1

Models and Basic Concepts

In this chapter we introduce three basic project scheduling problems: the time-constrained project scheduling problem, the resource-constrained project scheduling problem with renewable resources, and the resource-constrained project scheduling problem with cumulative resources. The time-constrained project scheduling problem consists in scheduling the activities of a project such that all temporal constraints are satisfied and some objective function is optimized. We review how temporal scheduling of the project can be performed by solving specific time-constrained project scheduling problems. We distinguish between two types of resources, namely renewable and cumulative resources, depending on whether or not resource availability at a given point in time is affected by the complete past project evolution. For both types of resources we show how to cope with resource constraints by establishing precedence relationships among the activities from so-called forbidden sets, whose joint resource requirements exceed the resource availability.

1.1 Temporal Constraints

1.1.1 Time-Feasible Schedules

A project can be considered to be a set of interacting tasks requiring time and resources for their completion. The structural analysis of the project provides a decomposition of the tasks into a set V of activities and a set E of precedence relationships among them. Set V consists of n activities $i = 1, \ldots, n$ to be scheduled and two auxiliary activities 0 and $n + 1$, representing the project beginning and the project termination, respectively. The precedence relationships can be represented as activity pairs (i, j) where $i \neq j$, saying that the start time of activity i affects the earliest start time of activity j. Thus, $E \subset V \times V$ is some irreflexive relation in set V. Note that this relation may not be asymmetric if there are two activities $i, j \in V$ which mutually influence their earliest start times. The time estimation associates a duration

$p_i \in \mathbb{Z}_{\geq 0}$ with each activity and a time lag $\delta_{ij} \in \mathbb{Z}$ with each pair $(i,j) \in E$. An activity $i \in V$ is referred to as *fictitious activity* or *event* if $p_i = 0$. Otherwise, we speak of a *real activity*. The project beginning and termination, the receipt of materials, or milestones are examples of events. V^a and V^e respectively denote the sets of real activities and events of the project. We assume that the real activities must not be interrupted once they have been begun. Let S_i denote the start time of activity i, which has to be determined when scheduling the project in the temporal scheduling and resource allocation steps. If i is a fictitious activity, S_i is also termed the occurrence time of event i. The time lags δ_{ij} give rise to the *temporal constraints*

$$S_j - S_i \geq \delta_{ij} \quad ((i,j) \in E) \tag{1.1}$$

If $(i,j) \in E$, activity j cannot be started earlier than δ_{ij} units of time after the start of activity i. A nonnegative value of δ_{ij} corresponds to a *minimum time lag* $d_{ij}^{min} = \delta_{ij} \geq 0$ between activities i and j, whereas a negative value of δ_{ij} can be viewed as a *maximum time lag* $d_{ji}^{max} = -\delta_{ij} > 0$ between activities j and i. If $d_{ij}^{min} = p_i$, inequality (1.1) is referred to as a *precedence constraint* between activities i and j. For what follows, we establish the following convention.

Remark 1.1. The project is started at time 0 and must be completed by a prescribed deadline $\overline{d}$, i.e., $S_0 = 0$ and $S_{n+1} \leq \overline{d}$. The deadline is represented as a maximum time lag $d_{0,n+1}^{max} = \overline{d}$ between the project beginning 0 and the project termination $n+1$.

The temporal constraints (1.1) connect the *start* times of activities i and j. Since by assumption activities must not be interrupted when being in progress,

$$C_i := S_i + p_i$$

is the completion time of activity i. Thus, start-to-start, start-to-completion, completion-to-start, and completion-to-completion relationships among activities can easily be transformed into one another (cf. e.g., Bartusch et al. 1988).

Remark 1.2. Some constraints that occur frequently in practice can be modelled by minimum and maximum time lags between activities (see Neumann and Schwindt 1997):

(a) Release date r_i for the start of activity i (head of i): $d_{0i}^{min} = r_i$.
(b) Deadline $\overline{d}_i$ for the completion of activity $i \in V$: $d_{0i}^{max} = \overline{d}_i - p_i$.
(c) Quarantine time q_i after the completion of activity i (tail of i): $d_{i,n+1}^{min} = p_i + q_i$.
(d) Fixed start time t_i for activity i: $d_{0i}^{min} = d_{0i}^{max} = t_i$.
(e) Simultaneous start of activities i and j: $d_{ij}^{min} = d_{ij}^{max} = 0$.
(f) Simultaneous completion of activities i and j with $p_i \geq p_j$: $d_{ij}^{min} = d_{ij}^{max} = p_i - p_j$.

(g) Consecutive execution of activities i and j without any delay in between: $d_{ij}^{min} = d_{ij}^{max} = p_i$.
(h) Overlapping of activities i and j for at least $x_{ij} \leq \min(p_i, p_j)$ units of time: $d_{ij}^{max} = p_i - x_{ij}$, $d_{ji}^{max} = p_j - x_{ij}$.

From $C_i = S_i + p_i$ it follows that the schedule for executing the activities $i \in V$ of the project is uniquely given by specifying the respective start times S_i. That is why we shall always represent solutions to project scheduling problems by a vector of activity start times.

Definition 1.3 (Time-feasible schedule). *A vector $S = (S_0, S_1, \ldots, S_{n+1})$ of start times for the activities where $S_i \geq 0$ $(i \in V)$ and $S_0 = 0$ is called a schedule. Schedule S is said to be time-feasible if it satisfies the temporal constraints (1.1). The set of all time-feasible schedules is denoted by $\mathcal{S}_T$.*

Obviously, set $\mathcal{S}_T$ represents an integral polytope in $\mathbb{R}_{\geq 0}^{n+2}$. Assume that $\mathcal{S}_T \neq \emptyset$. It is well-known that the partially ordered set $(\mathcal{S}_T, \leq)$ possesses exactly one minimum ES, where $S \leq S'$ precisely if $S_i \leq S_i'$ for all $i \in V$. We refer to ES as the *earliest schedule*. Furthermore, by Remark 1.1 $(\mathcal{S}_T, \leq)$ possesses exactly one maximum LS, which is termed the *latest schedule*. This means that there is no time-feasible schedule S such that $S_i < ES_i$ or $S_i > LS_i$ for any $i \in V$. The interval $[ES_i, LS_i]$ is termed the *time window* (for the start) of activity i.

Now let $f : \mathcal{S}_T \to \mathbb{R}$ be an *objective function* assigning a value $f(S)$ to each time-feasible schedule S. Without loss of generality we assume that the objective function has to be minimized. The basic *time-constrained project scheduling problem* can then be stated as follows:

$$\boxed{\begin{array}{ll} \text{Minimize} & f(S) \\ \text{subject to} & S \in \mathcal{S}_T \end{array}} \tag{1.2}$$

Definition 1.4 (Time-optimal schedule). *A time-feasible schedule S solving the time-constrained project scheduling problem (1.2) is called time-optimal.*

All objective functions that will be considered in this book are lower semi-continuous, i.e., any lower-level set $L_\alpha = \{S \in \mathcal{S}_T \mid f(S) \leq \alpha\}$, $\alpha \in \mathbb{R}$, is closed. Since set $\mathcal{S}_T$ is compact, this property ensures that there always exists a time-optimal schedule provided that $\mathcal{S}_T \neq \emptyset$.

1.1.2 Project Networks

In this subsection we shall show how the activities $i \in V$ and the temporal constraints $S_j - S_i \geq \delta_{ij}$ for $(i, j) \in E$ can be represented by a *project network*. Basically, there are three different types of project networks. *Activity-on-arc* or *CPM networks* associate an arc (u, v) with each activity i, where

the nodes u and v represent events (see Kelley 1961). CPM stands for "Critical Path Method", a temporal scheduling method based on activity-on-arc networks. u is the first start of all activities i belonging to arcs emanating from node u, whereas v is the last completion of all activities i belonging to arcs terminating at node v. Arc (u,v) is weighted by the duration p_i of the corresponding activity i. Though only precedence constraints can be modelled by CPM networks, this type of project network is widely used in practice. In general, dummy activities have to be introduced for modelling the precedence constraints among the activities and there is no unique representation of the project as a CPM network. The problem to assign a CPM network to the project in question using a minimum number of dummy activities is known to be NP-hard (cf. Garey and Johnson 1979, problem ND44). Neumann (1999*a*) devises an $\mathcal{O}(n^6)$ time algorithm for the construction of a CPM network with a small number of dummy activities, which is based on a procedure by Bricker (1973).

In *activity-on-node networks*, the nodes are identified with the activities. For each time lag δ_{ij}, the network contains one arc (i,j) with initial node i and terminal node j, i.e., V is the node set and E is the arc set of the network. An arc $(i,j) \in E$ is weighted by δ_{ij}. Activity-on-node networks belong to the class of *MPM networks* (cf. Roy 1964, Sect. II.2.1). MPM is the acronym of "Metra Potential Method", the temporal scheduling method for activity-on-node networks to be discussed in Subsection 1.1.3. Similar to CPM, MPM is based on calculating longest directed paths in the project network. Obviously, activity-on-node networks can cope with general temporal constraints. In addition, due to the one-to-one correspondence between precedence relationships and arcs, there is a unique activity-on-node representation of the project (cf. Neumann and Schwindt 1997).

Elmaghraby and Kamburowki (1992) have introduced the following *event-on-node network*. Each real activity i is represented by two events i^s and i^c in node set V. i^s corresponds to the start and i^c to the completion of activity i. Both nodes are linked by two arcs (i^s,i^c) and (i^c,i^s) with weights $\delta_{i^si^c} = p_i$ and $\delta_{i^ci^s} = -p_i$. For each time lag δ_{ij} between activities i and j, arc set E contains an arc (i^c,j^s) with weight $\delta_{i^cj^s} = \delta_{ij} - p_i$. Analogously to activity-on-node networks, the arcs of the resulting MPM network can be interpreted as minimum and maximum time lags between the incident events. The arcs (i^s,i^c) and (i^c,i^s) state that the completion of activity i must occur exactly p_i units of time after its start, i.e., activity i must not be interrupted. The arcs (i^c,j^s) represent completion-to-start time lags between activities i and j.

Example 1.5. We consider a project with four real activities $i = 1,2,3,4$ for which we assume that activities 3 and 4 cannot be started before activities 1 and 2 have been completed. The project must be completed by a prescribed deadline $\overline{d}$. Figure 1.1a shows the corresponding activity-on-arc project network, where the dashed-line arcs represent dummy activities required for modelling the precedence relationships. The arcs are labelled with the durations

of the respective activities. The activity-on-node network of the project is shown in Figure 1.1b, where square nodes correspond to real activities and circular nodes represent events. By splitting up each real activity into a start and a completion event, one obtains the representation of the project as an event-on-node network, which is shown in Figure 1.1c.

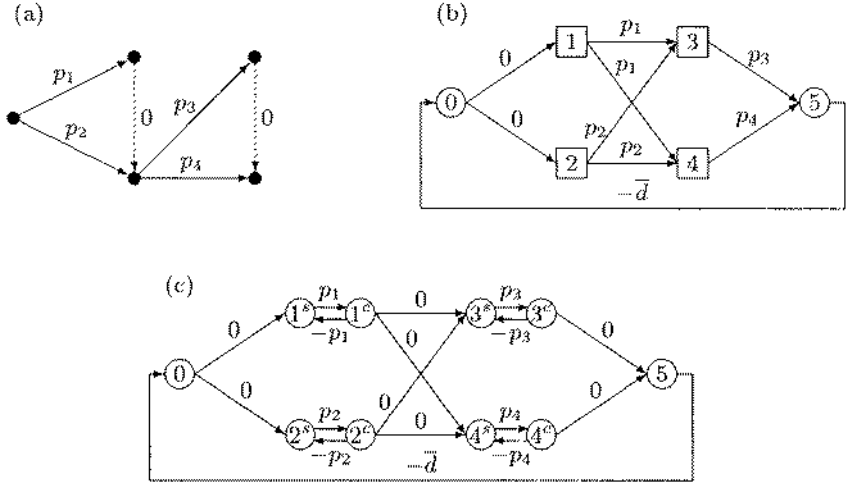

Fig. 1.1. Types of project networks: (**a**) activity-on-arc network; (**b**) activity-on-node network; (**c**) event-on-node network

Throughout this monograph, we shall represent projects by MPM networks. If not stated otherwise, the project network is an activity-on-node network. Event-on-node networks will be used when dealing with project scheduling problems where events instead of activities take up resources (the case of cumulative resources treated in Section 1.3).

