Chapter 9

Interval Scheduling, Reservations, and Timetabling

9.1 Introduction

Scheduling activities in an environment with resources in parallel may require at times a reservation system. Each activity (i.e., reservation) is supposed to occupy one of the resources for a given time period. Activity j, $j = 1, \ldots, n$, has a duration p_j and has to fit within a time window that is specified by an earliest starting time r_j and a latest termination time d_j. There may or may not be any slack, i.e., either

$$p_j < d_j - r_j$$

or

$$p_j = d_j - r_j.$$

Such a setting is equivalent to the environment described in Chapter 5 which has m machines in parallel and n jobs; job j has a processing time p_j, a release date r_j, a due date d_j, and a weight w_j.

It may not be possible to schedule all n activities and the decision-maker may have to decide which activities to schedule and which ones not. Several objectives may be of interest, e.g., maximizing the number of activities scheduled or maximizing the total utilization of the resources.

M.L. Pinedo, *Planning and Scheduling in Manufacturing and Services*,
DOI: 10.1007/978-1-4419-0910-7_9,

We first consider models with no slack in the time windows for the activities, i.e., the starting time and completion time of an activity have to be equal to its release date and due date. These models are at times referred to as fixed interval, or simply interval scheduling models. We consider also the more general case where there may be a slack in any given time window, i.e., the difference between the due date and the release date may be greater than the duration of the activity.

Reservation systems are ubiquitous. Consider, for example, a system in which customers attempt to reserve hotel rooms for specific time periods. The hotel has to decide which reservations to confirm and which ones not. This problem occurs in many other settings as well (e.g., in a manufacturing setting a customer may want to reserve one or more machines for specific time periods).

A different class of models that is also considered in this chapter is the class of timetabling models. In these models there are typically n activities to be scheduled with an *unlimited* number of identical resources in parallel. However, timetabling problems have an additional dimension: an activity can be done by one of the resources only if specific operators are available at the time of its execution. So an activity can be scheduled on any one of the resources at any time as long as the necessary operators are available during that period. The availability of the operators may be subject to certain constraints; these constraints may have as an effect that certain combinations of activities cannot be done at the same time, even though the resources are available. A typical objective may be to complete all activities in minimum time, i.e., to minimize the makespan. In a more general timetabling problem the timing of activity j may also be constrained by an earliest starting time r_j and a latest completion time d_j.

This chapter considers two types of timetabling problems. The first type of timetabling problem assumes that all operators are identical, i.e., the operators constitute a single homogeneous workforce. The total number of operators available is W and in order to do activity j on one of the resources W_j operators have to be present. If the sum of the people required by activities j and k is larger than W (i.e., $W_j + W_k > W$), then activities j and k may not overlap in time. This type of timetabling is in what follows referred to as timetabling with workforce or personnel constraints.

In the second type of timetabling problem each operator has its own identity and is unique. (An operator may now be equivalent to a specific tool or fixture that is required in order to perform certain activities.) Each activity now requires a specific subset of the operators and/or tools. In order for an activity to be scheduled all the operators or tools in its subset have to be available. Two activities that need the same operator can therefore not be processed at the same time.

This second type of timetabling can occur in many different settings. Consider, for example, a large repair shop for aircraft engines. In order to do certain types of repairs it is necessary to have specific tools, equipment, and operators. A given tool or piece of equipment may be required for certain types of repairs; timetabling may therefore be necessary. A second example of this type of timetabling occurs when meetings have to be scheduled. The operators are now the people who have to attend the meetings and each meeting has to be assigned to a time period in which those who have to attend are able to do so. The meeting rooms correspond to the resources. A third example of this type of timetabling occurs when exams have to be scheduled. Each operator represents a student (or a group of students). Two exams that have to be taken by the same student (or group of students) cannot be scheduled at the same time. The objective is to schedule all the exams within a given time period.

One reason for covering reservation problems and timetabling problems in the same chapter is that both types of problems lead to well-known graph coloring problems. The reservation problem with zero slack and the timetabling problem with operator or tooling constraints are both closely related to a well-known node coloring problem in graph theory. The reservation model with slack is, of course, a generalization of the reservation model without slack. The timetabling problem with workforce constraints cannot be compared that easily to the timetabling problem with operator constraints. In the timetabling problem with workforce constraints there is only one type of operator, but there are a number of them and they are interchangeable. In the timetabling problem with operator or tooling constraints there are several different types of operators, but of each type there is only one. In any case, both the timetabling problem with operator or tooling constraints and the timetabling problem with workforce constraints are special cases of the project scheduling problem with workforce constraints described in Section 4.6.

In this chapter we often make a distinction between the feasibility version of a problem and its optimization version. In the feasibility version we need to determine whether or not a feasible schedule exists; in the optimization version an objective has to be minimized. If no efficient algorithm exists for the feasibility version, then no efficient algorithm exists for the optimization version either.

Throughout this chapter we assume that all data are integer and that preemptions are not allowed.

9.2 Reservations without Slack

In this section we consider the following reservation model. There are m resources in parallel and n activities. Activity j has a release date r_j, a due date d_j, and a weight w_j. As stated before, all data are integer. The fact that there is no slack between release date and due date implies that

$$p_j = d_j - r_j.$$

If we decide to do activity j, then it has to be done within the specified time frame. However, it may be the case that activity j cannot be done by just any one of the m resources; it may have to be done by a resource that belongs to a specific subset M_j of the m resources. When all activities have equal weights, the objective is to maximize the number of activities done. In contrast, when the activities have different weights, the objective is to maximize the weighted number of activities scheduled. A weight is often equivalent to a profit that is made by doing the activity. In a more general model the weight of activity j may also depend on the resource to which it is assigned, i.e., the weight is w_{ij} (i.e., the profit depends on the activity as well as on the resource to which the activity is assigned).

Example 9.2.1 (A Car Rental Agency). Consider a car rental agency with four types of cars: subcompact, midsize, full size and sport-utility. Of each type there are a fixed number available. When customer j calls to make a reservation for p_j days, he may, for example, request a car of either one of two types and will accept the price quoted by the agency for either type. The set M_j for such a customer includes all cars belonging to the two types. The profit made by the agency for a car of type i is π_i dollars per day. So, the weight of this particular reservation is $w_{ij} = \pi_i p_j$.

