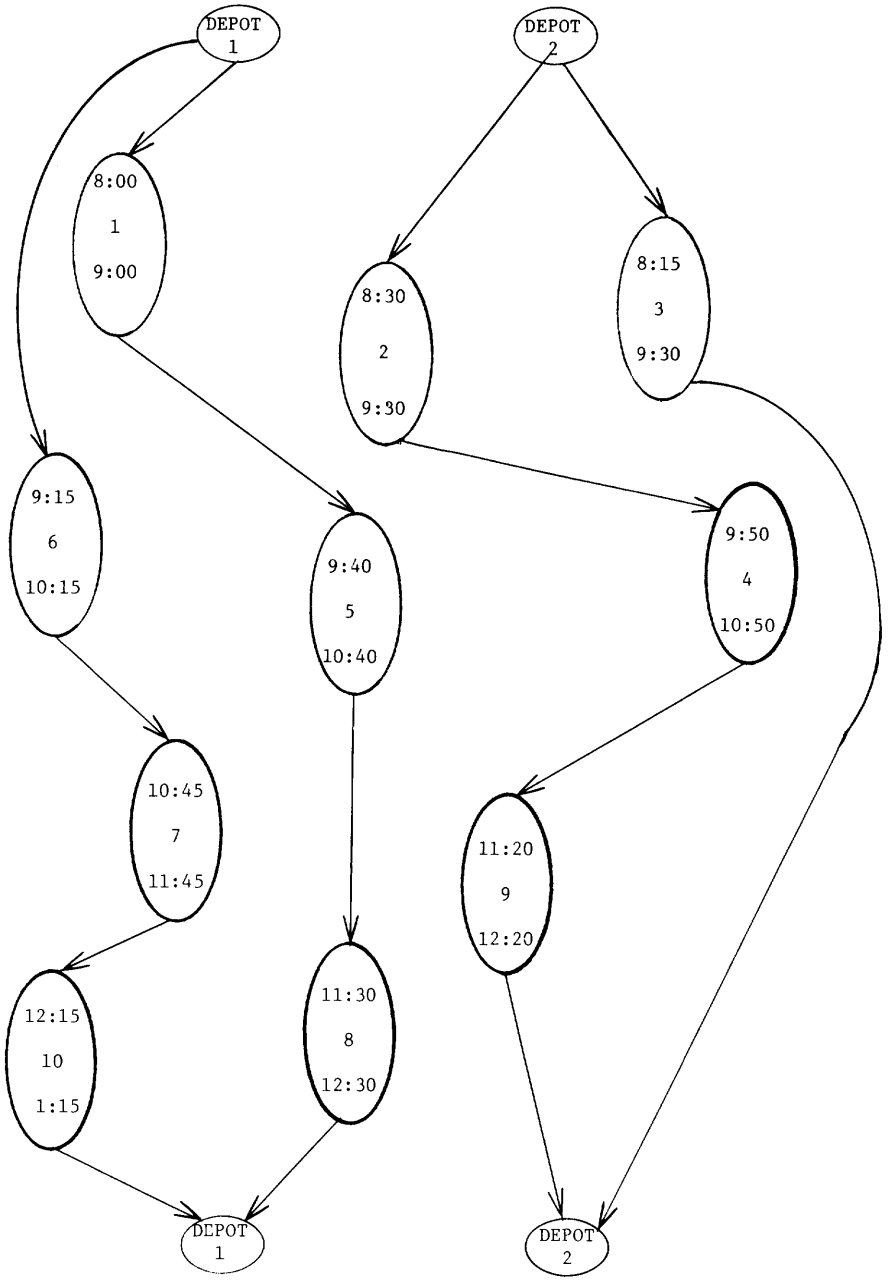


Fig. 3.4. Vehicle schedules with 2 depots.

schedules. These schedules describe the vehicle movements required for a certain transportation system. Each individual schedule has a set of points where one crew can relieve another. In the mass transit setting, a relief point is a designated stop along a transit line and in the airline case it is the end of any flight leg. To obtain crew schedules each vehicle schedule is split into pieces at one or more of the relief points. As Fig. 3.5 shows, an individual crew schedule is then obtained by grouping one or more of these pieces together. Of course, the feasibility of joining one piece with another depends not only on the end time of the first piece relative to the start time of the second, but also on the end location of the first piece relative to the start location of the second. The cost function will almost never be linear since the total cost of the crew schedule would rarely equal the sum of the costs of its component pieces.

Although we have described crew schedules in terms of vehicles, this by no means suggests that vehicles should be scheduled prior to and separately from crews. Unfortunately, in nearly all practical settings this is the approach taken. In the case of aircraft and air crew scheduling,

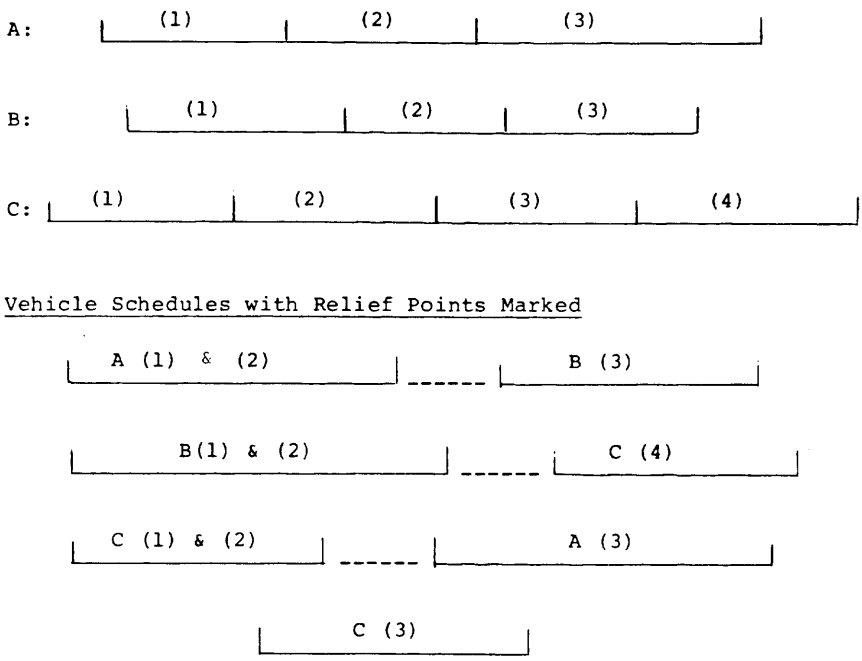


Fig. 3.5. Vehicle and crew schedules.

this approach is justifiable since aircraft costs significantly dominate air crew costs. In the mass transit case, however, crew costs dominate vehicle operating costs and in some cases reach as high as 80% of total operating costs. Under such circumstances, it seems foolish to allow vehicles to be scheduled independently of, and without regard to crews. For this reason, in this section rather than describing a "pure" mass transit crew scheduling problem, we describe a combined crew/vehicle scheduling problem.

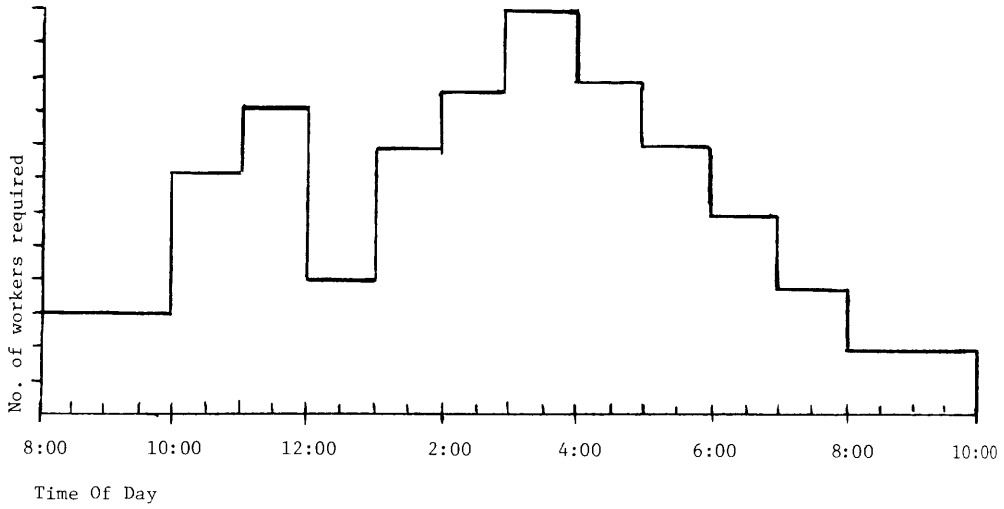
Although the primary thrust of this chapter is toward scheduling crews to operate vehicles, we start with a simpler case, that of scheduling workers at a fixed location, e.g. sales clerks or telephone operators. Considering this case will help in the conceptualization of more complicated situations where crew and vehicle schedules interact.

3.3.1 *Scheduling workers at a fixed location*

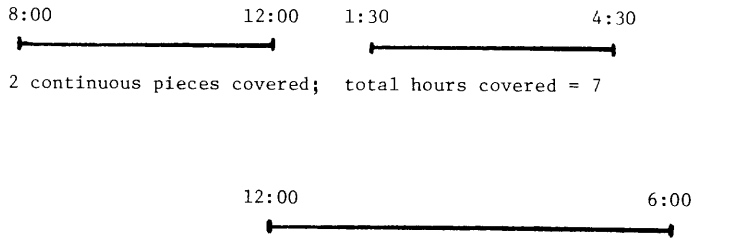
For this problem we do not start out with a set of tasks as described previously, but rather divide the workday into T time intervals and specify a demand for workers, d_t associated with each time interval $t = 1, 2, \ldots, T$. (In Section 3.4.2 we show how these specifications can be converted into a set of tasks). The worker scheduling problem is to find a set of worker schedules that cover all the required work. It is assumed that workers are interchangeable and that any worker can be relieved at the end of any time period and that any worker can start at the beginning of any time period. Figure 3.6 illustrates the demand for workers via a histogram and gives sample workdays for an individual. A cost is associated with each possible workday and the problem is to cover the histogram with a minimum cost set of workdays. In the case where workdays do not contain breaks this problem has a network flow structure. The problem just described applies to several real world situations including telephone operator scheduling[589]. It has also been used to estimate crew requirements for urban mass transit systems[76, 89]. In this case, the spatial considerations of transit crew scheduling are deemphasized in order to get a clear picture of the temporal considerations.

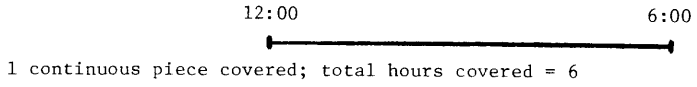
3.3.2 *Mass transit crew and vehicle scheduling*

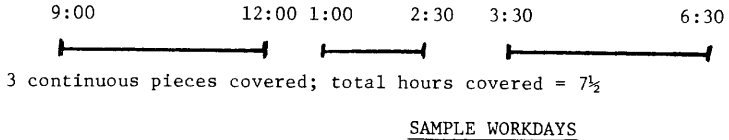
Passenger service for a mass transit system is defined by means of a set of *lines* and *line schedules* as illustrated in Fig. 3.7(A). Each traversal of a line must be performed by a single



WORKER DEMAND HISTOGRAM







SAMPLE WORKDAYS

Fig. 3.6. Fixed location worker scheduling.

vehicle so that continuous passenger service can be provided. We refer to each one-way traversal of a line as a *trip*. As Fig. 3.7(B) illustrates, we may characterize trips by their start time and start location and end time and end location. If vehicles are to be scheduled independently of crews, the appropriate problem description from Section 3.2 can be applied with the set of trips serving as the set of input tasks.

To define a combined crew/vehicle scheduling problem we must consider the nature of crew movements. Each line has one or more *relief points* which are stops along the line where one crew may relieve another. Relief points do not necessarily coincide with the start or end of a trip (see Fig. 3.7(A)). Thus, a crew's period of work on a single vehicle starts and ends at either a relief point or at the garage. We create the set of *d-trips* (driver trips) by partitioning each trip at its relief points. These are illustrated in Fig. 3.7(C). The set of *d*-trips makes up the set of input tasks for the mass transit crew/vehicle scheduling problem. Whereas a trip represents the minimal amount of work that must be performed by a single vehicle, a *d*-trip represents the minimal amount of work that must be performed by a single crew.