1.1.3 Temporal Scheduling Computations

In this subsection we review the Metra Potential Method for the temporal scheduling of the project. Let $N = (V, E, \delta)$ be the MPM network under consideration, where $\delta = (\delta_{ij})_{(i,j)\in E}$ denotes the vector of arc weights. Temporal scheduling consists of

(a) computing earliest and latest start time of activities,
(b) finding the shortest project duration,
(c) calculating total floats, early free floats, and late free floats of activities, and
(d) identifying the critical activities with zero total float

with respect to the temporal constraints (1.1).

A vector $\pi \in \mathbb{R}^{n+2}$ is called a *potential* on project network N if the corresponding *tensions* $\pi_j - \pi_i$ are greater than or equal to the respective lower bounds δ_{ij} (cf. Berge 1970, Sect. 5.3). Let S be some schedule and assume that $\mathcal{S}_T \neq \emptyset$. Clearly, S is a potential on N if and only if schedule S is time-feasible. The earliest schedule ES thus corresponds to the componentwise minimal potential $\pi \geq 0$, and the latest schedule LS equals the componentwise maximal potential $\pi \geq 0$ with $\pi_0 = 0$ (and thus $\pi_{n+1} \leq \overline{d}$, see Remark 1.1). In other words, ES is the unique solution to the following minimization problem:

$$\left.\begin{array}{lll} \text{Minimize} & \sum_{i \in V} \pi_i & \\ \text{subject to} & \pi_j - \pi_i \geq \delta_{ij} & ((i,j) \in E) \\ & \pi_i \geq 0 & (i \in V) \end{array}\right\} \tag{1.3}$$

Problem (1.3) corresponds to the time-constrained optimization problem (1.2) where $f(S) = \sum_{i \in V} S_i$. The latest schedule LS is obtained by solving (1.3) with objective function $-\sum_{i \in V} \pi_i$ and additional constraint $\pi_0 = 0$.

Now let $D = (d_{ij})_{i,j \in V}$ be the matrix solving the following system of equations

$$\left.\begin{array}{ll} d_{ii} = 0 & (i \in V) \\ d_{ij} = \max\limits_{(h,j) \in E} (d_{ih} + \delta_{hj}) & (i,j \in V : i \neq j) \end{array}\right\} \tag{1.4}$$

The values d_{ij} can be interpreted as time lags between activities i and j which are induced by the set of given time lags δ_{ij} $((i,j) \in E)$.

Remarks 1.6.

(a) Due to $\delta_{ij} \in \mathbb{Z}$ for all $(i,j) \in E$, matrix D is integral as well.

(b) For each activity $i \in V$, we assume that $d_{0i} \geq 0$ (i.e, activity i cannot be started before the project beginning) and $d_{i,n+1} \geq p_i$ (i.e., the project cannot be terminated before all activities have been completed).

(c) Each node $i \in V$ in project network N is reachable from node 0 and node $n+1$ is reachable from each node $i \in V$. Since we always have a maximum time lag $d^{max}_{0,n+1} = -\delta_{n+1,0} = \overline{d}$ between the project beginning and the project termination and thus $(n+1,0) \in E$, project network N is strongly connected.

(d) Without loss of generality we assume that $\overline{d}$ is the latest project termination time, i.e., $d_{n+1,0} = \overline{d}$.

(e) The minimum time lag between the project beginning and activity i equals the earliest start time ES_i of activity i, i.e.,

$$ES_i = d_{0i} \quad (i \in V)$$

(f) Likewise, the maximum time lag between the project beginning and activity i equals the latest start time of activity i, i.e.,

$$LS_i = -d_{i0} \quad (i \in V)$$

If there is no given maximum time lag $d_{0i}^{max} = -\delta_{i0}$ between the project beginning and activity i, then $LS_i = \overline{d} - d_{i,n+1}$.

(g) The earliest and latest completion times of activity i are

$$EC_i = ES_i + p_i \text{ and } LC_i = LS_i + p_i \quad (i \in V)$$

Recall that a directed walk in network N is a sequence $(i_1, i_2, \ldots, i_\nu)$ of nodes of N such that $(i_\mu, i_{\mu+1}) \in E$ for all $\mu = 1, \ldots, \nu - 1$, where the sum $\sum_{\mu=1}^{\nu-1} \delta_{i_\mu i_{\mu+1}}$ is referred to as the length of the directed walk. A directed walk without any repetition of nodes is called a directed path. A directed cycle is a directed walk $(i_1, i_2, \ldots, i_\nu, i_1)$ such that $(i_1, i_2, \ldots, i_\nu)$ is a directed path. The lower equations in (1.4) correspond to the Bellman equations for calculating longest directed walks in MPM networks. Thus, each induced time lag d_{ij} coincides with the length of a longest directed walk from node i to node j, provided that there is such a longest directed walk. Since according to Remark 1.6c network N is strongly connected, there is always a directed walk in N from any node $i \in V$ to any node $j \in V$. In Roy (1962) it is shown that there exists a *longest* directed walk from any node $i \in V$ to any node $j \in V$ in N if and only if N does not contain any directed cycle of positive length. On the other hand, system of equations (1.4) possesses a solution precisely if there is a longest directed walk from i to j for all $i, j \in V$. In the latter case, the longest directed walks in N coincide with the longest directed paths in N, and $D = (d_{ij})_{i,j \in V}$ is called the *distance matrix* of N. Thus, we have the following proposition.

Proposition 1.7 (Roy 1962). *There is a time-feasible schedule for a project (i.e., $\mathcal{S}_T \neq \emptyset$) if and only if project network N does not contain any directed cycle of positive length.*

Let $m := |E|$ denote the number of arcs in project network N. Problem (1.3) can be solved in $\mathcal{O}(mn)$ time by the label-correcting procedure shown in Algorithm 1.1 (cf. Bellman 1958), where Q is a queue. Although this algorithm has been devised more than four decades ago, it is still the most efficient algorithm for solving longest-path problems in cyclic networks with arbitrary arc weights. The procedure may be terminated if some node i has been examined $n+2$ times (see, e.g., Ahuja et al. 1993, Sect. 5.5). In that case, (1.3) is unsolvable and thus $\mathcal{S}_T = \emptyset$, which means that contradictory temporal constraints have been specified.

The solution D to equations (1.4) is the elementwise minimal matrix satisfying

$$\left.\begin{array}{ll} d_{ii} = 0 & (i \in V) \\ d_{ij} \geq \delta_{ij} & ((i,j) \in E) \\ d_{ij} \geq d_{ih} + d_{hj} & (h,i,j \in V) \end{array}\right\} \tag{1.5}$$

This formulation gives rise to the following Algorithm 1.2 by Floyd and Warshall (cf. Floyd 1962) for computing distances d_{ij} for all $i, j \in V$. After having

Algorithm 1.1. Earliest schedule

Input: MPM project network $N = (V, E, \delta)$.
Output: Earliest schedule ES.

set $d_{00} := 0$, $Q := \{0\}$, and $d_{0i} := -\infty$ for all $i \in V \setminus Q$;
while $Q \neq \emptyset$ **do**
  dequeue i from Q;
  for all $(i,j) \in E$ with $d_{0j} < d_{0i} + \delta_{ij}$ **do**
    set $d_{0j} := d_{0i} + \delta_{ij}$;
    if $j \notin Q$ **then** enqueue j to Q;
return earliest schedule $ES = (d_{0i})_{i \in V}$;

initialized the values d_{ij} according to the prescribed time lags δ_{ij}, the algorithm computes the transitive closure of those time lags by iteratively putting into force the *triangle inequalities*

$$d_{ij} \geq d_{ih} + d_{hj} \tag{1.6}$$

(1.5) and thus (1.4) is solvable exactly if the matrix D calculated by the Floyd-Warshall algorithm satisfies $d_{ii} = 0$ for all $i \in V$. The number of computations performed is $\mathcal{O}(n^3)$, which is the best possible time complexity for this problem (note that for checking whether or not distances d_{ij} satisfy (1.5), $\mathcal{O}(n^3)$ triangle inequalities must be evaluated).

Algorithm 1.2. Distance matrix

Input: MPM project network $N = (V, E, \delta)$.
Output: Distance matrix D.

for all $i, j \in V$ **do**
  if $(i,j) \in E$ **then** set $d_{ij} := \delta_{ij}$; **elsif** $i = j$ set $d_{ij} := 0$; **else** set $d_{ij} := -\infty$;
for all $h, i, j \in V$ with $d_{ih} > -\infty$ and $d_{hj} > -\infty$ **do**
  if $d_{ij} < d_{ih} + d_{hj}$ **then** set $d_{ij} := d_{ih} + d_{hj}$;
return distance matrix $D = (d_{ij})_{i,j \in V}$;

The next algorithm, which is due to Bartusch et al. (1988), achieves the update of the distance matrix D in $\mathcal{O}(n^2)$ time when adding some arc (i,j) to the project network (see Algorithm 1.3). The calculation of the distance matrix D from scratch by initializing the values d_{ij} as in the Floyd-Warshall algorithm and then applying the algorithm for all arcs $(i,j) \in E$ would require $\mathcal{O}(mn^2)$ time, which is more expensive than using the Floyd-Warshall algorithm. The former procedure, however, will prove useful later on when dealing with resource constraints and the resolution of so-called resource conflicts, where individual arcs are added to N.

Algorithm 1.3. Addition of arc (i,j) with weight δ_{ij}

Input: Distance matrix D, an arc (i,j) with weight δ_{ij}.
Output: Updated distance matrix D.

for all $g,h \in V$ **do**
  if $d_{gh} < d_{gi} + \delta_{ij} + d_{jh}$ **then** set $d_{gh} := d_{gi} + \delta_{ij} + d_{jh}$;
return distance matrix $D = (d_{ij})_{i,j \in V}$;

Remark 1.8. The update of distances d_{0h} (or, equivalently, earliest start times ES_h) after the addition of some arc (i,j) to project network N can be performed in $\mathcal{O}(n)$ time by putting $d_{0h} := \max(d_{0h}, d_{0i} + \delta_{ij} + d_{jh})$ for all $h \in V$. This can easily be seen by using the fact that the correctness of Algorithm 1.3 does not depend on the sequence in which pairs (g,h) are iterated. Symmetrically, distances d_{g0} (which coincide with the negative latest start times $-LS_g$) can be updated by putting $d_{g0} := \max(d_{g0}, d_{gi} + \delta_{ij} + d_{j0})$ for all $g \in V$. Moreover, adding some arc $(0,j)$ to N does not affect distances d_{i0} $(i \in V)$ and adding some arc $(i,0)$ to N does not affect distances d_{0j} $(j \in V)$.

Proposition 1.9 shows that the creation of a directed cycle of positive length when adding arc (i,j) to N can be tested before calling Algorithm 1.3.

Proposition 1.9. *Let N be a project network with distance matrix D. The addition of arc (i,j) with weight δ_{ij} to N generates a directed cycle of positive length if and only if $\delta_{ij} > -d_{ji}$.*

Proof. Sufficiency: After the addition of arc (i,j) with $\delta_{ij} > -d_{ji}$ to N it holds that $d_{ij} \geq \delta_{ij}$. Thus, we have $d_{ij} + d_{ji} \geq \delta_{ij} + d_{ji} > 0$, which means that there is a directed cycle of positive length containing nodes i and j.

