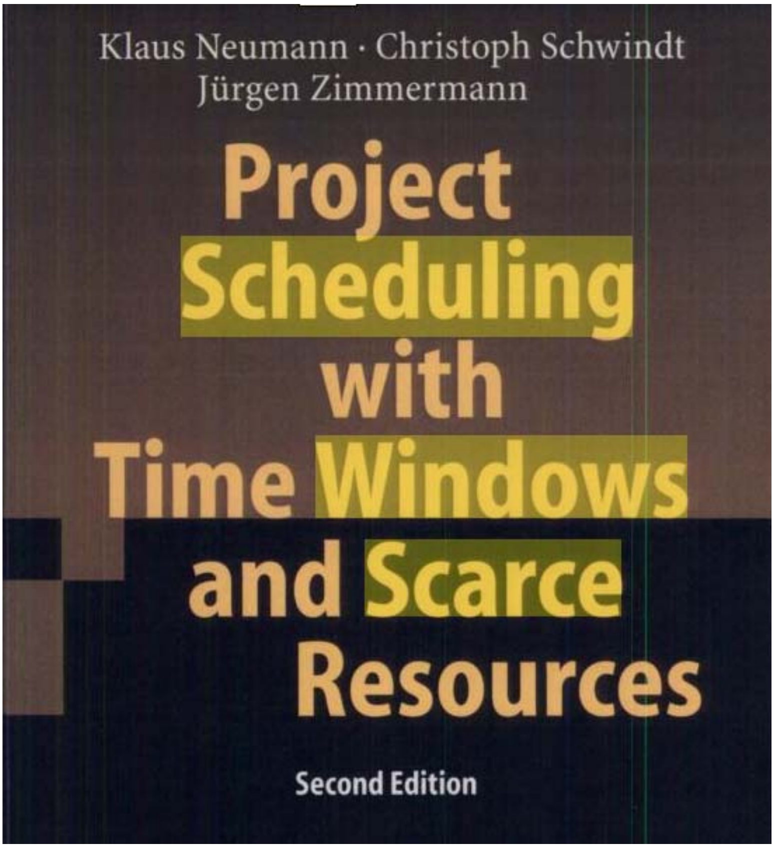

Klaus Neumann · Christoph Schwindt
Jürgen Zimmermann

Project Scheduling with Time Windows and Scarce Resources

Second Edition

2.6 Priority–rule methods

Priority–rule methods represent heuristic procedures which schedule the activities of the underlying project successively according to certain priority rules. Priority–rule methods generally run much faster than (truncated) branch–and–bound methods, however, they do not always provide a feasible schedule if there is one.

In what follows, we discuss two different types of priority–rule techniques. The *sequential* or *direct method* processes the activities or respectively nodes of the project network N one after another without evaluating the cycle structures separately. The *pr decomposition methods*, where "pr" stands for priority rule, are the analogues to the bb decomposition methods from Subsection 2.5.5 and again exploit the decomposition theorem (Theorem 2.2.3). First, a feasible subschedule S^C is determined for each cycle structure C of N. Second, in *pr decomposition method 1*, a cycle of length zero is added to each cycle structure C (which is based on subschedule S^C and contains all nodes of C) as described in Subsection 2.5.5. Finally, the resulting network $\widetilde{N}$ is treated by the direct method. In *pr decomposition method 2*, the "backward arc" within each cycle of length zero is deleted. The direct method and pr decomposition methods 1 and 2 are discussed in detail in Subsections 2.6.1 and 2.6.2, respectively.

As mentioned in Subsection 2.5.5, the addition of the cycle of length zero to each cycle structure C in decomposition method 1 can be viewed as replacing C by a single node or activity with appropriate processing time and resource requirements, which results in an acyclic "contracted" AoN network. That latter version of pr decomposition method 1 called *contraction method* has first been proposed by Zhan (1994) and Neumann and Zhan (1995). Both the direct and contraction methods have been presented and compared with each other by Brinkmann and Neumann (1996). Improved versions of both procedures have been studied in Franck and Neumann (1998). Recent versions of the direct method and pr decomposition methods 1 and 2 can be found in Franck (1999) and Franck et al. (2001a).

To construct a feasible schedule (whether for the original project network N in the direct method or for a cycle structure or the modified network $\widetilde{N}$ in the pr decomposition methods), a *serial* or a *parallel schedule generation scheme* can be used. In both generation schemes, the activities are scheduled successively, where the activity to be scheduled next is selected using some priority rule. The two generation schemes are studied in detail in Subsections 2.6.4 and 2.6.5.

2.6.1 Direct method

The direct method for a project network N consists of three steps.

In *Step 1*, we determine all cycle structures of N. This can be done by using an algorithm for finding all strong components in a directed graph, see for example, Even (1979, Sect. 3.4) or Bang–Jensen and Gutin (2000, Sect. 4.4). Although cycle structures are not evaluated separately in the direct method, we have to know them because in Step 3, once an activity from a cycle structure C has been scheduled, all remaining activities from C are scheduled before scheduling an activity outside C.

In *Step 2*, we first compute the upper bound $\overline{d}$ on the shortest project duration by (2.1.3). Second, we determine the longest path length d_{ij} from i to j in temporal scheduling network N^+ $(i, j \in V)$, for example, by the Floyd–Warshall algorithm. If $d_{ij} + d_{ji} > 0$ for some $i, j \in V$, N^+ contains a cycle of positive length. In this case, there is no feasible schedule, and the direct method is terminated. Otherwise, we perform the preprocessing phase by calling Algorithm 2.5.12 with $UB = \overline{d}$. Recall that preprocessing generally results in updating the longest path lengths d_{ij}. If $\overline{d}$ is rejected, there is no feasible schedule, and the direct method is terminated. Eventually, we compute the earliest and latest start times $ES_i = d_{0i}$ and $LS_i = -d_{i0}$, respectively ($i \in V$).

In *Step 3*, we first determine the precedence graph $G^{\prec}$ and thus the sets of immediate predecessors $Pred^{\prec}(i)$, $i \in V$, for the distance order $\prec = \prec_D$ (cf. Definition 1.4.3), for example, by the top–down algorithm discussed in Habib et al. (1993). Second, we compute a feasible schedule $(S_i)_{i \in V}$ for N using the serial or parallel generation scheme from Subsection 2.6.4 or 2.6.5, respectively. For those generation schemes, the earliest and latest start times ES_i and LS_i, respectively, and the predecessor sets $Pred^{\prec}(i)$ are needed ($i \in V$).

In short, the direct method for project network N is as follows:

(2.6.1) Algorithm: Direct method for $PS|temp|C_{\max}$.

Step 1. Determine all cycle structures of N.