However, if customer j specifically requests a subcompact and all subcompacts have been rented out, the agency may decide to give him a midsize for the price of a subcompact in order not to lose him as a customer. The set M_j includes subcompacts as well as midsizes (even though customer j requested a subcompact), but the agency's daily profit is a function of the car as well as of the customer, i.e., π_{ij} dollars per day, since the agency gives him a larger car at a lower price. The weight is $w_{ij} = \pi_{ij} p_j$.

Most reservation problems can be formulated as integer programs. Time is divided in periods or slots of unit length. If the number of slots is fixed, say H, then the problem is referred to as an H-slot problem. Assume, for the time being, that the activity durations are equal to 1 and let J_t denote the set of activities that need a resource in slot t, i.e., during period $[t-1,\ t]$. If x_{ij} denotes a binary variable that assumes the value 1 if activity j is assigned to resource i and 0 otherwise, then the following constraints have to be satisfied:

$$\sum_{i=1}^{m} x_{ij} \le 1 \qquad j = 1, \ldots, n$$

$$\sum_{j \in J_t} x_{ij} \le 1 \qquad i = 1, \ldots, n, \quad t = 1, \ldots, H.$$

The first set of constraints ensures that every activity is assigned to at most one resource and the second set ensures that a resource is not assigned to more than one activity in any given slot.

The easiest version of the reservation problem is a feasibility problem: does there exist an assignment of activities to resources with every activity being assigned to a resource? A slightly harder version of this feasibility problem would be the following: does there exist an assignment of activities to resources with activity j being assigned to a resource belonging to a given subset M_j? It turns out that this problem is still relatively easy and therefore left as an exercise.

In the optimization version of the reservation problem the objective is to maximize the total profit

$$\sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij},$$

where the weight w_{ij} is equivalent to a profit associated with assigning activity j to resource i. Some special cases of this optimization problem can actually be solved in polynomial time. For example, consider again the case with all n activities having a duration equal to 1, i.e., $p_j = 1$ for all j, and assume arbitrary resource subsets M_j and arbitrary weights w_{ij}. Each time slot can be considered as a separate subproblem that can be solved as an independent assignment problem (see Appendix A).

Another version of the reservation model that allows for an efficient solution assumes arbitrary durations, identical weights (i.e., $w_{ij} = 1$ for all i and j), and each set M_j consisting of all m resources (i.e., the m resources are identical). The durations, the starting times (release dates) and the completion times (due dates) are arbitrary integers and the objective is to maximize the number of activities assigned. This problem cannot be decomposed into a number of independent subproblems (one for each time slot), since the durations of the different activities may overlap. However, it can be shown that the following relatively simple algorithm maximizes the total number of activities. In this algorithm the activities are ordered in increasing order of their release dates, i.e.,

$$r_1 \le r_2 \le \cdots \le r_n.$$

Set J denotes the set of activities already selected.

Algorithm 9.2.2 (Maximizing Number of Activities Assigned).

Step 1.

Set $J = \emptyset$ and $j = 1$.

Step 2.

If a resource is available at time r_j, then assign activity j to that resource; include activity j in J, and go to Step 4.

Otherwise go to Step 3.

Step 3.

Let j^* be such that

$$C_{j^*} = \max_{k \in J}(C_k) = \max_{k \in J}(r_k + p_k).$$

If $C_j = r_j + p_j > C_{j^*}$, do not include activity j in J and go to Step 4.

Otherwise, delete activity j^* from J, assign activity j to the resource freed and include activity j in J.

Step 4.

If $j = n$, STOP,

otherwise set $j = j + 1$ and return to Step 2.

Another version of this reservation model with zero slack, arbitrary durations, equal weights, and identical resources is also of interest. Assume there are an unlimited number of identical resources in parallel and all activities have to be assigned. However, the assignment must be done in such a way that a minimum number of resources is used. This problem is, in a sense, a dual of the problem discussed before. It turns out that minimizing the number of resources when all activities have to be done is also an easy problem.

It can be solved as follows. Again, the activities are ordered in increasing order of their release dates, i.e., $r_1 \leq r_2 \leq \cdots \leq r_n$. First, activity 1 is assigned to resource 1. The algorithm then proceeds with assigning the activities, one by one, to the resources. Suppose that the first $j-1$ activities have been assigned to resources $1, 2, \ldots, i$. Some of these activities may have been assigned to the same resource. So $i \leq j - 1$. The algorithm then takes the next activity from the list, activity j, and tries to assign it to a resource that already has been utilized before. If this is not possible, i.e., resources $1, \ldots, i$ are all busy at time r_j, then the algorithm assigns activity j to resource $i+1$. The number of resources utilized after activity n has been assigned is the minimum number of resources required.

This last problem turns out to be a special case of a well-known node coloring problem in graph theory. Consider n nodes and let node j correspond to activity j. If there is an (undirected) arc (j, k) connecting nodes j and k, then the processing of activities j and k overlap in time and nodes j and k cannot be given the same color. If the graph can be colored with m (or less) colors, then a feasible schedule exists with m resources. This node coloring problem, which is a feasibility problem that is NP-hard, is more general than the reservation problem in which the number of resources used is minimized. This node coloring problem is actually equivalent to the timetabling problem with operator or tooling constraints described in Section 9.5. The node coloring problem establishes the links between interval scheduling, reservations, and timetabling.

9.3 Reservations with Slack

In the previous section we assumed that there was no slack between the release date and the due date of each activity, i.e.,

$$p_j = d_j - r_j.$$

A more general version of the reservation model allows for slack in the time window specified, i.e.,

$$p_j \leq d_j - r_j.$$

We again consider first the special case where all release dates and due dates are integer and all processing times are equal to one. The weights of all activities are identical and all M_j sets consist of all m resources. This case is trivial since a schedule can be constructed progressively in time and the maximum number of activities can be assigned.

The more general problem with non-identical durations does not have an easy solution. Maximizing the weighted number of activities assigned is NP-hard, so it is unlikely that there exists an efficient algorithm that would guarantee an optimal solution. We have to rely on heuristics.