Necessity: Now assume that $\delta_{ij} \leq -d_{ji}$ and consider an iteration of Algorithm 1.3 for some pair (g,h) such that distance d_{gh} is increased. Then the updated distance is $d_{gh} = d_{gi} + \delta_{ij} + d_{jh} \leq d_{gi} - d_{ji} + d_{jh}$. The triangle inequalities (1.6) say that $d_{ji} \geq d_{jh} + d_{hg} + d_{gi}$ and thus $d_{gh} + d_{hg} \leq d_{gi} - (d_{jh} + d_{hg} + d_{gi}) + d_{jh} + d_{hg} = 0$. This means that after applying Algorithm 1.3 it holds that $d_{gh} + d_{hg} \leq 0$ for all $g,h \in V$, i.e., N contains no directed cycle of positive length. □

Next, we consider three different floats or slack times of an activity $i \in V$. The *total float* TF_i is the maximum amount of time by which the start of activity i can be delayed beyond its earliest start time ES_i such that the project is terminated on time, i.e., $S_{n+1} \leq \overline{d}$. In other words,

$$TF_i = LS_i - ES_i = -d_{i0} - d_{0i} \quad (i \in V)$$

Activity $i \in V$ is called *critical* if i cannot be delayed, i.e., if the maximum time lag $-d_{i0}$ equals the minimum time lag d_{0i} between the project beginning

and activity i and thus fixes the start time S_i of i to $ES_i = LS_i$. Activity i is critical exactly if $TF_i = 0$.

The *early free float* EFF_i is the maximum amount of time by which the earliest start of activity i at time ES_i can increase given that any other activity j can be begun at its earliest start time ES_j. Hence,

$$EFF_i = \min_{(i,j)\in E}(ES_j - \delta_{ij}) - ES_i = \min_{j\in V: i\neq j}(d_{0j} - d_{ij}) - d_{0i} \quad (i \in V)$$

The *late free float* LFF_i is the maximum amount of time by which the latest start of activity i at time LS_i can decrease given that any other activity j can be begun at its latest start times LS_j. Thus,

$$LFF_i = LS_i - \max_{(j,i)\in E}(LS_j + \delta_{ji}) = \min_{j\in V: i\neq j}(d_{j0} - d_{ji}) - d_{i0} \quad (i \in V)$$

1.2 Renewable-Resource Constraints

To perform the activities of a project, different types of resources are required. Basically, we may distinguish between resources whose availability solely depends on the activities being in progress (like manpower or machinery) and resources for which the availability results from the entire project history (such as the project budget, materials, or storage space). In this section we deal with *renewable resources*, which belong to the former type and to which the overwhelming part of research in the field of resource-constrained project scheduling has been dedicated. The case of *cumulative resources*, corresponding to the second type, will be discussed in Section 1.3. In the present section, we suppose that no cumulative resources need to be considered. At first, we provide a formal statement of the constraints arising from the scarcity of renewable resources. We are then concerned with conditions on the start times of activities whose joint requirements for renewable resources exceed the resource capacities and which thus cannot be in progress simultaneously. Finally, we discuss consistency tests for detecting temporal constraints that are implied by the limited availability of renewable resources.

1.2.1 Resource-Feasible Schedules

Let $\mathcal{R}^\rho$ be the set of renewable resources k with *capacity* $R_k \in \mathbb{N} \cup \{\infty\}$ that have been assigned to the project during the project definition phase. $R_k = \infty$ means that the availability of resource k is not explicitly bounded from above but can be adapted, at a certain cost, to any usage over time. The resource estimation yields (resource) *requirements* $r_{ik} \in \mathbb{Z}_{\geq 0}$ for each real activity $i \in V^a$ and each resource $k \in \mathcal{R}^\rho$. r_{ik} corresponds to the number of capacity units of resource k which are taken up for processing activity i from start time S_i (inclusively) to completion time $C_i = S_i + p_i$ (exclusively).

$r_{ik} = 0$ means that activity i does not use resource k. Furthermore, we assume that

$$r_{ik} \leq R_k \quad (i \in V^a,\ k \in \mathcal{R}^\rho)$$

which ensures that sufficient resource capacity is available for processing each activity individually. For simplicity, we may omit resource index k when there is only one renewable resource available.

Now let S be some schedule and let t be some point in time. Then

$$\mathcal{A}(S,t) := \{i \in V^a \mid S_i \leq t < S_i + p_i\}$$

is the *active set* of activities being in progress at time t. The corresponding requirement for resource $k \in \mathcal{R}^\rho$ at time t is given by

$$r_k(S,t) := \sum_{i \in \mathcal{A}(S,t)} r_{ik}$$

For given schedule S, function $r_k(S,\cdot) : \mathbb{R} \to \mathbb{Z}_{\geq 0}$ is termed the *loading profile* for renewable resource k. $r_k(S,\cdot)$ is a right-continuous step function with at most $2n$ jump discontinuities. Obviously, we have $r_k(S,t) = 0$ for all $t < 0$.

The *renewable-resource constraints* can be stated as follows:

$$r_k(S,t) \leq R_k \quad (k \in \mathcal{R}^\rho,\ 0 \leq t \leq \overline{d}) \tag{1.7}$$

Definition 1.10 (Resource-feasible and feasible schedules). *A schedule S satisfying the renewable-resource constraints (1.7) is called resource-feasible with respect to renewable resources $k \in \mathcal{R}^\rho$. The set of all resource-feasible schedules is denoted by $\mathcal{S}_R$. $\mathcal{S} := \mathcal{S}_T \cap \mathcal{S}_R$ is the set of all feasible schedules.*

As we shall see later on, unlike the polytope of time-feasible schedules $\mathcal{S}_T$, set $\mathcal{S}_R$ represents a finite union of polytopes which is generally not connected. As an intersection of a polytope and a finite union of polytopes, $\mathcal{S}$ is the union of finitely many polytopes as well. Resource allocation consists in assigning start times S_i (and thus execution time intervals $[S_i, C_i[$) to the activities of the project such that the corresponding schedule $S = (S_i)_{i \in V}$ is feasible and minimizes the objective function on set $\mathcal{S}$.

The basic *resource-constrained project scheduling problem with renewable resources* reads as follows:

$$\boxed{\begin{array}{ll} \text{Minimize} & f(S) \\ \text{subject to} & S \in \mathcal{S}_T \cap \mathcal{S}_R \end{array}} \tag{1.8}$$

Recall that we have assumed objective function f to be lower semicontinuous. The compactness of $\mathcal{S}$ then implies that there exists an optimal solution to problem (1.8) precisely if $\mathcal{S} \neq \emptyset$. Note, however, that due to the presence of maximum time lags it may happen that $\mathcal{S}_T \neq \emptyset$ and $\mathcal{S}_R \neq \emptyset$ but $\mathcal{S} = \emptyset$.

Definition 1.11 (Optimal schedule). *A feasible schedule S solving the resource-constrained project scheduling problem (1.8) is called optimal.*

By replacing the set $\mathcal{S} = \mathcal{S}_T \cap \mathcal{S}_R$ of feasible schedules with the set of resource-feasible schedules $\mathcal{S}_R$ we obtain the *temporal relaxation* of the resource-constrained project scheduling problem (1.8). Since we have assumed that $r_{ik} \leq R_k$ for all $i \in V^a$ and all $k \in \mathcal{R}^\rho$, each schedule carrying out the activities one after another is resource-feasible. The *resource relaxation* of (1.8) arises from deleting the resource constraints (1.7) or, equivalently, setting $R_k := \infty$ for all $k \in \mathcal{R}^\rho$. The resource relaxation coincides with the basic time-constrained project scheduling problem (1.2). As we noticed in Subsection 1.1.3, the existence of a time-feasible schedule can be checked in $\mathcal{O}(mn)$ time by applying Algorithm 1.1 to project network N. The following theorem, however, shows that it cannot be decided in polynomial time whether or not there exists a feasible schedule.

Theorem 1.12 (Bartusch et al. 1988). *The following decision problem is NP-complete.*

Instance: A project with one renewable resource and requirements $r_i = 1$ for all $i \in V^a$.

Question: Does there exist a feasible schedule?

Proof. The feasibility of a given schedule S can be checked by computing $S_j - S_i$ for all arcs $(i,j) \in E$ of project network N as well as the resource requirements $r_k(S,t)$ for all resources $k \in \mathcal{R}^\rho$ and all start times $t = S_i$ of real activities $i \in V^a$. Thus, verification of schedule feasibility can be done in polynomial time, and the problem to decide upon the existence of a feasible schedule belongs to NP. In Bartusch et al. (1988) it is shown by transformation from problem PRECEDENCE CONSTRAINED SCHEDULING with m processors and strict order $\prec$ that the decision problem whether or not $\mathcal{S} \neq \emptyset$ is NP-hard. An equivalent instance of the latter problem is obtained by considering one renewable resource with capacity $R = m$ and choosing $r_i = 1$, $p_i = 1$, $d_{0i}^{min} = 0$, $d_{i,n+1}^{min} = 1$ for all $i \in V^a$, as well as $d_{ij}^{min} = 1$ if $i \prec j$ and $d_{0,n+1}^{max} = 3$. □

When dealing with the project duration problem, we may drop the assumption that there is a deadline $\overline{d}$ on the project termination because the objective is to maximize the slack $\overline{d} - S_{n+1}$ of the deadline constraint. The construction of a feasible schedule then turns into an easy problem if there are no maximum time lags given. In that case, project network N is acyclic, and the activities can be scheduled consecutively according to any topological ordering of the nodes $i \in V$ of N.

1.2.2 Forbidden Sets and Delaying Alternatives

The resource-feasibility of schedules is closely related to the concept of forbidden sets introduced by Radermacher (1978). The forbidden-set perspective of resource constraints is useful for investigating the set $\mathcal{S}$ of feasible schedules.

Definition 1.13 (Forbidden and feasible sets). *A set of real activities $F \subseteq V^a$ is called a forbidden set if there is some resource $k \in \mathcal{R}^\rho$ such that*

$$\sum_{i \in F} r_{ik} > R_k$$

If F is $\subseteq$-minimal in the set of all forbidden sets, we speak of a minimal forbidden set. By $\mathcal{F}$ we denote the set of all minimal forbidden sets. A set $A \subseteq V^a$ that is not forbidden is termed a feasible set. A is said to be a maximal feasible set if it is $\subseteq$-maximal in the set of all feasible sets.

When solving the resource-constrained project scheduling problem (1.8), the activities from a forbidden set F must be scheduled in such a way that they do not all overlap in time. In other terms, each forbidden set F has to be partitioned into a feasible set A and some nonempty set B, no activity from set B being executed simultaneously with all activities from set A. In literature, such a set B is called a delaying alternative (cf. e.g., Christofides et al. 1987 or Demeulemeester and Herroelen 1992, 1997).

Definition 1.14 (Delaying alternative). *Let F be a forbidden set. $B \subseteq F$ is called a delaying alternative for F if $F \setminus B$ is a feasible set. If additionally B is $\subseteq$-minimal in the set of all delaying alternatives for F (i.e., $F \setminus B$ is a maximal feasible set), we speak of a minimal delaying alternative for F.*

The number of minimal delaying alternatives for a forbidden set F grows exponentially in the cardinality of set F. Given some forbidden set F and a subset $B \subseteq F$, it can be decided in polynomial time whether or not B is a minimal delaying alternative for F by evaluating the following two conditions (1.9) and (1.10). This can be achieved in $\mathcal{O}(|\mathcal{R}^\rho||F|)$ time.

$$\sum_{i \in F \setminus B} r_{ik} \leq R_k \text{ for all } k \in \mathcal{R}^\rho \tag{1.9}$$

$$\sum_{i \in F \setminus B} r_{ik} + \min_{j \in B} r_{jk} > R_k \text{ for some } k \in \mathcal{R}^\rho \tag{1.10}$$

Nevertheless, Neumann et al. (2003*b*) have shown that a minimal delaying alternative B cannot be generated efficiently by iteratively transferring activities from set F to set B.