FIGURE 3.7 A: LINE SCHEDULE

College Park (I)	Mount Ranier*(II)	North Capitol & New York Ave.	Potomac Park (III)
9:00	9:20	9:55	10:15
9:15	9:35	10:10	10:30
⋮	⋮	⋮	⋮
10:20	10:00	9:25	9:05
10:35	10:15	9:40	9:20
⋮	⋮	⋮	⋮

* indicates relief point

FIGURE 3.7B: TRIPS (indicated by [(start time, start location),(end time, end location)])

[(9:00,I),(10:15,III)], [(9:15,I),(10:30,III)], [(9:05,III),(10:20,I)],

[(9:20,III),(10:35,I)].

FIGURE 3.7C: D-TRIPS

[(9:00,I),(9:20,II)], [(9:20,II),(10:15,III)], [(9:15,I),(9:35,II)],

[(9:35,II),(10:30,III)], [(9:05,III),(10:00,II)], [(10:00,II),(10:20,I)],

[(9:20,III),(10:15,II)], [(10:15,II),(10:35,I)]

Fig. 3.7. Input to mass transit crew/vehicle scheduling problem.

A crew operates a single vehicle during a continuous portion of work called a *piece.* A sequencing of pieces into a single crew schedule is called a *run.* As was illustrated at the beginning of the chapter, if the end location of one piece and the start location of another do not coincide, then the crew must walk or use the transit system to go from one location to another. In general, crews are paid for this dead time. Figure 3.8 illustrates a solution to the mass transit crew/vehicle scheduling problem. It consists of: (a) a set of *blocks*, which specify the work plan for vehicles over the day and which are made up of a set of *d*-trips; (b) the decomposition of blocks into pieces: and (c) the sequencing of pieces into runs.

Figure 3.9 illustrates a typical service profile for an urban mass transit system. Part of the challenge of mass transit crew scheduling results from the fact that this profile is rather ill-suited for decomposition into desirable crew runs. In particular, it is clear that, compared to mid-day, there will be a large number of pieces required during the AM peak and during the PM peak. It is natural to try to join these pieces into *split runs*, i.e. a run with a very long break between pieces. Transit crews generally prefer *straight runs* which consist of a single piece or contain a short break between pieces. The typical union contract places cost penalties on split runs or limits the percentage of straight runs used. To complicate matters further, the nature of the union regulations and the service profile can vary significantly from one municipality to another.

3.3.3 *Air crew scheduling*

The vehicle and crew scheduling problems which perhaps have received the most attention in recent years are the air crew and aircraft scheduling problems. As with mass transit crew and vehicle scheduling, a fixed timetable is assumed for the air crew scheduling problem. Aircraft, on the other hand, are almost always scheduled at the same time timetables are formed. The combined aircraft scheduling/timetable formation problem is a combined routing and scheduling problem which is discussed in Chapter 4.

The crew scheduling problem is broken down into two parts: generating pairings and

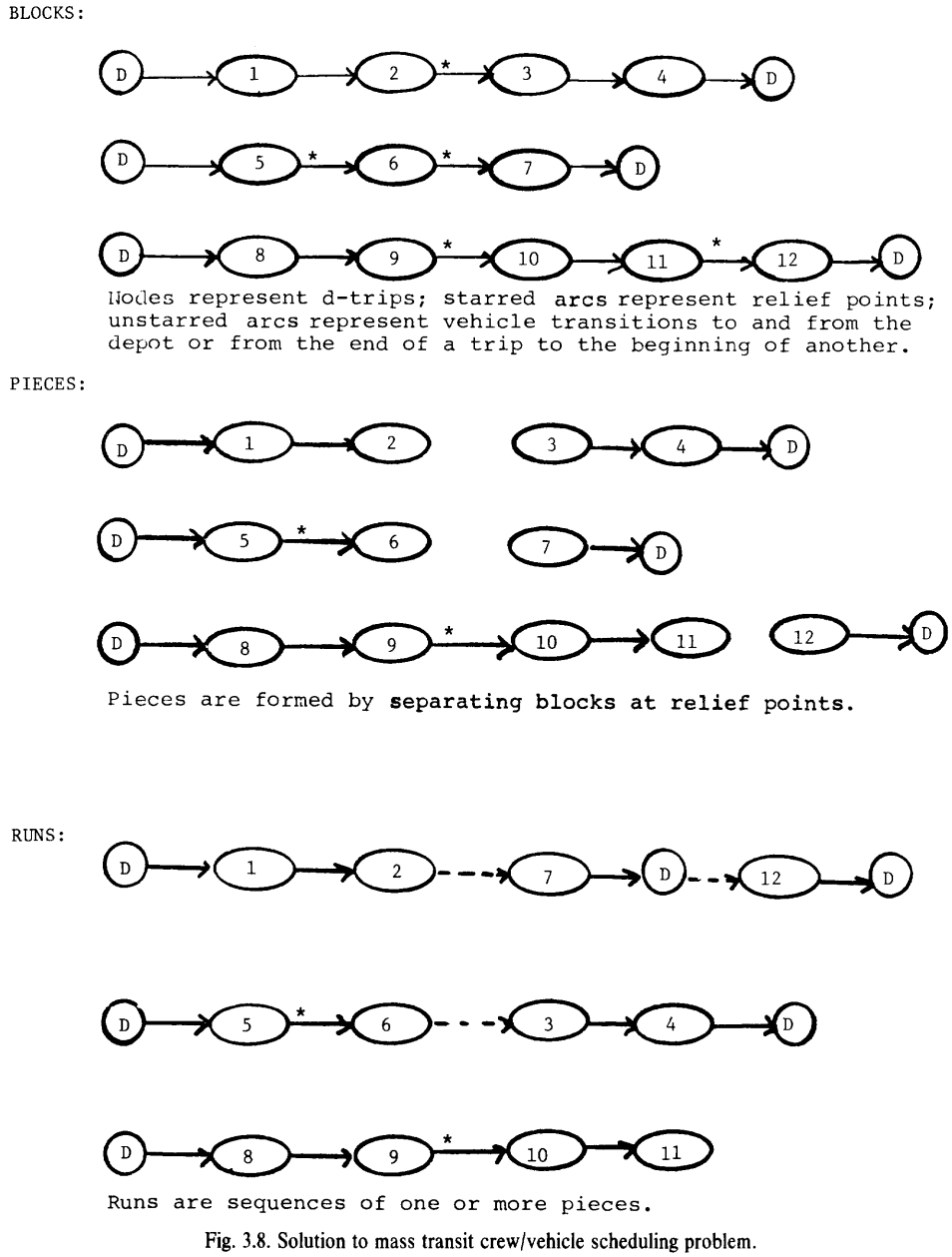
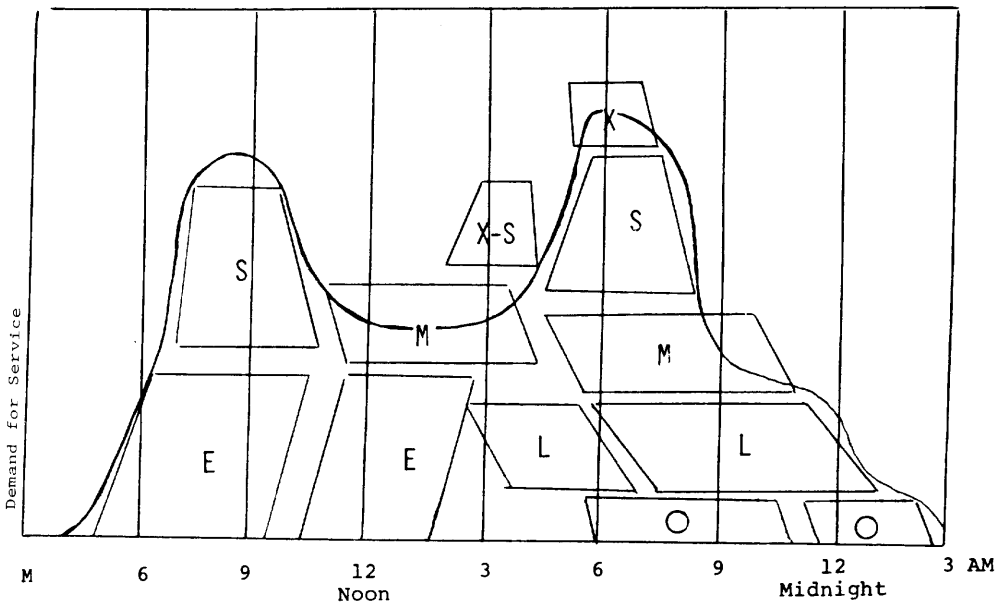


Fig. 3.8. Solution to mass transit crew/vehicle scheduling problem.

constructing bid lines. A *pairing* is a sequence of required service entities that a crew must complete which begins and ends at the same domicile. The construction of minimum cost pairings that cover all tasks arising from the timetable and follow the airplanes, has generally been referred to as the air crew scheduling problem.

Once the pairings have been constructed, bid lines are formed. The *bid lines* are sets of pairings that represent monthly work schedules for the crews. An overview of the pairings problem is now presented. The formation of minimum cost bid lines is described in Section 3.3.4.

The set of input tasks for the air crew scheduling problem consists of a set of flight legs. A *flight leg* corresponds to a flight between two cities, departing one city at a specified time and arriving at a second city at a specified time. A single crew pairing consists of a set of *duty periods*, which are continuous blocks of time when a crew is on duty. A duty period is in turn made up of a set of flight legs. Federal aviation regulations, union rules and company policies together impose a complex set of constraints in the formation of pairings. In particular, duty periods have a maximum length, 12 hours and a minimum rest time is required between duty



Run Types:

E - Early Straight Runs
L - Late Straight Runs
M - Middle Straight Runs
O - Owls (late late straight Runs)
S - Split Runs
X - Extra pieces (for use as Trippers, i.e., short one-piece runs, or the 3rd piece of work on a straight run)

Fig. 3.9. Service profile and decomposition into run types.

periods, 9.5 hours. Further, there are constraints on the manner in which flight legs can be combined into duty periods, e.g. a duty period can contain no more than 7 flight legs. Restrictions may also apply to the manner in which duty periods can be combined into pairings, e.g. no crew can fly more than 16 hours in any 48 hour interval. From a conceptual standpoint the air crew scheduling problem and the mass transit crew scheduling problem are similar. Roughly, flight legs correspond to *d*-trips, duty periods correspond to pieces and pairings correspond to runs. However, the time frames involved, the constraints, and the cost functions all differ significantly. These differences lead to differences in the solution algorithms as well.

Figure 3.10 illustrates a solution to an air crew scheduling problem. It consists of (a) a set of duty periods, which are in turn made up of flight legs and (b) a set of pairings, which consist of groups of duty periods. As we mentioned earlier, a final problem must be solved to group duty periods into bid lines. This problem is described in the next section.