Step 2. Compute upper bound $\overline{d}$ and longest path lengths d_{ij} in temporal scheduling network N^+ ($i, j \in V$). If $d_{ij} + d_{ji} > 0$ for some $i, j \in V$, then terminate (there is no feasible schedule).
Perform preprocessing with $UB = \overline{d}$. If preprocessing rejects $\overline{d}$, then terminate (there is no feasible schedule).
Determine earliest and latest start times ES_i and LS_i ($i \in V$).

Step 3. Determine predecessor sets $Pred^{\prec}(i)$ for all $i \in V$.
Find feasible schedule $(S_i)_{i \in V}$ for N by some generation scheme, where all activities of one and the same cycle structure are scheduled directly one after the other. □

2.6.2 Decomposition methods

The pr decomposition methods 1 and 2 consist of five steps each. First, we deal with decomposition method 1.

As in the direct method, *Step 1* consists of determining all cycle structures of the project network N in question. Also, *Step 2* is the same as in the direct method.

In *Step 3*, each cycle structure C with node set U^C is expanded to an AoN network N^C with node set V^C as shown in Section 2.2. Then, Steps 2 and 3 of the direct method are performed for network N^C instead for N. If for some cycle structure C, a feasible subschedule $(S_i^C)_{i \in V^C}$ cannot be found, no feasible schedule for N can be determined either, and the decomposition method is terminated.

In *Step 4*, for each cycle structure C, the nodes $i \in U^C$ are ordered according to nondecreasing S_i^C, say $0 = S_{i_1}^C \leq S_{i_2}^C \leq \cdots \leq S_{i_s}^C$. For every two successive nodes i_σ and $i_{\sigma+1}$, an arc $\langle i_\sigma, i_{\sigma+1}\rangle$ with weight $\delta_{i_\sigma i_{\sigma+1}} := S^C_{i_{\sigma+1}} - S^C_{i_\sigma}$ is introduced ($\sigma = 1, \ldots, s-1$). Moreover, the backward arc $\langle i_s, i_1\rangle$ with weight $\delta_{i_s i_1} := -S^C_{i_s}$ is added. This corresponds to the introduction of a cycle of length zero for each cycle structure C (cf. Subsection 2.5.5) and results in an AoN network $\widetilde{N}$.

Step 5 corresponds to Step 3 of the direct method with $\widetilde{N}$ instead of N. That is, a feasible schedule for $\widetilde{N}$ and thus for N as well is determined.

Pr decomposition method 2 differs from method 1 in that backward arc $\langle i_s, i_1\rangle$ is not added for any cycle structure C in Step 4. In short, pr decomposition method 1 is as follows.

(2.6.2) Algorithm: Pr decomposition method 1 for $PS|temp|C_{\max}$.

Steps 1 and *2* as in Algorithm 2.6.1.

Step 3. Expand each cycle structure C of project network N to an AoN network N^C. Perform Steps 2 and 3 of Algorithm 2.6.1 with N^C instead of N. If for some C, a feasible subschedule cannot be found, terminate.

Step 4. Add a cycle of length zero to each cycle structure C as described above, which results in AoN network $\widetilde{N}$.

Step 5. Perform Step 3 of Algorithm 2.6.1 with $\widetilde{N}$ instead of N. □

2.6.3 Priority rules

A schedule generation scheme schedules the activities i of the project in question, i.e., fixes their start times S_i, successively according to some priority rule. Several subsets of activity set V are used in a generation scheme. The *completed set* $\mathcal{C} \subseteq V$ contains all activities that have already been scheduled and are thus regarded as completed. At the beginning, we put $S_0 := 0$ and $\mathcal{C} := \{0\}$.

(2.6.3) Definition.
$S^{\mathcal{C}} = (S_i)_{i\in\mathcal{C}}$ is called a *partial schedule* corresponding to completed set $\mathcal{C} \subseteq V$ with $0 \in \mathcal{C}$ if $S_i \geq 0$ ($i \in \mathcal{C}$) and $S_0 = 0$. A partial schedule $S^{\mathcal{C}}$ is called *time–feasible* (or *resource–feasible*) if it satisfies the temporal constraints (1.2.1) with $i, j \in \mathcal{C}$ (or the resource constraints (2.1.4) with $S^{\mathcal{C}}$ instead of S, respectively). A partial schedule that is both time–feasible and resource–feasible is called *feasible*.

A generation scheme constructs a sequence of feasible partial schedules $S^{\mathcal{C}}$ beginning with $\mathcal{C} = \{0\}$ until a feasible schedule S is attained.

Set $\overline{\mathcal{C}} := V \setminus \mathcal{C}$ consists of all activities that have not been scheduled yet. The *eligible set* $\mathcal{E} \subseteq \overline{\mathcal{C}}$ contains those activities from $\overline{\mathcal{C}}$ which are eligible for scheduling. In the basic version of the serial generation scheme, which represents the simplest case, $\mathcal{E}$ contains those activities which have not been scheduled yet but all of whose immediate predecessors with respect to strict order $\prec = \prec_D$ (in short, $\prec$*-predecessors*) have already been scheduled, i.e.,

$$\mathcal{E} = \{j \in \overline{\mathcal{C}} \mid Pred^{\prec}(j) \subseteq \mathcal{C}\}$$

The activity to be scheduled next is always that activity $j^* \in \mathcal{E}$ with highest *priority* $\pi(j^*)$, where ties are broken on the basis of increasing activity numbers, that is,

$$j^* = \min\{j \in \mathcal{E} \mid \pi(j) = \operatorname*{ext}_{h \in \mathcal{E}} \pi(h)\}$$

where ext $\in \{\min, \max\}$. Once activity j^* has been scheduled, i.e., its start time S_{j^*} has been determined, the earliest and latest start times $ES_j^{\mathcal{C}}$ and $LS_j^{\mathcal{C}}$ of the activities $j \in \overline{\mathcal{C}}$ are updated:

$$\left.\begin{array}{lcl} ES_j^{\mathcal{C}} & := & \max(ES_j^{\mathcal{C}}, S_{j^*} + d_{j^*j}) \\ LS_j^{\mathcal{C}} & := & \min(LS_j^{\mathcal{C}}, S_{j^*} - d_{jj^*}) \end{array}\right\} \tag{2.6.1}$$

where $ES_j^{\{0\}} := ES_j = d_{0j}$ and $LS_j^{\{0\}} := LS_j = -d_{j0}$ for all $j \in V$.