Proposition 1.15 (Neumann et al. 2003*b*, Sect. 2.5). *The following decision problem is NP-complete.*

Instance: A project with one renewable resource, a forbidden set F, and an activity $h \in F$.

Question: Does there exist a minimal delaying alternative B for F containing h?

Proof. Since conditions (1.9) and (1.10) can be verified in polynomial time, the problem is contained in *NP*. Let B with $h \in B$ be an arbitrary set of activities using the single resource. Then B is a minimal delaying alternative if and only if $R - \min_{j \in B} r_j < \sum_{i \in F \setminus B} r_i \leq R$. For $r_h = 1$, we then have $r_h = \min_{j \in B} r_j$ and thus $R - 1 < \sum_{i \in F \setminus B} r_i \leq R$, i.e., $\sum_{i \in F \setminus B} r_i = R$. Hence, there is a minimal delaying alternative B containing h exactly if there is a set $A \subseteq F \setminus \{h\}$ with $\sum_{i \in A} r_i = R$. Now let I be an instance of problem SUBSET SUM with index set $\mathcal{I}$, sizes $s(i) \in \mathbb{N}$ for $i \in \mathcal{I}$, and threshold $M \in \mathbb{N}$ (cf. Garey and Johnson 1979, problem SP13). We obtain an equivalent instance of our decision problem by choosing $F = \mathcal{I} \cup \{h\}$, $r_i = s(i)$ for all $i \in \mathcal{I}$, $r_h = 1$, and $R = M$. □

Similarly it can be shown that it is also NP-complete to decide whether a given activity h is contained in some minimal forbidden set $F \in \mathcal{F}$ (cf. Stork and Uetz 2005, who devise a polynomial transformation from PARTITION).

In what follows, we describe a recursion for computing minimal delaying alternatives for a forbidden set F (see Neumann et al. 2003*b*, Sect. 2.5). Given a delaying alternative B, the set $\mathcal{B}$ of all minimal delaying alternatives $B' \subseteq B$ for F is either equal to $\{B\}$ if B is a minimal delaying alternative for F or equal to the set of all minimal delaying alternatives $B' \subseteq B \setminus \{i\}$ for F with $i \in B$. To avoid the multiple generation of one and the same minimal delaying alternative B' (as subset of two different sets $B \setminus \{i_1\}$ and $B \setminus \{i_2\}$), we restrict the recursion to subsets B' of $B \setminus \{i\}$ for which $j > i$ holds for all $j \in (B \setminus \{i\}) \setminus B'$. Since F itself is a delaying alternative, which includes all minimal delaying alternatives for F, we start the recursion with $B = F$. Algorithm 1.4 shows the corresponding recursive procedure, where $i = 0$ if $B = F$ at recursion level 0 and i is the number of the activity removed in the preceding call to the recursion, otherwise. A call to ***MinimalDelayingAlternatives***$(F, 0)$ determines the set $\mathcal{B}$ of all minimal delaying alternatives for forbidden set F.

An alternative approach to calculating all feasible subsets $A \subset F$ (and thus all delaying alternatives $B = F \setminus A$) has been proposed by Brucker et al. (1998). Assume that $F = \{i_1, i_2, \ldots, i_\nu\}$. Brucker's procedure constructs a binary decision tree, where each node at level $\mu = 1, \ldots, \nu$ corresponds to some feasible set $A' \subseteq \{i_1, i_2, \ldots, i_\mu\}$ and branching from a node at level $\mu - 1$ corresponds to the decision whether or not activity i_μ is contained in the respective child node at level μ. Each leaf of the decision tree belongs to one feasible set A.

Algorithm 1.4. *MinimalDelayingAlternatives*(B, i)

Input: A project, a forbidden set B, an index i.
Ensure: $\mathcal{B}$ contains all minimal delaying alternatives $B' \subseteq B$ for F with $\min(B \setminus B') > i$.

```
if B satisfies (1.9) then (* B is delaying alternative *)
  if B satisfies (1.10) then (* B is minimal delaying alternative *)
    B := B ∪ {B};
  else
    for all j ∈ B with j > i do MinimalDelayingAlternatives(B \ {j}, j);
```

The following proposition establishes the relationship between minimal delaying alternatives and minimal forbidden sets.

Proposition 1.16 (Schwindt 1998*c*). *A minimal delaying alternative B for a forbidden set F is an $\subseteq$-minimal set containing an activity j of each minimal forbidden set $F' \subseteq F$.*

Proof. We assume that there is a minimal delaying alternative B for F and a forbidden subset $F' \subseteq F$ with $B \cap F' = \emptyset$. Then set $F \setminus B \supseteq F'$ is feasible. Since every superset of a forbidden set is forbidden, this contradicts the fact that F' is forbidden. □

1.2.3 Breaking up Forbidden Sets

When scheduling the activities of a project, *resource conflicts* caused by the simultaneous execution of the activities of some forbidden set have to be resolved. The following theorem by Bartusch et al. (1988) shows how resource conflicts can be settled by introducing precedence constraints between activities of minimal forbidden sets.

Theorem 1.17 (Bartusch et al. 1988). *A schedule S is resource-feasible if and only if for each minimal forbidden set $F \in \mathcal{F}$, there are two activities $i, j \in F$ such that $S_j \geq S_i + p_i$.*

Proof. Sufficiency: We consider the active set $\mathcal{A}(S, t)$ for a resource-infeasible schedule S at some time $t \geq 0$ such that $\mathcal{A}(S, t)$ is forbidden. Since $\mathcal{A}(S, t)$ is forbidden, there is a subset F of $\mathcal{A}(S, t)$ that is minimally forbidden. By definition of $\mathcal{A}(S, t)$, all activities of F overlap at time t, which implies that there are no two activities $i, j \in F$ with $S_j \geq S_i + p_i$.

Necessity: Assume that there is some minimal forbidden set F for which no two activities $i, j \in F$ satisfy $S_j \geq S_i + p_i$. Then $[S_i, S_i + p_i[\cap [S_j, S_j + p_j[\neq \emptyset$ for any two activities $i, j \in F$. The Helly property of intervals (cf. e.g., Golumbic 2004, Sect. 4.5) then implies that $\cap_{i \in F} [S_i, S_i + p_i[\neq \emptyset$, and thus there is some point in time t at which all activities i from set F overlap. Since F is a forbidden set, $r_k(S, t) = \sum_{i \in F} r_{ik} > R_k$ for some $k \in \mathcal{R}^\rho$. □

As a direct consequence of Theorem 1.17 we obtain the following Corollary.

Corollary 1.18 (Bartusch et al. 1988). *The set S of all feasible schedules represents the union of finitely many integral polytopes.*

We say that a constraint C *breaks up* minimal forbidden set F if for each schedule satisfying C, there are two activities $i, j \in F$ such that $S_j \geq S_i + p_i$. Minimal forbidden sets can be broken up in different ways. According to Theorem 1.17, the first possibility consists in choosing two activities $i, j \in F$ and introducing an (ordinary) precedence constraint

$$S_j \geq S_i + p_i \tag{1.11}$$

between i and j. Alternatively one may define a *disjunctive precedence constraint*

$$S_j \geq \min_{i \in F: i \neq j} (S_i + p_i) \tag{1.12}$$

between set $F \setminus \{j\}$ and activity j saying that j must not be started before the earliest completion of some other activity i from set F. Disjunctive precedence constraint (1.12) is equivalent to the disjunction of the precedence constraints (1.11) for all $i \in F$, $i \neq j$ and represents a so-called linear reverse-convex constraint (see, e.g., Tuy 1995, Sect. 7). Whereas the number of alternatives for breaking up F by precedence constraints is $\mathcal{O}(|F|^2)$, this number is of linear order $\mathcal{O}(|F|)$ when using disjunctive precedence constraints. The set of all schedules satisfying a disjunctive precedence constraint is generally disconnected and thus in particular nonconvex. As will be shown in Subsection 3.1.2, the minimization of regular (i.e., componentwise nondecreasing) objective functions can nevertheless be done with a time complexity that is linear in the maximum project duration $\overline{d}$. In literature, disjunctive precedence constraints are also referred to as *AND/OR precedence constraints* or *waiting conditions* (cf. Möhring et al. 2004). They have been introduced by Igelmund and Radermacher (1983) in the form of preselective strategies for resource-constrained project scheduling with stochastic activity durations.

An arbitrary forbidden set F is said to be broken up if all minimal forbidden subsets of F are broken up. Let B be some minimal delaying alternative for F. From Proposition 1.16 it then follows that breaking up F can be achieved by imposing a set of precedence constraints

$$S_j \geq S_i + p_i \quad (j \in B)$$

between some activity i from the maximal feasible set $A = F \setminus B$ and all activities $j \in B$ or by a disjunctive precedence constraint

$$\min_{j \in B} S_j \geq \min_{i \in A} (S_i + p_i)$$

between set A and set B. Note that in the case of precedence constraints, one and the same activity $i \in F \setminus B$ can be chosen for all $j \in B$ because any conjunction of precedence constraints (1.11) for the activities j from delaying alternative B implies shifting all $j \in B$ behind the earliest finishing activity $i \in F \setminus B$, which breaks up forbidden set F.

1.2.4 Consistency Tests

The NP-hardness of finding feasible schedules implies that resource allocation can only be performed by enumerating alternative sets of precedence relationships among activities using common resources. *Consistency tests* designate algorithms for detecting constraints that must be satisfied by any feasible schedule and that can be evaluated without enumeration to rule out in advance certain inadmissible alternatives from further consideration. A consistency test is described through a condition and a constraint that can be established whenever the condition is satisfied. From a geometric point of view, applying consistency tests provides a convex set containing all feasible schedules. In the best case, this convex set coincides with the convex hull $\text{conv}(\mathcal{S})$ of the feasible region. From Theorem 1.12, however, it immediately follows that $\text{conv}(\mathcal{S})$ cannot be computed in polynomial time (otherwise, problem (1.8) with linear objective function f could be efficiently solved by finding some optimal vertex of $\text{conv}(\mathcal{S})$). Since $\mathcal{S}$ is the union of finitely many integral polytopes, the convex hull $\text{conv}(\mathcal{S})$ is integral as well.

In enumeration procedures, consistency tests are often applied dynamically to the search space of any enumeration node. The tests then refer to search spaces rather than to the feasible region. In scheduling literature, consistency tests are also known under the names preprocessing (if they are applied to the root node before starting the enumeration), immediate selection algorithms, edge finding rules, constraint propagation techniques, or satisfiability tests. Instead of directly checking given conditions, consistency tests may also try to refute additional, hypothetical constraints. If the test rejects the hypothesis, the alternative hypothesis has been shown to be true and thus can be used to reduce the search space. Consistency tests have been applied with great success in machine scheduling and for the resource-constrained project duration problem (see Brucker et al. 1998, Dorndorf et al. 2000*a*, or Dorndorf et al. 2000*c*). The algorithm of Carlier and Pinson (1989) that solved the famous Fisher and Thompson (1963) job shop scheduling problem with 10 jobs and 10 machines for the first time has become a classical reference in the field.

We review some consistency tests that have been proposed in literature for project scheduling with renewable resources (see, e.g., Dorndorf et al. 1999). All procedures to be discussed provide additional temporal constraints that can be added in the form of arcs to project network N. Let d_{ij} again denote the length of a longest directed path from node i to node j in project network N, where we assume that $\mathcal{S}_T \neq \emptyset$. Consistency tests are usually used in an iterative fashion as long as new temporal constraints can be identified and thus distance matrix D is modified (see Algorithm 1.5, where Γ denotes the set of consistency tests to be applied). The reason for this is that due to updating distance matrix D, certain tests that in previous iterations failed may possibly deduce additional constraints. In general, the distance matrix yielded depends on the sequence in which the different tests are applied. For the consistency tests to be discussed below, however, it can be shown that

the resulting matrix is unique (cf. Dorndorf et al. 2000*b*). More precisely, any consistency test can be interpreted as a function γ mapping distance matrices D to updated distance matrices $\gamma(D)$. If for all consistency tests $\gamma \in \Gamma$, $D \leq D'$ implies $\gamma(D) \leq \gamma(D')$, then there exists only one fixed-point matrix D with $D = \gamma(D)$.