3.3.4 *Rostering and bid line problems*

The crew schedules output by the procedures described in Sections 3.3.2. and 3.3.3 cover a time period of restricted length. For urban mass transit systems this time period is typically one day; for airline systems this period covers a few days. In certain instances, this is a "final" solution. For example, in most North American transit systems, a single driver follows the same crew schedule each day. This system is generally preferred by both the drivers and the transit companies since it alllows drivers to become familiar with particular routes and riders. However, each crew schedule has a different cost and desirability and drivers bid on schedules

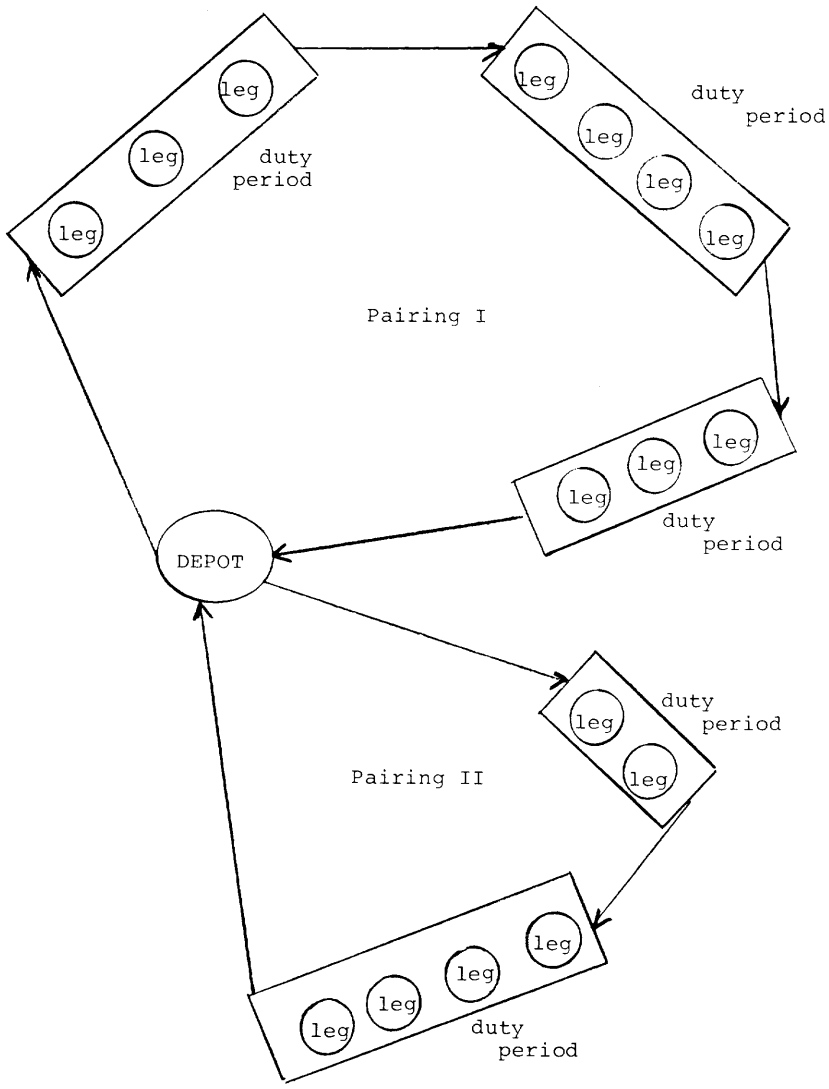


Fig. 3.10. Solution to air crew scheduling problem.

using a priority system based on seniority. Operating in this manner builds some inequities into the system. In most European transit systems each driver within a particular category receives the same pay and there is a need to insure that all schedules are of "equal difficulty". This is accomplished by changing the daily schedule that a particular driver follows every day so that all schedules are of about the same difficulty. This combining of daily schedules into weekly or monthly schedules is called the *crew rostering problem.*

The airline industry also must solve a very similar problem since the crew schedules output by the procedures outlined in Section 3.3.3 are for limited time periods. Just as North American transit crews bid on daily schedules, airline crews bid on monthly schedules. In the airline industry, the problem of combining pairings into monthly work schedules is called the *bid line problem.*

The constraints on these schedules are usually quite simple. The total number of hours worked on each bid line must be between some minimum and maximum. In some applications, this is a "hard constraint", while in other applications, there is a penalty attached to having a bid line fall below the minimum number of hours or exceed the maximum number of hours. For example in the airline industry, a crew is guaranteed a minimum number of hours per month even if the crew's bid line contains fewer than the minimum number of hours. There are also constraints on the minimum and maximum amount of time that may exist between two pairings (or crew schedules) on the same bid line (or roster).

3.4 VEHICLE SCHEDULING ALGORITHMS

All vehicle scheduling problems we discuss can be formulated as optimization problems on appropriately defined networks. In the case of the VSP, this formulation leads to an efficient solution using a minimum cost flow algorithm. In the other cases, complicating constraints make the problems much more difficult. Section 3.4.1 describes the network formulation of the VSP and shows how this problem may be efficiently solved using a network flow algorithm. Section 3.4.2. gives formulations for the constrained vehicle scheduling problems and presents approximate algorithms for solving these problems.

3.4.1 *Solving the VSP*

Recall that as input to the vehicle scheduling problem we start with a set of tasks with each task i characterized by a start location, SL_i, a start time, ST_i, an end location, EL_i, and an end time, ET_i. For any pair of locations L1 and L2 we denote by TM(L1, L2) the time to travel from L1 to L2. DL denotes the location of the depot. We construct a network similar to the one illustrated in Fig. 3.1. The node set N consists of a node representing each task together with a super source node s and a super sink node t. The arc set A is obtained by inserting an arc from task node i to task node j if it is feasible for a single vehicle to service both tasks, i.e. if $ST_j - ET_i \geq TM(EL_i, SL_j)$. Further, an arc is inserted from s to each task node and from each task node to t. These arcs represent trips to and from the depot. Each (s, t)-path through this network represents a possible schedule for a single vehicle. The solution to the vehicle scheduling problem is a set of (s, t)-paths that cover all task nodes. If we associate a cost c_{ij} with each arc $(i, j) \in A$, we formulate the partitioning problem as a minimum cost flow problem as follows:

$$\text{V1:} \quad \text{Min} \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{s.t.} \sum_{i:(i,j)\in A} x_{ij} - \sum_{i:(j,i)\in A} x_{ji} = 0 \text{ for all } j \in N - \{s, t\} \tag{3.1}$$

$$\sum_{i:(i,j)\in A} x_{ij} = 1 \text{ for all } j \in N - \{s, t\} \tag{3.2}$$

$$0 \leq x_{ij} \leq 1 \text{ and integer for all } (i, j) \in A \tag{3.3}$$

We interpret

$$x_{ij} = \begin{cases} 1 \text{ if some vehicle traverses arc } (i, j), \\ 0 \text{ otherwise.} \end{cases}$$

Constraint (3.1) is the standard conservation of flow constraint; constraint (3.2) insures that exactly one path (vehicle schedule) will pass through each task node. We may represent vehicle costs fairly accurately within this formulation. We assume that operating costs are a linear function of vehicle milage and travel time and that a fixed capital cost may be associated with each vehicle used. We then define

$$c_{ij} = \begin{cases} \text{the operating cost associated with deadheading from } EL_i \text{ to } SL_j \text{ for } i, j \in N - \{s, t\}, \\ \text{the operating cost associated with deadheading from } EL_i \text{ to DL for } j = t, \text{ the vehicle} \\ \text{capital cost plus the operating cost associated with deadheading from DL to } SL_i \text{ for} \\ i = s. \end{cases}$$

This cost function definition is based on the assumption that the number of paths (arcs in the solution out of node s) equals the number of required vehicles. Depending on how arcs and the problem's time horizon are defined the number of vehicles may or may not equal the number of paths. For example one path could represent a vehicle leaving the depot at 7 AM and returning at 10 AM and another path could represent a 3 PM to 6 PM vehicle schedule. It is clear these

could be covered by the same vehicle. Note from the definition of the arc set A given previously that there would be a "long arc" from the end task on the 7–10 AM schedule to the start task on the 3–6 PM schedule. Thus, a single path could cover these two work intervals (see Fig. 3.11).

However, in practice the vehicle might return to the depot between these work intervals and thus the cost of this long arc should be adjusted to reflect deadheading to and from the depot. The number of such long arcs can be quite large, i.e. proportional to the number of nodes squared, and consequently many times they are not included in the network. Rather, as was suggested in Section 3.1, arcs whose time duration is greater than some upper bound are not included in the network so that each path explicitly represents a vehicle trip away from the depot. In such situations many times, capital costs can be accurately included by taking into account temporal problem structure. For example, in mass transit systems, where demand for service has a dual peak structure (see Fig. 3.9), it is assumed that the number of vehicles required will equal the number required in the largest peak. If this were the AM peak then

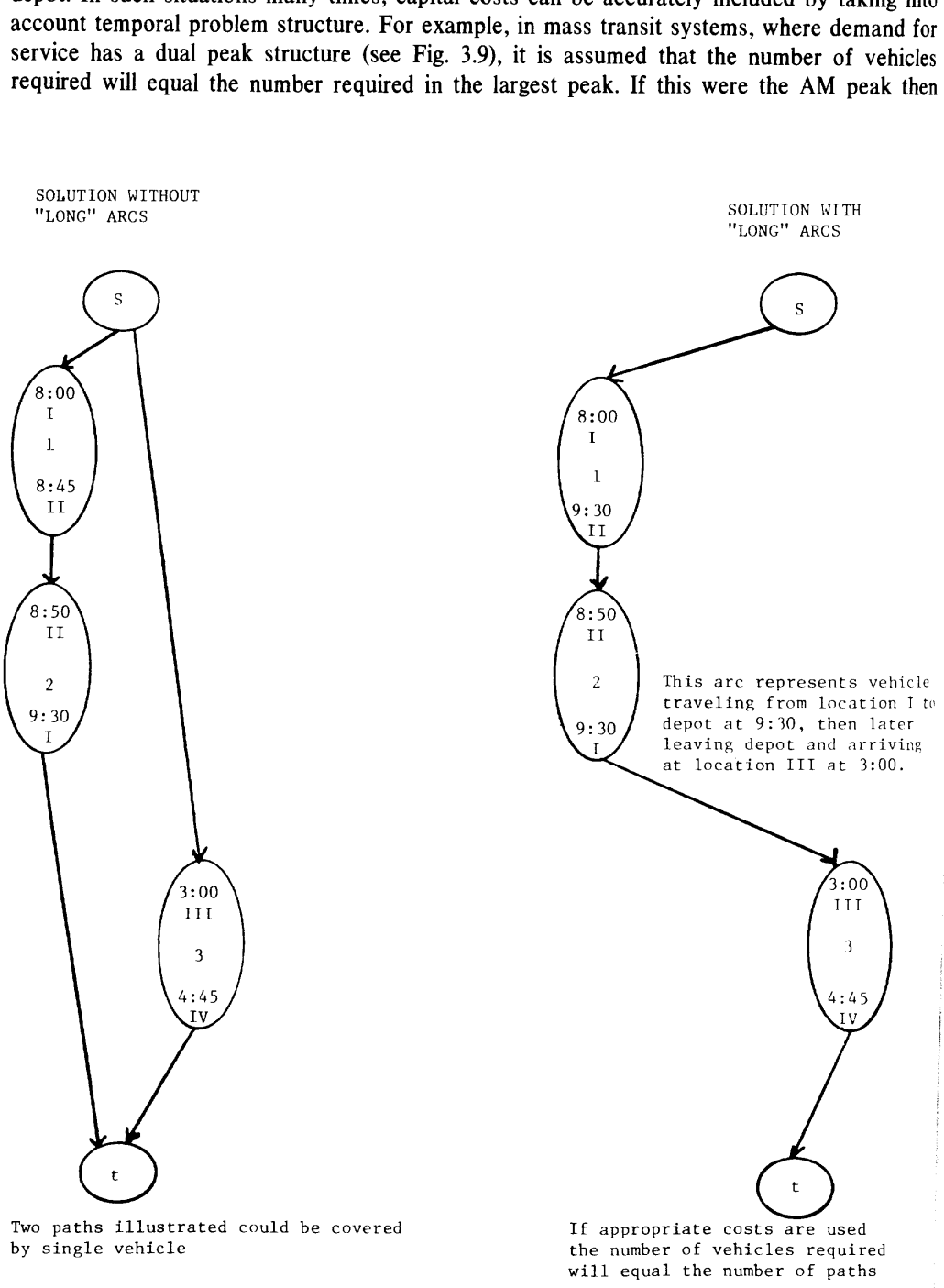


Fig. 3.11. Long arcs in a vehicle scheduling network.

capital cost could be included in the costs of arcs from s to tasks starting between say 5 AM and 9 AM. This is a fairly accurate approximation of total capital costs.

The preceding discussion has assumed that good estimates of capital and operating costs are available that allow the two types of costs to be compared. If this is not the case, then either operating costs alone or capital costs alone can be minimized by minimizing deadhead mileage or the number of paths (subject to previous reservations), respectively. A third alternative is to find the solution that minimizes the number of paths and then resolve the problem minimizing deadhead mileage subject to the restriction that the number of paths used equals the number found in the previous solution. The solution that results would be the least vehicle solution that minimized deadhead mileage.

The mathematical program V1 can easily be seen to be a minimum cost flow problem and consequently can be efficiently solved. The use of this formulation to solve vehicle scheduling problems was first devised by Dantzig and Fulkerson[164]. This formulation with cost per unit flow on the arcs between two nodes being deadheading time is the basis of the vehicle scheduling algorithm contained in the RUCUS package[69]. This formulation with a cost of one on all arcs pointing out of s, which is sometimes referred to as the *Dilworth formulation* (Ford and Fulkerson[227]) can be solved as a maximum flow problem. In this form, it has served as the basis for the vehicle scheduling algorithm in UCOST (Bodin, Rosenfield and Kydes[92]).