The following heuristic is basically a composite dispatching rule as described in Appendix C. It requires, as a first step, the computation of a number of statistics. Let ν_{it} denote the number of activities that may be assigned to resource i during interval $[t-1, t]$. This factor thus corresponds to a potential utilization of resource i in time slot t. The higher this number is, the more flexible resource i is in this time slot. A second factor is the number of resources to which activity j can be assigned, i.e., the number of resources in set M_j, which is denoted by $|M_j|$. The larger this number, the more flexible activity j is. Define for activity j a priority index I_j that is a function of w_j/p_j and $|M_j|$, i.e.,

$$I_j = f(w_j/p_j, |M_j|).$$

The higher w_j/p_j and the smaller $|M_j|$, the lower the index. The activities can now be ordered in increasing order of their indices, i.e.,

$$I_1 \leq I_2 \leq \cdots \leq I_n.$$

The algorithm takes the activity with the lowest index among the remaining activities and attempts to assign it to one of the resources, starting with the resource with the least flexible time intervals. If the activity needs a resource over the period $[t, t+p_j]$, then the selection of resource i depends on a function of the factors $\nu_{i,t+1}, \ldots, \nu_{i,t+p_j}$, i.e., $g(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j})$. Examples of such functions are:

$$g(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j}) = \Big(\sum_{l=1}^{p_j} \nu_{i,t+l}\Big)/p_j;$$

$$g(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j}) = \max(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j}).$$

The heuristic attempts to assign the activity to a resource in a period that has the lowest possible $g(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j})$ value. This one-pass heuristic can be summarized as follows.

Algorithm 9.3.1 (Maximizing Weighted Number of Activities).

Step 1.

Set $j = 1$

Step 2.

Take activity j *and select, among the resources and time slots available, the resource and time slots with the lowest* $g(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j})$ *rank.*

Discard activity j *if it cannot be assigned to any machine at any time.*

Step 3.

If $j = n$ *STOP,*

otherwise set $j = j + 1$ *and return to Step 2.*

The next example illustrates this heuristic.

Example 9.3.2 (Maximizing Weighted Number of Activities). Consider seven activities and three resources.

activities	1	2	3	4	5	6	7
p_j	3	10	9	4	6	5	3
w_j	2	3	3	2	1	2	3
r_j	5	0	2	3	2	4	5
d_j	12	10	20	15	18	19	14
M_j	{1,3}	{1,2}	{1,2,3}	{2,3}	{1}	{1}	{1,2}

Consider the index function

$$I_j = f(w_j/p_j, M_j) = \frac{|M_j|}{w_j/p_j}.$$

The indices for the activities can be computed and are tabulated below.

activities	1	2	3	4	5	6	7
I_j	3	6.67	9	4	6	2.5	2

The factors ν_{it} are tabulated below.

slot t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ν_{1t}	1	1	3	3	4	6	6	6	6	6	5	5	4	4	3	3	3	3	2	1
ν_{2t}	1	1	2	3	3	4	4	4	4	4	3	3	3	3	2	1	1	1	1	1
ν_{3t}	0	0	1	2	2	3	3	3	3	3	3	3	2	2	2	1	1	1	1	1

Applying the algorithm using the function

$$g(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j}) = \Big(\sum_{l=1}^{p_j} \nu_{i,t+l}\Big)/p_j$$

yields the schedule depicted in Figure 9.1.

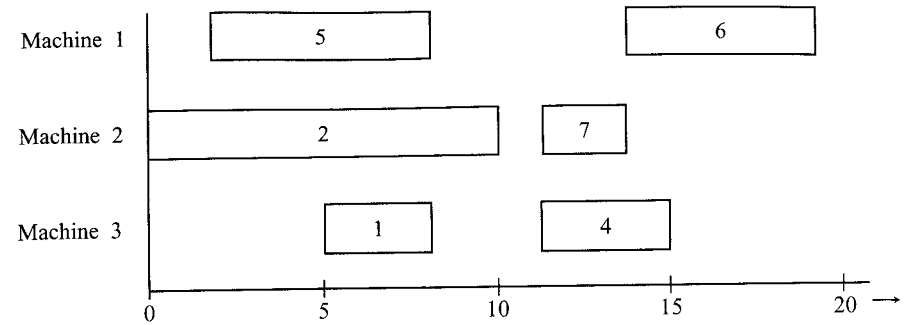

Fig. 9.1. Schedule in Example 9.3.2

Activity	Resource	Period
7	2	11-14
6	1	14-19
1	3	5-8
4	3	11-15
5	1	2-8
2	2	0-10

It turns out that activity 3 (the last activity) does not fit into the schedule. However, in the optimal schedule all activities are assigned (activity 7 starts with resource 1 at time 10 and activity 3 starts with resource 2 at time 11). So the heuristic yields in this case a suboptimal solution.

The function

$$f(w_j/p_j, |M_j|) = \frac{|M_j|^2}{w_j/p_j},$$

yields the same schedule as the one described above (although the sequence in which the activities are put on the resources is slightly different, the final result is still the same).

The function

$$f(w_j/p_j, |M_j|) = \frac{\sqrt{|M_j|}}{w_j/p_j}$$

yields a schedule with activity 3 assigned. However, now activity 5 ends up unassigned. Noting that $w_5 = 1$ and $w_3 = 3$, this last schedule is actually better than the previous one, but still not optimal.

9.4 Timetabling with Workforce Constraints

Consider now an infinite number of identical resources in parallel. There are n activities and all activities have to be done. Activity j can be done by or on any one of the resources, but once the activity has started it has to proceed without interruption until it is completed. There is a workforce that consists of a single pool of W_1 identical operators. In order to do activity j it is necessary to have W_{1j} operators at hand. If the sum of the requirements of activities j and k is larger than W_1, i.e., $W_{1j} + W_{1k} > W_1$, then activities j and k cannot be done at the same time. Actually, the sum of the requirements of any set of activities that are done at any given point in time may not exceed W_1. It is clear that this problem is a special case of the workforce constrained project scheduling problem described in Section 4.6.

Such a model with workforce constraints can be used for workforce scheduling applications. However, workforce scheduling problems in general tend to be more complicated and will be discussed in more detail in Chapter 13.

Example 9.4.1 (Project Management in the Construction Industry). A contractor has to complete n activities. The duration of activity j is p_j and it requires a crew of size W_{1j}. The activities are not subject to precedence constraints. The contractor has W_1 workers at his disposal and his objective is to complete all n activities in minimum time.