A large number of different *priority rules* have been examined in literature for resource–constrained project scheduling problems with and without maximum time lags (see, for example, Alvarez–Valdes and Tamarit, 1989, Kolisch, 1995, Sect. 5.2, Neumann and Zhan, 1995, Franck and Neumann, 1998, and Franck, 1999, Sect. 4.3). We list some priority rules which mostly provide "good" schedules (compare also Section 2.8):

LST rule (smallest "latest start time" first): $\operatorname*{ext}_{h \in \mathcal{E}} \pi(h) = \min_{h \in \mathcal{E}} LS_h$

MST rule ("minimum slack time" first): $\operatorname*{ext}_{h \in \mathcal{E}} \pi(h) = \min_{h \in \mathcal{E}} TF_h$,
where $TF_h = LS_h - ES_h$ is again the total float or slack time of activity h

MTS rule ("most total successors" first): $\operatorname*{ext}_{h \in \mathcal{E}} \pi(h) = \max_{h \in \mathcal{E}} | Reach^{\prec}(h)|$,
where $Reach^{\prec}(h)$ is the set of nodes i for which there is a path from h to i in precedence graph $G^{\prec}$, i.e. the *reachable set* of node h in $G^{\prec}$

LPF rule ("longest path following" first): $\operatorname*{ext}_{h \in \mathcal{E}} \pi(h) = \max_{h \in \mathcal{E}} l(h)$,
where the *level* $l(h)$ of node h is the maximum number of nodes on any longest path from h to $n+1$

GRD rule ("greatest resource demand" first): $\operatorname*{ext}_{h \in \mathcal{E}} \pi(h) = \max_{h \in \mathcal{E}} p_h \sum_{k \in \mathcal{R}} r_{hk}$

RSM rule ("resource scheduling method"):
$\operatorname*{ext}_{h \in \mathcal{E}} \pi(h) = \min_{h \in \mathcal{E}} \max[0, \max_{j \in \mathcal{E} \setminus \{h\}} (ES_h + p_h - LS_j)]$

The RSM rule says that an activity which induces the smallest delay of every other activity from the eligible set is scheduled next, where we assume that each activity j must be delayed up to the end of activity h.

We distinguish between static and dynamic priority rules. We speak of a *static priority rule* if $\pi(j)$ need only be computed at the beginning of the generation scheme. For a *dynamic priority rule*, $\pi(j)$ depends on eligible set $\mathcal{E}$ or partial schedule $S^{\mathcal{C}}$ and must be computed each time the activity to be scheduled next has to be selected. The MTS, LPF, and GRD rules are static, the RSM rule is dynamic, and the LST and MST rules may be static or dynamic. If for the earliest and latest start times, we use the values $ES_i = d_{0i}$ and $LS_i = -d_{i0}$ after the preprocessing phase, the LST and MST rules are static. If we employ the updated values

$$\left.\begin{array}{rcl} ES_i^{\mathcal{C}} &=& \max[d_{0i}, \max_{j\in\mathcal{C}}(S_j + d_{ji})] \\ LS_i^{\mathcal{C}} &=& \min[-d_{i0}, \min_{j\in\mathcal{C}}(S_j - d_{ij})] \end{array}\right\} \qquad (2.6.2)$$

(compare (2.6.1)), the LST and MST rules are dynamic. In the latter case, we use the acronyms LSTd and MSTd instead of LST and MST.

2.6.4 Serial generation scheme

We first present a basic version of the serial generation scheme. After that, we discuss some modifications.

Let $\mathcal{C}$ be again the completed set, $\mathcal{E} \subseteq \overline{\mathcal{C}} = V \setminus \mathcal{C}$ be the eligible set, and $j^* \in \mathcal{E}$ be the activity to be scheduled next. To fix the start time S_{j^*} of activity j^*, we have to take into account the resource profile $r_k(S^{\mathcal{C}}, \cdot)$ of each resource $k \in \mathcal{R}$ given partial schedule $S^{\mathcal{C}} = (S_i)_{i\in\mathcal{C}}$, which is defined as follows:

$$r_k(S^{\mathcal{C}}, t) := \sum_{i\in\mathcal{A}(S^{\mathcal{C}},t)} r_{ik} \quad (k \in \mathcal{R},\ 0 \le t \le \overline{d}) \qquad (2.6.3)$$

where

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

is the active set at time t given partial schedule $S^{\mathcal{C}}$. Then

$$t^* := \min\{t \ge ES_{j^*}^{\mathcal{C}} | r_k(S^{\mathcal{C}}, \tau) + r_{j^*k} \le R_k \text{ for } t \le \tau < t + p_{j^*} \text{ and all } k \in \mathcal{R}\}$$

is the earliest resource–feasible start time of activity j^*.

If $t^* \le LS_{j^*}^{\mathcal{C}}$, then t^* is time–feasible as well, and we assign the start time $S_{j^*} := t^*$ to activity j^*. Moreover, we update the earliest and latest start times $ES_j^{\mathcal{C}}$ and $LS_j^{\mathcal{C}}$, respectively, of all activities $j \in \overline{\mathcal{C}}$ by (2.6.1).

If $t^* > LS_{j^*}^{\mathcal{C}}$, then t^* is not time–feasible, and we perform a so–called *unscheduling step*. In general, the latest start time $LS_{j^*}^{\mathcal{C}}$ results from a maximum time lag $d_{ij^*}^{max} = -d_{j^*i}$ between the start of some real activity $i \in \mathcal{C}$ and activity j^*, i.e., $LS_{j^*}^{\mathcal{C}} = S_i - d_{j^*i}$. Let

$$\mathcal{U} := \{i \in \mathcal{C} \mid LS_{j^*}^{\mathcal{C}} = S_i - d_{j^*i}\}$$

be the set of all those activities scheduled. To increase $LS^{\mathcal{C}}_{j^*}$, we unschedule all activities $i \in \mathcal{U}$ and increase start times S_i by $t^* - LS^{\mathcal{C}}_{j^*}$ for all $i \in \mathcal{U}$. In addition, we unschedule all activities $i \in \mathcal{C}$ with $S_i > \min_{h\in\mathcal{U}} S_h$, which due to the right–shift of the activities from set $\mathcal{U}$ may possibly be started earlier. If there is no real activity $i \in \mathcal{C}$ whose scheduling has led to a decrease of $LS^{\mathcal{C}}_{j^*}$, i.e., $LS^{\mathcal{C}}_{j^*} = -d_{j^*0}$ and thus $0 \in \mathcal{U}$, the generation scheme terminates because no feasible schedule can be found. It is recommended to prescribe the maximum number $\overline{u}$ of unscheduling steps, e.g., $\overline{u} = |V|$. The basic version of the serial generation scheme is then as follows, where u is the number of unscheduling steps performed, and $\pi(j)$ is computed for all $j \in \mathcal{E}$ each time an activity $j^* \in \mathcal{E}$ has to be selected.