3.4.2 *Solving the VSPLPR, VSPMVT and VSPMD*

All the constrained vehicle scheduling problems discussed here, VSPLPR, VSPMVT and VSPMD, are known to be *NP*-hard ([33] and [436]). We can generalize formulation V1 for the VSP to include these problems, however, the generalizations involve complications that preclude them from being solved as minimum cost flow problems. In this section we give formulations for the constrained scheduling problems and present several approximate solution algorithms. We should note that these problems are treated as a group because each of the solution approaches discussed applies to two or more of the problems.

Formulations for the VSPLPR, VSPMVT and VSPMD. We first show how to extend formulation V1 to the VSPLPR and then give a single formulation that applies to the VSPMVT and VSPMD. Recall that the VSPLPR is essentially the same as the VSP except that no paths with time duration greater than some constant T_{MAX} are allowed. We start with the network defined for the VSP but do not include the nodes s and t. Rather we define a set of "back arcs" that are used to characterize feasible paths. A back arc is inserted from task node j to task node i if paths *starting at* i and *ending at* j are feasible, i.e., if $\mathrm{TM(DL, SL}_i) + (\mathrm{ET}_j - \mathrm{ST}_i) + \mathrm{TM(EL}_j, \mathrm{DL}) \leq T_{MAX}$. We denote by A_1 the set of forward arcs defined previously and by A_2 the set of back arcs. Variables x_{ij} represent arcs in A_1 and y_{ij} arcs in A_2. We associate a cost c_{ij} with all $(i, j) \in A_1 \cup A_2$. We may now define VSPLPR as

$$\text{V2:} \quad \text{Min} \sum_{(i,j)\in A_1} c_{ij}x_{ij} + \sum_{(i,j)\in A_2} c_{ij}y_{ij}$$

$$\text{s.t.} \sum_{i:(i,j)\in A_1} x_{ij} + \sum_{i:(i,j)\in A_2} y_{ij} - \sum_{i:(j,i)\in A_1} x_{ji} - \sum_{i:(j,i)\in A_2} y_{ji} = 0 \qquad \text{for all } j \in N, \tag{3.4}$$

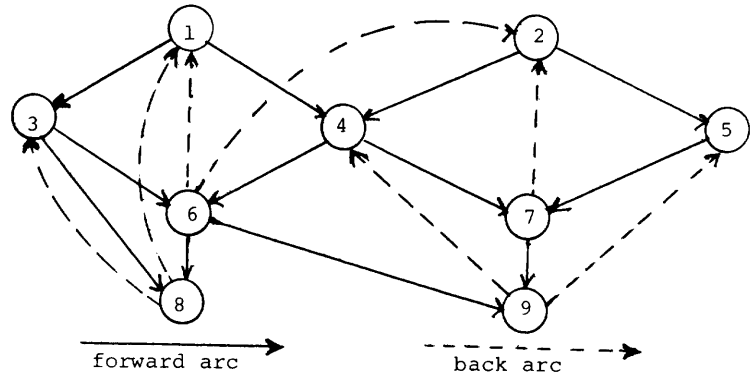
$$\sum_{i:(i,j)\in A_1} x_{ij} + \sum_{i:(i,j)\in A_2} y_{ij} = 1 \qquad \text{for all } j \in N, \tag{3.5}$$

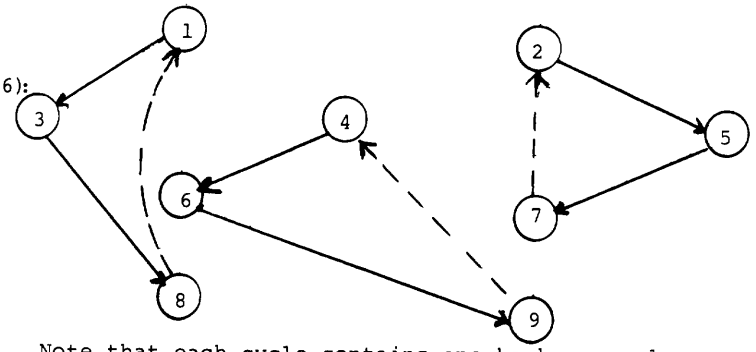
$$\sum_{(i,j)\in A_1\cap C} x_{ij} + \sum_{(i,j)\in A_2\cap C} y_{ij} \leq |C| - 1 \text{ for all cycles } C \text{ with } |A_2 \cap C| \geq 2, \tag{3.6}$$

$$0 \leq x_{ij} \leq 1 \text{ and integer for all } (i, j) \in A_1, \tag{3.7}$$

$$0 \leq y_{ij} \leq 1 \text{ and integer for all } (i, j) \in A_2. \tag{3.8}$$

Where for any set G, $|G|$ is the number of elements in G. Constraints (3.4), (3.5), (3.7) and (3.8)





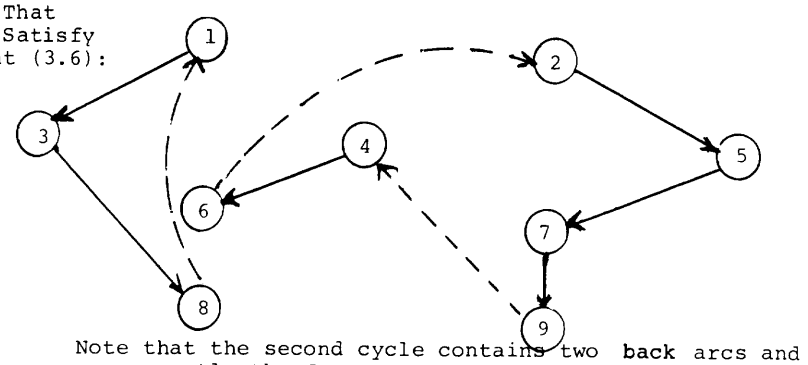


Fig. 3.12. Feasible solutions to V2 with and without constraint (3.6).

constitute a network flow formulation that seeks a feasible circulation. Constraint (3.6) is required to ensure that no cycle includes more than one back arc. Cycles with more than one back arc do not correspond to paths and consequently are not allowed. Figure 3.12 illustrates a feasible solution to V2 and also illustrates a feasible solution to V2 with constraint (3.6) dropped. The previous discussion concerning determination of the c_{ij}'s applies to V2 except that vehicle capital costs and costs for trips to and from the depot are associated with the appropriate back arc.

Although more compact formulations of the VSPLPR exist, the preceding formulation captures some of this problem's structure. In particular, it will be used in describing and solving certain crew scheduling problems in Section 3.5 where, due to the structure of the network considered, constraint (3.6) may be dropped.

We may formulate both VSPMVT and VSPMD as multicommodity flow problems. We start

with the same node set defined for the VSP but rather than adding the single pair of nodes s and t, we include several source/sink pairs, $\{s_1, t_1\}, \{s_2, t_2\}, \ldots, \{s_k, t_k\}$. In the case of VSPMVT each pair corresponds to a vehicle type and in the case of VSPMD each pair corresponds to a depot. We define arcs as in VSP except that the arcs defined out of s and into t are defined out of each s_l and into each t_l. A set of flow variables $\{x_{ij}^l\}$ and costs $\{c_{ij}^l\}$ is associated with each source/sink pair $\{s_l, t_l\}$. Further, we associate lower and upper bounds $\underline{b}_l$ and $\bar{b}_l$ with each vehicle category l. In this case, an x_{ij}^l variable is only associated with arc (i, j) if traversal of the arc is allowed by a vehicle in category l. We denote by A_l all those arcs for which a traversal by a vehicle in category l is allowed. The multicommodity flow formulation is:

$$\text{V3:} \quad \text{Min} \sum_{l=1}^{k} \sum_{(i,j)\in A_l} c_{ij}^l x_{ij}^l$$

$$\text{s.t.} \quad \sum_{i:(i,j)\in A_l} x_{ij}^l - \sum_{i:(j,i)\in A_l} x_{ji}^l = 0$$

$$\text{for all tasks } j \text{ and categories } l, \tag{3.9}$$

$$\underline{b}_l \le \sum_{j:(s,j)\in A_l} x_{sj}^l \le \bar{b}_l \text{ for } l = 1, 2, \ldots, k \tag{3.10}$$

$$\sum_{l=1}^{k} \sum_{i:(i,j)\in A_l} x_{ij}^l = 1 \quad \text{for all task nodes } j \tag{3.11}$$

$$0 \le x_{ij} \le 1 \text{ and integer for } l = 1, 2, \ldots, k \text{ and all } (i, j) \in A_l. \tag{3.12}$$

For VSPMVT, an arc from task node i to task node j is included in A_l if a type l vehicle can service both tasks i and j. For VSPMD, an arc from task node i to task node j is included in A_l if both tasks i and j can be serviced by a vehicle housed at depot l. In both cases arcs out of s_l nodes and into t_l nodes are only included in A_l for $i = l$. We should note that the "driving force" of this formulation for VSPMVT is the definition of the feasible arc set A_l and the bounds $\underline{b}_l$ and $\bar{b}_l$ whereas the driving force for VSPMD is the definition of the arc costs for arcs out of each s_l and into each t_l.

The remaining sections describe algorithms for solving these problems.

The concurrent scheduler. The concurrent scheduler is a simple intuitive approach that has proved quite successful in practice for solving a variety of constrained scheduling problems. Bodin, Rosenfeld and Kydes [92] describe this algorithm as follows:

Step 1. Order tasks by starting time. Assign task 1 to vehicle 1.

Step 2. For $k = 2$ to the number of tasks. If it is feasible to assign task k to an existing vehicle then assign it to the vehicle that involves the minimum deadhead time. Otherwise, create a new vehicle and assign task k the new vehicle.

Depending on whether VSPLPR, VSPMVT, or VSPMD are involved, the appropriate feasibility check would be applied in Step 2. For VSPMD, the new vehicles created in step 2 would be assigned to the nearest depot.

We have found in our discussions with practitioners that this heuristic is widely used in practice. In addition, since it is easy to code and computationally efficient, it is useful for generating starting solutions for other algorithms. It can be interpreted as a member of the class of "greedy" heuristics, which have become popular in recent years because of their simplicitly and analyzability [150, 502].

Two step approaches. VSPMVT and VSPMD can both be viewed as vehicle scheduling/clustering problems in the sense that the output of each is a set of vehicle schedules clustered either by vehicle type or depot. This interpretation suggests two classes of approaches: one which clusters tasks and then schedules vehicles over each cluster and another which schedules vehicles and then clusters vehicle schedules. These approaches are not documented in the literature formally but have nonetheless been widely used by practitioners.

For the task clustering approach, we associate a weight with each task/cluster pair that measures how appropriate it is to include the task in the cluster. For VSPMVT, this weight

would be a 0/1 value depending on whether or not the vehicle type is feasible for that task. For VSPMD, the weight would measure the proximity of the task to the corresponding depot. Given these weights, a minimum weight assignment that satisfies lower and upper bounds on vehicle availability can be found by solving a transportation problem. The VSP is then solved over each cluster.

For the second approach, the VSP is solved over the entire task set. Subsequently, a transportation problem is set up in a manner similar to the one described in the previous paragraph. In this case, the transportation problem assigns entire vehicle schedules to clusters. This second approach has been applied successfully to VSPMDs, however, it is less appropriate for VSPMVTs since feasibility cannot be guaranteed and may be difficult to obtain.

It is interesting to note the similarity in philosophy of the approaches with multiple vehicle node routing approaches described in Section 2.4.4. The first procedure can be thought of as a "cluster first" approach while the second procedure can be thought of as a "route first" approach. In both the routing and scheduling problems we either group the required tasks into clusters, one cluster representing each depot, and then solve a single depot routing or scheduling problem or we form routes or schedules first and then assign these routes or schedules to the appropriate depot.

Finally, we should note that in many cases, a natural clustering exists within the physical system. For example, in mass transit systems, lines provide a natural clustering for trips. This clustering may be quite appropriate to use for either the VSPMVT or the VSPMD since trips arising from a single line have common characteristics with respect to locations and vehicle types required. We should caution, however, that very often clustering by line is enforced by transit agencies to an undesirable extent due to convenience or historical reasons.