Consider now the following special case of the workforce constrained timetabling problem with the number of available resources being unlimited. All activities have now the same duration. The activities are not subject to precedence constraints but are subject to workforce constraints and the objective is to minimize the makespan. This problem may be regarded as a discrete counterpart of a famous combinatorial problem known as the *bin packing* problem. In the bin packing problem each bin has capacity W_1 and activity j is equivalent to an item of size W_{1j}. Each bin corresponds to one time slot and the items packed in one bin correspond to the activities done in that time slot. The objective is to pack all the items in a minimum number of bins. This problem has many applications in practice.

Example 9.4.2 (Exam Scheduling). All the exams in a community college have the same duration. The exams have to be held in a gym with W_1 seats. The enrollment in course j is W_{1j} and all W_{1j} students have to take the exam at the same time. The goal is to develop a timetable that schedules all n exams in minimum time.

This workforce constrained scheduling problem with all activities having the same duration is known to be NP-hard (even in the absence of precedence constraints). However, a number of heuristics have been developed that perform reasonably well.

The *First Fit (FF)* heuristic first orders the activities (items) in an arbitrary way. The slots (bins) are numbered $1, 2, 3, \ldots$ The procedure starts at the beginning of the activity list and checks whether the activity fits in slot 1. If it fits, it is inserted there. Otherwise, the procedure checks whether it fits in slot 2, and so on. It has been shown that for any instance of this problem

$$C_{\max}(FF) \leq \frac{17}{10} C_{\max}(OPT) + 2,$$

where $C_{\max}(FF)$ denotes the makespan under the FF rule and $C_{\max}(OPT)$ denotes the makespan under a (possibly unknown) optimal rule. It is easy to find instances for which

$$\frac{C_{\max}(FF)}{C_{\max}(OPT)} = \frac{5}{3}.$$

Example 9.4.3 (Application of the FF Heuristic). Let $W_1 = 2100$. There are 18 activities.

activities	$1, \ldots, 6$	$7, \ldots, 12$	$13, \ldots, 18$
W_{1j}	301	701	1051

Under the optimal schedule the makespan $C_{\max}$ is 6 and under the FF schedule the makespan is equal to 10. The optimal schedule assigns to each one of the six slots three activities: one of 301, one of 701 and one of 1051.

The FF rule assigns six activities of 301 to slot 1. To each one of the next three slots it assigns two activities of 701. To each one of the last six slots it assigns a single activity of 1051.

This example shows that an FF schedule may be far from optimal when the activities initially are listed in a haphazard way. If the activities are ordered initially in a clever way the First Fit heuristic may perform better. The worst case performance of the next heuristic, that is based on this idea, is significantly better.

The *First Fit Decreasing (FFD)* heuristic first orders the activities in decreasing order of W_{1j}. The slots are again numbered $1, 2, 3$, and so on. The procedure starts at the beginning of the activity list and checks whether the activity fits in slot 1. If it fits, it is inserted. Otherwise, the procedure checks whether the activity fits in slot 2, and so on.

It has been shown that for any instance of the problem

$$C_{\max}(FFD) \leq \frac{11}{9} C_{\max}(OPT) + 4,$$

where $C_{\max}(FFD)$ denotes the makespan under the FFD rule. There are instances for which

$$\frac{C_{\max}(FFD)}{C_{\max}(OPT)} = \frac{11}{9}.$$

The following example shows how this worst case bound can be attained.

Example 9.4.4 (Application of the FFD Heuristic). Let $W_1 = 1000$.

activities	$1, \ldots, 6$	$7, \ldots, 12$	$13, \ldots, 18$	$19, \ldots, 30$
W_{1j}	501	252	251	248

Under the optimal schedule the makespan $C_{\max}$ is 9 and under the FFD schedule the makespan is 11. The optimal schedule assigns to each one of the first six slots three activities: one of 501, one of 251 and one of 248. To each one of the remaining three slots it assigns four activities: two of 252 and two of 248.

The FFD rule assigns to each one of the first six slots one activity of 501 and one activity of 252. To the next two slots it assigns three activities of 251. To each one of the last three slots it assigns four activities of 248.

The two heuristics described above can be applied with some minor modifications to cases where the activities have different release dates. If the activities have due dates and the objective is the minimization of a due date related penalty function, then a different heuristic is required. If the activities have deadlines and the objective is to find a feasible schedule, then also additional modifications are needed.

9.5 Timetabling with Operator or Tooling Constraints

In the previous section we considered models with W_1 identical operators. The operators were basically interchangeable.

In what follows the operators are not identical. Each operator is unique, has his own identity and his own skill. An operator in this model may actually be equivalent to a specific piece of machinery, a fixture, or a tool. An activity either needs or does not need any given operator or tool. Each activity needs for its execution a specific set of different operators and/or tools.

It is easy to see that this problem is also a special case of the workforce constrained project scheduling problem discussed in Section 4.6. It is a project scheduling problem with workforce constraints and without precedence constraints. There are now N_p different pools of operators and each pool of operators consists of a single operator, i.e., $W_\ell = 1$ for $\ell = 1, \ldots, N_p$.

The difference between timetabling with operator or tooling constraints and timetabling with workforce constraints is significant. In one sense the model in the previous section is more restrictive (there is only one type of operator, i.e., $N_p = 1$), and in another sense it is more general (there are a number of that type of operator available, i.e., $W_1 > 1$).

Each activity now requires one or more different operators or tools. If two activities require the same operator, then they cannot be done at the same time. In the feasibility version of this problem, the goal is to find a schedule or timetable that completes all n activities within the time horizon H. In the optimization version, the objective is to do all the activities and minimize the makespan.

Throughout this section we assume that all activity durations are equal to 1. Even this special case with all activity durations being equal does not have an easy solution. In what follows we first focus on the feasibility version when all durations are equal to 1. Finding for this case a conflict-free timetable is structurally equivalent to the node coloring problem described at the end of Section 9.2. In the node coloring problem a graph is constructed by representing each activity as a node. Two nodes are connected by an arc if the two activities require the same operator(s). The two activities, therefore, cannot be scheduled in the same time slot. If the length of the time horizon is H time slots, then the question is: can the nodes in the graph be colored with H different colors in such a way that no two connected nodes receive the same color? This is a feasibility problem. The associated optimization problem is to determine the minimum number of colors needed to color the nodes of the graph in such a way that no two connected nodes have the same color. This minimum number of colors is referred to as the chromatic number of the graph and is equivalent to the makespan in the timetabling problem.

