Chapter 3

Resource-Constrained Project Scheduling

As shown in Section 2.1, even special cases of the RCPSP are NP-hard, i.e. with high probability they cannot be exactly solved in polynomial time. Thus, for larger instances in practice approximation algorithms have to be applied. In this chapter we present different solution methods for the RCPSP with the makespan objective and some of its generalizations (like the multi-mode RCPSP).

After introducing some basic concepts in Section 3.1, in Section 3.2 foundations for heuristic procedures like priority-based heuristics, local search algorithms, genetic algorithms, or ant colony optimization are described. In Section 3.3 different mixed integer linear programming formulations are presented, in Section 3.4 the RCPSP is formulated as a satisfiability problem. Since in recent years solvers for mixed integer linear programs or satisfiability problems became much more efficient, these formulations may be an alternative for calculating exact (or heuristic) solutions.

In Section 3.5 more general objective functions for problems without resource constraints are considered. In Section 3.6 we discuss constraint propagation techniques, which are useful to reduce the solution space. Section 3.7 is devoted to methods for calculating lower bounds, in Section 3.8 different branch-and-bound algorithms are discussed.

3.1 Basics

In this section we introduce some basic concepts (temporal analysis, classification of schedules), which are useful for many algorithms solving the RCPSP.

3.1.1 Temporal analysis

Usually, before solution algorithms are applied to the RCPSP, a temporal analysis is performed. If no generalized precedence relations are given, the activity-on-

DOI 10.1007/978-3-642-23929-8_3, © Springer-Verlag Berlin Heidelberg 2012

node network $G = (V, A)$ of an RCPSP-instance must be acyclic (since otherwise no feasible schedule exists). Thus, the activities can be numbered according to a topological ordering, i.e. we may assume $i < j$ for all precedences $i \rightarrow j \in A$.

We assume $S_0 = 0$, i.e. no activity can start before time $t = 0$. Furthermore, an upper bound UB for the $C_{\max}$-value is given, i.e. we have $S_{n+1} \leq UB$. For each activity i we define a

- **head** r_i as a lower bound for the earliest starting time of i, and a
- **tail** q_i as a lower bound for the length of the time period between the completion time of i and the optimal makespan.

Heads and tails may be calculated as follows. The length of a (directed) path P from activity i to j in G is the sum of processing times of all activities in P excluding the processing time of j. A valid head r_i is the length of a longest path from the dummy activity 0 to i. Symmetrically, a valid tail q_i is the length of a longest path from i to the dummy node $n + 1$ minus the processing time p_i of i. Given an upper bound UB, a deadline for the completion time of activity i in any schedule with $C_{\max} \leq UB$ may be defined by $d_i := UB - q_i$. Thus, activity i must be completely processed within its **time window** $[r_i, d_i]$ in any feasible schedule with $C_{\max} \leq UB$.

In the acyclic graph $G = (V, A)$ heads r_i and deadlines d_i for the activities $i = 0, 1, \ldots, n, n+1$ may be calculated in a topological ordering by the recursions

$$\begin{array}{lll} r_0 := 0; & r_i := \max\limits_{\{j \mid j \rightarrow i \in A\}} \{r_j + p_j\} & \text{for } i = 1, 2, \ldots, n+1 \text{ and} \\ d_{n+1} := UB; & d_i := \min\limits_{\{j \mid i \rightarrow j \in A\}} \{d_j - p_j\} & \text{for } i = n, n-1, \ldots, 0. \end{array}$$

A longest path CP from 0 to $n+1$ is also called a **critical path**. The length of a critical path r_{n+1} is equal to the minimal makespan if all resource constraints are relaxed. For the problem with resource constraints the critical path length is a lower bound for the optimal makespan. The schedule in which all activities i start as early as possible at $S_i := r_i$ is called **earliest start schedule**, the schedule in which all activities i start as late as possible at $S_i := d_i - p_i$ is called **latest start schedule**. Both schedules are feasible if all resource constraints are relaxed. In this case for each activity i the so-called **slack** $s_i := d_i - p_i - r_i$ defines the amount of time by which i can be moved in any feasible schedule with $C_{\max} \leq UB$. Activities with $s_i = 0$ are called **critical** since they cannot be moved in any feasible schedule with $C_{\max} \leq UB$.

Example 3.1 : Consider the project with $n = 6$ activities shown in Figure 3.1.

For $UB = 7$ we obtain the following heads r_i, tails q_i, deadlines d_i and slacks s_i:

i	1	2	3	4	5	6
r_i	0	0	2	3	2	5
q_i	3	5	2	0	2	0
d_i	4	2	5	7	5	7
s_i	1	0	2	1	0	0

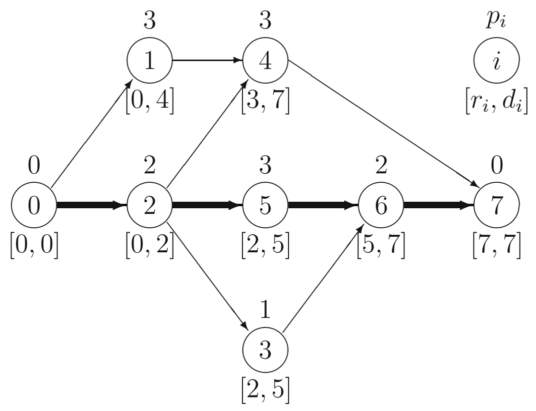


Figure 3.1: Time windows for a project with $n = 6$ and $UB = 7$

For example, for activity $i = 4$ we get

$$r_4 := \max\{r_1 + p_1, r_2 + p_2\} = \max\{0 + 3, 0 + 2\} = 3,$$

for activity $i = 2$ we get

$$d_2 := \min\{d_5 - p_5, d_4 - p_4, d_3 - p_3\} = \min\{5 - 3, 7 - 3, 5 - 1\} = 2.$$

Activities 2, 5, and 6 are critical, a corresponding critical path is $CP = (0 \to 2 \to 5 \to 6 \to 7)$ with length $r_7 = 7$. □

Usually, in an activity-on-node network the vertices (corresponding to the activities) are weighted with the processing times. In an equivalent alternative representation instead of the vertices the arcs may be weighted. Then, all arcs $(i, j) \in A$ get the weight p_i. In such a model the vertices correspond to start events of the activities. This alternative representation may especially be used when time-lags d_{ij} are given. In this situation the arcs (i, j) get the weights d_{ij}.

3.1.2 A classification of schedules

In the following we give a classification of schedules for the RCPSP, which is useful to compare different exact and heuristic algorithms. Usually, the set of considered schedules is restricted to the set of **feasible** schedules, which are feasible with respect to the precedence and resource constraints. Moreover, if a regular objective function has to be minimized, activities may be shifted to the left without increasing the objective function value.

More specifically, a **left shift** of an activity i in a schedule S transforms S into a feasible schedule S' with $S'_i < S_i$ and $S'_j = S_j$ for all other activities $j \neq i$. If $S'_i = S_i - 1$ holds, the shift is called an **one-period left shift**. A **local left shift** is a shift which can be obtained by successive one-period left shifts, i.e. all intermediate schedules (in which the starting time of i is successively decreased by one time unit) have to be feasible. A left shift which is not a local left shift is called a **global left shift**. Then at least one intermediate schedule must be infeasible with respect to the resource constraints.

According to the notion of left shifts, schedules may be classified as follows. A feasible schedule is called **semi-active** if for all activities no local left shift is feasible. A feasible schedule is called **active** if for all activities no local or global left shift can be performed. A feasible schedule is called a **non-delay** schedule if for all activities no local or global left shift can be performed even if activities can be preempted at integer time points (i.e. only parts of activities are shifted). In non-delay schedules no resource is left idle when an activity can be processed.

Example 3.2: Consider the instance with $n = 5$ activities and one resource with constant capacity $R_1 = 2$ shown in Figure 3.2.

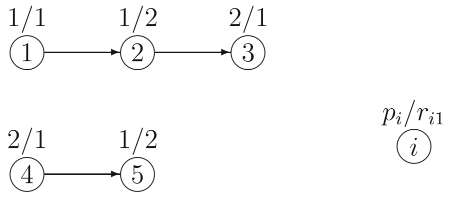


Figure 3.2: An RCPSP instance with $n = 5$ activities

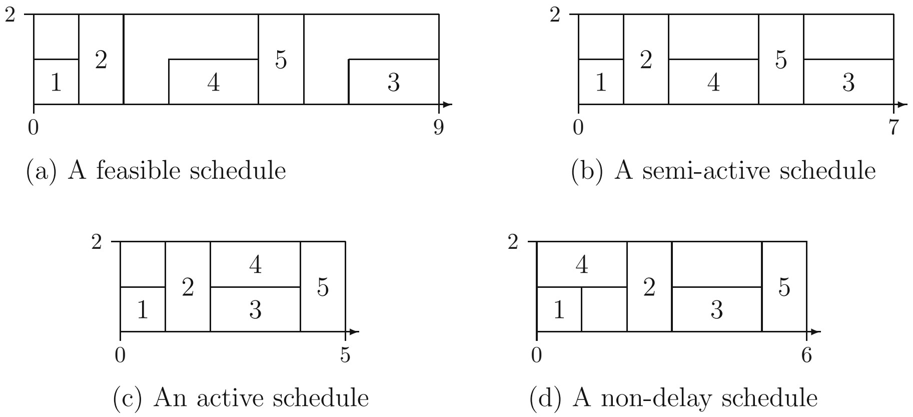


Figure 3.3: Four different schedules

A feasible schedule for this instance is depicted in Figure 3.3(a). This schedule is not semi-active since activities 4, 5 and 3 can be locally shifted to the left. By performing one-period left shifts for activities 4, 5 and two one-period left shifts of activity 3 (all intermediate schedules are feasible) the semi-active schedule shown in (b) is derived. For this schedule no further local left shift is possible, but activity 3 can be globally shifted to the left by three time periods. As a result the active schedule in (c) is obtained (which is the unique optimal solution for the given instance). This schedule is not a non-delay schedule, since the first unit of activity 4 can be feasibly shifted to the left starting at time 0. On the other hand, the schedule in (d) is a non-delay schedule, since no part of activity 3 can be feasibly shifted to the left due to the precedence relation $2 \rightarrow 3$. $\square$

According to the previous definitions the non-delay schedules are a subset of all active schedules and the active schedules are a subset of the semi-active schedules. Moreover, each feasible schedule can be transformed into a semi-active schedule by a series of local left shifts and into an active schedule by local and global left shifts.

If optimization algorithms are applied, it is often advantageous to use a small search space. Unfortunately, if we want to guarantee an optimal solution, we may not choose the set of non-delay schedules since it may happen that this set does not contain an optimal solution. This can be seen at the instance from Example 3.2. Its unique optimal schedule is the active schedule shown in Figure 3.3(c) with makespan 5. Since this schedule is not a non-delay schedule, for this instance the set of non-delay schedules does not contain an optimal solution.

But if we restrict our search space to the set of active schedules, it can be proved that in this set for any regular objective function (not only for the makespan) always an optimal solution exists.

Theorem 3.1 For the RCPSP with a regular objective function always an optimal active schedule exists.