An interchange heuristic. In this section, we describe an interchange heuristic [605] that has been applied very successfully in several mass transit agencies in England. It can handle a wide variety of cost functions and constraints. Side constraints are typically handled by using Lagrangean relaxation in the manner described in Section 2.5.

The procedure can be viewed as an adaptation of the 2-opt algorithm for the traveling salesman problem [445] (see Section 2.3). A starting solution can be found using a variety of approaches such as the concurrent scheduler. Assuming that a starting solution is already available, the procedure effects interchanges between the components of this schedule to improve costs. An interchange affects only two vehicle schedules, say, 1 and 2. It joins the first half of vehicle schedule 1 with the second half of vehicle schedule 2 and the first half of 2 with the second half of 1. In this case, the cost of the interchange cannot be evaluated by simply looking at the four arcs involved, but rather, the costs of the two new vehicle schedules must be compared to the cost of the two old vehicle schedules. This more global evaluation strategy will allow a variety of complex cost functions to be considered. Two cost functions that relate to the problems mentioned in this section are now described.

The first cost function is appropriate for handling the VSPMVT. The cost of a vehicle schedule is:

$$\underset{\text{vehicle type } l}{\text{Min}} \quad \sum_{i \in B(l)} (\mathrm{ET}_i - \mathrm{ST}_i)$$

where $B(l)$ is the set of tasks i in the schedule that are not feasible for type l vehicles. Note that this cost is zero if the vehicle schedule is feasible for some vehicle type. This objective function does not consider vehicle availability ($\underline{b}_l$ and $\bar{b}_l$). If the interchange heuristic produces a set of schedules that violate the vehicle availability, then a transportation problem is solved that assigns the vehicle schedules generated to vehicle types using the cost function described above and the restrictions on vehicle availability ($\underline{b}_l$ and $\bar{b}_l$). If the value of the transportation solution obtained is greater than zero (i.e. solution to vehicle scheduling problem is infeasible) then the interchange heuristic is reentered where certain vehicle types are penalized based on the shadow prices obtained from the transportation problem solution. The procedure continues to iterate in this fashion until a feasible solution is obtained or no further improvement is found.

The following cost function is appropriate for the VSPMD. The cost of a vehicle schedule is

$$\min_{\text{all depots } l} \{\text{cost to travel from } EL_j \text{ to depot } l \text{ plus the cost of travel from depot } l \text{ to } SL_i \text{ where } i \text{ is the first trip on the schedule and } j \text{ is the last}\}$$

Although reference [605] does not describe explicitly how to handle depot capacities ($\underline{b}_l$ and $\bar{b}_l$), presumably these could be handled by modifications in the technique described in the previous paragraph.

The two objective functions described above are aimed only at obtaining feasible solutions. In a practical application of this procedure, a weighted objective that includes the objective function components described above as well as components related to vehicle capital and operating costs would be appropriate.

3.5 CREW SCHEDULING ALGORITHMS

This section is divided into five subsections. Section 3.5.1 gives a general formulation that applies to all the crew scheduling problems discussed. Sections 3.5.2, 3.5.3, 3.5.4 and 3.5.5 present algorithms for scheduling workers at a fixed location, mass transit crew/vehicle scheduling, air crew scheduling, and rostering and bid line problems, respectively. As we noted in Section 3.3, fixed location worker scheduling provides insight and algorithmic intuition for the more complicated crew scheduling problems. When this problem is discussed in Section 3.5.2, we also discuss its application to estimating mass transit crew costs.

3.5.1 *General formulation for crew scheduling problems*

As with all scheduling problems, we start with a set of tasks as input. We represent the crew scheduling problem with a hierarchical set of constraints. The first level defines continuous crew work periods (CWPs). The key assumption concerning CWPs is that the crew pay depends only on the starting and ending times of the CWP. The second level of constraints groups sets of CWPs into full work schedules (FWSs). For fixed location worker scheduling, Figure 3.6 illustrates three FWPs consisting of two, one and three CWPs. In mass transit crew scheduling a CWP corresponds to a piece and an FWS corresponds to a run. In air crew scheduling a CWP corresponds to a duty period and an FWS corresponds to a pairing.

The first level of constraints consists of the constraint set for V2 (see 3.4 through 3.8). We start with a node set N that represents each task by a node. The arc set A_1 includes the arc (i, j) if a crew can perform task i followed by task j in a single CWP. Arc set A_2 includes all "back arcs" (j, i) for which a CWP starting at task i and ending at task j is feasible. The constraint set for V2 applied to this network yields the first level of constraints.

To complete the formulation, we must describe the manner in which CWPs are formed into FWSs. An FWS is simply a set of CWPs. Typically an FWS consists of a small number, e.g. 2, 3 or 4, of CWPs. We call a potential FWS a *pattern* and define ρ = the set of all FWS patterns, $p(i, j)$ = the set of all patterns that cover a CWP starting with task node i and ending with task node j.

The constraint required to insure that all CWPs are covered by an FWS is

$$\sum_{l \in p(i,j)} z_l - y_{ji} = 0 \quad \text{for all } (j, i) \in A_2 \tag{3.13}$$

$$0 \le z_l \le 1 \text{ and integer} \quad \text{for all } l \in \rho \tag{3.14}$$

where

$$z_l = \begin{cases} 1 & \text{if pattern } l \text{ is used,} \\ 0 & \text{otherwise.} \end{cases}$$

To define the mathematical program completely, we associate costs d_l with each $l \in \rho$ as

well as c_{ij}'s with each $(i, j) \in A_1 \cup A_2$. We now have

$$\text{C1: Min} \sum_{l \in \rho} c_l z_l + \sum_{(i,j) \in A_1} c_{ij} x_{ij} + \sum_{(i,j) \in A_2} c_{ij} y_{ij}$$

$$\text{s.t. (3.4), (3.5), (3.6), (3.7), (3.8), (3.13), (3.14).}$$

Generally, all crew costs can be captured in the d_l's. In Section 3.5.3 we apply this formulation to mass transit crew/vehicle scheduling. In that case, the c_{ij}'s are used to handle vehicle costs.

We should note that it typically is impractical to use this formulation explicitly to solve scheduling problems. One reason is that the amount of data required to "write it down" can be quite large. In particular, the number of patterns can grow in proportion to n^{2k} where n is the number of tasks and k is the number of CWPs per FWS. Even with k equal to 2 or 3 this growth rate can be prohibitively large.

We will describe several algorithms in terms of this formulation. In most cases, the algorithms employ two or more phases, with the early phases assigning values to a subset of the variables. Once certain variables are fixed, the resultant problem becomes smaller and much more tractable.

3.5.2 *Algorithms for scheduling workers at a fixed location*

As we discussed in Section 3.3.1, this problem is characterized by a demand histogram and a set of possible work days. We may specify the demand histogram by defining

T = number of time intervals,

d_t = number of workers required during interval t for $t = 1, 2, \ldots, T$.

To make this demand specification compatible with the model given in the previous section we must define a set of tasks. In the most direct representation, each time interval t corresponds to a set of d_t tasks whose start and end times are the start and end times of interval t. Rather than explicitly representing each of these tasks, we represent each time interval t with a single node in N and replace constraint (3.5) with the following constraint which states that task t must be covered by at least d_t workers,

$$\sum_{i:(i,j) \in A_1} x_{ij} + \sum_{i:(i,j) \in A_2} y_{ij} \geq d_t \quad \text{for } t = 1, \ldots, T. \tag{3.15}$$

We also allow all variables to take on values greater than one which means that several crews may follow the corresponding arc or pattern. This is accomplished by dropping the upper bounds from (3.7), (3.8) and (3.14). As Fig. 3.13 illustrates in the network associated with specification of CWPs for the fixed location worker scheduling problem, the set of all forward arcs forms a single path. It is not difficult to see that in such networks constraint (3.6) may be deleted to obtain a set of network flow constraints. This observation does not mean that the entire problem can always be solved as a network flow problem. However, if each FWS consists of exactly one CWP, i.e. if all workdays contain no rest or lunch breaks, then since constraints (3.13) and (3.14) are not required, we may solve the fixed location worker scheduling problem as a network flow problem.

When breaks in the workdays are allowed, the problem remains quite difficult even though the CWP constraints have a network flow structure. In this case, we give another formulation of the problem [76]. We define

n = number of possible workdays,

$$a_{tj} = \begin{cases} 1 & \text{if work day } j \text{ covers period } t \\ 0 & \text{otherwise} \end{cases}$$

$$\text{for } j = 1, 2, \ldots, n,$$

c_j = cost of work day j, for $j = 1, 2, \ldots, n$,

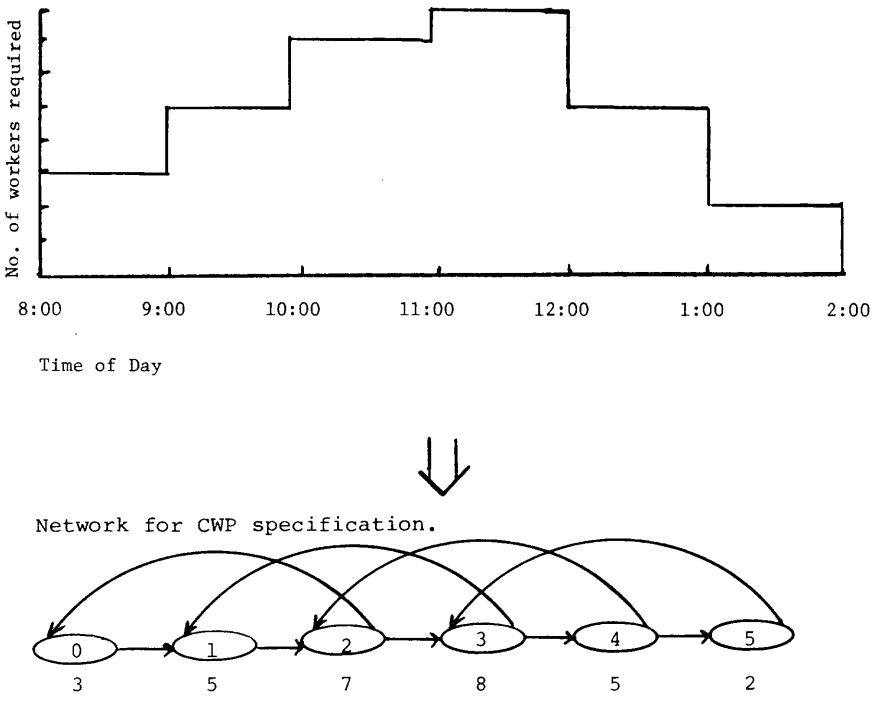


Fig. 3.13. Network associated with CWP specification for fixed location worker scheduling problem.

then we may write the fixed location worker scheduling problem as,

F1: Minimize $\sum_{j=1}^{n} c_j x_j$

subject to

$$\sum_{j=1}^{n} a_{tj} x_j \geq d_t \quad \text{for } t = 1, 2, \ldots, T, \tag{3.16}$$

$$x_j \geq 0 \text{ and integer for } j = 1, 2, \ldots, n. \tag{3.17}$$

Note that this formulation is, in fact, a generalization of the set covering problem.

Examples of the use of the model discussed in this section can be found in Bennett and Potts [67], Bodin *et al.* [91] and [92], Heller [340], Monroe [494], Guha and Browne [321], Jenkins [365], Miller [484] and Segal [589].

This model has also been used in conjunction with mass transit crew scheduling to estimate crew requirements [76], [92] and as a first step in a crew scheduling algorithm [437]. In this case, a time interval length is arbitrarily chosen with accuracy increasing as the time interval length decreases. We assume that vehicles have already been scheduled and define

d_t = the number of vehicles that operate during the entire time interval t.

Note that d_t is a lower bound on the number of crews required to be on duty during time period t. Even with time intervals defined as small as possible, e.g. 1 minute, this is still an approximate model in that it ignores crew movements required to get from the end location of one piece to the start location of another.

This model has been used as part of the UCOST system developed by the Department of Transportation [92] for long range planning. In this case, the histogram was partitioned into continuous $\frac{1}{2}$ day shifts no more than about $4\frac{1}{2}$ hours in length. It was assumed that each shift

could be paired with some other shift to form a full day run. In this way an estimate of the total number of required crews (half the number of shifts) was obtained using a flow algorithm in a situation where non-continuous work days were allowed.