The optimization version of the timetabling problem with all durations being equal to 1 is closely related to the zero slack reservation problem with *arbitrary* durations described at the end of Section 9.2. That this reservation problem (with the number of resources being minimized) is not equivalent to the timetabling problem but rather a special case can be shown as follows: two activities that need the same operator in the timetabling problem are equivalent to two activities that have an overlapping time slot in the reservation problem. If two activities in the reservation problem have an overlapping time slot, then the two nodes are connected. Each color in the coloring process represents a resource and minimizing the number of colors is equivalent to minimizing the number of resources in the reservation problem. That the reservation problem is a special case follows from the fact that the time slots required by an activity in a reservation problem are *adjacent*. However, it may not be possible to order the tools in the timetabling problem in such a way that the tools required for each activity are adjacent to one another. It is this adjacency property that makes the reservation problem easy, while the lack of adjacency makes the timetabling problem with operator constraints hard.

There are a number of heuristics for this timetabling problem with durations equal to 1. In this section we describe only one such procedure. First some graph theory terminology is needed: the degree of a node is the number of arcs connected to a node; in a partially colored graph, the saturation level of a node is the number of differently colored nodes already connected to it. In the coloring process, the first color to be used is labeled Color 1, the second Color 2, and so on.

Algorithm 9.5.1 (Graph Coloring Heuristic).

Step 1.

Arrange the nodes in decreasing order of their degree.

Step 2.

Color a node of maximal degree with Color 1.

Step 3.

Choose an uncolored node with maximal saturation level.

If there is a tie, choose any one of the nodes with maximal degree in the uncolored subgraph.

Step 4.

Color the selected node with the color with the lowest possible number.

Step 5.

If all nodes are colored, STOP. Otherwise go to Step 3.

Example 9.5.2 (Application of the Graph Coloring Heuristic). Gary, Hamilton, Izak and Reha are university professors attending a national conference. During this conference seven one hour meetings have to be scheduled in such a way that each one of the four professors can be present at all the meetings he has to attend. The goal is to schedule all seven meetings in a single afternoon between 2 p.m. and 6 p.m.

meetings	1	2	3	4	5	6	7
Gary	1	0	0	1	1	0	1
Hamilton	1	1	1	0	0	0	0
Izak	0	0	1	0	1	1	0
Reha	1	0	1	1	1	0	0

This problem can be transformed into a timetabling problem with operator constraints by assuming that the seven meetings are activities and the four professors are operators. Consider the following set of data.

activities	1	2	3	4	5	6	7
operator 1	1	0	0	1	1	0	1
operator 2	1	1	1	0	0	0	0
operator 3	0	0	1	0	1	1	0
operator 4	1	0	1	1	1	0	0

If the activities are regarded as nodes, then their degrees can be computed (see Figure 9.2).

activities (nodes)	1	2	3	4	5	6	7
degree	5	2	5	4	5	2	3

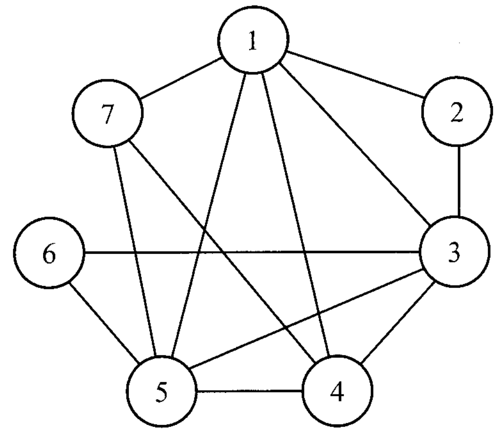

Fig. 9.2. Graph in Example 9.5.2

Based on the degrees, activity 5 may be colored first, say with the color red (Color 1). The saturation levels of all nodes connected to node 5, i.e., nodes 1, 3, 4, 6, 7, are equal to 1. Of these nodes, nodes 1 and 3 have the highest degrees. Color node 3 blue (Color 2). The saturation levels and the degrees in the uncolored subgraph are presented in the table below.

activities (nodes)	1	2	3	4	5	6	7
saturation level	2	1	-	2	-	2	1
degree	3	1	-	2	-	0	2

Based on these numbers node 1 is selected as the node to be colored next, say yellow (Color 3). Node 4 is selected after that and colored green (Color 4). Node 7 follows and is colored with the color that has the lowest number, Color 2 (blue). Node 6 is colored last and colored yellow. Since four colors were needed to color the graph, the makespan of the corresponding schedule is equal to 4. It can easily be seen that this schedule is optimal. Both operators 1 and 4 are needed for 4 activities.

To see why it makes sense to schedule the activity with the highest degree first, consider scheduling the activity with the lowest degree first. Activities 2, 6, and 7 then have to be done in the same time slot. However, activities 4, 1, 3, and 5 are scheduled afterwards in four different time slots and the makespan is 5. If the high degree activities are not scheduled early on, they often end up requiring new colors at the end of the process.

The next example illustrates the relationship between the reservation problem and the timetabling problem.

Example 9.5.3 (Timetabling Compared to Reservations). Consider the following timetabling problem.

activities	1	2	3	4	5	6	7
p_j	1	1	1	1	1	1	1
operator 1	1	0	1	1	1	0	1
operator 2	1	1	1	0	0	0	0
operator 3	0	0	1	0	1	1	0
operator 4	1	0	1	1	1	0	0

Note that the only difference between this example and the previous one is that activity 3 now needs all four operators. The operators can be transformed into time slots as follows. Operators 3 and 4 are equivalent to time slots 1 and 2 and operators 1 and 2 are equivalent to time slots 3 and 4. The time slots required by each activity are now contigious (i.e., adjacent) and the problem is equivalent to a reservation problem.

Consider now a more general timetabling model with all activities again having duration 1. There are a number of feasible slots. However, there is an aversion cost c'_{jt} for assigning activity j to slot t. There is also a proximity cost for scheduling two conflicting activities (that require the same operator) too close to one another. The penalty for scheduling two conflicting activities τ slots apart is $\psi(\tau)$, where $\psi(\tau)$ is decreasing in τ. The objective function is, for any schedule, the sum of these two costs for every occurrence. The following multi-pass heuristic is applicable to this more general timetabling problem.