(2.6.4) Algorithm: Serial generation scheme for $PS|temp|C_{\max}$.

```
$S_0 := 0$, $\mathcal{C} := \{0\}$, $u := 0$
While $\mathcal{C} \neq V$ do
  $\mathcal{E} := \{j \in V \setminus \mathcal{C} \mid Pred^{\prec}(j) \subseteq \mathcal{C}\}$
  For all $j \in \mathcal{E}$ compute $\pi(j)$
  $j^* := \min\{j \in \mathcal{E} \mid \pi(j) = \text{ext}_{h\in\mathcal{E}}\, \pi(h)\}$
  $t^* := \min\{t \geq ES_{j^*} | r_k(S^{\mathcal{C}},\tau)+r_{j^*k} \leq R_k$ for $t \leq \tau < t+p_{j^*}$ and all $k \in \mathcal{R}\}$
  If $t^* > LS_{j^*}$ then $u := u + 1$ and Unschedule$(j^*, t^* - LS_{j^*})$
  Else (* schedule $j^*$ at time $t^*$ *)
    $S_{j^*} := t^*$, $\mathcal{C} := \mathcal{C} \cup \{j^*\}$
    For all $j \in V \setminus \mathcal{C}$ do (* update $ES_j$ and $LS_j$ *)
      $ES_j := \max(ES_j, S_{j^*} + d_{j^*j})$
      $LS_j := \min(LS_j, S_{j^*} - d_{jj^*})$
    End (* for *)
  End (* if *)
End (* while *)
$S := S^{\mathcal{C}}$
Return $S$
```
□

(2.6.5) Algorithm: *Unschedule*(j^*, Δ).

```
$\mathcal{U} := \{i \in \mathcal{C} \mid LS_{j^*} = S_i - d_{j^*i}\}$
If $0 \in \mathcal{U}$ or $u > \overline{u}$ then terminate (* no feasible schedule is found *)
Step 1 (* right–shift of activities $i \in \mathcal{U}$ *)
For all $i \in \mathcal{U}$ do
  $ES_i := S_i + \Delta$, $\mathcal{C} := \mathcal{C} \setminus \{i\}$
  If $ES_i > -d_{i0}$ then terminate (* no feasible schedule is found *)
End (* for *)
```

```
Step 2 (* unschedule all activities $i$ with $S_i > \min_{h \in \mathcal{U}} S_h$ *)
For all $i \in \mathcal{C}$ with $S_i > \min_{h \in \mathcal{U}} S_h$ do $\mathcal{C} := \mathcal{C} \setminus \{i\}$
Step 3 (* compute $ES_j$ and $LS_j$ for all $j \in V \setminus \mathcal{C}$ again *)
For all $j \in V \setminus \mathcal{C}$ do
    $ES_j := \max[d_{0j}, \max_{h \in \mathcal{U}}(ES_h + d_{hj})]$
    $LS_j := -d_{j0}$
    For all $i \in \mathcal{C}$ do
        $ES_j := \max(ES_j, S_i + d_{ij})$
        $LS_j := \min(LS_j, S_i - d_{ji})$
    End (* for *)
End (* for *)
```
□

In Step 3 of procedure ***Unschedule***, we have to compute the times $ES_j^{\mathcal{C}}$ and $LS_j^{\mathcal{C}}$ for all $j \in \overline{\mathcal{C}}$ again which were valid before scheduling activities $h \in \mathcal{U}$ because the "history" of the scheduling process is not stored.

If the resource requirements r_{ik} of activity $i \in V$ and resource capacities R_k $(k \in \mathcal{R})$ are no longer constant but depend on time t, we only have to replace r_{ik} and R_k by $r_{ik}(t)$ and $R_k(t)$, respectively, in Algorithm 2.6.4.

An efficient way of keeping resource profiles $r_k(S^{\mathcal{C}}, \cdot)$ for resources $k \in \mathcal{R}$ is the so-called *support-point representation* of step functions. For each jump discontinuity (or equivalently, *support point*) t of function $r_k(S^{\mathcal{C}}, \cdot)$, we consider the pair (t, r_t) with $r_t = r_k(S^{\mathcal{C}}, t)$. Each support point of $r_k(S^{\mathcal{C}}, \cdot)$ corresponds to the start or completion time S_i or $S_i + p_i$, respectively, of some scheduled activity $i \in \mathcal{C}$. Thus, for partial schedule $S^{\mathcal{C}}$ and resource $k \in \mathcal{R}$, step function $r_k(S^{\mathcal{C}}, \cdot)$ can be represented by a list Λ_k of pairs (t, r_t). For simplicity, we assume that list Λ_k contains two pairs (t, r_t) with $t = S_i$ and $t = S_i + p_i$ for each scheduled activity $i \in \mathcal{C}$ (and hence, one and the same support point t may occur several times in list Λ_k, and there may be pairs (t, r_t) in list Λ_k for which t is not a support point of $r_k(S^{\mathcal{C}}, \cdot)$). When scheduling activity $j^* \in \mathcal{E}$ at time t^*, we add two pairs (t^*, r_{t^*}) and $(t^* + p_{j^*}, r_{t^* + p_{j^*}})$ to each list Λ_k $(k \in \mathcal{R})$. r_{t^*} equals r_{j^*k} if Λ_k does not contain any pair (t, r_t) with $t \le t^*$, and $r_{t^*} = r_t + r_{ik}$ otherwise, where r_t is the resource utilization at the greatest support point $t \le t^*$. If Λ_k does not contain any pair $(t', r_{t'})$ with $t' \le t^* + p_{j^*}$, we have $r_{t^* + p_{j^*}} = 0$, else $r_{t^* + p_{j^*}}$ coincides with the resource utilization $r_{t'}$ at the greatest support point $t' \le t^* + p_{j^*}$. Moreover, all pairs (t, r_t) in lists Λ_k $(k \in \mathcal{R})$ with $t^* \le t < t^* + p_{j^*}$ have to be replaced by pairs $(t, r_t + r_{j^*k})$. Given lists Λ_k for partial schedule $S^{\mathcal{C}}$, the computational effort for scheduling activity j^* is then $\mathcal{O}(|\mathcal{R}||\mathcal{C}|)$.

Let $q(|\mathcal{R}|, |V|)$ be a bivariate polynomial describing the time complexity of computing the priority $\pi(j)$ for some activity $j \in \mathcal{E}$. As shown above, the update of resource profiles $r_k(S^{\mathcal{C}}, \cdot)$ after the scheduling of activity j^* can be done in $\mathcal{O}(|\mathcal{R}||\mathcal{C}|)$ time. The time complexity for one iteration of the serial generation scheme if procedure ***Unschedule*** is not called is $\mathcal{O}(\max(|\mathcal{R}||V|, |V|q(|\mathcal{R}|, |V|)))$. One execution of ***Unschedule*** requires $\mathcal{O}(|\mathcal{R}||V|^2)$ time. Before procedure ***Unschedule*** is called for the first time or between two calls to ***Unschedule***, at most n iterations of the serial generation scheme are performed. Thus, the time complexity of Algorithm 2.6.4 is $\mathcal{O}(\overline{u}\,(|V|^2 \max[|\mathcal{R}|, q(|\mathcal{R}|, |V|)] + |\mathcal{R}||V|^2)) = \mathcal{O}(\overline{u}\,|V|^2 \max[|\mathcal{R}|, q(|\mathcal{R}|, |V|)])$.