Algorithm 1.5. Search space reduction by consistency tests $\gamma \in \Gamma$

Input: A project, a set Γ of consistency tests.
Output: Updated distance matrix D.

```
compute distance matrix D; (* Algorithm 1.2 *)
repeat
  for all consistency tests γ ∈ Γ do
    apply γ;
    if new temporal constraint S_j − S_i ≥ δ_ij has been established then
      update distance matrix D, i.e., set D := γ(D); (* Algorithm 1.3 *)
until distance matrix D has not been changed during last iteration;
return distance matrix D;
```

Disjunctive activities tests try to establish precedence constraints between activities which cannot be processed at the same time. Let $i, j \in V^a$ be two different real activities that, with respect to the temporal constraints, can be executed in parallel and for which j cannot be completed before i is started, i.e.,

$$-p_j < d_{ij} < p_i \text{ and } d_{ji} < p_j$$

We say that i and j are *in disjunction* if due to the resource constraints they cannot be processed at the same time. In that case, we can introduce a new precedence constraint $S_j \geq S_i + p_i$ between i and j that will be satisfied by any feasible schedule S.

Obviously, the activities of two-element forbidden sets are in disjunction. However, i and j may also be in disjunction if $r_{ik} + r_{jk} \leq R_k$ for all $k \in \mathcal{R}^p$. Brucker et al. (1998) have used the concept of *symmetric triples* for finding such activities. We call (h, i, j) a symmetric triple if $\{h, i, j\}$ is a forbidden set and activity h must be executed simultaneously with activity i (i.e., $d_{hi} > -p_i$ and $d_{ih} > -p_h$) and with activity j (i.e., $d_{hj} > -p_j$ and $d_{jh} > -p_h$). For a symmetric triple (h, i, j), activities i and j cannot be in progress at the same time because this would imply that h, i, and j were carried out in parallel, which is impossible because $\{h, i, j\}$ is a forbidden set. Obviously, detecting all symmetric triples takes $\mathcal{O}(n^3)$ time. After having established a new precedence constraint, distance matrix D must be updated, which can be done in $\mathcal{O}(n^2)$ time by using Algorithm 1.3.

Many consistency tests are based on lower bounds on the work that must be performed in certain time intervals $[a, b[$ with $0 \leq a < b \leq \overline{d}$. Those tests are referred to as *energetic reasoning* ("raisonnement énergétique", see Lopez

et al. 1992) or interval capacity tests (Dorndorf et al. 1999). If $b - a = 1$, we speak of *unit-interval capacity tests*. Given some schedule S, $\int_a^b r_k(S,t)dt$ is the *workload* to be processed by resource $k \in \mathcal{R}^\rho$ in time interval $[a,b[$. $R_k(b-a)$ is termed the *interval capacity* of resource k in interval $[a,b[$. The execution time of activity i in interval $[a,b[$ equals $(\min(b-a, p_i, C_i - a, b - S_i))^+$, where $(x)^+ := \max(0,x)$. It follows that the workload of resource k in interval $[a,b[$ can be written as $\sum_{i \in V^a} r_{ik}(\min(b-a, p_i, C_i - a, b - S_i))^+$. Now let

$$p_i(a,b) := (\min(b-a, p_i, EC_i - a, b - LS_i))^+ \tag{1.13}$$

denote the minimum time activity i has to be processed in interval $[a,b[$. For any time-feasible schedule $S \in \mathcal{S}_T$,

$$w_k(a,b) := \sum_{i \in V^a} r_{ik} p_i(a,b) \tag{1.14}$$

then represents a lower bound on the workload of resource $k \in \mathcal{R}^\rho$ in $[a,b[$.

Dorndorf et al. (2000*c*) have used energetic reasoning for finding further activities i, j being in disjunction. i and j are in disjunction if for all times t at which the temporal constraints allow both activities to be in progress, the combined resource requirements of i and j for some resource $k \in \mathcal{R}^\rho$ exceed the maximum residual capacity of k at time t. This condition can be formulated as follows. Activities i and j may be executed in parallel at time t if $t_1 \leq t < t_2$ where $t_1 = \max[\max(ES_i, ES_j), \min(EC_i, EC_j) - 1]$ and $t_2 = \min[\min(LC_i, LC_j), \max(LS_i, LS_j) + 1]$. The minimum workload in interval $[t, t+1[$ (or, equivalently, the minimum requirement at time t) that is due to the execution of activities from set $V^a \setminus \{i,j\}$ equals $w_k(t,t+1) - r_{ik}p_i(t,t+1) - r_{jk}p_j(t,t+1)$. Accordingly, activities i and j cannot overlap in time if there exists a resource $k \in \mathcal{R}^\rho$ such that for all $t \in [t_1, t_2[$

$$r_{ik} + r_{jk} > R_k - [w_k(t,t+1) - r_{ik}p_i(t,t+1) - r_{jk}p_j(t,t+1)] \tag{1.15}$$

For given resource $k \in \mathcal{R}^\rho$, the *core loading profile* $r_k^c : \mathbb{R} \to \mathbb{Z}_{\geq 0}$ where $r_k^c(t) = w_k(t,t+1)$ represents a lower approximation to the loading profiles $r_k(S,\cdot)$ of all time-feasible schedules $S \in \mathcal{S}_T$. By using a support-point representation of step function r_k^c, all disjunctive activities $i, j \in V^a$ satisfying (1.15) can be identified in $\mathcal{O}(|\mathcal{R}^\rho| n^2)$ time (cf. Dorndorf et al. 2000*c*). Each time a new precedence constraint has been established, we have to recalculate the earliest and latest start times of activities and to update the core loading profiles of renewable resources, which, for given distance matrix D, requires $\mathcal{O}(|\mathcal{R}^\rho| n \log n)$ time. Recall that after the addition of an arc (i,j) to project network N, the earliest and latest start times can be updated in linear time (see Remark 1.8).

The **shaving** technique is intended to tighten the time windows $[ES_i, LS_i]$ of activities $i \in V^a$ by falsifying hypothetical earliest or latest start times. We first consider the case of a hypothetical earliest start time t_i. Assume that after

the addition of the respective arc $(0, i)$ with weight t_i to project network N it holds that

$$w_k(t, t+1) > R_k \tag{1.16}$$

for some resource $k \in \mathcal{R}^\rho$ and some time t. Then the capacity of resource k is not sufficient to match the requirements for resource k at time t, i.e., we have shown that any feasible schedule S satisfies $S_i \leq t_i - 1$ (recall that $\mathrm{conv}(\mathcal{S})$ is integral). For each activity i, the values for t_i can be tested according to a binary search in set $[ES_i, LS_i] \cap \mathbb{Z}$, where t_i is decreased if the test fails in refuting the hypothesis, and increased, otherwise. Testing hypothetical latest start times can be performed analogously. When we apply the test to a given activity $i \in V^a$, we have to update the core loading profiles r_k^c at each iteration of the binary search, which again takes $\mathcal{O}(|\mathcal{R}^\rho| n \log n)$ time. Obviously, inequality (1.16) needs only to be evaluated at jump-up discontinuities of the core loading profiles, i.e., at points $t = LS_j$ $(j \in V)$. Thus, the time complexity of applying shaving to activity i is $\mathcal{O}(\log d |\mathcal{R}^\rho| n \log n)$. Since updating the core loading profiles is included in the shaving procedure, establishing a new earliest or latest start time does not incur any additional effort.

The following **unit-interval capacity test** determines points in time at which certain activities cannot be executed. Consider some real activity $i \in V^a$ that, at a given time t, is not necessarily in progress (i.e., $ES_i \leq t - p_i$ or $LS_i \geq t+1$). In this case, activity i cannot be carried out at time t if for some resource $k \in \mathcal{R}^\rho$

$$w_k(t, t+1) + r_{ik} > R_k$$

which implies $S_i \in [ES_i, t - p_i] \cup [t+1, LS_i]$ for any feasible schedule S (note that due to $p_i(t, t+1) = 0$, requirement r_{ik} does not enter into workload $w_k(t, t+1)$). Two particular cases allow the introduction of additional temporal constraints. If t is less than the earliest completion time EC_i of activity i, we obtain $S_i \geq t+1$, and if t is greater than or equal to the latest start time LS_i of i, it follows that $S_i \leq t - p_i$. Again, it suffices to consider points in time t coinciding with the latest start time LS_j of some $j \in V^a$. Accordingly, applying the unit-interval capacity test to activity i requires $\mathcal{O}(|\mathcal{R}^\rho| n)$ time. The update of core loading profiles after having established a new earliest or latest start time can again be performed in $\mathcal{O}(|\mathcal{R}^\rho| n \log n)$ time.

The **activity-interval capacity test** generalizes several consistency tests that have been devised for machine scheduling (see Domdorf et al. 1999). Let $U \subseteq V^a$ be a nonempty set of real activities and let $U', U'' \subset U$ be two subsets of U. If for some resource $k \in \mathcal{R}^\rho$, the interval capacity in the interval from the earliest start of an activity from set $U \setminus U'$ to the latest completion of an activity from set $U \setminus U''$ is less than the workload of the activities from set U, i.e.,

$$\sum_{h \in U} r_{hk} p_h > R_k \max_{\substack{g \in U \setminus U', \\ h \in U \setminus U''}} (LC_h - ES_g) \tag{1.17}$$

then there is some activity from set U' that is started first or some activity from set U'' that is completed last among the activities from set U:

$$\min_{g\in U'} S_g < \min_{h\in U\setminus U'} S_h \quad \text{or} \quad \max_{h\in U''} C_h > \max_{g\in U\setminus U''} C_g \tag{1.18}$$

For certain choices of sets U' and U'', the disjunction (1.18) results in temporal constraints (cf. Table 1.1). The corresponding consistency tests are known as input, output, input negation, and output negation tests. The computational effort associated with the different activity-interval consistency tests will arise from the analysis of the next consistency test.

Table 1.1. Specific implementations of the activity-interval capacity test

Test	(U', U'')	Temporal constraint(s)
Input	$(\{i\}, \emptyset)$	$S_j - S_i \geq 1$ for all $j \in U, j \neq i$
Output	$(\emptyset, \{j\})$	$S_j - S_i \geq p_i - p_j + 1$ for all $i \in U, i \neq j$
Input negation	$(U \setminus \{j\}, \{j\})$	$S_j \geq \min(\min_{i\in U\setminus\{j\}} ES_i, \max_{i\in U\setminus\{j\}} EC_i - p_j) + 1$
Output negation	$(\{i\}, U \setminus \{i\})$	$S_i \leq \max(\min_{j\in U\setminus\{i\}} LS_j, \max_{j\in U\setminus\{i\}} LC_j - p_i) - 1$

The **general interval capacity test** refers to time intervals $[a, b[$ for which the residual interval capacity $R_k(b-a) - w_k(a,b)$ for given resource $k \in \mathcal{R}^\rho$ is minimum. In Schwindt (1998c), Sect. 3.3, and, independently, in Baptiste et al. (1999) it has been shown that intervals $[a, b[$ with minimum residual interval capacity can be determined by investigating $\mathcal{O}(n^2)$ critical intervals (where interestingly it is not sufficient to consider only intervals whose endpoints coincide with earliest or latest start or completion times). Similarly to the shaving technique, we may establish a hypothesis on the consistency of some temporal constraint $S_i - S_j \geq t_{ji}$. If under this assumption there is a resource k with

$$\max_{0\leq a<b\leq \overline{d}} w_k(a,b) > R_k(b-a)$$

the hypothesis has been refuted and thus we can introduce the reverse temporal constraint $S_j - S_i \geq -t_{ji} + 1$. For each pair $(i,j) \in V^a \times V^a$ where $i \neq j$, a binary search in set $[d_{ji}, -d_{ij}] \cap \mathbb{Z}$ provides, within $\mathcal{O}(\log \overline{d})$ iterations, the minimum t_{ji} for which $S_i - S_j \geq t_{ji}$ can be disproved. Since for given resource k, an interval $[a, b[$ with minimum residual interval capacity can be found in $\mathcal{O}(n^2 \log n)$ time (cf. Schwindt 1998c, Sect. 3.3), the time required for applying the general interval capacity test to a given pair (i,j) is of order $\mathcal{O}(\log \overline{d} |\mathcal{R}^\rho| n^2 \log n)$.