Proof: Consider an optimal schedule S which is not active. Then an activity i exists which can be feasibly shifted to the left without violating the precedence or resource constraints. Let S' be the resulting schedule which is obtained from S by the left shift of i. We have $S'_i < S_i$ and $S'_j = S_j$ for all other activities $j \neq i$, i.e. for any regular objective function the schedule S' must also be optimal. By repeating these arguments and shifting further activities to the left an optimal schedule is obtained for which no further left shift is possible, i.e. this schedule is an optimal active schedule. $\square$

Since the set of active schedules is a subset of all semi-active schedules, the result implies that also always an optimal semi-active schedule exists (some algorithms enumerate the larger set of semi-active schedules instead of only active schedules).

3.1.3 Reference notes

The calculation of time windows for activities and the notion of critical paths are the oldest techniques proposed in project management. They have their origin in the methods called PERT (Program Evaluation and Review Technique, cf. Malcolm et al. [137]) and CPM (Critical Path Method, cf. Kelley [108]) which were invented in 1958 in connection with military projects. The classification into semi-active, active and non-delay schedules is due to Sprecher et al. [181].

3.2 Heuristic Methods

In this section we discuss methods which calculate heuristic solutions for the RCPSP or more generally, for the RCPSP with time-dependent resource profiles. In Section 3.2.1 we present so-called schedule generation schemes, which are used to generate a schedule from a given sequence of activities. Priority-based heuristics are discussed in Section 3.2.2. Foundations for local search and genetic algorithms can be found in Sections 3.2.3 and 3.2.4, an ant colony optimization approach is presented in Section 3.2.5. Finally, the multi-mode case is reviewed in Section 3.2.6.

3.2.1 Schedule generation schemes

In order to apply an optimization algorithm to a problem, at first a suitable representation of solutions has to be chosen. Often, a schedule S is not directly represented by the corresponding starting times S_i of the activities since this representation has two drawbacks. On the one hand, the corresponding solution space is very large (if for each activity every starting time in the interval $[0, T]$ has to be considered, we get $O(T^n)$ possible schedules), on the other hand, it is often difficult to check feasibility of a given starting time vector. For this reason, schedules are often represented in an indirect way by sequences of the activities (so-called **activity lists**). From these lists feasible starting times are derived by appropriate decoding procedures (so-called **schedule generation schemes**). In the following we will present different schedule generation schemes for the RCPSP. Since it is not much more difficult to handle also time-dependent resource profiles, we dicuss this more general situation.

We assume that the time-dependent resource profile $R_k(t)$ of resource k for time periods $[t-1, t[$ with $t = 1, \ldots, T$ is compactly represented by pairs (t_k^μ, R_k^μ) for $\mu = 1, \ldots, m_k$, where $0 = t_k^1 < t_k^2 < \ldots < t_k^{m_k} = T$ are the jump points of the availability function and R_k^μ denotes the resource capacity in the time interval $[t_k^\mu, t_k^{\mu+1}[$ for $\mu = 1, \ldots, m_k - 1$. Note that it is already NP-complete to decide whether a feasible schedule satisfying the resource constraints exists. Since we ask for a feasible schedule with $C_{\max} \leq T$ for a given time horizon T, the decision version of the RCPSP (cf. Example 2.2) is a special case of this problem. In order to guarantee that a feasible solution exists, we assume that the last interval $[\max\limits_{k=1}^{r}\{t^{m_k-1}\}, T]$ is chosen in such a way that enough space and sufficient resources are available to process all activities in this interval.

We represent solutions by lists of all activities which are compatible with the precedence relations, i.e. i is placed before j in the list if $i \rightarrow j \in A$ holds. With each list $L = (i_1, \ldots, i_n)$ of all activities we associate an "earliest start schedule" by planning the activities in the order induced by the list. This is done by the procedure `Earliest Start Schedule` $(i_1, \ldots, i_n)$ shown in Figure 3.4. For each activity j the procedure calculates the earliest time t where all predecessors of j are finished and sufficient resources are available. After scheduling j in the interval $[t, t + p_j[$ the resource profiles are updated and the next activity is considered.

```
Procedure Earliest Start Schedule $(i_1, \ldots, i_n)$
1. FOR $\lambda := 1$ TO $n$ DO
2.     $j := i_\lambda$;
3.     $t := \max\limits_{i \to j \in A} \{S_i + p_i\}$;
4.     WHILE a resource $k$ with $r_{jk} > R_k(\tau)$ for some time
       $\tau \in \{t+1, \ldots, t+p_j\}$ exists DO
5.         Calculate the smallest time $t_k^\mu > t$ such that $j$
           can be scheduled in the interval $[t_k^\mu, t_k^\mu + p_j[$ if
           only resource $k$ is considered and set $t := t_k^\mu$;
6.     ENDWHILE
7.     Schedule $j$ in the interval $[S_j, C_j[ := [t, t+p_j[$;
8.     Update the current resource profiles by setting
       $R_k(\tau) := R_k(\tau) - r_{jk}$ for $k = 1, \ldots, r$; $\tau \in \{t+1, \ldots, t+p_j\}$;
9. ENDFOR
```

Figure 3.4: Calculation of an earliest start schedule for a given list

In each iteration we have to check all jump points of the resource profiles at most once. Furthermore, scheduling an activity creates at most two new jump points where the resource profile changes. Thus, procedure `Earliest Start Schedule` can be implemented such that it needs at most $O(\sum\limits_{\lambda=1}^{n} \sum\limits_{k=1}^{r} (m_k + 2(\lambda - 1)) + |A|) = O(n^2 r + n \sum\limits_{k=1}^{r} m_k)$ time, i.e. it runs in polynomial time. For the RCPSP with constant resource capacities (i.e. $m_k = 1$ for all resources k) the running time reduces to $O(n^2 r)$.

It remains to show that with this procedure a dominant set of schedules is achieved, i.e. that in the set of schedules which can be generated by applying the list scheduling procedure to all possible lists, always an optimal solution exists. By construction of the procedure `Earliest Start Schedule` each activity is processed as early as possible (respecting the precedence and resource constraints). Thus, in the constructed schedules no activity can be (locally or globally) shifted to the left, i.e. the list schedules are active.

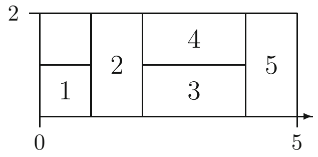
Theorem 3.2 For the RCPSP with a regular objective function a list of all activities exists such that the procedure `Earliest Start Schedule` provides an optimal schedule.

Proof: Let S be an optimal schedule and order the activities according to non-decreasing starting times in S. If we apply procedure `Earliest Start Schedule` to this list, a feasible schedule S' is produced. Furthermore, we have

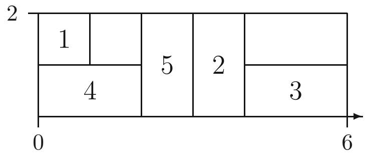
$S'_j \leq S_j$ for all activities $j = 1, \ldots, n$ since each activity is processed as early as possible in S'. Thus, for any regular objective function the schedule S' must also be optimal. $\square$

On the other hand, by similar arguments it can be shown that each active schedule can be generated by this procedure. This implies that the set of schedules which can be generated by applying the procedure `Earliest Start Schedule` to all possible lists equals the set of all active schedules.

Example 3.3: Consider again the instance from Example 3.2 with $n = 5$ activities and one resource with constant capacity $R_1 = 2$.



(a) List schedule for $(1, 2, 3, 4, 5)$



(b) List schedule for $(4, 5, 1, 2, 3)$

Figure 3.5: Two list schedules for Example 3.2

In Figure 3.5 the corresponding list schedules for the lists $(1, 2, 3, 4, 5)$ and $(4, 5, 1, 2, 3)$ are shown. For the lists $(1, 2, 4, 3, 5)$ and $(4, 1, 5, 2, 3)$ also the schedules in (a) and (b), respectively, are generated. This shows that in general the scheduling procedure does not produce different schedules for different lists. $\square$

Another list scheduling procedure, which is often used for scheduling problems, also produces a dominant set of schedules for the described situation. This procedure generates schedules where the starting times are ordered according to the given list, i.e. $S_{i_1} \leq S_{i_2} \leq \ldots \leq S_{i_n}$ holds. The resulting schedules are in general not active, but with the same arguments as in the proof of Theorem 3.2 it can be shown that for any regular objective function an optimal schedule is contained in the set of all schedules generated by this procedure. More precisely, the set of schedules which can be generated by applying this procedure contains all semi-active schedules and some additional schedules that are not semi-active.

In order to calculate such schedules in Step 3 of the procedure `Earliest Start Schedule` we only have to set t equal to the maximum of $\max\limits_{i \to j \in A} \{S_i + p_i\}$ and the starting time $S_{i_{\lambda-1}}$ of the previous activity. The resulting complexity is $O(\sum\limits_{k=1}^{r}(m_k + 2n) + |A|) = O(nr + n^2 + \sum\limits_{k=1}^{r} m_k)$ since during the check of the resource profiles we never have to go back to previous time periods.

The first list scheduling procedure described above may be seen as a special case of the so-called "serial schedule generation scheme" for the RCPSP. A **schedule generation scheme** (SGS) iteratively generates a feasible schedule by extending a partial schedule in each iteration (in general not always a list is needed). In the literature two schemes are distinguished:

- the activity-oriented **serial SGS**, where in each iteration one activity is scheduled, and
- the time-oriented **parallel SGS**, where in each iteration a new decision point on the time axis is considered and a subset of activities is scheduled at this time point.

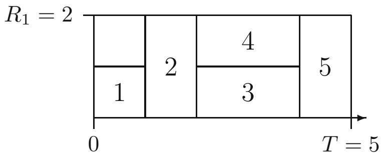
Schedule generation schemes are a basic component of heuristics based on priority rules. In such heuristics the activities are iteratively planned and in each iteration an eligible activity is chosen according to some priority rule. More details on priority-based heuristics will be described in Subsection 3.2.2.