In the basic version of the serial generation scheme, the activities of one and the same cycle structure are not necessarily scheduled directly one after the other as required for the direct method (see Step 3 in Algorithm 2.6.1). Let

$$V(i) := \begin{cases} U^C, & \text{if there is a cycle structure } C \text{ with } i \in U^C \\ \{i\}, & \text{otherwise} \end{cases} \tag{2.6.4}$$

be the node set of the strong component containing node $i \in V$. For what follows, we establish the assumption that $i \prec j$ implies $l \nprec h$ for all $h \in V(i)$ and all $l \in V(j)$. Figuratively speaking, this means that $\prec$ induces an asymmetric relation in the set of strong components $V(i)$ of project network N. It can easily be seen that the distance order $\prec_D$ complies with this assumption. The eligible set $\mathcal{E}$ is now redefined as follows: If for some cycle structure C with node set U^C, some but not all activities from C have already been scheduled, $\mathcal{E}$ consists of those activities from $\overline{\mathcal{C}} \cap U^C$ whose $\prec$–predecessors have been scheduled, i.e., $\mathcal{E} := \{j \in \overline{\mathcal{C}} \cap U^C \mid \mathit{Pred}^{\prec}(j) \subseteq \mathcal{C}\}$. Otherwise $\mathcal{E}$ contains all activities from $\overline{\mathcal{C}}$ whose $\prec$–predecessors have been scheduled and which either are outside any cycle structure or belong to some cycle structure C for which all $\prec$–predecessors $h \in V \setminus U^C$ of any activity $j \in U^C$ have been scheduled. Next, we show that the above assumption on strict order $\prec$ ensures that the eligible set is nonempty as long as $\mathcal{C} \subset V$.

For given strict order $\prec$, we define relation $\prec_C$ by

$$\left.\begin{array}{l} i \prec_C j \text{ exactly if (a) } i \prec j, \text{ or if (b) } V(i) \neq V(j) \\ \text{and } h \prec l \text{ for some } h \in V(i), l \in V(j), \\ \text{or if (c) } i \prec_C m \prec_C j \text{ for some } m \in V \end{array}\right\} \quad (i, j \in V) \tag{2.6.5}$$

Let j be a node of some cycle structure C. Then $\mathit{Pred}^{\prec_C}(j)$ consists of all nodes $i \in U^C$ with $i \prec j$ and all nodes $i \notin U^C$ for which there is a $\prec$–predecessor $l \in V(i)$ of some node $h \in U^C$. In particular, $\mathit{Pred}^{\prec_C}(j) \subseteq \mathcal{C}$ implies that all $\prec$–predecessors l outside cycle structure C of any activity $h \in U^C$ have been scheduled. The (redefined) eligible set then reads as

$$\mathcal{E} := \begin{cases} \{j \in \overline{\mathcal{C}} \cap U^C \mid \mathit{Pred}^{\prec_C}(j) \subseteq \mathcal{C}\}, & \text{if some but not all nodes from some cycle structure } C \text{ have been scheduled} \\ \{j \in \overline{\mathcal{C}} \mid \mathit{Pred}^{\prec_C}(j) \subseteq \mathcal{C}\}, & \text{otherwise} \end{cases} \tag{2.6.6}$$

Since we have assumed that $i \prec j$ excludes $l \prec h$ for any $h \in V(i)$, $l \in V(i)$, relation $\prec_C$ is asymmetric. Moreover, $\prec_C$ is transitive by definition and thus represents a strict order in set V. Let i and j be two different nodes of one and the same cycle structure of N. Then $i \prec_C j$ exactly if $i \prec j$, and thus $\mathcal{E} \neq \emptyset$ provided that $\mathcal{C} \cap U^C \neq \emptyset$ and $\overline{\mathcal{C}} \cap U^C \neq \emptyset$ for some cycle structure C in N (first part of (2.6.6)). Furthermore, the asymmetry of $\prec_C$ implies that $\mathcal{E} \neq \emptyset$ if there is no cycle structure C with $\mathcal{C} \cap U^C \neq \emptyset$ and $\overline{\mathcal{C}} \cap U^C \neq \emptyset$ (second part of (2.6.6)).

(2.6.6) Example.
We illustrate the serial generation scheme by scheduling the project of Example 2.5.13, for which we have already performed the preprocessing phase in Subsection 2.5.2. There is one resource with capacity $R = 3$. The project network is depicted in Fig. 2.6.1, and the longest path lengths after preprocessing are given by Table 2.5.4. The corresponding earliest and latest start times of activities $i \in V$ are listed in Table 2.6.1. Fig. 2.6.2 shows the precedence graph for distance order $\prec_D$, where $cr(\prec_D)$ is again the covering relation of strict order $\prec_D$ (cf. Definition 1.4.1). We schedule the eligible activities $j \in \mathcal{E}$ according to (dynamic) rule LSTd.

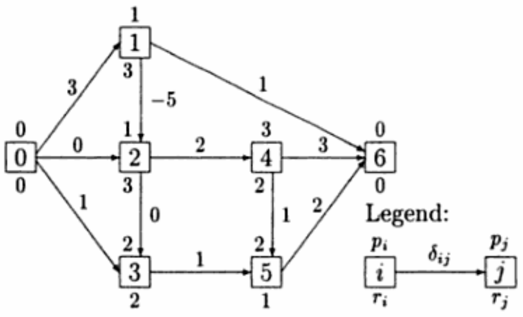


Figure 2.6.1: AoN project network with a single resource

Table 2.6.1: Earliest and latest start times

i	0	1	2	3	4	5	6
ES_i	0	3	0	1	2	4	6
LS_i	0	10	7	10	10	11	13

Table 2.6.2 shows the iterations of Algorithm 2.6.4, where "It." designates the iteration number and "Update" refers to increasing earliest start times and decreasing latest start times after the scheduling of selected activity j^* at time t^*.