The general interval capacity test represents a generalization of all activity-interval consistency tests listed in Table 1.1. This can be seen as follows. Consider, for given sets U, U', U'', the time interval $[a, b[$ where $a := \min_{g\in U\setminus U'} ES_g$ and $b := \max_{h\in U\setminus U''} LC_h$. Then the right-hand side of inequality (1.17) coincides with the interval capacity $R_k(b-a)$ of interval $[a, b[$. We first show that the general interval capacity test generalizes the input test. Let $U \subseteq V^a$

be a set containing two different activities i, j. According to Table 1.1, we choose $U' = \{i\}$ and $U'' = \emptyset$, i.e., $a = \min_{g \in U \setminus \{i\}} ES_g$ and $b = \max_{h \in U} LC_h$. Now assume that $S_i - S_j \geq 0$. Then $\min_{g \in U \setminus \{i\}} ES_g = \min_{g \in U} ES_g$ and thus $[ES_h, LC_h[\subseteq [a, b[$ for all $h \in U$, which implies $\sum_{h \in U} r_{hk} p_h \leq w_k(a, b)$. This means that any temporal constraint that can be deduced by using the input test also arises from applying the general interval capacity test where for each pair (i, j), time lag t_{ji} is chosen to be equal to 0. We now turn to the input negation test with $U' = U \setminus \{j\}$ and $U'' = \{j\}$, i.e., $a = ES_j$ and $b = \max_{h \in U \setminus \{j\}} LC_h$. We apply the general interval capacity test with hypothesis $S_0 - S_j \geq -\min(\min_{g \in U \setminus \{j\}} ES_g, \max_{h \in U \setminus \{j\}} EC_h - p_j)$. From $S_j \leq \min_{g \in U \setminus \{j\}} ES_g$ it follows that $ES_g \geq ES_j = a$ for all $g \in U \setminus \{j\}$, and $S_j + p_j \leq \max_{h \in U \setminus \{j\}} EC_h$ implies $LC_j \leq \max_{h \in U \setminus \{j\}} LC_h = b$. We then again have $[ES_h, LC_h[\subseteq [a, b[$ for all $h \in U$. For reasons of symmetry, the output and output negation tests can be dealt with analogously.

The **energy precedence test** has been devised by Laborie (2003). If there is an (implied) minimum time lag $d_{ij} \geq p_i$ between the starts of activities i and j, then a workload of $r_{ik} p_i$ units has to be processed on each resource $k \in \mathcal{R}^\rho$ between S_i and S_j, which takes at least $\max_{k \in \mathcal{R}^\rho} r_{ik} p_i / R_k$ units of time. Thus, for each feasible schedule S we have

$$S_j \geq \min_{i \in V^a : d_{ij} \geq p_i} ES_i + \max_{k \in \mathcal{R}^\rho} \left\lceil \sum_{i \in V^a : d_{ij} \geq p_i} r_{ik} p_i / R_k \right\rceil$$

Note that in contrast to the preceding interval capacity tests, the effectiveness of the energy precedence test is independent of the tightness of time windows $[ES_i, LS_i]$. Applying the energy precedence test to activity j requires $\mathcal{O}(|\mathcal{R}^\rho| n)$ time. If the energy precedence test is applied to all activities, the amortized computational effort per activity can be decreased to $\mathcal{O}(|\mathcal{R}^\rho| + n)$.

1.3 Cumulative-Resource Constraints

Cumulative resources represent a generalization of *nonrenewable resources* like money or raw materials, which have been studied in the context of project scheduling problems where activities can be performed in one out of several alternative execution modes differing with respect to duration and resource requirements (cf. e.g., Węglarz 1980 or Słowiński 1981). Unlike renewable resources, which are *used* during the execution time of activities and released after completion, nonrenewable resources are *consumed*. Since the availability of nonrenewable resources is nonincreasing over time, the feasibility of a resource allocation and the respective cost incurred is independent of the schedule S established and solely depends on the assignment of execution modes to activities. Thus, nonrenewable resources can be disregarded if each activity can only be performed in one mode. How to solve the mode assignment problem in case of multiple execution modes will be discussed in Section 5.3.

In practice, resources that are consumed are generally renewed later on. If the replenishment occurs during the project execution, the availability of the resource increases at certain points in time. In that case, the feasibility of a schedule generally depends on the sequence of depletions and replenishments. For example, in many real-life projects certain project activities are associated with disbursements for materials or staff leasing, and progress payments arise for completed subprojects. It may then be necessary to delay certain disbursements behind payments in order to avoid a negative cash balance. Resources that are depleted and replenished over time are called *cumulative resources*. The concept of cumulative resources has been introduced by Schwindt (1998*c*). A cumulative resource can be regarded as the inventory level in some storage facility of finite capacity. The inventory level is bounded from below by some safety stock and bounded from above by the capacity of the storage facility.

Carlier and Rinnooy Kan (1982) and Carlier (1989) have dealt with the special case where activities consume nonrenewable resources that become available at given points in time. The authors provide a polynomial-time algorithm for minimizing regular and max-separable objective functions f. In addition they show that in presence of replenishing activities the optimization problem becomes NP-hard.

Shewchuk and Chang (1995) have considered scheduling problems with *recyclable resources*, i.e., renewable resources whose availability expires after a given lifespan and which may be reused after a certain repair time (like cutters that have to be re-ground from time to time). Such a recyclable resource can be viewed as the combination of a classical renewable resource and a cumulative resource keeping the residual time before recycling becomes necessary.

Of course, cumulative resources can also be used to formulate part availability constraints arising, e.g., in construction projects or assembly manufacturing (see, e.g., Kolisch 2000, who has devised a mixed-integer linear program for scheduling in assembly environments). If certain intermediate products represent *common parts*, which are components of different subassemblies or final products, one has to decide on the sequence in which completed items of those common parts are allotted to the respective products into which they are installed (assignment-sequence problem, cf. Neumann and Schwindt 1997). The concept of cumulative resources permits to integrate the assignment-sequence problem into the resource allocation problem (see Section 6.1). A further application of cumulative resources in the context of assembly management is the modelling of spatial capacity constraints, which are due to the limited assembly area. Kolisch and Heß (2000) have developed schedule-improvement methods for assembly scheduling problems including the latter type of constraints (see also Kolisch 2001*b*, Ch. 10).

The case of general cumulative resources has been considered by Neumann and Schwindt (2002), who have discussed structural issues and have proposed a branch-and-bound algorithm for project scheduling subject to inventory constraints. Constraint-based methods for solving scheduling problems with cumulative resources have been developed by Beck (2002) and Laborie (2003).

For what follows, we assume that cumulative resources are depleted and replenished discontinuously at the occurrence of certain events like the starts and completions of real activities. Accordingly, we associate the resource requirements with events instead of real activities, and we represent the project under study by an event-on-node network (see Subsection 1.1.2). The case where cumulative resources are replenished and depleted continuously over the processing time of activities is treated in Section 5.4.

1.3.1 Resource-Feasible Schedules

Let $\mathcal{R}^\gamma$ be the set of cumulative resources. For each resource $k \in \mathcal{R}^\gamma$ a minimum inventory level or *safety stock* $\underline{R}_k \in \mathbb{Z} \cup \{-\infty\}$ and a maximum inventory level or *storage capacity* $\overline{R}_k \in \mathbb{Z} \cup \{\infty\}$ is given, where $\overline{R}_k \geq \underline{R}_k$. The (storage) *requirement* $r_{ik} \in \mathbb{Z}$ of event $i \in V^e$ for resource k equals the increase in the inventory level of resource k at the occurrence of i. r_{ik} is positive if i replenishes k and negative if i depletes k. For example, a replenishing event may represent the completion of some real activity producing an intermediate product that is stocked in resource k, whereas a depleting event may coincide with the start of some real activity consuming the intermediate product. Another example of replenishing and depleting events are progress payments received and disbursements for materials and subcontractors. Resource requirement r_{0k} can be regarded as the *initial inventory level* in resource k. We assume that

$$\underline{R}_k \leq \sum_{i \in V^e} r_{ik} \leq \overline{R}_k \quad (k \in \mathcal{R}^\gamma) \tag{1.19}$$

which ensures that the inventories $\sum_{i \in V^e} r_{ik}$ of resources $k \in \mathcal{R}^\gamma$ at the project termination neither fall below the safety stocks $\underline{R}_k$ nor exceed the storage capacities $\overline{R}_k$.

Now let $V_k^{e^-} := \{i \in V^e \mid r_{ik} < 0\}$ and $V_k^{e^+} := \{i \in V^e \mid r_{ik} > 0\}$ denote the sets of events depleting and replenishing, respectively, resource $k \in \mathcal{R}^\gamma$. Given a schedule S,

$$\mathcal{A}(S,t) := \{i \in V^e \mid S_i \leq t\}$$

is the *active set* of events that have taken place by time t and thus determine the inventory level in resource $k \in \mathcal{R}^\gamma$ at time t. By

$$r_k(S,t) := \sum_{i \in \mathcal{A}(S,t)} r_{ik}$$

we denote the inventory level of resource $k \in \mathcal{R}^\gamma$ at time t given schedule S. $r_k(S,t)$ corresponds to the cumulative resource demands for resource k in time interval $[0,t]$. The right-continuous step function $r_k(S,\cdot)$ is again called the *loading profile* of resource k. The *cumulative-resource constraints* can be written as

$$\underline{R}_k \leq r_k(S,t) \leq \overline{R}_k \quad (k \in \mathcal{R}^\gamma,\ 0 \leq t \leq \overline{d}) \tag{1.20}$$

Definition 1.19 (Resource-feasible and feasible schedules). *A schedule S satisfying the cumulative-resource constraints (1.20) is called resource-feasible with respect to cumulative resources $k \in \mathcal{R}^\gamma$. The set of all resource-feasible schedules is denoted by $\mathcal{S}_C$. $\mathcal{S} := \mathcal{S}_T \cap \mathcal{S}_C$ is the set of all feasible schedules.*

Notice that conditions (1.19) are necessary and sufficient for the existence of a resource-feasible schedule. Under conditions (1.19), a resource-feasible schedule is obtained by scheduling all events at time 0.

The basic *resource-constrained project scheduling problem with cumulative resources* can be stated as follows:

$$\boxed{\begin{array}{ll} \text{Minimize} & f(S) \\ \text{subject to} & S \in \mathcal{S}_T \cap \mathcal{S}_C \end{array}} \tag{1.21}$$

Definition 1.20 (Optimal schedule). *A feasible schedule S solving the resource-constrained project scheduling problem (1.21) is called optimal.*

Remarks 1.21.

(a) Without loss of generality we may assume that $\overline{R}_k = \infty$ for all $k \in \mathcal{R}^\gamma$ because the storage capacity of resource k can be taken into account by introducing a fictitious resource k' with $\underline{R}_{k'} = -\overline{R}_k$, $\overline{R}_{k'} = \infty$, and $r_{ik'} = -r_{ik}$ for all $i \in V^e$. Since $\mathcal{S}_R$ remains unchanged when adding some integer $r \in \mathbb{Z}$ to r_{0k}, $\underline{R}_k$, and $\overline{R}_k$, we may in addition assume that $\underline{R}_k = 0$ for all $k \in \mathcal{R}^\gamma$.