In the **serial schedule generation scheme** a schedule is generated in n stages. With each stage $\lambda \in \{1, \dots, n\}$ two disjoint activity sets are associated: the set of scheduled activities and the set E_λ of all eligible activities (i.e. all activities for which all predecessors are scheduled). In each stage one eligible activity $j \in E_\lambda$ is chosen and scheduled at the earliest precedence- and resource-feasible time. Afterwards, the resource profiles of the partial schedule and the set of eligible activities are updated. The serial SGS is summarized in Figure 3.6. Similar to the procedure `Earliest Start Schedule` it can be implemented in such a way that it runs in $O(n^2 r + n \sum_{k=1}^{r} m_k)$ time.

```
Procedure Serial Schedule Generation Scheme
 1. Let $E_1$ be the set of all activities without predecessor;
 2. FOR $\lambda := 1$ TO $n$ DO
 3.   Choose an activity $j \in E_\lambda$;
 4.   $t := \max_{i \to j \in A} \{S_i + p_i\}$;
 5.   WHILE a resource $k$ with $r_{jk} > R_k(\tau)$ for some time
      $\tau \in \{t+1, \dots, t+p_j\}$ exists DO
 6.      Calculate the smallest time $t_k^\mu > t$ such that $j$
         can be scheduled in the interval $[t_k^\mu, t_k^\mu + p_j[$ if
         only resource $k$ is considered and set $t := t_k^\mu$;
 7.   ENDWHILE
 8.   Schedule $j$ in the interval $[S_j, C_j[ := [t, t+p_j[$;
 9.   Update the current resource profiles by setting
      $R_k(\tau) := R_k(\tau) - r_{jk}$ for $k = 1, \dots, r$; $\tau \in \{t+1, \dots, t+p_j\}$;
10.   Let $E_{\lambda+1} := E_\lambda \setminus \{j\}$ and add to $E_{\lambda+1}$ all successors
      $i \notin E_\lambda$ of $j$ for which all predecessors are scheduled;
11. ENDFOR
```

Figure 3.6: Serial schedule generation scheme

Example 3.4: Consider again the instance from Example 3.2. If in Step 3 we always choose an eligible activity with smallest index, then the serial SGS generates the following schedule:



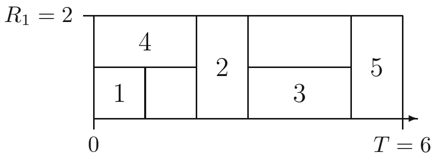
In the first iteration we choose activity 1 from the set of eligible activities $E_1 = \{1,4\}$ and schedule it at time 0. Afterwards, we have $E_2 = \{2,4\}$ and schedule activity 2 at time 1. In stage $\lambda = 3$ we get $E_3 = \{3,4\}$ and schedule activity 3 at time 2. Then $E_4 = \{4\}$ and activity 4 is scheduled also at time 2. Finally, activity 5 is scheduled at time 4. □

While the serial SGS is activity-oriented, the **parallel schedule generation scheme** is time-oriented. With each stage λ a time point t_λ and three disjoint activity sets are associated: the set of finished activities, the set A_λ of all active activities (i.e. activities which are already scheduled in the partial schedule, but finish after time t_λ), and the set E_λ of all eligible activities (i.e. all unscheduled activities i for which all predecessors are completed up to time t_λ and for which sufficient resources are available when i is started at time t_λ).

In each stage a maximal resource-feasible subset of eligible activities in E_λ is chosen and scheduled at time t_λ. Afterwards, the resource profiles of the partial schedule and the sets of active and eligible activities are updated. The next decision point $t_{\lambda+1}$ is given by the minimum of the next value t_k^μ where a resource profile changes and the minimal completion time of all active activities.

The parallel SGS is summarized in Figure 3.7. As the serial SGS, it generates feasible schedules in $O(n^2 r + n \sum\limits_{k=1}^{r} m_k)$ time.

Example 3.5 : Consider again the instance from Example 3.2. If in Step 5 we always choose an eligible activity with smallest index, then the parallel SGS generates the following schedule:



In the first iteration we consider time $t_1 = 0$ and schedule activity 1 and then activity 4 from the set of eligible activities $E_1 = \{1,4\}$. The next decision point is given by the completion time of activity 1, i.e. $t_2 = 1$. Since at this time no activity can be scheduled, we proceed with $t_3 = 2$. Then we have $E_3 = \{2,5\}$ and schedule activity 2. Since afterwards activity 5 cannot be processed, we go to $t_4 = 3$ with $E_4 = \{3,5\}$. After scheduling activity 3 at time 3, in the last stage activity 5 is scheduled at the next decision point $t_5 = 5$. □

```
Procedure Parallel Schedule Generation Scheme
 1. $\lambda := 1$; $t_1 := 0$; $A_1 := \emptyset$;
 2. Let $E_1$ be the set of all activities $i$ without predecessor
    and $r_{ik} \leq R_k(\tau)$ for $k = 1, \ldots, r$ and all $\tau \in \{1, \ldots, p_i\}$;
 3. WHILE not all activities are scheduled DO
 4.   WHILE $E_\lambda \neq \emptyset$ DO
 5.     Choose an activity $j \in E_\lambda$;
 6.     Schedule $j$ in the interval $[S_j, C_j[ := [t_\lambda, t_\lambda + p_j[$;
 7.     Update the current resource profiles by setting
        $R_k(\tau) := R_k(\tau) - r_{jk}$ for $k = 1, \ldots, r$; $\tau \in \{t_\lambda + 1, \ldots, t_\lambda + p_j\}$;
 8.     Add $j$ to $A_\lambda$ and update the set $E_\lambda$ by eliminating
        $j$ and all activities $i \in E_\lambda$ with $r_{ik} > R_k(\tau)$ for some
        resource $k$ and a time $\tau \in \{t_\lambda + 1, t_\lambda + p_i\}$;
 9.   ENDWHILE
10.   Let $t_{\lambda+1}$ be the minimum of the smallest value $t_k^\mu > t_\lambda$
      and $\min\limits_{i \in A_\lambda} \{S_i + p_i\}$;
11.   $\lambda := \lambda + 1$;
12.   Calculate the new sets $A_\lambda$ and $E_\lambda$;
13. ENDWHILE
```

Figure 3.7: Parallel schedule generation scheme

For the RCPSP with constant resource capacities contrary to the serial SGS (which may generate all active schedules), the parallel SGS generates only a subset of all active schedules, namely the set of non-delay schedules (cf. Section 3.1.2).

Theorem 3.3 For the RCPSP with constant resource capacities the parallel SGS generates non-delay schedules.

Proof: To prove that the parallel SGS only generates non-delay schedules, assume to the contrary that it generates a schedule S which is not a non-delay schedule. Thus, in S some part of an activity j can be feasibly shifted to the left to some time $t < S_j$. W.l.o.g. we may assume $t = t_\lambda$ for some index λ since due to the constant resource capacities the starting times of activities may be restricted to completion times of other activities. Because the left shift is feasible, all predecessors must be completed at time t_λ and sufficient resources have to be available in the interval $[t_\lambda, t_\lambda + 1[$. Since constant resource profiles are given, in stage λ of the parallel SGS the resource availabilities from interval $[t_\lambda, t_\lambda + 1[$ to $[t_\lambda, t_\lambda + p_j[$ do not decrease. Thus, at the end of stage λ activity j was contained in the set E_λ of eligible activities. Although it could be scheduled in $[t_\lambda, t_\lambda + p_j[$, it was not scheduled which contradicts Step 4 of the parallel SGS, where activities are scheduled until the set E_λ becomes empty. □

Note that this property does not hold for the RCPSP with time-dependent resource profiles since in Step 8 activities i are eliminated from E_λ if they cannot be resource-feasibly scheduled in the interval $[t_\lambda, t_\lambda + p_i[$. However, parts of i may be processed in an interval $[t_\lambda, t_\lambda + x[$ for some $x \geq 1$.

Since the set of non-delay schedules is a subset of the set of active schedules, in general the solution space of the parallel SGS is smaller than the solution space of the serial SGS. But unfortunately, as shown in Section 3.1.2 it may happen that the set of non-delay schedules does not contain an optimal solution. Example 3.5 shows this effect: independently of the chosen priority rule, the parallel SGS does not generate an optimal schedule since at time 0 always activities 1 and 4 are started. Instead of scheduling this maximal resource-feasible subset, it would be better to leave one unit of the resource idle and start another activity at the next decision point 1.

However, from this observation it cannot be deduced that in general the serial generation scheme is superior to the parallel one. Computational experiments of various authors have shown that for some instances the serial SGS produces better schedules, for other instances the parallel SGS is more suitable.

Another variant of the proposed schemes are their **backward** counterparts. While in the described (forward) schemes, the schedules are generated from left to right, in the corresponding backward schemes the schedules are constructed in the reverse direction from right to left. Finally, in a **bidirectional scheduling** procedure the forward and the backward scheme are combined. In each iteration an activity is scheduled either in a forward or in a backward schedule. At the end both schedules are concatenated and transformed into an active schedule by moving activities to the left.

The idea of scheduling activities from both directions is also used in so-called **forward-backward improvement** procedures. After applying a forward SGS to a given list, a new list is created by sorting the activities according to non-increasing completion times in the forward schedule. This new list is used for a backward scheduling step and after sorting the activities according to non-decreasing starting times in the backward schedule again the forward SGS is applied. This process may be iterated until no new schedules are created. Finally, the best schedule in this process is taken as output for the original list.

3.2.2 Priority-based heuristics

As described in the previous subsection, the main two components of priority-based heuristics for the RCPSP are schedule generation schemes and priority rules. In a schedule generation scheme in each stage an activity is selected which is scheduled next in the partial schedule. For this purpose, a priority value is calculated for each activity and an activity with smallest (or largest) value is chosen. In case of ties (i.e. if several activities have the same priority value), an additional rule is used to choose a unique activity.

activity-based:

SPT	choose an activity with the smallest processing time
LPT	choose an activity with the largest processing time

network-based:

MIS	choose an activity with the most immediate successors
LIS	choose an activity with the least immediate successors
MTS	choose an activity with the most total successors
LTS	choose an activity with the least total successors
GRPW	choose an activity with the greatest rank positional weight, i.e. with the largest total processing time of all successors

critical path-based:

EST	choose an activity with the smallest earliest starting time
ECT	choose an activity with the smallest earliest completion time
LST	choose an activity with the smallest latest starting time
LCT	choose an activity with the smallest latest completion time
MSLK	choose an activity with a minimum slack

resource-based:

GRR	choose an activity with the greatest resource requirements

Figure 3.8: Priority rules for the RCPSP

Many different priority rules for scheduling problems have been proposed in the literature. They may be classified as activity-based, network-based, critial path-based or resource-based (depending on the information which is used by the rule). In Figure 3.8 some often used rules are listed.

Another distinction of priority rules is based on the dynamics of a rule. For example, the SPT-rule is always static since the processing time of an activity does not change over time. On the other hand, the EST-rule may be used in a static or dynamic way: in the static case the earliest starting times are calculated once (only based on the project network), in the dynamic case they are recalculated after each scheduling step (taking into account the current partial schedule).

Additionally, priority-based heuristics may be classified as single- or multi-pass methods. While in **single-pass methods** only one schedule is generated (using a single priority rule and a single SGS), in **multi-pass methods** several schedules are generated (for example, by using several priority rules, different generation schemes or different scheduling directions).

Priority-based heuristics are often used in practice since they have small running times and are easy to implement. Furthermore, they are often used to calculate a starting solution for improvement procedures like local search or genetic algorithms, which will be discussed next.

3.2.3 Local search algorithms

In this subsection we describe some problem-specific parts of local search algorithms for the RCPSP with time-dependent resource profiles. As discussed in Subsection 2.10.1, the problem-specific parts of local search algorithms are the representation of solutions and the definition of appropriate neighborhoods (which are usually defined by operators). As described in Subsection 3.2.1, a solution for the RCPSP with time-dependent resource profiles may be represented by an activity list (sequence of the activities), which is compatible with the precedence constraints. In order to change such a list $L = (i_1, \ldots, i_n)$ in a local search procedure, the following neighborhoods may be defined:

Adjacent pairwise interchange-neighborhood

The adjacent pairwise interchange (api)-neighborhood $\mathcal{N}_{api}$ is defined by operators api_λ for $\lambda = 1, \ldots, n-1$, where api_λ interchanges the elements i_λ and $i_{\lambda+1}$ in L.