Algorithm 9.5.4 (Minimizing Timetabling Costs).

Step 1.

Take activity j from the set of activities not yet scheduled.

Step 2.

Find all feasible slots where activity j can be assigned, i.e., where there is no operator conflict.

If no such slot is found go to Step 4.

Step 3.

For each feasible slot, compute the increase in the cost function (aversion as well as proximity costs);

Assign activity j to the slot with the lowest cost.

Go to Step 1.

Step 4.

Activity j conflicts in every possible slot with other activities.

Find the slots in which activity j can be scheduled by rescheduling all activities that conflict with j.

If there are no such slots go to Step 6. Otherwise go to Step 5.

Step 5.

For each slot calculate the cost of rescheduling all conflicting activities.

Assign activity j to that slot with the lowest rescheduling cost.

Step 6.

If there are no slots for which conflicting activities can be rescheduled without conflict, count for each slot the number of activities that cannot be rescheduled.

Assign activity j to that slot with the least number of such conflicts.

Reschedule as many conflicts as possible and bump the others back on the list of unscheduled activities.

If the number of times activity k is bumped by the same activity j reaches N, then activity k is dropped and considered unschedulable.

The last step of the algorithm can be viewed as a bounded backtracking mechanism that allows it to reconsider earlier decisions. If a pair of activities is difficult to schedule, then they most likely will bump each other relatively early in the backtracking process. The next example illustrates the manner in which the backtracking mechanism works when the algorithm is applied to the more specific model described earlier in this section.

Example 9.5.5 (Minimizing Timetabling Costs). Consider the instance discussed in Example 9.5.2. Assume that the number of feasible slots is 4 (this implies that the algorithm will either find the optimal schedule or it will conclude that there is no feasible schedule). Assume that all aversion costs and proximity costs are zero.

Since Algorithm 9.5.4 does not specify the order in which the unscheduled activities are taken, we assume here that the activities are going to be considered in the order

$$2, 6, 7, 4, 1, 3, 5.$$

Going through Steps 1, 2, and 3 a number of times results in activities 2, 6, and 7 being done in slot $[\,0,1\,]$, activity 4 in slot $[\,1,2\,]$, activity 1 in $[\,2,3\,]$, and activity 3 in $[\,3,4\,]$. However, when the algorithm attempts to insert the last activity, activity 5, in one of the four slots, then there are conflicts in each one of them. If activity 5 is put in the slot of activity 4, then activity 4 has to be rescheduled, but activity 4 cannot be rescheduled in any of the other three slots. The same thing happens if activity 5 is assigned to the third or fourth slot. So the first slot remains to be checked. If activity 5 is inserted in the first slot, then activities 6 and 7 are in conflict and have to be rescheduled. Activity 6 can be scheduled together with activity 4 in the second slot and activity 7 can be scheduled together with activity 3 in the fourth slot. So the optimal schedule has been obtained.

The University of California at Berkeley enrolls about 30,000 students in over 80 academic departments. Each semester, all departments provide the scheduling office an estimated enrollment, a requested meeting time, and special requirements (e.g., with regard to audiovisual equipment) for each section of each course. The scheduling office must assign 4000 classes to about 250 classrooms. The office consists of three schedulers (one for each academic unit) and one supervisor.

The assignment has to take a number of objectives into consideration. A room with fewer seats than students is undesirable, as is a room that is much too large. In addition, some courses require special equipment. The location of the room is also important. From a professor's point of view, it is nice to have a room that is close to his or her office. From a students' point of view it is convenient to have consecutive classes close together.

It is not easy to state a formal objective for this optimization problem, since there are often no clear priorities. For example, if there is no room to accomodate both Chemistry 201 and Russian 101 at the same time, then it is not easy to make a choice based on some general principle. Fortunately, some policy guidelines had been established by a campus committee. The policy guidelines are based on standard time patterns for offering courses. The nine-hour day, starting at 8 a.m., is partitioned into nine one hour blocks and, at the same time, also into 6 one and one-half-hour blocks. Classes may be scheduled only for whole blocks. Certain time blocks are defined as prime time and departments may not request more than 60% of their courses during prime time. Standard courses have priority over nonstandard courses. These policies provide some means of deciding which courses should not be assigned during overloaded time blocks. However, they still do not provide a watertight method for resolving conflicts between departments.

This classroom assignment problem can be formulated as a large $0-1$ integer programming problem. The objective function of this integer program is rather complicated and contains many terms. First, there is a penalty associated for not assigning a class at all. By making this penalty large relative to the other terms in the objective, the total number of unassigned classes is minimized. The cost terms in the objective associated with the assignment variables account for distances, overutilized facilities, and empty seats.

Since the integer program is huge (approximately 500,000 variables and 30,000 constraints), it is solved heuristically even though the problem does not have to be solved in real time. The heuristic works in a sequential manner and is based on the principle of always solving the hardest remaining subproblem next.

Some notation is needed in order to describe the heuristic. Let J denote the set of all classes to be scheduled. Let t denote a timeslot and let J_t denote the set of all classes for timeslot t. Let M denote the set of all classrooms and let M_j denote the set of all classrooms that can accomodate class j.

The heuristic can be summarized in the following four steps.

Algorithm 9.6.1 (Room Assignment Heuristic).

Step 1. *(Select Time Slot)*

Select, among time slots not yet considered, slot t with the smallest supply/demand ratio.

Step 2. *(Greedy Algorithm)*

Rank all classes j in J_t in decreasing order of class size.

Go in a single pass through the list of classes, and assign class j to the (still vacant) room in M_j with lowest cost.

Step 3. *(Improvement Phase)*

Rank all classes j in J_t in decreasing order of current cost.

Go in a single pass through the list of classes and do the following: If class j is not assigned, find all feasible interchanges in which class j moves into an occupied room displacing the assigned class k into a vacant room; if this set is not empty, make the interchange with maximum cost reduction.

If class j is assigned, find the set of feasible assignment interchanges for class j that reduce total cost; if this set is not empty, apply the interchange with the maximum cost reduction.