(b) The resource-constrained project scheduling problem (1.8) with renewable resources is a special case of problem (1.21). To formulate the renewable-resource constraints in terms of temporal and cumulative-resource constraints, we replace each real activity i by two events activities i^s and i^c with $d_{i^s i^c}^{min} = d_{i^s i^c}^{max} = p_i$. For each renewable resource $k \in \mathcal{R}^\rho$, we introduce a cumulative resource k' with safety stock $\underline{R}_{k'} = 0$, storage capacity $\overline{R}_{k'} = \infty$, as well as requirements $r_{0k'} = R_k$, $r_{n+1,k'} = 0$ and $r_{i^s k'} = -r_{ik}$, $r_{i^c k'} = r_{ik}$ for all real activities $i \in V^a$.

In analogy to Section 1.2, the problem without temporal constraints is termed *temporal relaxation*. The *resource relaxation* again coincides with time-constrained project scheduling problem (1.2).

The NP-hardness of finding some feasible schedule follows from the fact that first, the respective problem for the case of renewable resources is NP-hard (cf. Theorem 1.12) and that second, renewable-resource constraints can be expressed by temporal and cumulative-resource constraints without changing the order of magnitude of the problem size. The following theorem shows that, unlike the case of renewable resources, the problem remains NP-hard even if all maximum time lags are deleted.

Theorem 1.22 (Neumann and Schwindt 2002). *The following decision problem is NP-complete.*

Instance: A project with one cumulative resource of infinite storage capacity, with $\delta_{ij} \geq 0$ for all $(i,j) \in E$, $(i,j) \neq (n+1,0)$, and with an arbitrarily large project deadline $\overline{d}$.

Question: Does there exist a feasible schedule?

Proof. Clearly, the resource-feasibility of a schedule S can be verified in polynomial time by evaluating the resource constraints for all $k \in \mathcal{R}^\gamma$ and all occurrence times $t = S_i$ of events $i \in V^e$. Hence, the decision problem is contained in *NP*.

Consider an instance of the NP-complete decision problem 3-PARTITION (cf. Garey and Johnson 1979, problem SP15). Given a set $\mathcal{I}$ of 3ν indices $i = 1, \ldots, 3\nu$ with sizes $s(i) \in \mathbb{N}$ and given a bound $M \in \mathbb{N}$ such that $M/4 < s(i) < M/2$ for all $i \in \mathcal{I}$ and $\sum_{i \in \mathcal{I}} s(i) = \nu M$. The question is whether or not $\mathcal{I}$ can be partitioned into ν sets $\mathcal{I}_1, \ldots, \mathcal{I}_\nu$ such that $\sum_{i \in \mathcal{I}_\mu} s(i) = M$ for all $\mu = 1, \ldots, \nu$. An equivalent instance of our decision problem can be constructed as follows. Besides the project beginning 0 and the project termination $n+1$, set V^e contains $n = 4\nu$ events $i = 1, \ldots, 4\nu$. There is one cumulative resource with safety stock $\underline{R} = 0$ and infinite storage capacity $\overline{R} = \infty$. The requirements for the cumulative resource are $r_0 = r_{n+1} = 0$, $r_i = s(i)$ for $i = 1, \ldots, 3\nu$, and $r_i = -M$ for $i = 3\nu + 1, \ldots, 4\nu$. In addition, we define $\nu - 1$ minimum time lags $d_{i,i+1}^{min} = 1$ for $i = 3\nu + 1, \ldots 4\nu - 1$, which prevent the simultaneous occurrence of any two depleting events. Due to $\underline{R} = 0$, each unit consumed must immediately be replenished, which can be achieved precisely if the replenishing events can be assigned to the depleting events such that at each depletion time t, the total replenishment by those events $i = 1, \ldots, 3\nu$ with $S_i = t$ equals M. □

1.3.2 Forbidden Sets and Delaying Alternatives

In the case of cumulative resources, we have to consider depletions and replenishments of resources. Moreover, in addition to upper bounds $\overline{R}_k$, there are lower bounds $\underline{R}_k$ on the inventories ($k \in \mathcal{R}^\gamma$). This results in two different types of forbidden sets: so-called surplus sets if the storage capacity is exceeded and shortage sets if the inventory falls below the safety stock.

Definition 1.23 (Surplus and shortage sets). *For a resource $k \in \mathcal{R}^\gamma$, a set of events $F \subseteq V^e$ is called a k-surplus set if*

$$\sum_{i \in F} r_{ik} > \overline{R}_k$$

F is termed a minimal k-surplus set if F is a k-surplus set and there is no k-surplus set $F' \subset F$ with $F \setminus F' \subseteq V_k^{e^+}$ and no k-surplus set $F'' \supset F$ with $F'' \setminus F \subseteq V_k^{e^-}$. Likewise, a set of events $F \subseteq V^e$ is called a k-shortage set if

$$\sum_{i \in F} r_{ik} < \underline{R}_k$$

F is termed a minimal k-shortage set if F is a k-shortage set and there is no k-shortage set $F' \subset F$ with $F \setminus F' \subseteq V_k^{e^-}$ and no k-shortage set $F'' \supset F$ with $F'' \setminus F \subseteq V_k^{e^+}$. By $\mathcal{F}_k^+$ and $\mathcal{F}_k^-$ we denote the sets of all minimal k-surplus and all minimal k-shortage sets, respectively.

Note that one and the same set F can be a surplus set with respect to a resource $k \in \mathcal{R}^\gamma$ and a shortage set with respect to a different resource $k' \in \mathcal{R}^\gamma$. In the following, we refer to sets F being k-surplus or k-shortage sets for some resource $k \in \mathcal{R}^\gamma$ as *forbidden sets*. A *minimal forbidden set* is a minimal k-surplus or a minimal k-shortage set for some resource $k \in \mathcal{R}^\gamma$.

Remark 1.24. We assume that $\underline{R}_k \leq 0$ and $\overline{R}_k \geq 0$ for all $k \in \mathcal{R}^\gamma$, which ensures that $F = \emptyset$ is not a forbidden set. It follows from Remark 1.21a that this convention does not mean any loss of generality.

Similarly to the case of renewable resources, the concept of minimal delaying alternatives can be used for breaking up several minimal forbidden sets at once.

Definition 1.25 (Delaying alternative). *Let F be a k-surplus set (k-shortage set). $B \subseteq F$ is called a delaying alternative for F and k if $F \setminus B$ is not a k-surplus set (not a k-shortage set). If additionally B is $\subseteq$-minimal in the set of all delaying alternatives for F and k, we speak of a minimal delaying alternative for F and k.*

The following two conditions (1.22) and (1.23) are necessary and sufficient for a set $B \subseteq V^e$ to be a minimal delaying alternative for F and k.

$$\sum_{i \in F \setminus B} r_{ik} \leq \overline{R}_k \quad \left(\sum_{i \in F \setminus B} r_{ik} \geq \underline{R}_k \right) \tag{1.22}$$

$$\sum_{i \in F \setminus B} r_{ik} + \min_{j \in B} r_{jk} > \overline{R}_k \quad \left(\sum_{i \in F \setminus B} r_{ik} + \max_{j \in B} r_{jk} < \underline{R}_k \right) \tag{1.23}$$

From (1.23) it immediately follows that minimal delaying alternatives for surplus sets only contain replenishing events and that conversely, minimal delaying alternatives for shortage sets only contain depleting events.

To prove the basic theorem that will show how to resolve resource conflicts in a systematic way, we need the following preliminary lemma.

Lemma 1.26 (Neumann and Schwindt 2002).

(a) *For each k-surplus set F, there exists some set $F' \in \mathcal{F}_k^+$ satisfying the conditions $\emptyset \neq F' \cap V_k^{e^-} \subseteq F \cap V_k^{e^+}$ and $F' \cap V_k^{e^-} \supseteq F \cap V_k^{e^-}$.*

(b) *For each k-shortage set F, there exists some set $F' \in \mathcal{F}_k^+$ satisfying the conditions $\emptyset \neq F' \cap V_k^{e^-} \subseteq F \cap V_k^{e^-}$ and $F' \cap V_k^{e^+} \supseteq F \cap V_k^{e^+}$.*

Proof. Let F be a k-surplus set. We construct a minimal k-surplus set F' satisfying the condition of (a) as follows. We set $F' := F$ and scan the events $j \in F' \cap V_k^{e^+}$. Event j is removed from set F' if $F' \setminus \{j\}$ is still a k-surplus set. Remark 1.24 implies that the resulting set F' contains a replenishing event. Then, we scan the events $j \in V_k^{e^-} \setminus F$ and add j to set F' if $F' \cup \{j\}$ is still a k-surplus set. Consequently, for all events $j \in F'$ replenishing resource k, $F' \setminus \{j\}$ is no longer a k-surplus set and for all events $j \notin F'$ depleting resource k, $F' \cup \{j\}$ is not a k-surplus set, either. Thus, F' represents a minimal k-surplus set meeting the condition of (a). The reasoning for a k-shortage set F is analogous. □

Proposition 1.27 (Neumann and Schwindt 2002). *Let F be a k-surplus set (resp. k-shortage set). Set B represents a minimal delaying alternative for F and k if and only if B is an $\subseteq$-minimal set containing one event $j \in V_k^{e^+}$ of each minimal k-surplus set $F' \in \mathcal{F}_k^+$ with $F' \cap V_k^{e^+} \subseteq F \cap V_k^{e^+}$ and $F' \cap V_k^{e^-} \supseteq F \cap V_k^{e^-}$ (resp. one event $j \in V_k^{e^-}$ of each minimal k-shortage set $F' \in \mathcal{F}_k^-$ with $F' \cap V_k^{e^-} \subseteq F \cap V_k^{e^-}$ and $F' \cap V_k^{e^+} \supseteq F \cap V_k^{e^+}$).*

Proof. Let F be a k-surplus set for some $k \in \mathcal{R}^\gamma$.

Sufficiency: We consider a set B satisfying

$$\left.\begin{array}{l} F' \cap V_k^{e^+} \cap B \neq \emptyset \text{ for all } F' \in \mathcal{F}_k^+ \text{ with} \\ F' \cap V_k^{e^+} \subseteq F \cap V_k^{e^+} \text{ and } F' \cap V_k^{e^-} \supseteq F \cap V_k^{e^-} \end{array}\right\} \tag{1.24}$$

Now assume that $\sum_{j \in F \setminus B} r_{jk} > \overline{R}_k$. Then $F \setminus B$ is a k-surplus set, and Lemma 1.26 implies the existence of a set $F' \in \mathcal{F}_k^+$ with $F' \cap V_k^{e^+} \subseteq (F \setminus B) \cap V_k^{e^+}$ and $F' \cap V_k^{e^-} \supseteq (F \setminus B) \cap V_k^{e^-}$. From $F' \cap V_k^{e^+} \subseteq (F \setminus B) \cap V_k^{e^+}$ it then follows that $F' \cap V_k^{e^+} \cap B = \emptyset$, which contradicts the assumption. Consequently, we have $\sum_{j \in F \setminus B} r_{jk} \leq \overline{R}_k$ for any set B with property (1.24), and thus each $\subseteq$-minimal set B meeting condition (1.24) is a minimal delaying alternative.