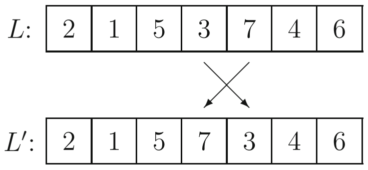


Figure 3.9: Operator api_λ for $\lambda = 4$

For example, by applying api_λ with $\lambda = 4$ to $L = (2, 1, 5, 3, 7, 4, 6)$, we get $L' = (2, 1, 5, 7, 3, 4, 6)$ (cf. Figure 3.9).

Since we only consider lists which are compatible with the precedence constraints, such an interchange is only feasible if no precedence relation $i_{\lambda+1} \to i_\lambda$ exists. Since we have at most $n-1$ feasible *api*-operators for a list, the size of the neighborhood $\mathcal{N}_{api}$ is bounded by $O(n)$.

If the neighborhood $\mathcal{N}_{api}$ is used in a tabu search procedure, usually the two interchanged jobs i, j are stored in a tabu list. As long as this pair is in the list, it is not allowed to swap jobs i and j again.

Swap-neighborhood

The swap-neighborhood $\mathcal{N}_{swap}$ generalizes the neighborhood $\mathcal{N}_{api}$ and is defined by operators $swap_{\lambda,\mu}$ for $\lambda, \mu = 1, \ldots, n; \lambda < \mu$, where $swap_{\lambda,\mu}$ interchanges the elements i_λ and i_μ in L.

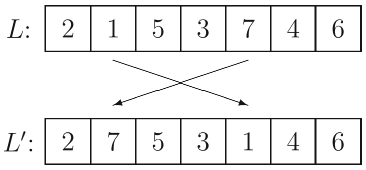


Figure 3.10: Operator $swap_{\lambda,\mu}$ with $\lambda = 2, \mu = 5$

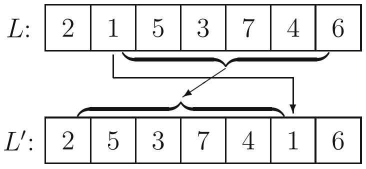
For example, by applying $swap_{\lambda,\mu}$ with $\lambda = 2, \mu = 5$ to $L = (2, 1, 5, 3, 7, 4, 6)$, we get $L' = (2, 7, 5, 3, 1, 4, 6)$ (cf. Figure 3.10).

Such an interchange is only feasible if for all $\nu \in \{\lambda + 1, \ldots, \mu\}$ no precedence relation $i_\lambda \to i_\nu$ or $i_\nu \to i_\mu$ exists. Since we have at most $n(n-1)/2$ feasible *swap*-operators for a permutation, the size of the neighborhood $\mathcal{N}_{swap}$ is bounded by $O(n^2)$.

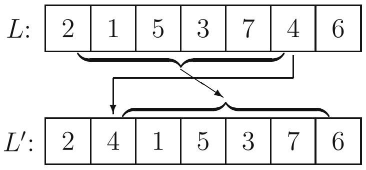
Note that if the neighborhood $\mathcal{N}_{swap}$ is used in a tabu search procedure, cycles are not prevented if only the two interchanged jobs i, j are stored. Therefore, other tabu criteria have to be used.

Shift-neighborhood

The shift-neighborhood $\mathcal{N}_{shift}$ also generalizes the neighborhood $\mathcal{N}_{api}$ and is defined by operators $shift_{\lambda,\mu}$ for $\lambda, \mu = 1, \ldots, n; \lambda \neq \mu$, where $shift_{\lambda,\mu}$ shifts the element i_λ to position μ in L. For $\lambda < \mu$ we have a right shift, symmetrically, for $\lambda > \mu$ we have a left shift.



(a) Right shift operator $shift_{\lambda,\mu}$ for $\lambda = 2 < \mu = 6$



(b) Left shift operator $shift_{\lambda,\mu}$ for $\lambda = 6 > \mu = 2$

Figure 3.11: Operators $shift_{\lambda,\mu}$

For example, by applying $shift_{\lambda,\mu}$ with $\lambda = 2, \mu = 6$ to $L = (2, 1, 5, 3, 7, 4, 6)$, we get $L' = (2,5,3,7,4,1,6)$ (cf. Figure 3.11(a)). By applying $shift_{\lambda,\mu}$ with $\lambda = 6, \mu = 2$ to $L = (2,1,5,3,7,4,6)$, we get $L' = (2,4,1,5,3,7,6)$ (cf. Figure 3.11(b)).

Right shifts are only feasible if for no $\nu \in \{\lambda+1,\dots,\mu\}$ a precedence relation $i_\lambda \to i_\nu$ exists; left shifts are feasible if for no $\nu \in \{\mu,\dots,\lambda-1\}$ a relation $i_\nu \to i_\lambda$ exists. Since we have at most $n(n-1)/2$ feasible left shift operators and at most $n(n-1)/2$ feasible right shift operators for a permutation, the size of the neighborhood $\mathcal{N}_{shift}$ is bounded by $O(n^2)$. Furthermore, note that a swap-operator can be performed by two consecutive shift-operators.

Example 3.6 : Consider again the instance from Example 3.2. There are 10 lists which are compatible with the precedence constraints. In Figure 3.12 the neighborhood graph for all these lists and the neighborhood $\mathcal{N}_{api}$ is shown. Associated with these lists are the four different earliest start schedules S_1, S_2, S_3, S_4 depicted in the right part of the figure.

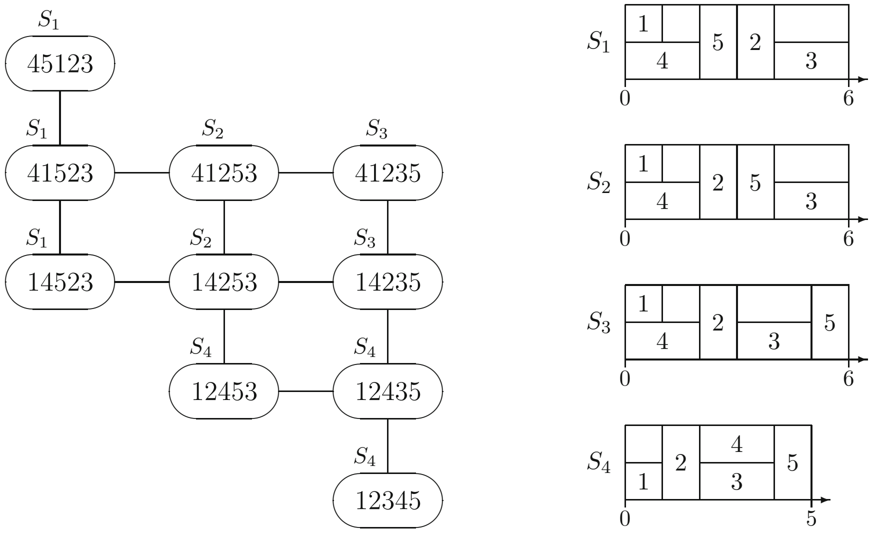


Figure 3.12: Neighborhood $\mathcal{N}_{api}$ for Example 3.6

□

3.2.4 Genetic algorithms

In this subsection we describe some problem-specific parts of genetic algorithms for the RCPSP with time-dependent resource profiles. For genetic algorithms the population *POP* may be defined as a set of precedence-feasible sequences (activity lists). Mutations may be achieved by changing a list slightly (e.g. by a swap or shift move) or by performing some iterations of simulated annealing or tabu search using one of these neighborhoods. Compared to the situation in Section 2.10.2 (where solutions were represented by bit strings), for sequences the basic crossover operators have to be slightly adapted in order to guarantee that all activities from the parent solutions are contained exactly once in the child solutions.

One-point crossover

Given two parent sequences, a mother sequence $L^M = (i_1^M, \dots, i_n^M)$ and a father sequence $L^F = (i_1^F, \dots, i_n^F)$, two child sequences, a daughter sequence $L^D = (i_1^D, \dots, i_n^D)$ and a son sequence $L^S = (i_1^S, \dots, i_n^S)$ are created by the following procedure. Let $q \in \{1, \dots, n\}$ be a random number. Then the daughter sequence L^D inherits the positions $\lambda = 1, \dots, q$ from the mother sequence L^M, i.e.

$$i_\lambda^D := i_\lambda^M \text{ for } \lambda = 1, \dots, q.$$

The remaining positions $q+1, \dots, n$ are taken from the father sequence L^F. They are ordered in L^D in the same way as they are ordered in L^F, i.e. for $\lambda = q+1, \dots, n$ we set

$$i_\lambda^D := i_k^F \text{ where } k \text{ is the smallest index with } i_k^F \notin \{i_1^D, \dots, i_{\lambda-1}^D\}.$$

The son sequence L^S is symmetrically constructed by interchanging the roles of L^M and L^F.

For example, with $L^M = (7,1,3,2,5,8,4,6,9)$, $L^F = (1,4,2,6,3,9,8,7,5)$, and $q = 4$ we get $L^D = (7,1,3,2,4,6,9,8,5)$ (cf. Figure 3.13).

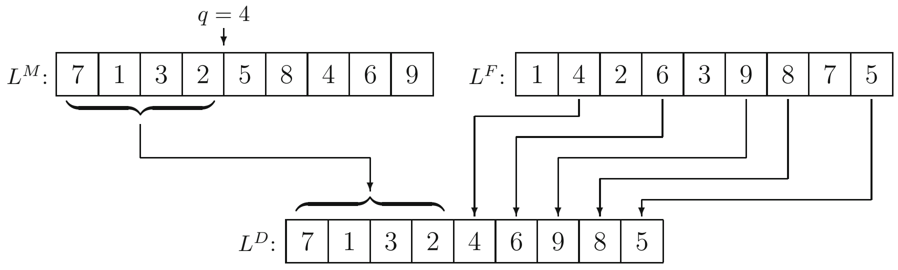


Figure 3.13: One-point crossover

If L^M and L^F are compatible with the precedence constraints, then the child sequences L^D and L^S also have this property. Let us consider L^D. For the first subsequence $I_1 = (i_1^D, \dots, i_q^D)$ of L^D the property is inherited from L^M, for the second subsequence $I_2 = (i_{q+1}^D, \dots, i_n^D)$ from L^F. Furthermore, it is not possible that $i \rightarrow j$ with $i \in I_2$, $j \in I_1$ holds, since this would contradict the fact that L^M is compatible with the precedence constraints.

Two-point crossover

Firstly, we draw two random numbers $q_1, q_2 \in \{1, \ldots, n\}$ with $q_1 < q_2$. The daughter sequence L^D inherits positions $1, \ldots, q_1$ and $q_2 + 1, \ldots, n$ from the mother L^M. The remaining positions $q_1 + 1, \ldots, q_2$ are taken from the father sequence L^F. They are ordered in L^D in the same way as they are ordered in L^F. The son sequence L^S is constructed by changing the roles of L^M and L^F.