Step 4. *(Stopping Criterion)*

If Step 3 has resulted in a reduction of the total cost return to Step 3; otherwise delete from the unscheduled list all classes scheduled for the current time slot.

If all time slots have been considered STOP, otherwise go to Step 1.

The heuristic has proven to be a very fast method for generating near-optimal solutions. Combining the rule of "selecting the hardest subproblem next" with a dynamic recalculation of the costs of wasted resources seems very effective. The decision support system is designed so that it is easy to use interactively and it is flexible enough to accomodate future policy modifications without extensive reprogramming.

The system is used in the following manner. Approximately six months before the start of the semester, departments submit room request forms that list all classes scheduled for the semester. Within a couple of weeks, a preliminary schedule is generated, showing those classes that could not be assigned to rooms. Departments then submit revised requests and negotiate with the scheduling office about possible pre-assignments. The system is then run again, with the unchanged standard lectures that had already been assigned flagged as pre-assigned. The resulting set of assignments is then published in time for pre-enrollment. Using the system, the scheduling office is able to complete its part of the cycle several weeks earlier than with the original manual procedure.

system has been used for a number of years. A number of factors contributed to the success of the system. The most important one is a flexible user interface. While an optimization model is being used, its behavior is easily altered to explore different trade-off strategies. Furthermore, the system is designed so that partial solutions, in the form of easily generated pre-assignments, can be incorporated, while allowing the heuristic to generate a solution for the remaining problem. Special needs that were not anticipated when the model was designed can be accomodated and the schedulers can evaluate their own heuristics.

This classroom assignment problem is very similar to the reservation problems with slack and without slack described in Sections 9.2 and 9.3. The meeting rooms are the resources and activity j can only be assigned to a resource belonging to the subset M_j. However, in the classroom assignment problem *all* activities have to be scheduled. The multi-pass heuristic implemented here is clearly more sophisticated than the one-pass heuristic described in Algorithm 9.3.1.

9.7 Discussion

Even though the four sections of this chapter deal with four different types of scheduling and timetabling problems, it is not hard to imagine real world scheduling problems that have all the features discussed in this chapter, i.e., release dates and due dates (with or without slack), workforce constraints as well as operator constraints. The objective may also be a combination of the minimization of the makespan and the maximization of the (weighted) number of activities done. The separate analyses of these four different aspects give an indication of how hard real world problems can be.

The interval scheduling models and reservation models discussed in this chapter are relatively simple. They only give a flavor of the thinking behind these problems. A company that has to deal with these types of problems typically relies on models that are significantly more complicated. First of all, the models have to be dynamic rather than static. Calls for reservations come in continuously and the decision making process embedded in the company's systems depends heavily on forecasts of future demands. Second, the reservation systems depend on the existing pricing structure and the pricing structure depends on the current occupancy as well as on forecast demand. A significant amount of research has been done recently on reservation models that include a pricing mechanism. These models are beyond the scope of this book.

A natural generalization of timetabling problems with operator constraints and timetabling problems with workforce constraints is the following: Suppose there are a number of different types of operators, say N_p. A limited number W_ℓ of operators of type ℓ, $\ell = 1, \ldots, N_p$, are available. Doing activity j requires $W_{\ell j}$ operators of type ℓ. The activities have to be scheduled subject to the operator availability constraints. The FF and FFD rules can be adapted to this situation with multiple types of operators. This problem is basically equivalent to the problem discussed in Section 4.6 without precedence constraints.

In this chapter we have not considered any preemptions. When the durations of the activities are not equal to 1, preemptions may improve the performance measures. However, heuristics for preeemptive models may be very different from those for nonpreemptive models.

Exercises

9.1. Consider the following reservation problem with 10 activities and zero slack. There are three identical resources in parallel.

activities	1	2	3	4	5	6	7	8	9	10
p_j	6	1	4	2	3	3	6	2	1	3
r_j	2	7	5	2	1	0	4	8	0	0
d_j	8	8	9	4	4	3	10	10	1	3

(a) Apply Algorithm 9.2.1 to find the schedule with the maximum number of activities done.

(b) Find the schedule that maximizes the total amount of processing (i.e., the sum of the durations of the activities done).

(c) What is the minimum number of resources needed to satisfy the total demand?

9.2. Consider a car rental agency where the following 10 reservations have been made.

reservations	1	2	3	4	5	6	7	8	9	10
p_j	3	1	4	2	3	1	1	3	3	3
r_j	1	3	0	1	0	0	2	3	4	2
d_j	4	4	4	3	3	1	3	6	7	5

Determine the minimum number of cars needed to satisfy the demand.

9.3. Consider the following instance with eight activities and three resources.

activities	1	2	3	4	5	6	7	8
p_j	4	3	10	9	4	6	5	3
w_j	3	2	3	3	2	1	2	3
r_j	8	5	0	2	3	2	4	5
d_j	12	12	10	20	15	18	19	14
M_j	{2}	{1,3}	{1,2}	{1,2,3}	{2,3}	{1}	{1}	{1,2}

appropriate index functions I_j and $g(\nu_{i,t+1},\ldots,\nu_{i,t+p_j})$ and apply Algorithm 9.3.1.

(b) Is the solution obtained optimal? If not, can you modify the algorithm to obtain an optimal solution?

9.4. Consider a hotel with two types of rooms: suites and regular rooms. There are n_1 suites and n_2 regular rooms. If someone wants a suite, then the hotel makes a profit of w_1 dollars per night. If someone wants a regular room, the hotel makes a profit of w_2 dollars per night ($w_2 < w_1$). If a person wants a regular room and all regular rooms are taken, then the hotel can put that person up in a suite. However, the hotel makes then only w_2 dollars per night.

(a) Assume that the hotel cannot ask the guest to change rooms in the middle of his stay; once assigned to a room or suite the guest will stay there until he leaves (i.e., preemptions are not allowed). Explain how this problem fits the framework described in Section 9.2.

(b) Assume that the hotel can ask a guest to change rooms in the middle of his stay. How does this affect the problem? Can the hotel rent out more rooms this way? Illustrate your answer with a numerical example.