In each iteration, one eligible activity j^* with minimum latest start time is scheduled, where ties are broken on the basis of increasing activity numbers. Since in each iteration, the earliest resource–feasible start time t^* for

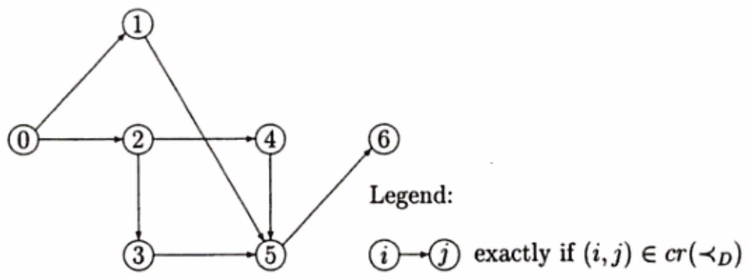


Figure 2.6.2: Precedence graph $G^{\prec_D}$

Table 2.6.2: Serial generation scheme – LSTd rule

It.	$\mathcal{C}$	$\mathcal{E}$	j^*	t^*	Update
1	$\{0\}$	$\{1,2\}$	2	0	$LS_1 := 5$
2	$\{0,2\}$	$\{1,3,4\}$	1	3	
3	$\{0,1,2\}$	$\{3,4\}$	3	1	
4	$\{0,1,2,3\}$	$\{4\}$	4	4	$ES_5 := 5$, $ES_6 := 7$
5	$\{0,1,\ldots,4\}$	$\{5\}$	5	5	
6	$\{0,1,\ldots,5\}$	$\{6\}$	6	7	

activity j^* is always less than or equal to latest start time $LS^{\mathcal{C}}_{j^*}$, procedure ***Unschedule*** is not called. As we will see later on, this implies that the generated schedule $S = (0,3,0,1,4,5,7)$ is active. Fig. 2.6.3 shows the corresponding Gantt chart. Schedule S is optimal because the project duration $S_{n+1} = 7$ coincides with lower bounds LBD and LBW (see Example 2.5.15). □

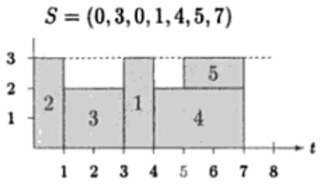


Figure 2.6.3: Gantt chart for generated schedule

2.6.5 Parallel generation scheme

Recall that in the serial generation scheme, exactly one activity j^* is scheduled in each iteration. In the parallel generation scheme, more than one activity can be scheduled in each iteration. At first, we determine an auxiliary set $\mathcal{G}$, which corresponds to eligible set $\mathcal{E}$ in the serial scheme, and the smallest of the earliest start times of the activities from $\mathcal{G}$, say t^+. Eligible set $\mathcal{E}$ then consists of those activities from $\mathcal{G}$ whose earliest start times equal t^+. The basic version of the parallel generation scheme is then as follows:

(2.6.7) Algorithm: Parallel generation scheme for $PS|temp|C_{\max}$.

```
$S_0 := 0, \mathcal{C} := \{0\}, u := 0$
While $\mathcal{C} \neq V$ do
   $\mathcal{G} := \{j \in V \setminus \mathcal{C} \mid Pred^{\prec}(j) \subseteq \mathcal{C}\}$
   $t^+ := \min_{j\in\mathcal{G}} ES_j$
   $\mathcal{E} := \{j \in \mathcal{G} \mid ES_j = t^+\}$
   While $\mathcal{E} \neq \emptyset$ do
      For all $j \in \mathcal{E}$ compute $\pi(j)$
      $j^* := \min\{j \in \mathcal{E} \mid \pi(j) = \text{ext}_{h\in\mathcal{E}}\, \pi(h)\}$
      $\mathcal{E} := \mathcal{E} \setminus \{j^*\}$
      $t^* := \min\{t \geq ES_{j^*} \mid r_k(S^{\mathcal{C}},\tau) + r_{j^*k} \leq R_k$ for $t \leq \tau < t + p_{j^*}$ and all $k \in \mathcal{R}\}$
      If $t^* > LS_{j^*}$ then $u := u+1$, Unschedule$(j^*, t^* - LS_{j^*})$, and $\mathcal{E} := \emptyset$
      Else
         If $t^* > t^+$ then
            For all $j \in V \setminus \mathcal{C}$ do $ES_j := \max(ES_j, t^* + d_{j^*j})$
         Else (* schedule $j^*$ at time $t^*$ *)
            $S_{j^*} := t^*, \mathcal{C} := \mathcal{C} \cup \{j^*\}$
            For all $j \in V \setminus \mathcal{C}$ do (* update $ES_j$ and $LS_j$ *)
               $ES_j := \max(ES_j, S_{j^*} + d_{j^*j})$
               $LS_j := \min(LS_j, S_{j^*} - d_{jj^*})$
            End (* for *)
         End (* if *)
      End (* if *)
   End (* while *)
End (* while *)
$S := S^{\mathcal{C}}$
Return $S$
```
□

Similarly to the serial generation scheme, it can be shown that the time complexity of Algorithm 2.6.7 is $\mathcal{O}(\overline{u}\,|V|^2 \max[|\mathcal{R}||V|, q(|\mathcal{R}|, |V|)])$.

To guarantee that the activities of one and the same cycle structure are scheduled directly one after another, we have to modify the basic version of the parallel generation scheme analogously to the serial scheme. Moreover, the parallel scheme can be adapted to the case where r_{ik} and R_k $(i \in V,\ k \in \mathcal{R})$ are no longer constant in the same way as the serial scheme in Subsection 2.6.4.

(2.6.8) Remark.
If the serial or the parallel generation scheme yield a feasible schedule S without calling procedure ***Unschedule***, then for each activity $i \in V$, S_i is the earliest resource–feasible start time of activity i. Hence, there is no global left–shift from S, and S represents an active schedule.

(2.6.9) Example.
We return to the project of Example 2.6.6 for which we construct a schedule using the parallel generation scheme in combination with (static) rule GRD. Again, we start after the preprocessing phase. The resource demands $r_i p_i$ of activities $i \in V$ are listed in Table 2.6.3.

Table 2.6.3: Resource demands

i	0	1	2	3	4	5	6
$r_i p_i$	0	3	3	4	6	2	0

Table 2.6.4 shows the individual iterations of applying Algorithm 2.6.2. Bold entries for j^* and t^* indicate that the selected activity $j^* \in \mathcal{E}$ is scheduled at time t^*.

After having scheduled activities 2, 3, and 4, the earliest time– and resource–feasible start time t^* for activity 1 in iteration 4 equals $6 > LS_1 = 5$, and thus activity 1 cannot be started on time (cf. Fig. 2.6.4a). Accordingly, we unschedule and right–shift activities $2, 3 \in \mathcal{U}$ by $t^* - LS_1 = 1$ time unit and unschedule all activities $i \in \mathcal{C}$ with $S_i > S_2 = 0$, that is, activity 4 is deleted from set $\mathcal{C}$. The unscheduling of activities 2, 3, and 4 must be repeated after iterations 8 (cf. Fig. 2.6.4b) and 12 (cf. Fig. 2.6.4c).