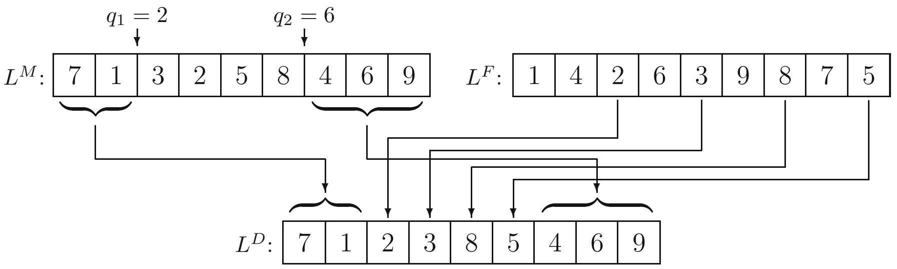


Figure 3.14: Two-point crossover

For example, with $L^M = (7, 1, 3, 2, 5, 8, 4, 6, 9)$, $L^F = (1, 4, 2, 6, 3, 9, 8, 7, 5)$, and $q_1 = 2, q_2 = 6$ we get $L^D = (7, 1, 2, 3, 8, 5, 4, 6, 9)$ (cf. Figure 3.14).

Again, it is easy to show that the child sequences are compatible with the precedence constraints if L^M and L^F have this property.

Another possibility for a two-point crossover would be the following. Again, two random numbers $q_1, q_2 \in \{1, \ldots, n\}$ with $q_1 < q_2$ are chosen. The daughter sequence L^D inherits positions $q_1, \ldots, q_2$ from the mother L^M. The remaining positions $1, \ldots, q_1 - 1$ and $q_2 + 1, \ldots, n$ are filled with those elements from L^F which are not contained in $i^M_{q_1}, \ldots, i^M_{q_2}$ (in the same order as in L^F).

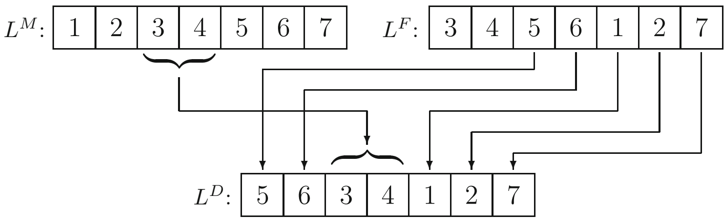


Figure 3.15: A non-compatible two-point crossover

For example, with $L^M = (1, 2, 3, 4, 5, 6, 7)$, $L^F = (3, 4, 5, 6, 1, 2, 7)$, and $q_1 = 3, q_2 = 4$ we get $L^D = (5, 6, 3, 4, 1, 2, 7)$ (cf. Figure 3.15). This example shows that in general such a crossover does not preserve compatibility with the precedence constraints. For example, the precedences $3 \rightarrow 6$ and $4 \rightarrow 6$ are satisfied in both parents L^M and L^F, but they are not respected in L^D.

Uniform crossover

For $\lambda = 1, \dots, n$ we randomly draw values $\xi_\lambda \in \{0, 1\}$ and fill the positions in L^D in the following way:

- If $\xi_\lambda = 1$, then we choose from L^M the activity which has the smallest index and is not yet inserted in positions $1, \dots, \lambda - 1$ in L^D, i.e.

$$i^D_\lambda := i^M_k \text{ where } k \text{ is the smallest index with } i^M_k \notin \{i^D_1, \dots, i^D_{\lambda-1}\}.$$

- If $\xi_\lambda = 0$, then we set

$$i^D_\lambda := i^F_k \text{ where } k \text{ is the smallest index with } i^F_k \notin \{i^D_1, \dots, i^D_{\lambda-1}\}.$$

For the construction of L^S we again change the roles of L^M and L^F.

For example, with $L^M = (7, 1, 3, 2, 5, 8, 4, 6, 9)$, $L^F = (1, 4, 2, 6, 3, 9, 8, 7, 5)$, and $\xi = (1, 1, 0, 1, 0, 0, 1, 0, 1)$ we get $L^D = (7, 1, 4, 3, 2, 6, 5, 9, 8)$.

Also for this crossover the children are compatible with the precedences if the parents have this property. This can easily be proved as follows. Assume that in L^D an activity i with $j \to i$ is placed before j. Then in L^M and L^F activity j must be placed before i. Is i inherited from L^F, then we have a contradiction to the fact that j is placed after i in L^D, since then j had to be scheduled in L^D instead of i. We get a similar contradiction if i is inherited from L^M.

In connection with genetic algorithms and schedule generation schemes for the RCPSP also so-called **self-adapting genetic algorithms** were proposed. In such algorithms the activity list representation is extended by an additional gene which determines whether the serial or parallel SGS is used as decoding procedure. Then during the selection process the algorithm tries to "learn" which schedule generation scheme is better.

3.2.5 Ant colony optimization

In order to apply an ACO algorithm to the RCPSP, the problem is considered as a sequencing problem similar to the TSP (cf. Section 2.10.3). Each ant constructs a sequence of all activities (activity list) that is converted to a schedule by a schedule generation scheme (serial or parallel SGS).

```
Ant algorithm RCPSP
 1.  REPEAT
 2.     FOR $k := 1$ TO $m$ DO
 3.         FOR $i := 1$ TO $n$ DO
 4.             Choose an unscheduled eligible activity
                $j \in V$ for position $i$ with probability
                $p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum\limits_{l \in V^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}$;
 5.         ENDFOR
 6.     ENDFOR
 7.     Calculate the makespans $C^k$ of the schedules
        constructed by the ants $k = 1, \ldots, m$;
 8.     Determine the best makespan $C^* = \min\limits_{k=1}^{m} \{C^k\}$ and a
        corresponding list $L^*$;
 9.     FOR ALL activities $j \in V$ and their corresponding
        positions $i$ in $L^*$ DO
10.         $\tau_{ij} := (1 - \varrho)\tau_{ij} + \varrho \frac{1}{2C^*}$;
11.  UNTIL a stopping condition is satisfied
```

Figure 3.16: Ant algorithm for the RCPSP

For the construction process the ants use pheromone values τ_{ij} as well as some heuristic information η_{ij} indicating how good it seems to put activity j at position i in the list. For sequencing problems where positions are encoded by the pheromone values τ_{ij}, instead of the so-called "direct evaluation" of the values τ_{ij} often a so-called "summation evaluation" is used. In such an approach the probability for placing activity j to position i is not only given by the value τ_{ij}, but the (accumulated) sum $\sum\limits_{\mu=1}^{i} \tau_{\mu j}$. The heuristic values η_{ij} may be obtained by some priority-based heuristic (e.g. using the priority rules ECT, LCT, MTS, or GRPW).

In Figure 3.16 an ant algorithm for the RCPSP is summarized. Here, V^k denotes the set of unscheduled activities of ant k. For the pheromone updates in Steps 9-10 an elitist strategy is used, i.e. only the best schedule is used to update the pheromone values.

3.2.6 Heuristics for the multi-mode case

In order to solve the multi-mode RCPSP heuristically, often a procedure with two stages is applied. While in one stage the mode assignment is determined, in the other stage a corresponding schedule is calculated. If this is done in a hierarchical way where at first the modes are fixed, in the second stage a classical single-mode RCPSP has to be solved.

The genetic algorithm based on activity lists has been extended to the multi-mode RCPSP by introducing an additional mode list M which assigns a mode to each activity. From such a solution (M, L) a schedule is derived by fixing the modes according to M and applying a schedule generation scheme to the activity list L afterwards. Different mutation and crossover operators have been proposed for M. For example, in a mutation the mode of a single activity may be changed. In the crossover operators mode sequences are inherited from the parents (e.g. the modes for the first q activities are taken from the mother, the remaining ones from the father).

3.2.7 Reference notes

The first heuristic for the RCPSP based on priority rules was proposed by Kelley [108] in 1963. A review of heuristics developed before 1973 can be found in Davies [48], a review of priority-based heuristics up to 1989 in Alvarez-Valdés and Tamarit [4]. Different methods based on the serial and the parallel schedule generation scheme are reviewed and computationally tested by Kolisch [117]. A simulated annealing algorithm based on activity lists was proposed by Bouleimen and Lecocq [25]. Overviews about various heuristics (including different local search and genetic algorithms) and a computational comparison of them can be found in Hartmann and Kolisch [95] and Kolisch and Hartmann [118], [119].

The forward/backward scheduling technique was first introduced by Li and Willis [134] and afterwards also used by Valls et al. [185]. The described genetic algorithms are due to Hartmann [89], [91], [92]. Another efficient genetic algorithm can be found in Debels and Vanhoucke [51]. The ant algorithm was developed by Merkle et al. [144].

3.3 MIP Formulations of the RCPSP

In this section we present different mixed integer linear programming (MIP) formulations of the RCPSP, which can be given to commercial or non-commercial MIP solvers. Unfortunately, up to now only small instances can be solved to optimality and the continuous relaxations often provide weak lower bounds. In Sections 3.3.1-3.3.3 discrete and continuous time formulations as well as event-based formulations are discussed. Finally, in Section 3.3.4 a MIP formulation for an extended project scheduling problem with labor constraints is presented.

3.3.1 Discrete time formulations

In discrete time formulations the time horizon T is partitioned into T time slots $[t-1, t[$ for $t = 1, \dots, T$. We introduce so-called **time-indexed variables** x_{it} for $i \in V = \{0, 1, \dots, n, n+1\}$ and $t = 0, \dots, T$ with

$$x_{it} := \begin{cases} 1, & \text{if activity } i \text{ starts at time } t \\ 0, & \text{otherwise.} \end{cases}$$

In the basic discrete time (DT) formulation the RCPSP may be formulated as the following binary linear program:

$$\min \quad \sum_{t=0}^{T} t x_{n+1,t} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{t=0}^{T} x_{it} = 1 \qquad (i \in V) \tag{3.2}$$

$$\sum_{t=0}^{T} t x_{jt} - \sum_{t=0}^{T} t x_{it} \geq p_i \qquad ((i,j) \in A) \tag{3.3}$$

$$\sum_{i=1}^{n} r_{ik} \sum_{\tau=\max\{0, t-p_i+1\}}^{t} x_{i\tau} \leq R_k \qquad (k = 1, \dots, r;\ t = 0, \dots, T-1) \tag{3.4}$$

$$x_{it} \in \{0, 1\} \quad (i \in V;\ t = 0, \dots, T) \tag{3.5}$$

In (3.1) the start time of the dummy terminating activity $n+1$ (i.e. the makespan) is minimized. Due to the constraints (3.2) and (3.5) exactly one start time is chosen for each activity. Since the term $\sum\limits_{t=0}^{T} t x_{it}$ corresponds to the start time S_i of activity i, constraints (3.3) ensure that the precedence constraints $S_j \geq S_i + p_i$ for $(i,j) \in A$ are respected. Finally, restrictions (3.4) guarantee that the resource constraints are satisfied (since activity i is processed in the interval $[t, t+1[$ if and only if it starts in the interval $[t - p_i + 1, t]$).

Example 3.7 : Example 1.1 may be used to illustrate the DT-formulation. For the optimal schedule shown in Figure 1.2(b) we have $x_{20} = x_{33} = x_{43} = x_{18} = 1$.