Necessity: Now let B be a minimal delaying alternative. We assume the existence of a set $F' \in \mathcal{F}_k^+$ with $F' \cap V_k^{e^+} \subseteq F \cap V_k^{e^+}$, $F' \cap V_k^{e^-} \supseteq F \cap V_k^{e^-}$, and $F' \cap V_k^{e^+} \cap B = \emptyset$. Clearly, we have $r_{jk} > 0$ for all $j \in B$, which then implies $B \cap V_k^{e^+} = B$, i.e., $F' \cap B = \emptyset$ and $F' = F' \setminus B$. Thus, $\sum_{j \in F'} r_{jk} = \sum_{j \in F' \setminus B} r_{jk} \leq \sum_{j \in F \setminus B} r_{jk} \leq \overline{R}_k$, i.e., F' is not a k-surplus set, which contradicts the assumption. Moreover, we have $\sum_{j \in F \setminus B'} r_{jk} > \overline{R}_k$ for all subsets $B' \subset B$, which implies that for each $B' \subset B$, there is a set $F' \in \mathcal{F}_k^+$ with $F' \cap V_k^{e^+} \subseteq F \cap V_k^{e^+}$ and $F' \cap V_k^{e^+} \cap B' = \emptyset$ (see the proof of sufficiency). Thus, B is $\subseteq$-minimal in the set of all sets satisfying (1.24).

The proofs for the case of a shortage set F are analogous. □

Algorithm 1.6, which is a variant of Algorithm 1.4, shows the corresponding recursive procedure used for computing the set $\mathcal{B}$ of all minimal delaying alternatives for a forbidden set F and a resource k. Since the project beginning 0 may be contained in minimal delaying alternatives, the procedure is invoked by ***MinimalDelayingAlternatives***$(F, k, -1)$.

Algorithm 1.6. ***MinimalDelayingAlternatives***(B, k, i)

Input: A project, a forbidden set B, a resource k, an index i.
Ensure: $\mathcal{B}$ contains all minimal delaying alternatives $B' \subseteq B$ for F and k with $\min(B \setminus B') > i$.

```
if B satisfies (1.22) then (* B is delaying alternative *)
  if B satisfies (1.23) then (* B is minimal delaying alternative *)
    B := B ∪ {B};
  else
    for all j ∈ B with j > i do MinimalDelayingAlternatives(B \ {j}, k, j);
```

1.3.3 Breaking up Forbidden Sets

The following theorem provides a sufficient and necessary condition on the resource-feasibility of schedules with respect to cumulative resources.

Theorem 1.28 (Neumann and Schwindt 2002). *A schedule S is resource-feasible if and only if*

(a) *for each $F \in \mathcal{F}_k^+$ with $k \in \mathcal{R}^\gamma$, there exist two events $j \in F \cap V_k^{e^+}$ and $i \in V_k^{e^-} \setminus F$ such that $S_j \geq S_i$, and*
(b) *for each $F \in \mathcal{F}_k^-$ with $k \in \mathcal{R}^\gamma$, there exist two events $j \in F \cap V_k^{e^-}$ and $i \in V_k^{e^+} \setminus F$ such that $S_j \geq S_i$.*

Proof. Sufficiency: Let S be a schedule with $r_k(S,t) > \overline{R}_k$ for some resource $k \in \mathcal{R}^\gamma$ and some point in time $t \geq 0$. Lemma 1.26 then provides the existence of a minimal k-surplus set $F \in \mathcal{F}_k^+$ for which $\emptyset \neq F \cap V_k^{e^+} \subseteq \mathcal{A}(S,t) \cap V_k^{e^+}$ and $F \cap V_k^{e^-} \supseteq \mathcal{A}(S,t) \cap V_k^{e^-}$. Moreover, (1.19) ensures that $V_k^{e^-} \setminus F \neq \emptyset$. Due to $F \cap V_k^{e^+} \subseteq \mathcal{A}(S,t)$ we have $S_j \leq t$ for all $j \in F \cap V_k^{e^+}$. In addition, $V_k^{e^-} \setminus F \subseteq V^e \setminus \mathcal{A}(S,t)$ implies $S_i > t$ for all $i \in V_k^{e^-} \setminus F$. Thus, $S_j < S_i$ holds for all $j \in F \cap V_k^{e^+}$ and all $i \in V_k^{e^-} \setminus F$, which contradicts condition (a). Similarly it can be shown that from a shortage in some resource k at a time $t \geq 0$ it follows that condition (b) is not met.

Necessity: Let $F \in \mathcal{F}_k^+$ be a minimal k-surplus set violating (a), i.e., for all $j \in F \cap V_k^{e^+}$ and all $i \in V_k^{e^-} \setminus F$, we have $S_j < S_i$. From Remark 1.24 it follows that F contains an event replenishing resource k. Let $t := \max_{j \in F \cap V_k^{e^+}} S_j$ be the point in time at which the last replenishing event $j \in F$ occurs. Due to $F \cap V_k^{e^+} \subseteq \mathcal{A}(S,t)$ and $(V_k^{e^-} \setminus F) \cap \mathcal{A}(S,t) = \emptyset$, we obtain $r_k(S,t) \geq \sum_{j \in F} r_{jk} > \overline{R}_k$, i.e., S is not resource-feasible. The case of $F \in \mathcal{F}_k^-$ can be dealt with analogously. □

Theorem 1.28 states that any *resource conflict* caused by the occurrence of the events of some forbidden set can be resolved by adding precedence constraints $S_j \geq S_i$ to the original temporal constraints. As a consequence, the set $\mathcal{S}_C$ of all resource-feasible schedules represents a union of polyhedral

cones, and the set S of all feasible schedules again is a finite union of integral polytopes. Since each project scheduling problem with renewable-resource constraints can be represented as an equivalent project scheduling problem with cumulative-resource constraints, this union is generally disconnected.

Similarly to the case of renewable resources, forbidden sets F can be broken up by introducing (ordinary) precedence constraints or disjunctive precedence constraints. Let F be a k-surplus set for some resource k and let B be some minimal delaying alternative for F and k. Then we may either impose a set of precedence constraints

$$S_j \geq S_i \quad (j \in B)$$

between some depleting event i from set $A = V_k^{e^-} \setminus F$ and all replenishing events j from set B or, alternatively, a disjunctive precedence constraint

$$\min_{j \in B} S_j \geq \min_{i \in A} S_i$$

between sets A and B. For breaking up a k-shortage set F, we may introduce a set of precedence constraints

$$S_j \geq S_i \quad (j \in B)$$

between some replenishing event i from set $A = V_k^{e^+} \setminus F$ and all events j from a corresponding minimal delaying alternative B or by a disjunctive precedence constraint

$$\min_{j \in B} S_j \geq \min_{i \in A} S_i$$

between sets A and B. Since compared to project scheduling with renewable-resource constraints, set A typically contains a large number of elements, the use of disjunctive precedence constraints instead of ordinary precedence constraints generally leads to a tremendous decrease in the size of the enumeration tree of branch-and-bound methods.

1.3.4 Consistency Tests

As for project scheduling problems with renewable-resource constraints, consistency tests can be used to draw conclusions about temporal constraints that must necessarily be satisfied by resource-feasible schedules.

Neumann and Schwindt (2002) have used the **profile test** for calculating lower bounds on the minimum project duration. Assume that some event i cannot take place before a hypothetical earliest occurrence time t_i. We add the corresponding arc $(0, i)$ with weight t_i to project network N. Let S^k with $k \in \mathcal{R}^\gamma$ be the (generally not time-feasible) schedule where replenishments arise as early as possible and depletions occur as late as possible, i.e.,

$$\left.\begin{array}{l} S_i^k = ES_i, \text{ if } r_{ik} > 0 \\ S_i^k = LS_i, \text{ otherwise} \end{array}\right\} \quad (i \in V^e)$$

The corresponding loading profile $r_k(S^k, \cdot)$ then provides an upper approximation to the loading profile of any resource-feasible schedule. If $r_k(S^k, t) < \underline{R}_k$ for some time t, it has thus been shown that event i must arise before time t_i, i.e., $S_i \leq t_i - 1$ (notice that $\mathrm{conv}(\mathcal{S})$ is again an integral polytope). The contradiction may also be derived from comparing the storage capacity $\overline{R}_k$ of resources k with lower approximations to resource-feasible loading profiles obtained by scheduling depletions at earliest and replenishments at latest occurrence times. Similarly to the shaving technique for project scheduling with renewable resources, the tentative values for t_i can be tested according to a binary search in set $[ES_i, LS_i] \cap \mathbb{Z}$. Hence, the profile test can be implemented to run in $\mathcal{O}(\log \overline{d} |\mathcal{R}^\gamma| n \log n)$ time per event i. Recalculating the earliest occurrence times after having applied the test takes $\mathcal{O}(n)$ time (cf. Remark 1.8). Instead of earliest occurrence times we can also establish hypotheses on latest occurrence times, which may then be falsified by the same techniques.

The following **balance test** has been devised by Laborie (2003). Event $h \in V^e$ *must* occur before event $j \in V^e$ precisely if $d_{hj} > 0$, and h *may* occur before j exactly if $d_{jh} < 0$. Now let $d_{0j} > 0$. By considering all depleting events that must occur before j and all replenishing events that may occur before j, we obtain the upper bound

$$\overline{r}_k^<(j) = \sum_{h \in V_k^{e^-}: d_{hj} > 0} r_{hk} + \sum_{h \in V_k^{e^+}: d_{jh} < 0} r_{hk}$$

on the inventory level in resource k just before the occurrence of j. By rearranging the terms, $\overline{r}^<(j)$ can also be written as

$$\overline{r}_k^<(j) = \sum_{h \in V^e: d_{hj} > 0} r_{hk} + \sum_{\substack{h \in V_k^{e^+}: \\ d_{jh} < 0, d_{hj} \leq 0}} r_{hk}$$

i.e., as the sum of all requirements that must take place before j and all replenishments that possibly but not necessarily occur before j. Now assume that $\sum_{h \in V^e: d_{hj} > 0} r_{hk} < \underline{R}_k$, which implies that some of the latter replenishments must arise before j. Let $h_1, \ldots, h_\nu$ be a numbering of the events from set $V_k^{e^+}(j) := \{h \in V_k^{e^+} \mid d_{jh} < 0,\ d_{hj} \leq 0\}$ according to nondecreasing earliest occurrence times ES_h, and let μ be the smallest index such that

$$\sum_{h \in V^e: d_{hj} > 0} r_{hk} + \sum_{\lambda=1}^{\mu} r_{h_\lambda k} \geq \underline{R}_k$$

Then j must occur after time ES_{h_μ}, and we obtain the temporal constraint $S_j \geq ES_{h_\mu} + 1$. If distance matrix D is given, the time needed for applying the balance test to activity j is of order $\mathcal{O}(|\mathcal{R}^\gamma| n \log n)$. Updating matrix D after having increased ES_j takes $\mathcal{O}(n^2)$ time.

The balance test can be strengthened as follows. We consider one event $i \in V_k^{e^+}(j)$ and we assume that $S_i \geq S_j$. Then upper bound $\overline{r}_k^<(j)$ on the

inventory level in resource k at time S_j-1 can be reduced by all replenishments from set $V_k^{e^+}(j)$ which cannot occur strictly before i (and due to $S_i \geq S_j$ thus cannot occur strictly before j). This means that if

$$\overline{r}_k^<(j) - \sum_{\substack{h \in V_k^{e^+}: \\ d_{jh}<0, d_{hj} \leq 0, d_{ih} \geq 0}} r_{hk} < \underline{R}_k$$

for some $k \in \mathcal{R}^\gamma$, then it must hold that $S_j \geq S_i + 1$. This variant of the test takes $\mathcal{O}(|\mathcal{R}^\gamma|n)$ time per pair (i,j) of events.

Similar consistency tests can be performed based on the upper bound

$$\overline{r}_k^\leq(j) = \sum_{i \in V_k^{e^-}: d_{ij} \geq 0} r_{ik} + \sum_{i \in V_k^{e^+}: d_{ji} \leq 0} r_{ik}$$

on the inventory level at the occurrence of event j and the corresponding lower bounds $\underline{r}_k^<(j)$ and $\underline{r}_k^\leq(j)$.