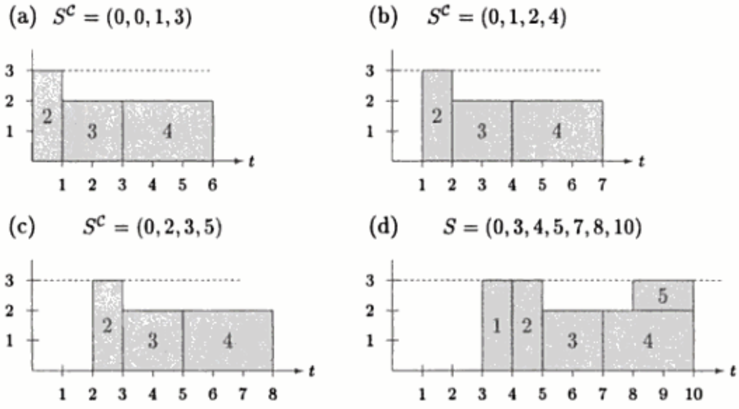


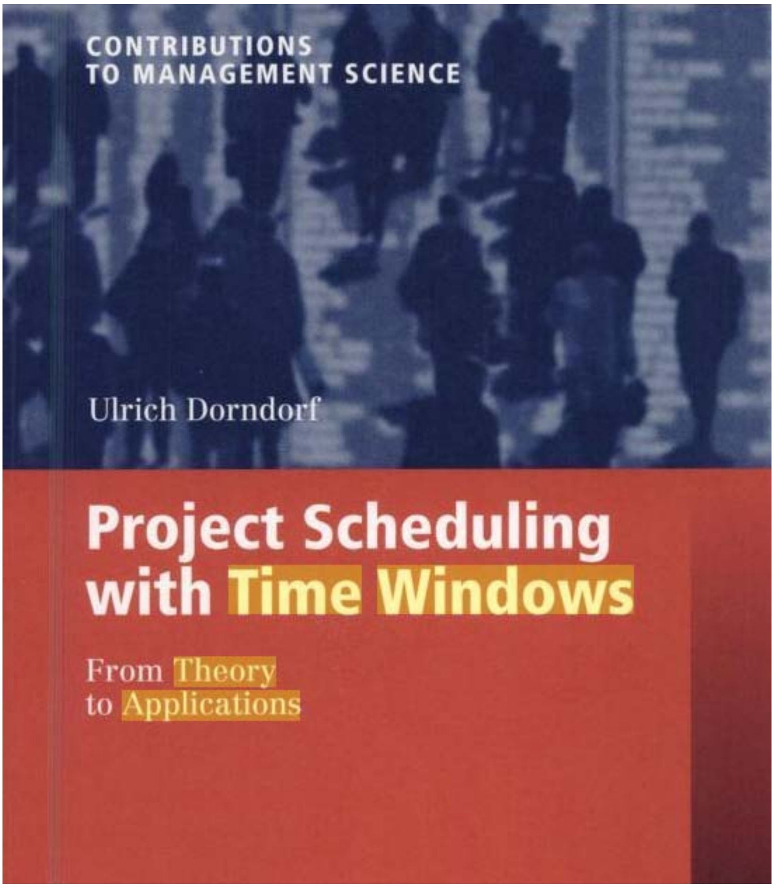
Figure 2.6.4: Gantt charts for generated (partial) schedules

Since in iteration 13, we have $ES_1 = ES_2 = 3$ and $\pi(1) = \pi(2) = 3$, activity 1 is scheduled before activity 2, and no further unscheduling step is needed

Table 2.6.4: Parallel generation scheme – GRD rule

It.	$\mathcal{C}$	$\mathcal{G}$	t^+	$\mathcal{E}$	j^*	t^*	Update
1	$\{0\}$	$\{1,2\}$	0	$\{2\}$	**2**	**0**	$LS_1 := 5$
2	$\{0,2\}$	$\{1,3,4\}$	1	$\{3\}$	**3**	**1**	
3	$\{0,2,3\}$	$\{1,4\}$	2	$\{4\}$	4	3	$ES_4 := 3$
4	$\{0,2,3\}$	$\{1,4\}$	3	$\{1,4\}$	**4**	**3**	
				$\{1\}$	1	6	
Unschedule$(1,1)$: $ES := (0,3,1,2,3,5,7)$, $LS := (0,10,7,10,10,11,13)$							
5	$\{0\}$	$\{1,2\}$	1	$\{2\}$	**2**	**1**	$LS_1 := 6$
6	$\{0,2\}$	$\{1,3,4\}$	2	$\{3\}$	**3**	**2**	
7	$\{0,2,3\}$	$\{1,4\}$	3	$\{1,4\}$	4	4	$ES_4 := 4$
				$\{1\}$	1	4	$ES_1 := 4$
8	$\{0,2,3\}$	$\{1,4\}$	4	$\{1,4\}$	**4**	**4**	
				$\{1\}$	1	7	
Unschedule$(1,1)$: $ES := (0,3,2,3,4,6,8)$, $LS := (0,10,7,10,10,11,13)$							
9	$\{0\}$	$\{1,2\}$	2	$\{2\}$	**2**	**2**	$LS_1 := 7$
10	$\{0,2\}$	$\{1,3,4\}$	3	$\{1,3\}$	**3**	**3**	
				$\{1\}$	1	5	$ES_1 := 5$
11	$\{0,2,3\}$	$\{1,4\}$	4	$\{4\}$	4	5	$ES_4 := 5$
12	$\{0,2,3\}$	$\{1,4\}$	5	$\{1,4\}$	**4**	**5**	
				$\{1\}$	1	8	
Unschedule$(1,1)$: $ES := (0,3,3,4,5,7,9)$, $LS := (0,10,7,10,10,11,13)$							
13	$\{0\}$	$\{1,2\}$	3	$\{1,2\}$	**1**	**3**	
				$\{2\}$	2	4	$ES_2 := 4$, $ES_3 := 5$ $ES_4 := 6$, $ES_5 := 8$, $ES_6 := 10$
14	$\{0,1\}$	$\{2\}$	4	$\{2\}$	**2**	**4**	
15	$\{0,1,2\}$	$\{3,4\}$	5	$\{3\}$	**3**	**5**	
16	$\{0,1,2,3\}$	$\{4\}$	6	$\{4\}$	4	7	$ES_4 := 7$
17	$\{0,1,2,3\}$	$\{4\}$	7	$\{4\}$	**4**	**7**	
18	$\{0,1,\ldots,4\}$	$\{5\}$	8	$\{5\}$	**5**	**8**	
19	$\{0,1,\ldots,5\}$	$\{6\}$	10	$\{6\}$	**6**	**10**	

until the algorithm terminates. Fig. 2.6.4d shows the resulting schedule S. Notice that S is not active (activities 2 and 3 can be left–shifted by four, activities 4 and 5 by three units of time). The example demonstrates that priority rule GRD (and similarly priority rules MTS and LPF), which does not take into account the latest start times LS_j of eligible activities $j \in \mathcal{E}$, tends to reach a deadlock where partial schedule $S^{\mathcal{C}}$ cannot be extended to a feasible schedule without unscheduling some activities $i \in \mathcal{C}$. This observation will be confirmed by the results of the experimental performance analysis presented in Section 2.8. □


CONTRIBUTIONS
TO MANAGEMENT SCIENCE

Ulrich Dorndorf

Project Scheduling with Time Windows

From Theory
to Applications

Chapter 1

Introduction

1.1 Motivation and Objectives

Project scheduling is concerned with the allocation of resources over time to perform a collection of activities. The decision models that fit within this framework cover a multitude of practical problems that arise, for example, in such diverse areas as research and development, software engineering, construction engineering, repair and maintenance, as well as make-to-order and small batch production planning.