All other x_{it}-values are equal to 0. To illustrate that (3.4) correctly covers the resource constraints, consider for example $t = 9$ corresponding to the interval $[9, 10[$. In this case for $i = 1$ and $i = 4$ we have

$$\sum_{\tau=\max\{0,t-p_1+1\}}^{t} x_{1\tau} = \sum_{\tau=9-4+1}^{9} x_{1\tau} = x_{16} + x_{17} + x_{18} + x_{19} = x_{18} = 1,$$

$$\sum_{\tau=9-8+1}^{9} x_{4\tau} = x_{42} + \ldots + x_{49} = x_{43} = 1.$$

For $i = 2, 3$ we get $\sum\limits_{\tau=9-p_i+1}^{9} x_{i\tau} = 0$. Thus, for $t = 9$ the left hand side of (3.4) equals the resource requirements $r_{1k} + r_{4k}$. □

The DT-formulation is a pseudo-polynomial formulation since it has $O(nT)$ binary variables and $O(|A| + rT + n)$ constraints. If time windows $[r_i, d_i]$ are calculated for the activities, it is sufficient to introduce the variables x_{it} for all possible starting times $t \in \{r_i, \ldots, d_i - p_i\}$ since activity i cannot start outside this time window. Thus, smaller time windows may reduce the number of variables considerably.

If we have time-dependent resource profiles, the right hand side of (3.4) has to be replaced by $R_k(t)$. If time-lags d_{ij} are given, constraints (3.3) have to be replaced by

$$\sum_{t=0}^{T} t x_{jt} - \sum_{t=0}^{T} t x_{it} \geq d_{ij} \qquad ((i,j) \in A). \tag{3.6}$$

In order to model the precedence constraints, inequalities (3.3) may also be replaced by the stronger inequalities

$$\sum_{\tau=t}^{T} x_{i\tau} + \sum_{\tau=0}^{\min\{t+p_i-1,T\}} x_{j\tau} \leq 1 \quad ((i,j) \in A;\ t = 0, \ldots, T-1). \tag{3.7}$$

Here, if activity i starts at time t or later (i.e. $\sum\limits_{\tau=t}^{T} x_{i\tau} = 1$), activity j cannot start in the interval $[0, t + p_i[$.

The formulation in which inequalities (3.7) instead of (3.3) are used, is also called disaggregated discrete time (DDT) formulation. It requires the same number of binary variables, but more (namely $O(|A|T + rT + n)$) constraints. Since (3.2) and (3.7) together imply (3.3), the DDT-formulation is stronger than the DT-formulation and the corresponding continuous relaxation provides better lower bounds for the RCPSP than the continuous relaxation of the DT-formulation. However, longer computation times are needed.

The formulations in this section can be generalized to cover the multi-mode RCPSP by introducing binary variables x_{imt} for $i \in V; m \in \mathcal{M}_i; t = 0, \ldots, T$ where $x_{imt} = 1$ if and only if activity i is processed in mode m and starts at time t.

3.3.2 Continuous time formulations

In continuous time formulations the usual continuous starting time variables S_i for all activities $i \in V$ are used. Additionally, binary sequencing variables $x_{ij} \in \{0,1\}$ for all pairs of activities $i, j \in V$ are needed with

$$x_{ij} := \begin{cases} 1, & \text{if activity } j \text{ does not start before activity } i \text{ is completed} \\ 0, & \text{otherwise.} \end{cases}$$

The first formulation is based on the concept of so-called forbidden sets. A subset $F \subset \{1, \ldots, n\}$ of activities is called a **forbidden set** if

- all activities in F can be processed simultaneously, i.e. there are no precedence constraints among them, and
- there exists a resource k such that the resource requirements of the activities in F exceed the resource capacity, i.e. $\sum\limits_{i \in F} r_{ik} > R_k$.

A forbidden set F is called **minimal**, if it does not contain a forbidden set as a subset (i.e. for all subsets $F' \subset F$ we have $\sum\limits_{i \in F'} r_{ik} \le R_k$ for all resources k). In order to obtain a feasible schedule every minimal forbidden set has to be "broken", i.e. for any minimal forbidden set a precedence relation has to be introduced between at least two activities (such that they are not processed simultaneously).

Let $\mathcal{F}$ be the set of all minimal forbidden sets. Then the forbidden set formulation FS reads as follows:

$$\begin{array}{llll}
\min & S_{n+1} & & (3.8) \\
\text{s.t.} & x_{ij} + x_{ji} \le 1 & (i, j \in V, i < j) & (3.9) \\
& x_{ij} + x_{jh} - x_{ih} \le 1 & (i, j, h \in V) & (3.10) \\
& x_{ij} = 1 & ((i, j) \in A) & (3.11) \\
& S_j - S_i - M x_{ij} \ge p_i - M & (i, j \in V) & (3.12) \\
& \sum\limits_{i,j \in F} x_{ij} \ge 1 & (F \in \mathcal{F}) & (3.13) \\
& x_{ij} \in \{0,1\} & (i, j \in V) & (3.14) \\
& S_i \ge 0 & (i \in V) & (3.15)
\end{array}$$

In constraints (3.12) the value M is a big constant, which can be set to any valid upper bound for the makespan of the RCPSP (e.g. $M = \sum\limits_{i=1}^{n} p_i$). Constraints (3.9) and (3.10) guarantee that no cycles can occur in the sequencing decisions: While inequalities (3.9) state that for two activities $i \neq j$, it cannot happen that i precedes j and j precedes i, constraints (3.10) prevent cycles involving more than two activities. Due to constraints (3.11) the given precedence constraints are respected. Constraints (3.12) link the variables x_{ij} with the starting times of the activities: If $x_{ij} = 1$ (i.e. i precedes j), then $S_j - S_i \geq p_i$ must hold. Otherwise (i.e. $x_{ij} = 0$), inequality (3.12) is always satisfied. Finally, the resource constraints are taken into account by (3.13) which ensure that in every forbidden set at least one sequencing decision must be taken.

The FS-formulation has $O(n^2)$ binary variables, $O(n)$ continuous variables, and $O(2^n + n^3)$ constraints. Usually, due to the exponential number of constraints (induced by the large number of forbidden sets), this formulation cannot be used in practice.

In order to eliminate the large number of constraints, a more compact flow-based continuous time (FCT) formulation has been proposed. Besides the continuous start time variables S_i and the binary sequencing variables x_{ij} as above, continuous flow variables f_{ijk} for $i, j \in V; k = 1, \ldots, r$ are used. The variable f_{ijk} indicates the quantity of resource k that is transfered from activity i (at the end of its processing) to activity j (at the start of its processing). In this formulation for all resources k we set $r_{0k} = r_{n+1,k} := R_k$ for the dummy start activity 0 (acting as resource source) and the dummy finishing activity $n+1$ (acting as resource sink).

Then, the FCT-formulation can be written as follows:

$$\begin{array}{llll}
\min & S_{n+1} & & (3.16) \\
\text{s.t.} & x_{ij} + x_{ji} \leq 1 & (i, j \in V, i < j) & (3.17) \\
& x_{ij} + x_{jh} - x_{ih} \leq 1 & (i, j, h \in V) & (3.18) \\
& x_{ij} = 1 & ((i, j) \in A) & (3.19) \\
& S_j - S_i - M x_{ij} \geq p_i - M & (i, j \in V) & (3.20) \\
& f_{ijk} - \min\{r_{ik}, r_{jk}\} x_{ij} \leq 0 & (i \in V \setminus \{n+1\};\ j \in V \setminus \{0\}; & \\
& & \ k = 1, \ldots, r) & (3.21) \\
& \sum_{j=0}^{n+1} f_{ijk} = r_{ik} & (i \in V;\ k = 1, \ldots, r) & (3.22) \\
& \sum_{i=0}^{n+1} f_{ijk} = r_{jk} & (j \in V;\ k = 1, \ldots, r) & (3.23) \\
& f_{n+1,0,k} = R_k & (k = 1, \ldots, r) & (3.24) \\
& x_{ij} \in \{0, 1\} & (i, j \in V) & (3.25) \\
& S_i \geq 0 & (i \in V) & (3.26) \\
& f_{ijk} \geq 0 & (i, j \in V;\ k = 1, \ldots, r) & (3.27)
\end{array}$$

The constraints (3.17)-(3.20) are the same as (3.9)-(3.12). Constraints (3.21) ensure that there can only be a positive flow f_{ijk} if i is processed before j (i.e. $x_{ij} = 1$) and in this case the flow is bounded by r_{ik} as well as by r_{jk}. Otherwise, if i is not processed before j, the flow f_{ijk} must be zero. Equalities (3.22) and (3.23) are resource flow conservation constraints stating that for each activity i

the inflow of resource k is equal to the outflow. Condition (3.24) says that for each resource k the flow from $n+1$ to 0 must be equal to R_k, which implies that also R_k units must be sent from node 0 to node $n+1$.

Example 3.8: Consider again Example 1.1 and the optimal schedule shown in Figure 1.2(b). For the first resource the positive flow values are $f_{021} = f_{031} = 1, f_{041} = 2, f_{051} = 1, f_{151} = 2, f_{231} = 1, f_{311} = 2, f_{451} = 2, f_{501} = 5$ (cf. also Figure 3.17).

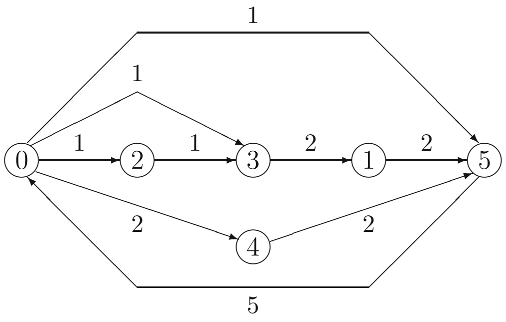


Figure 3.17: Resource flows

For the second resource we have $f_{022} = 5, f_{042} = 2, f_{152} = 3, f_{212} = 1, f_{232} = f_{242} = 2, f_{312} = 2, f_{452} = 4, f_{502} = 7$. □

The FCT-formulation involves $O(n^2)$ binary variables, $O(rn^2)$ continuous variables, and $O(rn^2 + n^3)$ constraints. Since (contrary to the discrete time formulations) the number of variables and constraints does not increase with the time horizon T, this formulation may especially be useful if T is very large.

3.3.3 Event-based formulations

The event-based formulations use the notion of events which correspond to times where an activity starts or ends. These formulations are based on the fact that for the RCPSP always an optimal semi-active schedule exists (cf. Section 3.1.2) in which the start time of any activity is either 0 or coincides with the completion time of another activity. Therefore, at most $n+1$ events have to be considered. Furthermore, dummy activities can be neglected (i.e. we only schedule the activity set $V := \{1, \ldots, n\}$). Like the FCT-formulation the event-based formulations are able to deal with instances containing non-integer processing times and the number of variables as well as the number of constraints do not depend on the time horizon T. Furthermore, the presented event-based formulations do not involve any big-M constant. There are two types of event-based formulations, which will be introduced next.