9.5. Design a heuristic for the reservation problem described in part (a) of the previous exercise. If guest j requests a regular room then the weight is $p_j w_2$ (even if he is put up in a suite); if he requests a suite his weight is $p_j w_1$. Apply your heuristic to the data set below.

guests	1	2	3	4	5	6	7	8	9
p_j	4	3	10	9	4	6	5	3	4
type	1	1	2	2	2	2	2	2	2
r_j	8	5	0	2	3	2	4	5	7
d_j	12	12	10	15	15	15	13	14	13

There are 3 regular rooms and 1 suite. The value $w_1 = 3$ and the value $w_2 = 2$. Check if your heuristic yields the optimal solution.

9.6. Consider a hotel with m identical rooms. Arriving guests request either a single room or two rooms for a certain period. If the hotel cannot provide two rooms to families that request two rooms, then they go elsewhere. Design a heuristic that has as goal to maximize the number of rooms rented out over time.

9.7. Consider again the instance described in Exercise 9.1. Assume now that it is possible to do an activity either before its release date or after its due date. Doing an activity for τ time units before its actual release date costs $c_1\tau$, where $c_1 = 1$, and doing an activity for τ time units after its due date costs $c_2\tau$, where $c_2 = 3$. Assume that now all the activities *must* be done.

(a) Develop a heuristic to find a low cost schedule.

(b) Apply the heuristic to the instance of Exercise 9.1 with three resources.

(c) Apply the heuristic to the instance of Exercise 9.1 with two resources.

9.8. Consider a timetabling problem with all processing times equal to 1 and tooling constraints. Let T_j denote the set of tools that are required for the processing of job j. Assume that the T_j sets are nested. That is, for any pair of sets T_j and T_k one and only one of the following three statements hold:

(i) Sets T_j and T_k are identical.
(ii) Set T_j is a subset of set T_k.
(iii) Set T_k is a subset of T_j.

(a) Develop a polynomial time algorithm for this problem.

(b) Does the equivalent reservation problem have any special properties?

9.9. Consider the following timetabling problem with tool sets.

jobs	1	2	3	4	5
p_j	1	1	1	1	1
tool 1	1	0	0	1	0
tool 2	0	1	1	0	0
tool 3	1	1	0	1	1
tool 4	0	1	1	1	1

(a) Can the tool sets be numbered in such a way that all the tools needed by each job are adjacent?

(b) Develop an algorithm for verifying whether the tool sets can be numbered in that way.

9.10. Consider the following timetabling problem with two types of personnel. The total number of personnel of type 1 is $W_1 = 3$ and the total number of personnel of type 2 is $W_2 = 4$.

activities	1	2	3	4	5	6	7
p_j	1	1	1	1	1	1	1
W_{1j}	2	0	1	2	1	2	1
W_{2j}	2	4	0	2	3	1	2

(a) Determine first which type of personnel is the most critical (the tightest).

(b) Use the information under (a) to develop a heuristic for this problem with two types of personnel (your heuristic may be a generalization of the FFD heuristic described in Section 9.5).

(c) Give a numerical example of bad behavior of your heuristic.

9.11. Consider the workforce constrained timetabling problem with the activities having arbitrary processing times. In this problem the makespan has to be minimized subject to the workforce constraint W_1. Consider now the

... of this problem. All activities have to be completed by a fixed time C_{max}. However, the maximum number of people used for these activities has to be minimized. Prove or disprove any mathematical relationship between these two problems.

Comments and References

The first part of Section 9.2 is based primarily on the paper by Bouzina and Emmons (1996). The last part of Section 9.2 that focuses on the processing of all activities with a minimum number of resources is (as described at the end of the section) a special case of a node coloring problem on an arbitrary graph. This problem is actually equivalent to a node coloring problem on a graph with a certain special property; the graphs that satisfy this special property are usually referred to as interval graphs.

A fair amount of research has been done on single and parallel machine models with release dates and due dates or deadlines. A partial list includes McNaughton (1959), Moore (1968), Emmons (1969), Sahni and Cho (1979), Potts (1980), Garey, Johnson, Simons, and Tarjan (1981), Martel (1982a, 1982b), Posner (1985), Potts and van Wassenhove (1988), Ow and Morton (1989), Hall and Posner (1991), Hall, Kubiak and Sethi (1991), and Dondeti and Emmons (1992).

Section 4 of this chapter is based on the research that has been done in bin packing. A very good, but slightly dated, overview of the bin packing literature is presented in Garey and Johnson (1979). Duin and van der Sluis (2006) consider an interesting application of a 2-dimensional bin packing problem.

A significant amount of research has been done on the timetabling problem with operator or tooling constraints and the problems that are equivalent, i.e., graph coloring and examination timetabling. Brelaz (1979) presents a number of methods to color the nodes of a graph. Algorithm 9.5.4 is due to Laporte and Desroches (1984). Carter (1986) presents a survey of practical applications of examination timetabling algorithms. For excellent overviews of the recent developments in timetabling, see Burke and Ross (1996), Burke and Carter (1998), Burke and Erben (2001), Burke and De Causmaecker (2003), Burke and Trick (2004), and Burke and Rudova (2006). For an in-depth analysis of local search techniques applied to course timetabling and examination timetabling problems, see Di Gaspero (2003).

The class assignment system developed at UC Berkeley is described in detail by Glassey and Mizrach (1986). Mulvey (1982), Gosselin and Truchon (1986), and Carter and Tovey (1992) also consider the classroom assignment problem.

Chapter 10

Scheduling and Timetabling in Sports and Entertainment

10.1 Introduction

The previous chapter covered the basics of interval scheduling and timetabling. The models considered were relatively simple and their main goal was to provide some insights. In practice, there are many, more complicated applications of interval scheduling and timetabling. For example, there are important applications in sports as well as in entertainment, e.g., the scheduling of games in tournaments and the scheduling of commercials on network television.

This chapter covers basically two topics, namely tournament scheduling and the scheduling of programs on network television. These two topics turn out to be somewhat related. The next section focuses on some theoretical properties of tournament schedules that are prevalent in U.S. college basketball, major league baseball, and European soccer; this section also presents a general framework for tackling the associated optimization problems via integer programming. The third section describes a completely different procedure for dealing with the same problem, namely the constraint programming approach. The fourth section looks at two tournament scheduling problems that are slightly different from the one discussed in the second and third section,

M.L. Pinedo, *Planning and Scheduling in Manufacturing and Services*,
DOI: 10.1007/978-1-4419-0910-7_10, © Springer Science + Business Media, LLC 2009