Jean Marc Rousseau and his colleagues [76, 437] have used the model both to estimate crew requirements and as a first step in crew scheduling. Their model allows work days to contain explicit breaks and therefore rules out using a flow algorithm. Instead, they use formulation F1 and relax the integrality restriction on the x_j's so that the problem may be efficiently solved using a linear programming code. In this case, an additional set of linear constraints of the form

$$\sum_{j=1}^{n} b_{ij}x_j \leq e_i \quad \text{for } i = 1, 2, \ldots, m \tag{3.18}$$

is added to the model. These constraints serve two purposes: first, to enforce a variety of different union restrictions and crew work day requirements, and second to make the model itself conform more closely to reality. A constraint of the first type might be that at least 50% of all runs must be straight runs. A constraint of the second type might be that if there are 10 short vehicle blocks starting at 7:30 AM and ending at 9:00 AM then there must be 10 runs with pieces starting at 7:30 AM and ending at 9:00 AM. This procedure has been used to estimate the economic impact of new union regulations proposed during union negotiations. The additional set of constraints (3.18) is particularly useful in this regard.

3.5.3 *Mass transit crew/vehicle scheduling algorithms*

Two major workshops dealing with mass transit crew and vehicle scheduling have been held: one in Chicago in 1975 [74] and the other in Leeds, England in 1980 [692]. The proceedings of the latter [692] provide an accurate representation of the state of the art in this area. The first attempts at computerized crew scheduling either followed closely the techniques used by manual schedulers ("run cutting") or viewed the problem explicitly as a set partitioning problem. Initially, both of these approaches were unsuccessful. However, over the past few years, a large amount of research has been directed toward the two approaches and both have produced good results. In addition to discussing these two areas, we present two new approaches, one which initially solves the crew estimation problem described in Section 3.5.2 [437] and one which simultaneously generates crew and vehicle schedules using several matching-based heuristics [35]. We should mention that aside from this last approach, all other approaches described start with a solution to the vehicle scheduling problem (i.e. operate sequentially).

Before describing individual procedures we interpret, in terms of the formulation given in Section 3.5.1, the effect of solving the vehicle scheduling problem prior to scheduling crews. Recall that in the mass transit crew/vehicle scheduling problem, the set of input tasks is the set of d-trips. The start of each d-trip is either the beginning of a trip (BT) or a relief point (RP) and the end of each d-trip is either the end of a trip (ET) or a relief point. Forward arcs (arcs in A_1) are drawn from each ET to BT exactly as in the vehicle scheduling network except that no "long arcs" are allowed. One forward arc is drawn from each d-trip ending in an RP to the d-trip that starts with the corresponding RP. The use of an ET to BT arc has the same interpretation as in the VSP, namely that the crew and vehicle proceed from the end of the first trip to the beginning of the next. The use of an RP to RP arc simply means that a crew does not exit the vehicle at that relief point. Back arcs represent feasible pieces, for example, a back arc from a node ending in an RP to a node beginning with a BT would represent a piece which started with the crew pulling the vehicle out of the garage and ended with the crew exiting the vehicle at a relief point.

The effect of scheduling vehicles is to specify values for the BT to ET arc variables. This greatly simplifies the resulting network. As Fig. 3.14 illustrates, after scheduling vehicles the network consists of a set of isolated subnetworks, each one corresponding to a single vehicle. Each of these subnetworks has the network structure of the fixed location worker scheduling problem of Section 3.3.2. This structure is used in solving a subproblem in the algorithm of [437]. A further effect of scheduling vehicles is that the number of potential patterns is greatly reduced. Set partitioning algorithms must enumerate all or a substantial number of the patterns

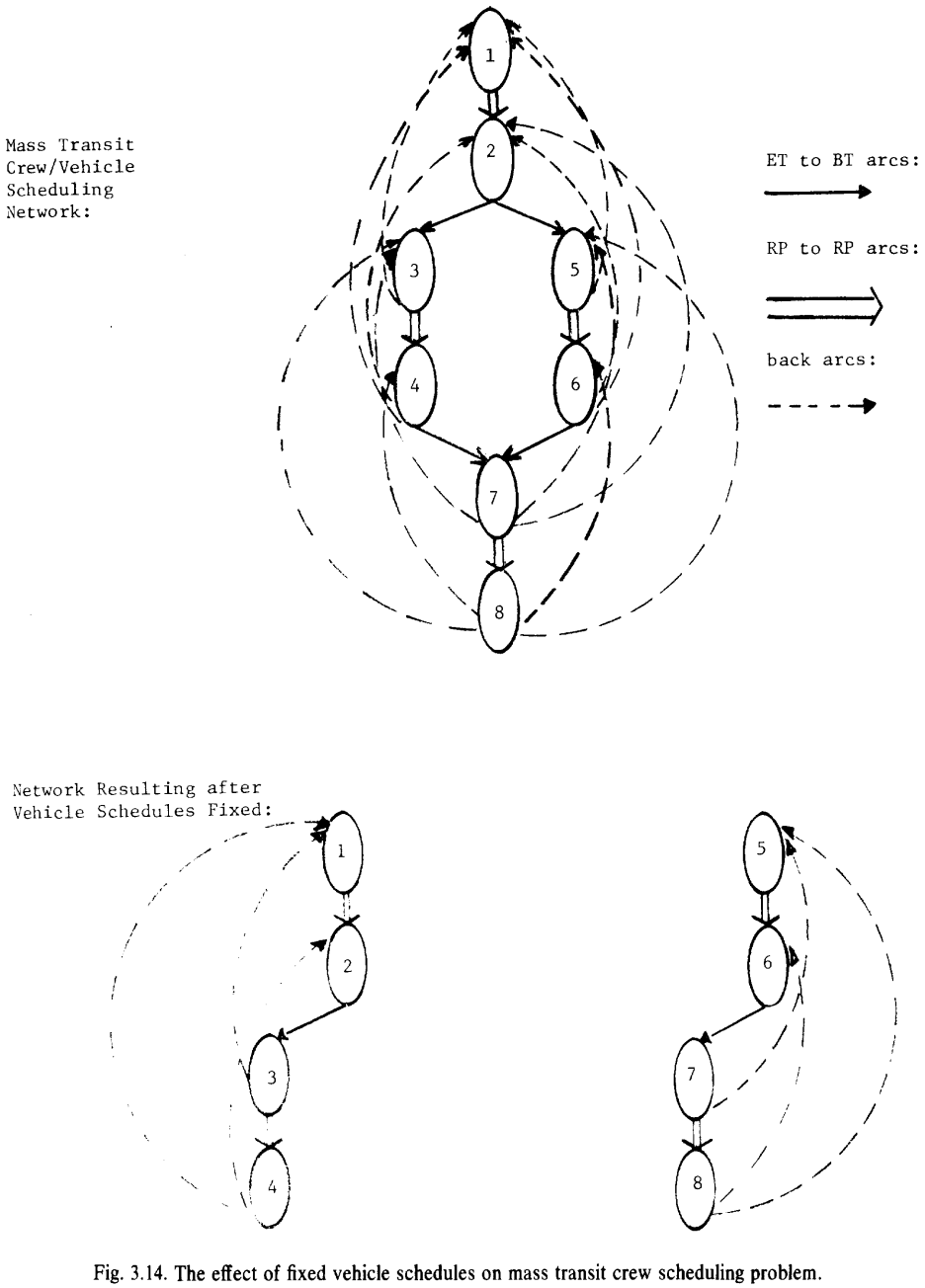


Fig. 3.14. The effect of fixed vehicle schedules on mass transit crew scheduling problem.

Thus, this reduction is extremely important. Of course, it is possible that scheduling vehicles first can eliminate many cost effective solutions (including the optimal solution).

Set partitioning based approaches. Set partitioning and covering methods have generally not been used in operational settings within mass transit agencies. However, a large amount of recent research [490, 539, 574, 665] has been directed toward limiting problem size by decomposing the problem or restricting the set of columns generated. To understand these approaches we must reexamine the service profile graph given in Fig. 3.9. Ward *et al.* [665] use a decomposition approach which classifies runs according to where their component pieces fit into the service profile. They then solve a sequence of set partitioning problems with the first problem generating the runs that end the latest (owls), the second problem generating the runs that end the second latest (late straights) and so forth until all trips are covered. Of course, the set of columns generated at any step will be constrained by the runs generated during previous steps. Since all columns are not considered simultaneously the approach cannot guarantee an optimal solution to the set partitioning problem.

Parker and Smith[539] include all trips in the service profile within one set covering problem, however, they use a variety of criteria to restrict the set of columns considered. These include: (a) lower and upper bounds on piece length; (b) lower and upper bounds on length of breaks; (c) restrictions on piece start and end times based on run classification similar to the one shown in Fig. 3.9.

It is widely felt that the service profile contains valuable information that should be used in generating crew schedules. These two approaches attempt to use this information. The approach of LESSARD *et al.*[437], which is described later in this section, includes profile information within the crew scheduling algorithm by solving the estimation problem described in Section 3.5.2. The two set partitioning/covering methods just described have undergone some computational testing and the results appear promising. However, they have not yet been used in an operational environment.

The matching problem is the set partitioning problem with no more than two ones per column. An efficient exact solution procedure exists for this problem[193]. Several approaches (e.g. [35], [437]) to mass transit crew scheduling initially generate a set of pieces or a subset of the runs together with some "left-over" pieces and then solve the problem of generating runs from pieces using a matching algorithm.

Run cutting approaches. It was generally felt that manual schedulers produced extremely good solutions, with the primary disadvantage of the manual approach being the time it took to produce a new solution (several weeks) and the time it took to train new schedulers (several years). While this assessment is true to a certain extent, there have been significant cost savings produced in recent years[413, 437, 582] using computerized techniques. We refer to the general set of heuristic approaches used by manual schedulers as *run cutting*. The first large-scale computerized implementation of run cutting was called RUCUS; it was developed under sponsorship of the Urban Mass Transportation Administration. The results were mixed. The system was cumbersome to use, rather inflexible and operated in a "batch" environment. Consequently, it became very difficult to incorporate minor changes in work rules and problem parameters. Some of the earlier computational experience with RUCUS is presented in Goeddel[285].

A recent implementation[345], which is used in an operational environment, is based on heuristic methods inspired by the manual schedulers. This method can be viewed as a refinement of the initial RUCUS technique. Some of the main embellishments that contributed to the success of this package include: (a) a highly flexible interactive user interface; (b) a large set of parameters for specifying the problem and for fine-tuning the algorithm; (c) a flexible system for manipulating input data.

We will now describe a procedure which is typical of the methods employed in [673] and [345]. The algorithm "cuts" pieces of work from a pair of vehicle schedules and fixes this pair as a run. To choose the first piece the algorithm scans all vehicle blocks for the earliest relief point beginning a piece of unscheduled work (block *A* in Fig. 3.15). It then determines a set of possible relief points along this same vehicle block for ending the piece. These relief points are determined based on maximum and minimum piece lengths for both the first piece on the run being generated and the leftover piece. For each one of these relief points, candidate pieces from other blocks (block *B* in Fig. 3.15) are found for forming possible two piece runs. Each candidate piece is evaluated in terms of the run formed and in terms of the quality of the segments remaining after the piece is cut out of the vehicle schedule. For example, having a very short piece among the remaining segments would be considered undesirable since it would be difficult to form a good run using this piece. This procedure depends heavily on the fine-tuning of several subjective cost parameters (e.g. cost of left-over segments).

An algorithm based on service profile decomposition. Lessard *et al.*[437] use a two step approach for solving the mass transit crew scheduling problem: the first step generates pieces and the second merges pieces into runs. They use information contained in the service profile to guide the piece generation phase. This information is obtained by solving the linear programming relaxation of the histogram partitioning problem described in Section 3.5.2. We should note that this relaxation can be shown to be a relaxation of the crew scheduling problem itself.