Start/End Event-based formulation

Let $\mathcal{E} = \{0, 1, \ldots, n\}$ be the index set of the events corresponding to the starting and completion times of the activities. In the start/end event-based formulation SEE two types of binary variables and two types of continuous variables are considered. There are binary variables $x_{ie}, y_{ie} \in \{0, 1\}$ for $i \in V, e \in \mathcal{E}$ with

$$x_{ie} := \begin{cases} 1, & \text{if activity } i \text{ starts at event } e \\ 0, & \text{otherwise} \end{cases}$$

and

$$y_{ie} := \begin{cases} 1, & \text{if activity } i \text{ ends at event } e \\ 0, & \text{otherwise.} \end{cases}$$

The continuous variables t_e for $e \in \mathcal{E}$ represent the dates of the events e. We assume that the events are enumerated such that $t_0 \leq t_1 \leq \ldots \leq t_n$ holds. The continuous (auxiliary) variable r^*_{ek} for $e \in \mathcal{E}, k = 1, \ldots, r$ represents the quantity of resource k required immediately after event e.

Then, the SEE-formulation can be written as follows:

$$\begin{array}{llll}
\min & t_n & & (3.28) \\
\text{s.t.} & t_0 = 0 & & (3.29) \\
& t_{e+1} - t_e \geq 0 & (e \in \mathcal{E} \setminus \{n\}) & (3.30) \\
& \sum_{f \in \mathcal{E}} f y_{if} - \sum_{e \in \mathcal{E}} e x_{ie} \geq 1 & (i \in V) & (3.31) \\
& t_f - t_e - p_i x_{ie} + p_i(1 - y_{if}) \geq 0 & (i \in V;\ e, f \in \mathcal{E}, e < f) & (3.32) \\
& \sum_{e \in \mathcal{E}} x_{ie} = 1 & (i \in V) & (3.33) \\
& \sum_{e \in \mathcal{E}} y_{ie} = 1 & (i \in V) & (3.34) \\
& \sum_{e'=e}^{n} y_{ie'} + \sum_{e'=0}^{e-1} x_{je'} \leq 1 & ((i,j) \in A;\ e \in \mathcal{E}) & (3.35) \\
& r^*_{0k} - \sum_{i \in V} r_{ik} x_{i0} = 0 & (k = 1, \ldots, r) & (3.36) \\
& r^*_{ek} - r^*_{e-1,k} + \sum_{i \in V} r_{ik}(y_{ie} - x_{ie}) = 0 & (e \in \mathcal{E} \setminus \{0\};\ k = 1, \ldots, r) & (3.37) \\
& r^*_{ek} \leq R_k & (e \in \mathcal{E}; k = 1, \ldots, r) & (3.38) \\
& x_{ie} \in \{0, 1\} & (i \in V;\ e \in \mathcal{E}) & (3.39) \\
& y_{ie} \in \{0, 1\} & (i \in V;\ e \in \mathcal{E}) & (3.40) \\
& t_e \geq 0 & (e \in \mathcal{E}) & (3.41) \\
& r^*_{ek} \geq 0 & (e \in \mathcal{E};\ k = 1, \ldots, r) & (3.42)
\end{array}$$

The objective function (3.28) consists in minimizing the completion time t_n of an activity processed last. (3.29) indicates that the first event starts at time 0, (3.30) takes care of the ordering of the events. The inequalities (3.31) guarantee that an activity must end at a later event than its starting event. Inequalities (3.32) ensure that if $x_{ie} = y_{if} = 1$ (i.e. i starts at event e and completes at event f), then $t_f \geq t_e + p_i$ holds. For all other combinations of values for x_{ie} and y_{if} we have either $t_f \geq t_e$ or $t_f \geq t_e - p_i$, which are covered by (3.30). Constraints (3.33) and (3.34) guarantee that each activity starts and ends exactly once. Constraints (3.35) ensure that the given precedence constraints are respected: If a predecessor i of j ends at event e or later (i.e. $\sum\limits_{e'=e}^{n} y_{ie} = 1$), then $\sum\limits_{e'=0}^{e-1} x_{je'}$ must be zero, i.e. j cannot start before event e. Equalities (3.36) set the quantities of resources k needed immediately after time 0. Equalities (3.37) describe the recursion for calculating the r^*_{ek}-values for the other events, namely the quantity of resource k needed immediately after time t_e is equal to the quantity of resource k needed immediately after time t_{e-1} plus the quantity of resource k needed by the activities starting at time t_e minus the quantity of resource k needed by the activities completing at time t_e. Finally, (3.38) limits the quantity of resource k needed immediately after time t_e to the availability of resource k.

Note that the auxiliary variables r^*_{ek} can be replaced by the expressions in (3.36) and (3.37), i.e. they are not counted below.

Example 3.9: Consider again Example 1.1 and the optimal schedule shown in Figure 1.2(b). For the SEE-formulation we have the following events:

$$\begin{array}{llll}
e = 0: & t_0 = 0, & x_{20} = 1 & \\
e = 1: & t_1 = 3, & x_{31} = x_{41} = 1, & y_{21} = 1 \\
e = 2: & t_2 = 8, & x_{12} = 1, & y_{32} = 1 \\
e = 3: & t_3 = 11, & & y_{43} = 1 \\
e = 4: & t_4 = 12, & & y_{14} = 1
\end{array}$$

All other x_{ie}, y_{if}-values are equal to zero. □

It remains to show the correctness of the SEE-formulation, i.e. we have to prove that an optimal solution of the SEE-formulation provides an optimal schedule for the RCPSP.

Theorem 3.4 An optimal solution (t, x, y) of the SEE-formulation provides an optimal RCPSP schedule $S = (S_i)_{i\in V}$ where activity $i \in V$ starts at time $S_i = \sum\limits_{e\in\mathcal{E}} t_e x_{ie}$.

Proof: We have already shown that any feasible solution for the RCPSP provides a feasible solution of the MIP-formulation and t_n corresponds to the makespan. Therefore, it remains to show that if (t, x, y) is a feasible solution for (3.28)-(3.31), then the schedule S with $S_i = \sum\limits_{e\in\mathcal{E}} t_e x_{ie}$ is a feasible solution for the RCPSP. Note that due to (3.33) for any activity $i \in V$ exactly one x_{ie} is equal to one, i.e. the starting time S_i is determined by exactly one event. Furthermore, the interpretation of $y_{if} = 1$ is slightly different: if $y_{if} = 1$ holds, then t_f is only an upper bound for the completion time of activity i (it may be larger than the starting time of activity i plus p_i). But, since the y_{if}-values are ignored in the definition of S, this causes no problem.

In the following we show that the schedule S is feasible, i.e. satisfies all precedence and resource constraints:

- Let $(i, j) \in A$, i.e. j is a successor of i. If $x_{ie} = 1$, then by constraints (3.31) we must have $y_{if} = 1$ for some $f > e$. Thus, due to (3.32) we have
$$0 \le t_f - t_e - p_i x_{ie} + p_i(1 - y_{if}) = t_f - t_e - p_i,$$
which implies $t_f \ge t_e + p_i$. Furthermore, with $y_{if} = 1$ constraints (3.33) and (3.35) induce that $x_{jg} = 1$ for some event $g \ge f$. Therefore, by (3.30) we get $t_g \ge t_f$, which implies $t_e + p_i \le t_f \le t_g$, i.e. activity j cannot start before the completion time of activity i.
- Due to (3.32) for each activity i and events e, f with $x_{ie} = y_{if} = 1$ the inequality $t_f \ge t_e + p_i$ must hold. Therefore, by (3.36)-(3.38) the capacity constraints are satisfied even when (in the MIP solution) activity i completes at a time $t_f \ge t_e + p_i$ instead of time $t_e + p_i$. This implies that all resource constraints are fulfilled. $\square$

If time windows $[r_i, d_i]$ are given for the activities, the formulation can be strengthened by the inequalities

$$\begin{aligned} r_i x_{ie} &\le t_e \le (d_i - p_i)x_{ie} + T(1 - x_{ie}) \quad (i \in V;\ e \in \mathcal{E}) \qquad (3.43)\\ (r_i + p_i)y_{ie} &\le t_e \le d_i y_{ie} + T(1 - y_{ie}) \quad (i \in V;\ e \in \mathcal{E}) \qquad (3.44)\end{aligned}$$

which ensure that we have $r_i \le t_e \le d_i - p_i$ for the starting event e of i and $r_i + p_i \le t_e \le d_i$ for the ending event of i.

Formulation SEE involves $O(n^2)$ binary variables, $O(n)$ continuous variables, and $O(n^3 + (|A| + r)n)$ constraints. Compared to DT and DDT, this formulation has a polynomial number of variables and constraints. Compared to CFT, SEE has no big-M constraints, but about twice as many binary variables.

On/Off Event-based formulation

The on/off event-based formulation OOE is a variant of SEE which uses only one type of binary variables per event, and one type of continuous variables. Let $\mathcal{E} = \{0, 1, \ldots, n-1\}$ be the index set of the events corresponding to the starting times of the activities. There are binary variables $z_{ie} \in \{0, 1\}$ for $i \in V, e \in \mathcal{E}$ where z_{ie} is equal to 1 if and only if activity i starts at event e or if it is still being processed during other events immediately after event e. Thus, z_{ie} remains 1 for the time activity i is processed. Additionally, we introduce an auxiliary variable $z_{i,-1} = 0$ for every activity $i \in V$. The continuous variables t_e for $e \in \mathcal{E}$ represent again the dates of the events e. Finally, a continuous variable $C_{\max}$

is introduced corresponding to the makespan which has to be minimized. The OOE-formulation can be written as follows:

$$\begin{aligned}
\min \quad & C_{\max} && && (3.45)\\
\text{s.t.} \quad & t_0 = 0 && && (3.46)\\
& t_{e+1} - t_e \geq 0 && (e \in \mathcal{E} \setminus \{n-1\}) && (3.47)\\
& \sum_{e \in \mathcal{E}} z_{ie} \geq 1 && (i \in V) && (3.48)\\
& \sum_{e'=0}^{e-1} z_{ie'} - e(1-(z_{ie} - z_{i,e-1})) \leq 0 && (i \in V; e \in \mathcal{E} \setminus \{0\}) && (3.49)\\
& \sum_{e'=e}^{n-1} z_{ie'} - (n-e)(1+(z_{ie} - z_{i,e-1})) \leq 0 && (i \in V; e \in \mathcal{E} \setminus \{0\}) && (3.50)\\
& t_f - t_e - p_i(z_{ie} - z_{i,e-1} - (z_{if} - z_{i,f-1})) \geq -p_i && (i \in V; e, f \in \mathcal{E}, e < f) && (3.51)\\
& C_{\max} - t_e - p_i(z_{ie} - z_{i,e-1}) \geq 0 && (i \in V;\ e \in \mathcal{E}) && (3.52)\\
& \sum_{i \in V} r_{ik} z_{ie} \leq R_k && (k = 1, \ldots, r;\ e \in \mathcal{E}) && (3.53)\\
& z_{ie} + \sum_{e'=0}^{e} z_{je'} - e(1 - z_{ie}) \leq 1 && ((i,j) \in A;\ e \in \mathcal{E}) && (3.54)\\
& z_{i,-1} = 0 && (i \in V) && (3.55)\\
& z_{ie} \in \{0,1\} && (i \in V; e \in \mathcal{E}) && (3.56)\\
& t_e \geq 0 && (e \in \mathcal{E}) && (3.57)\\
& C_{\max} \geq 0 && && (3.58)
\end{aligned}$$

As in the SEE-formulation constraints (3.46) and (3.47) model the timing of the events according to $t_0 = 0 \leq t_1 \leq \ldots \leq t_{n-1}$. Note that $z_{ie} - z_{i,e-1} = 1$ if $z_{ie} = 1$ and $z_{i,e-1} = 0$, i.e. if activity i is processed at event e and not at event $e-1$. Symmetrically, $z_{ie} - z_{i,e-1} = -1$ if i is processed at event $e-1$ and not at event e. In all other cases $z_{ie} - z_{i,e-1} = 0$.