A project is a one-of-a-kind undertaking with specific objectives that has to be performed within a certain time-frame and with limited resource supply. Its management roughly consists of (1) a project definition and data acquisition phase, (2) a scheduling phase and (3) an execution and termination phase during which the schedule is realised and the performance is analysed.

This work deals with the scheduling aspect. The aim is to develop methods for finding an optimal schedule for a project; this involves the assignment of activities to resources and the definition of exact activity start and completion times, a task that is generally difficult whenever multiple activities simultaneously compete for the same resources. We will not address the topics related to the conception, selection, and definition of a project, but will rather assume that the project structure is given, including data on resource availabilities and requirements as well as the necessary processing times. Likewise, we will not deal with the issues that typically arise during the realisation phase of a project.

We shall investigate a very general class of deterministic project scheduling problems that is expressive enough to capture many features commonly found in practical problems, such as precedence constraints, activity time windows, fixed activity start times, synchronisation of start or finish times, maximal or minimal activity overlaps, non-delay execution of activities, setup times, or time varying resource supply and demand.

In the basic model, technological or organisational requirements are represented through generalised precedence constraints that allow to specify minimal and/or maximal time lags, or *time windows*, between any pair of activities. An activity may require different amounts of several resource types. Resource requirements and availabilities may vary in discrete steps over time. While we usually consider the objective of minimising the overall completion time of a project, most of the results apply at least for any performance measure that is a non-decreasing function of the completion or start times of the activities. We will also address multi-mode scheduling, i.e., the situation where a choice must be made between several modes in which an activity may be processed, reflecting time-resource or resource-resource tradeoffs. Due to its generality, the basic model also covers many difficult special problems that have been extensively studied in scheduling research, for example, shop scheduling problems (Błażewicz et al. 1996).

Throughout this work, we study deterministic project scheduling problems, where all parameters that define a problem instance are known with certainty in advance. Deterministic scheduling models are best suited if any possible random influences in the project execution phase can be expected to be low, and if the problem parameters can thus be estimated with high accuracy. This may, for instance, be the case if the activities of a project show a high degree of similarity with previous projects. In situations where the problem parameters are difficult to estimate and are subject to significant random influences, the use of deterministic scheduling techniques may, however, be problematic. As a typical example, deterministic scheduling in the presence of stochastic activity processing times generally leads to an underestimation of the expected project duration, as already observed by Fulkerson (1962).

The first models and methods for dealing with large scale projects have been devised in the late 1950's and early 1960's. The well known Critical Path Method (CPM, Kelley 1961) and the Metra Potential Method (MPM, Roy 1962) have been designed for deterministic project scheduling with ordinary or generalised precedence constraints, respectively, while the Project Evaluation and Review Technique (PERT, Malcolm et al. 1959) considers probabilistic activity processing times; the Graphical Evaluation and Review Technique (GERT, Pritsker and Happ 1966) additionally takes probabilistic precedence relations into account. These approaches have received great attention in the following years. In the early 1970's, Davis (1973) already reported more than 15 books and 300 papers on the subject.

The original models and methods simplified the problem by concentrating only on temporal constraints, i.e., by assuming that the availability of resources is not a limiting factor. Beginning in the late 1960's, the models were extended by additionally considering scarcity of resources. In order to distinguish between the classic CPM, MPM, and PERT or GERT models on the one hand and models that consider limited resource availability on the other hand, the latter are usually referred to as *resource-constrained*. The underlying problems are much more difficult to solve, as the computational effort for finding an optimal solution usually grows exponentially with the problem size. For a long time, this has prohibited the use of exact algorithms for scheduling large practical projects with resource constraints.

In the past years, interest and research efforts in the field of resource-constrained project scheduling have strongly increased, and many new modelling concepts and algorithms have been developed. Overviews of the advances in models and solution methods are given in the survey papers of Brucker et al. (1999), Herroelen et al. (1998), Kolisch and Padman (2001), Drexl et al. (1997), Elmaghraby (1995), Özdamar and Ulusoy (1995), Icmeli et al. (1993), or Domschke and Drexl (1991). A gentle introduction to network models for project planning and control is given by Elmaghraby (1977). Descriptions of the basic classic project scheduling models for the temporal analysis of projects can be found in many introductory Management Science textbooks (e.g. Domschke and Drexl 1998). Applications within the area of production planning have been described, e.g., by Hax and Candea (1984) and Günther and Tempelmeier (2000); Drexl et al. (1994) discuss a special type of project scheduling software, called Leitstand system, for make-to-order manufacturing management.

The resource-constrained project scheduling problems studied in this work can be understood as extensions of the basic problem covered by the Metra Potential Method. Due to the general form of the temporal constraints, the resource-constrained version of the problem is particularly difficult to solve. Even the question for the existence of a feasible schedule can in general only be answered with exponentially growing effort. This may be one the main reasons why, despite the expressiveness and high practical relevance of the models, very few attempts have so far been made to design solution procedures for this class of problems.

The main objective of this work is to help overcome this deficiency by developing effective and efficient solution methods. The focus will be on the design and evaluation of exact branch-and-bound algorithms for finding optimal schedules, but we shall also study the performance of heuristics based upon truncated versions of these procedures.

The scheduling methods that will be developed make use of a general purpose problem solving paradigm that originated in the area of Artificial Intelligence. *Constraint propagation* is an elementary technique for simplifying difficult search and optimisation problems by exploiting implicit constraints that are discovered through the repeated analysis of the domains of decision variables and the interrelation between the variables and domains that is induced by the constraints. In the past years, constraint propagation techniques have been applied with growing success for solving a number of difficult, idealised scheduling problems, mostly in the area of machine scheduling. The successful application for solving special cases of the general problem class studied here suggests that the approach may also be valuable in this context. As a second objective of this work, we shall therefore study the application of constraint propagation techniques in project scheduling.

A third objective is to demonstrate the practical relevance of the approach taken in this work. To this end we shall describe possible applications of the models and methods and extensions thereof in the area of airport operations management.