A solution to the linear programming relaxation in effect specifies a "shape" for the crew scheduling solution. For example, it might specify that there should be about 10 runs consisting

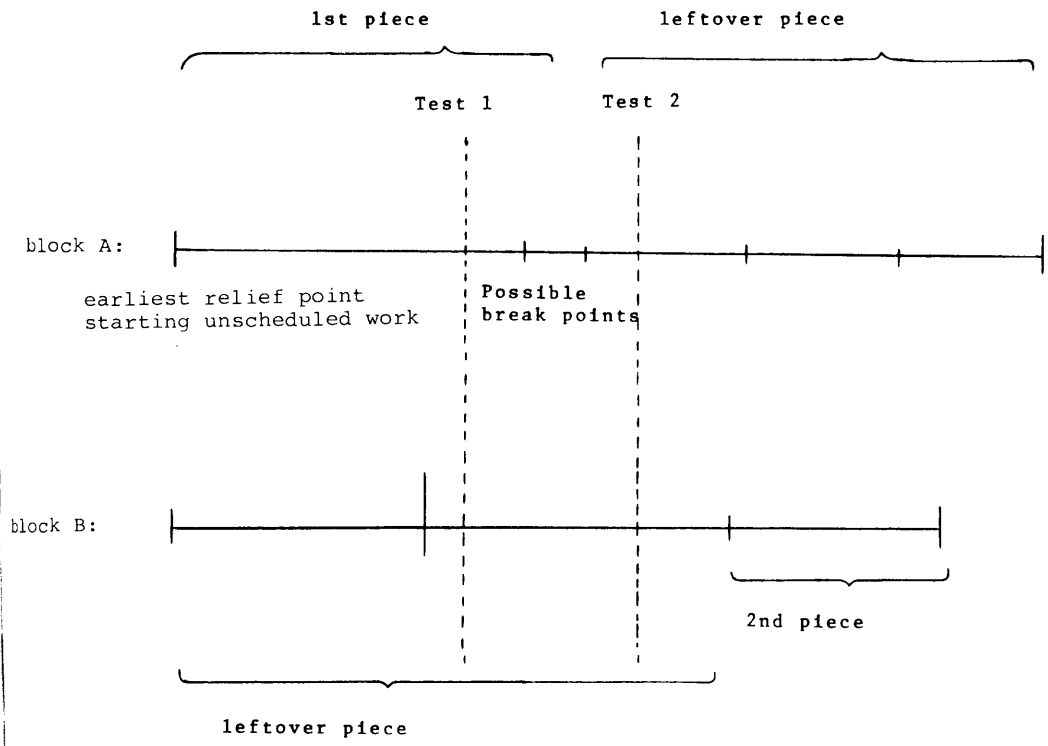


Fig. 3.15. Creation of two-piece runs using a run cutting algorithm: first piece of block A and second piece of block B are joined subject to tests 1 and 2.

of a short peice during the AM peak and a long peice during the PM peak, about 15 runs with two equal size pieces, one in the AM peak and one during the mid-day, etc. This in turn may be viewed purely as information concerning pieces, e.g. 10 pieces should start at 7:00 AM and end at 10:15 AM, 5 pieces should start at 7:30 AM and end at 11:30 AM, etc. Since this set of pieces makes up the solution to a relaxation of the crew scheduling problem the corresponding cost is a lower bound on the optimal solution. Consequently, if a real set of pieces is found that is "close to" this lower bound piece set, its cost should be close to the optimal solution. A quadratic objective function is used for the piece generation phase which minimizes the discrepancy between the linear programming solution and the piece set obtained. The overall solution procedure can now be outlined:

Step 1. Solve the linear programming relaxation of the histogram partitioning problem.

Step 2. Partition the vehicle blocks into pieces, so that the resulting piece set conforms as closely as possible to the set of pieces specified by the solution to the linear program.

Step 3. Combine peices into runs.

Step 4. Try to improve the solution by making marginal changes in the partition of vehicle schedules into pieces.

Step 1, of course, can be solved efficiently using a linear programming code. A heuristic is used for Step 2 which solves a sequence of shortest path problems. This algorithm makes use of the network structure illustrated in Fig. 3.14. Note that, in terms of the formulation given in Section 3.5.1, once Step 2 is completed the only unspecified variables are the z_l's. Thus the only relevant constraints are (3.13) and (3.14) where in (3.13) the y_{ji} values are fixed. This remaining problem can be viewed as a matching problem. Since it is rather large, Step 3 solves it using a heuristic that makes use of a matching code. This procedure is being successfully used in an operational environment in Quebec City and Montreal. In Montreal it produced a savings of 3% in crew costs. Experimentation with using other relaxations in Step 1 are given in [113].

A matching based algorithm. In Ball, Bodin and Dial[35, 36], the crews and vehicles are scheduled simultaneously. The procedure consists of two phases corresponding to the two levels of constraints in the formulation of Section 3.5.1. The first phase specifies pieces, i.e. the values for the x_{ij} and y_{ij} variables and the second phase merges pieces into runs, i.e. specifies values for the z_l variables.

The network used in the first phase does not include the back arcs. Rather the problem is viewed as partitioning the nodes in an acyclic network into paths restricted in length. This path length restriction, T, would be the maximum piece time, e.g. 4–1/2 hours.

The major steps of the algorithm are:

Step 1. Form the crew/vehicle scheduling network defined at the beginning of this section but do not include back arcs. Partition the nodes in this acyclic network into a set of paths (pieces) whose lengths are less than or equal to T.

Step 2. Improve the set of pieces by performing combination/splits of pairs of pieces.

Step 3. Combine pairs of pieces into runs or crew schedules.

Step 1 starts by partitioning the nodes into a set of levels and initially defines a set of "partial paths" as the nodes on level 1. It iteratively augments the partial paths by the nodes on the next higher numbered level using a matching based heuristic. Step 2 uses a matching based interchange heuristic. As was mentioned earlier, the problem of combining pieces into runs (Step 3) can also be formulated and solved as a matching problem. This matching problem usually has side constraints associated with it such as a constraint that at least 50% of all runs must be straight runs.

Although this procedure has not been used in an operational environment, it has been tested on a data base from the city of Baltimore and has produced significant cost savings (see Ball, Bodin and Dial [35]).

3.5.4 *Air crew scheduling algorithms*

A scheduling problem which has tantalized combinatorial analysts for many years is the air crew scheduling problem, i.e. the problem of forming a set of pairings which cover all the flight legs in the timetable. As is demonstrated in Section 3.5.1, the air crew scheduling problem and the mass transit crew/vehicle scheduling problem have a number of similarities. However, while a variety of methods have been proposed for the mass transit problem, research on air crew scheduling has centered on using the set partitioning model. The primary reason for this is that air crew scheduling problems are typically much smaller. In particular, problems with 100 to 300 flight legs are considered quite realistic in air crew scheduling whereas in mass transit crew scheduling a small to moderately sized problem might have 1500 d-trips and larger problems could have several thousand d-trips. Other differences are that: (1) the air crew scheduling problem does not have the highly peaked demand structure illustrated in Fig. 3.9, (2) the air crew scheduling problem covers a longer time horizon, i.e. a set of pairings covers a month whereas a set of runs a day, and (3) a duty period is not necessarily carried out on a single airplane whereas a piece is always carried out on a single vehicle.

The set partitioning model was first used to solve airline crew scheduling problems a number of years ago [610]. However, the early attempts to solve these problems were largely unsuccessful due to two difficulties: (1) the problems attempted were simply too large for existing exact codes, e.g. 400 rows and 30,000 columns, and (2) effective heuristic codes were not available. More recently, the situation has improved dramatically [471] due to the development of better algorithms and codes and more importantly, as a result of an explicit consideration of problem size in choosing an approach. Current set partitioning codes can solve problems with up to approximately 150 rows. Heuristics have been used to solve problems of larger size. Problems within this range are obtained by looking at smaller airlines or by restricting one's attention to smaller aircraft fleets. In addition, in many cases, larger problems can be decomposed into components in such a way that it suffices to solve a few small set partitioning problems.

In this section, we present an overview of the exact set partitioning based approaches of Rubin [570] and Marsten and his colleagues [470, 471]. In addition, we describe a set of heuristic set partitioning based procedures and solution improvement methods developed by Baker and others [20, 23–25].

Exact set partitioning approaches. The two approaches described in this subsection are based on solving a set partitioning problem using an exact set partitioning algorithm, e.g. branch and bound. However, since in many cases the set partitioning problem solved is an aggregation or restriction of the original crew scheduling problem, optimality is not necessarily guaranteed.

One of the first uses of the set partitioning approach for air crews was the algorithm developed by Rubin[570]. It proceeds as follows:

Step 0. Find an initial feasible set of pairings P.

Step 1. For $i = 2$ to $\bar{k}$. For each subset S of P of size i let R be the set of legs (rows) covered by S. Find the set, A, of all pairings that cover exactly the set of rows R. Solve the set partitioning problem with row set R and column set A. If the cost of the solution is less than the cost of S, then replace S in P with the solution found.

Generally, $\bar{k}$ is chosen to be rather small, i.e. 2 or 3. This procedure became a standard approach in the airline industry and several adaptations of this procedure are still in use by various airline companies. This procedure is very slow computationally, but it tends to give good answers in most applications attempted.

Marsten and Shepardson[471] and Marsten, Muller and Killion[470] report on three highly successful uses of Marsten's linear programming-based set partitioning algorithm[469] for solving the air crew scheduling problem. The airlines involved in the study were The Flying Tiger Line, Pacific Southwest Airways, and Continental Airlines. In most cases, problems with fewer than 130 rows and 30,000 columns were solved to optimality or to within 1% of optimality. The yearly cost savings produced by these solutions were on the order of hundreds of thousands of dollars. The problem sizes involved are fairly modest due to the fact that the airlines involved had relatively small fleets and that the schedulers created *resolved legs.* A resolved leg is a sequence of regular flight legs that have to be flown together as a unit. On a resolved leg, the trips are so grouped that there is no opportunity for the crew to change planes. Also, a sophisticated column generation procedure was used to eliminate columns that were not likely to appear in an optimal solution. This use of an exact set partitioning algorithm in this context does not guarantee optimality since the resolved leg represents an aggregation of the rows in the overall set partitioning problem and since the elimination of columns could discard the optimal solution from the set of solutions considered by the algorithm.

Heuristic set partitioning. Heuristics are the only recourse for most of the larger crew scheduling problems encountered in practice. Baker[20] and Baker, Bodin and Fisher[24] describe the use of heuristic set covering algorithms for solving air crew scheduling problems. These algorithms, together with solution improvement methods which are described in the next subsection, are being used by the Federal Express Corporation for both single and multiple depot crew scheduling. Similar approaches have been used by the Selective Bidding Corporation. These algorithms are quite fast and have been compared favorably with competitive procedures.

The Federal Express crew scheduling system contains three steps: (i) a pairings generator used to enumerate candidate crew schedules, (ii) a heuristic set covering algorithm, used to obtain an initial covering solution, and (iii) several solution improvement techniques used to reduce the total cost of the covering solution. Steps (i) and (ii) are described below and Step (iii) is described in the next subsection.

The pairings generator enumerates new pairings which are feasible vis-a-vis the constraints described in Section 3.3.3 and computes the cost of each pairing. In this enumeration, subjective reduction procedures are incorporated to limit the number of pairings generated and to reduce the computer run time of the generator module. The imposition of subjective reduction procedures in the enumeration process was used to enforce desired operational policies associated with the crew pairings (e.g. maximum time away from base, maximum layover time, maximum time away from base to flight time ratio, etc.).

Four distinct heuristics for finding feasible solutions to the set covering problem were evaluated. The first procedure, termed Algorithm A, begins with an empty solution set and adds to the solution the crew schedule that minimizes the cost per uncovered flight. The procedure continues until all flights are covered. The second procedure, Algorithm B, considers the crew schedules in ascending order of cost per flight leg covered. The algorithm begins with an empty solution set and adds the first crew schedule from the sorted list that contains no previously covered flights. If a solution is not obtained after a complete pass through the crew schedules, the process is repeated allowing candidate crew schedules to contain one previously covered flight. This process is repeated, allowing increased overcoverages until a complete solution is obtained.

The third procedure considered, Algorithm *C*, was proposed by Johnson[367] for the case where all columns have costs equal to one; this gives the minimum number of pairings needed to cover all the legs. Algorithm *C* executes in a manner identical to Algorithm *A* except that crew schedules are selected based on the maximum number of uncovered flights. The rationale of this approach is that if the column generator contains subjective tests that produce only efficient crew schedules, then a minimum set of such schedules may produce a near-optimal solution. The fourth procedure, termed Algorithm *D*, is Johnson's technique for the exact covering problem. This procedure starts with an empty solution set and adds to the solution the crew schedule that minimizes the ratio of covered to uncovered flight segments. The rationale of this algorithm is similar to that of Algorithm *C*.

Baker experimented with several data bases from Federal Express Corporation. The computational results indicate that Algorithm *A*, the selection of crew schedules according to a minimum cost per uncovered flight leg criterion, dominates the other set covering heuristics considered. Additional computational evidence in support of Algorithm *A* as an effective set covering heuristic may be found in the recent papers of Balas and Ho[30]. In addition, in Baker[20] and in Ho[351], discussions of worst case bounds for set covering heuristics may be found.

Solution improvement procedures. Baker, Bodin, Finnegan and Ponder[23] and Baker, Bodin and Fisher[24] propose three solution improvement procedures: an extended 2-opt algorithm, a solution merge technique and a hub turn determination procedure. A brief description of these solution improvement procedures is given below.

The extended 2-opt procedure, which assumes a single feasible solution to the set covering problem as input, is an adaptation of the 2-opt procedure proposed by Lin[443] for the traveling salesman problem (see Section 2.3). In this interpretation of the air crew scheduling problem, each flight leg to be scheduled is viewed as a city to be visited. The exchange of arcs connecting the cities on a traveling salesman's route is analogous to changing the sequence in which the flight legs are scheduled. Since any such exchange must take place at a stop common to all the candidate pairings, each city is examined for possible exchanges of pairings completions at that point. If k pairings transit a particular city, then a $k \times k$ assignment problem may be solved to determine the optimal reconfiguration of the crew schedules.

The second solution improvement procedure employed was a solution merging technique. Input to the merge procedure is a *set* of feasible solutions. Solution merging, which was originally proposed by Hinson and Mulherkar[350] for the Federal Express aircraft routing problem (see Chapter 4), exploits the availability of feasible covering solutions provided by the set covering heuristics. The first solution found becomes the incumbent. As subsequent solutions are found they are compared to the incumbent to determine if a reduced cost composite solution can be formed. Since this technique compares only two solutions at a time, the determination of the lower cost composite solution can be made very efficiently. The merge problem can be viewed as a restricted set partitioning problem where the rows are the legs and the columns are the pairings from the two solutions. This problem is restricted in the sense that only a fraction of the set of all possible columns are considered.

The final solution improvement technique employed involved joining two pairings together into a single pairing which is hubturned. A hubturn occurs when a crew transits the crew base on a pairing. All pairings start and end at the crew base. It is often possible to exploit the nonlinearity of the crew scheduling objective function by allowing two individual pairings to be performed sequentially as a reduced cost hubturn pairing. A matching algorithm can be used to find a hubturned solution. The nodes in the matching problem are the pairings in the best solution found so far and the arcs represent joining two pairings into a pairing with a hubturn. The costs on the arcs are the actual cost for a hubturned solution. The node costs are the costs of corresponding pairings without hubturns.

These procedures together with a heuristic set partitioning algorithm are being used by Federal Express Corporation to solve single hub crew scheduling problems. Computational results may be found in [23] and [24].

3.5.5 *Algorithms for rostering and bid line problems*

As described in Section 3.3.4, the output of the algorithms described in Section 3.5.3 and 3.5.4 may or may not be the final solution to the corresponding crew scheduling problem. The

output of the algorithms in Section 3.5.3 are mass transit crew schedules (runs) that cover all work on a particular day. The rostering problem [67, 321, 341, 112] refers to the problem of sequencing runs over a week, month or longer period to form the work plans for individual crews over that period. The output of the algorithms in Section 3.5.4 are air crew schedules (pairings) lasting a few days that typically cover all work in a given month. The bid line problem [214] refers to the problem of sequencing sets of these pairings into monthly work plans for individual crews. An important distinction between these two problems is that in the rostering problem the grouping of runs by day provides important structure to the problem, namely, that no crew can perform two runs on the same day and that the same runs are repeated each day with variations occurring only on weekends. No such structure exists in the bid line problem.

We may describe these problems in the same general terms with which we have characterized other scheduling problems. In this case, the input set of tasks is the set of runs or set of pairings defined over the appropriate time period. Note that with each run or pairing we may associate start and end times and start and end locations. In this case, however, the start and end locations are always the depot. Thus, many of the spatial issues associated with previous problems are no longer relevant. The precise nature of rostering and bid line problems varies significantly depending on the particular real world setting. In addition, these problems have not been as well studied as some of the other scheduling problems. We present two different approaches here that apply to two different well-structured problem settings. We summarize some of the other references that generally do not introduce new mathematical models or algorithms but rather give experience in solving these problems in real-world settings.

A large body of research has been directed toward so-called cyclic scheduling problems [26, 27, 45]. The model used fits into the class of fixed location worker scheduling problems described in Sections 3.3.1 and 3.5.2 and may be applied to certain rostering problems. In this case the time interval, t, is one day and the demand d_t used in constraint (3.16) is the number of runs required for day t. In the simplest form of this problem, the time horizon is one week and the d_t's are equal for weekdays. We wish to cover all requirements with consecutive sets of 5 day periods where the last day (Saturday) and first day (Sunday) are considered consecutive. The associated matrix $A = \{a_{ij}\}$ is illustrated in Fig. 3.16. This particular case of the fixed location worker scheduling problem with all $c_j = 1$ can be easily solved optimally in polynomial time [26, 27] by finding the solution to a set of equations. Several slight variations of this problem can also be efficiently solved. The key property of the matrix A is that ones appear consecutively in either the rows or columns. For the case shown to be solvable by network flow methods in Section 3.5.2, the ones always appear consecutively in the columns of A. Such matrices are known to be totally unimodular [246], which, of course, is related to the fact that they are solvable using network flow methods. The columns in the matrix in Fig. 3.16 have consecutive ones where the last and first columns are defined to be consecutive. Such matrices,

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

Each column represents a possible worker schedule;
the 7 rows correspond to the 7 days of the week;
the "consecutive ones" property found in this matrix
arises from the requirement that workers must work
in consecutive days (Saturday and Sunday are consec-
utive).

Fig. 3.16. Cyclic scheduling matrix for single week.

in general, are not totally unimodular, however, in [45] it is shown that in most cases these problems can be solved in polynomial time by iteratively solving several network flow problems. Reference [44] solves further generalizations using linear programming roundoff heuristics.

The model just described essentially considered all runs in a particular day with equal weight. This assumption is generally not a good one. The typical objective of rostering for European mass transit systems is to balance work requirements among all crews since each crew receives the same pay, which is independent of the particular work performed. In such cases the objective is to produce rosters that are as equitable as possible in terms of total hours worked. In [112], this problem is addressed by using an objective function that minimizes the maximum weight (time) roster. In this case an n-level network, (N, A), is defined where n is the number of days in the time period over which the roster is to extend. We assume that the same set of runs is to be repeated on each day. A node in the network denoted by $\langle i, k\rangle$ represents run i in day (level) k. An arc from node $\langle i, k\rangle$ to node $\langle j, k+1\rangle$ is included in the arc set A if runs i and j can be performed on the same roster in succeeding days (see Fig. 3.17). We denote such an arc by $(i, j,; k)$ and associate 0/1 variables x_{ij}^k with each such arc. A weight w_i is associated with each run i. This weight represents the time duration of the run i. The rostering problem may now be formulated as a multi-level bottleneck assignment problem [112]:

R1:

$$\text{Min } z$$

$$\text{s.t.} \quad \sum_{i:(i,j;k)\in A} x_{ij}^k = 1 \quad \text{for} \quad k = 2, \ldots, m \text{ and all runs } j \tag{3.19}$$

$$\sum_{i:(j,i;k)\in A} x_{ji}^k = 1 \quad \text{for} \quad k = 1, \ldots, m-1 \text{ and all runs } j \tag{3.20}$$

$$z_j^1 = w_j \quad \text{for all runs } j \tag{3.21}$$

$$z_j^k = w_j + \sum_{i:(i,j;k-1)\in A} z_i^{k-1} x_{ij}^{k-1} \quad \text{for } k = 2, \ldots, m \text{ and all runs } j \tag{3.22}$$

$$z_j^n \le z \quad \text{for all runs } j \tag{3.23}$$

$$0 \le x_{ij}^k \le 1 \quad \text{for all } (i, j; k) \in A. \tag{3.24}$$

Constraints (3.19) and (3.20) imply that the 0/1 x_{ij}^k variables will represent a partition of the nodes into paths of length n, where each path passes through exactly one node on each level. Constraints (3.21), (3.22) and (3.23) allow us to interpret z_j^k as the weight of the partial path starting with a node on level l and ending with a node on level k and z as the weight of the maximum weight path.

In [112], it is shown that his problem is NP-complete. A heuristic solution algorithm is presented which involves iteratively solving several bottleneck assignment problems over nodes on level k and $k+1$. Efficient algorithms exist for solving the bottleneck assignment problem [178], which is an assignment problem with a mini–max objective function. Theoretical as well as computational results are given in [112] which indicate that this algorithm is quite effective.

We should point out that neither of the approaches just described provides a complete solution to the rostering problem. The objective function used in the first model is not that realistic and the second model does not treat the issue of days off. Neither model is easily extendable to time horizons larger than one week whereas in most realistic settings a single

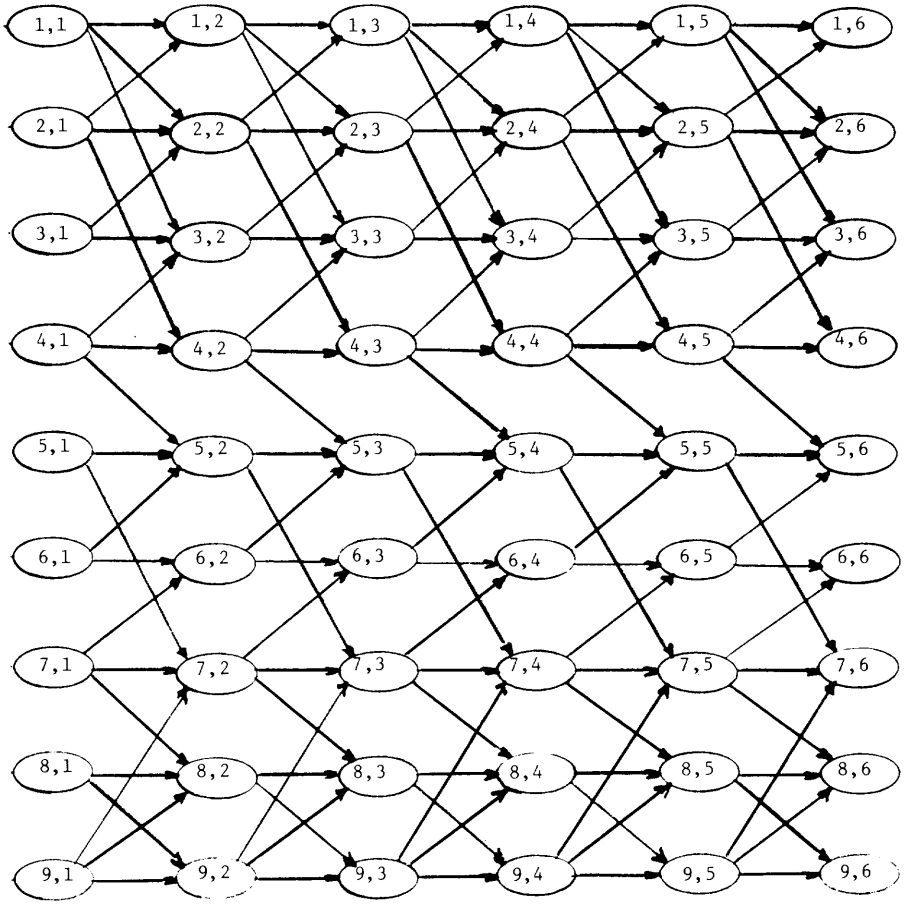


Fig. 3.17. Multi-level network for rostering problem.

roster covers several weeks. We should point out that both of these models could be quite useful when combined with procedures that treated other aspects of the problem.

References [67] and [321] give more comprehensive solutions to the rostering problem. In particular, [321] uses the first model presented in this section as a subprocedure in determining rosters. Other references which give practical experience with solving rostering problems are [341], [397] and [361]. This last reference describes a dynamic system which allows roster changes based on employee absentees.