Due to conditions (3.48)-(3.50) each activity i is processed in one block of contiguous events. Inequality (3.48) states that i is processed at all. Now consider two events e, f with $f > e+1$ and assume that i is processed at both events e and f, but not at the events $e+1, \ldots, f-1$ in between. Then, (3.49) implies $\sum\limits_{e'=0}^{f-1} z_{ie'} = 0$ and (3.50) implies $\sum\limits_{e'=e+1}^{n-1} z_{ie'} = 0$, which contradicts (3.48). Thus, we must have $z_{ie'} = 1$ for some events $e' = e, e+1, \ldots, f$ and $z_{ie''} = 0$ for all other events e''. Note that (3.49) and (3.50) provide no restrictions if $z_{ie} - z_{i,e-1} \neq 1$ and $z_{ie} - z_{i,e-1} \neq -1$, respectively.

Constraints (3.51) link the binary variables with the continuous variables: If $z_{ie} - z_{i,e-1} = 1$ (i.e. activity i starts at event e) and $z_{if} - z_{i,f-1} = -1$ (i.e. i finishes at event f) for some indices $f > e$, then $t_f \geq t_e + p_i$ must hold. In all other cases one has $t_f \geq t_e$, $t_f \geq t_e - p_i$, $t_f \geq t_e - 2p_i$, or $t_f \geq t_e - 3p_i$, which is redundant due to (3.47).

Inequalities (3.52) ensure that the makespan is correctly calculated by imposing $C_{\max} \geq t_e + p_i$ when activity i starts at event e. The resource constraints can easily be formulated as (3.53).

Finally, inequalities (3.54) represent the precedence constraints because if e is a latest event where activity i is processed, then (3.54) implies $\sum\limits_{e'=0}^{e} z_{je'} = 0$, i.e. activity j must start at a later event than the last event where activity i is processed. If $z_{ie} = 0$, then (3.54) is redundant.

If time windows $[r_i, d_i]$ are given for the activities, the formulation can again be strengthened by the following inequalities:

$$r_i z_{ie} \leq t_e \leq (d_i - p_i)(z_{ie} - z_{i,e-1}) + T(1 - (z_{ie} - z_{i,e-1})) \quad (i \in V;\ e \in \mathcal{E})$$

Example 3.10 : Consider again Example 1.1 and the optimal schedule shown in Figure 1.2(b). For the OOE-formulation we have the following events:

$$\begin{aligned} e = 0 :&\quad t_0 = 0, \quad z_{20} = 1 \\ e = 1 :&\quad t_1 = 3, \quad z_{31} = 1 \\ e = 2 :&\quad t_2 = 3, \quad z_{32} = z_{42} = 1 \\ e = 3 :&\quad t_3 = 8, \quad z_{43} = z_{13} = 1 \end{aligned}$$

All other z_{ie}-values are equal to zero. □

It remains to show the correctness of the OOE-formulation.

Theorem 3.5 An optimal solution (t, z) of the OOE-formulation provides an optimal RCPSP schedule $S = (S_i)_{i \in V}$ in which activity $i \in V$ starts at time $S_i = t_{e(i)}$ where $e(i) = \min\{e \mid z_{ie} = 1\}$.

Proof: From the previous discussion we may conclude that any feasible solution of the RCPSP induces a feasible solution for the OOE-formulation if we introduce events e corresponding to the start times S_i of the activities and set $t_e := S_i$, and $z_{i\nu} := 1$ if and only if ν is an event where i is processed.

On the other hand, for a feasible solution (t, z) of the OOE-formulation we consider the schedule S in which activity $i \in V$ starts at time $S_i = t_{e(i)}$ where $e(i) = \min\{e \mid z_{ie} = 1\}$.

By the definition of $e = e(i)$ we have $z_{i,e-1} = 0$. Furthermore, by (3.49) and (3.50) there exists an event $f > e$ such that $z_{i\nu} = 1$ for $\nu = e, \ldots, f-1$ and $z_{i\nu} = 0$ for $\nu \notin \{e, \ldots, f-1\}$. Due to (3.51) the inequality $t_f \geq t_e + p_i$ holds. If $(i, j) \in A$, i.e. j is a successor of i, then j cannot start before time t_f because (3.54) must hold for $f - 1$. Thus, the precedence constraints are fulfilled.

Furthermore, due to (3.53) also the resource constraints are satisfied. That the solution provided by the OOE-formulation is optimal is an immediate consequence of the constraints (3.52). □

Formulation OOE involves $O(n^2)$ binary variables, $O(n)$ continuous variables, and $O(n^3 + (|A| + r)n)$ constraints.

3.3.4 Project scheduling with labor constraints

In this subsection we consider an extended project scheduling problem with **labor constraints** and present a binary linear programming formulation for it. Given is a project with n activities $i = 1, \dots, n$ and precedences $(i, j) \in A$ between the activities. Furthermore there is a time horizon of T periods $[t-1, t[$, $t = 1, \dots, T$. Associated with each activity i is a processing time p_i and a labor profile $(l_{i1}, \dots, l_{ip_i})$ where $l_{i\nu}$ is the number of employees needed to process activity i during the ν-th processing period for $\nu = 1, \dots, p_i$. Let $n + 1$ be a dummy finishing activity with processing time zero which is a successor of all activities and needs no employee.

There is a finite set E of employees which have to perform the activities. Associated with each employee $e \in E$ is a set P_e of working patterns π indicating in which time periods e is available. π may be represented by a zero-one vector $(\pi(t))_{t=1}^T$ where $\pi(t) = 1$ if and only if the employee is available in period $[t-1, t[$. An employee may not be able to process any job. Let Q_e be the set of all activities which can be processed by employee e due to his qualification.

One has to

- choose a working pattern π_e for each employee e,
- assign to all working periods of the working pattern π_e an activity $i \in Q_e$ to be performed by e,

and to schedule the activities in such a way that

- the precedence constraints are satisfied,
- the number of employees needed by the activities in every processing period is available, and
- the makespan (or some other objective function) is minimized.

Under the assumption that a working pattern π_e is already fixed for each employee, a time-indexed IP for this problem with makespan objective can be formulated as follows. In this formulation we have binary variables x_{it} for all $i \in V = \{0, 1, \dots, n, n+1\}$ and $t = 0, \dots, T$ with

$$x_{it} := \begin{cases} 1, & \text{if activity } i \text{ starts at time } t \\ 0, & \text{otherwise} \end{cases}$$

and binary variables y_{ite} for all $i \in V; t = 1, \dots, T; e \in E$ with

$$y_{ite} := \begin{cases} 1, & \text{if employee } e \text{ is assigned to activity } i \text{ in period } [t-1, t[\\ 0, & \text{otherwise.} \end{cases}$$

Then the problem can be formulated as follows.

$$\begin{aligned}
\min \quad & \sum_{t=0}^{T} t x_{n+1,t} && && (3.59)\\
\text{s.t.} \quad & \sum_{t=0}^{T} x_{it} = 1 && (i \in V) && (3.60)\\
& \sum_{t=0}^{T} t x_{jt} - \sum_{t=0}^{T} t x_{it} \geq p_i && ((i,j) \in A) && (3.61)\\
& \sum_{i \in Q_e} y_{ite} = \pi_e(t) && (e \in E; t = 1, \ldots, T) && (3.62)\\
& \sum_{\nu=1}^{p_i} l_{it} x_{i,t-\nu+1} - \sum_{e \in E} y_{ite} \leq 0 && (i \in V; t = 1, \ldots, T) && (3.63)\\
& x_{it} \in \{0,1\} && (i \in V;\ t = 0, \ldots, T) && (3.64)\\
& y_{ite} \in \{0,1\} && (i \in V;\ t = 1, \ldots, T; e \in E) && (3.65)
\end{aligned}$$

Due to (3.60) exactly one starting time is assigned to each activity i, constraints (3.61) represent the precedence constraints. By the constraints (3.62) an activity is assigned to employee e for each of the active periods in his working pattern π_e. The constraints (3.63) guarantee that the labor demand of each activity during its processing time is satisfied.

This formulation involves $O(nT|E|)$ binary variables and $O(n+|A|+T(n+|E|))$ constraints.

In a more general version it may be required that additionally a working pattern $\pi_e \in P_e$ has to be assigned to each employee e. In this case, binary variables $z_{e\pi} \in \{0,1\}$ for all $e \in E, \pi \in P_e$ may be introduced where $z_{e\pi} = 1$ if and only if the working pattern $\pi \in P_e$ is assigned to e. Furthermore, the following constraints must be added:

$$\sum_{\pi \in P_e} z_{e\pi} = 1 \quad (e \in E) \tag{3.66}$$

$$\sum_{\pi \in P_e} \pi z_{e\pi} = \pi_e \quad (e \in E) \tag{3.67}$$

In a simpler version $L(t)$ employees are available in time period $[t-1, t[$ for $t = 1, \ldots, T$ and all employees are qualified to process all activities. To model this special situation, the y_{ite}-variables with constraints (3.62) may be dropped and the constraints (3.63) have to be replaced by

$$\sum_{i=1}^{n} \sum_{\nu=1}^{p_i} l_{it} x_{i,t-\nu+1} \leq L(t) \quad (t = 1, \ldots, T). \tag{3.68}$$

3.3.5 Reference notes

The basic MIP formulation DT was first published by Pritsker et al. [168], while the DDT-formulation is due to Christofides et al. [44]. The formulation based on the concept of forbidden sets can be found in Alvarez-Valdés and Tamarit [5]. Artigues et al. [6] proposed the FCT-formulation. Event-based formulations are presented in Koné et al. [124] where also a computational comparison of different formulations can be found. Linear programming based lower bounds are also surveyed in Néron et al. [150] and computationally tested in Demassey et al. [53]. Project scheduling problems with labor constraints are, for example, studied in Drezet and Billaut [71] and Brucker et al. [39].

3.4 A SAT Formulation of the RCPSP

Another possibility to model the RCPSP was presented by Horbach [102] who gave a SAT formulation and applied a very effective adapted DPLL-algorithm (cf. Section 2.8) to it. We consider the decision version of the RCPSP where an upper bound T for the makespan is given and one has to find a schedule with $C_{\max} \leq T$. Given an instance of this problem, we calculate time windows $[r_i, d_i]$ for all activities $i \in V$ by longest path calculations. Let $\mathcal{S}_i := \{r_i, \ldots, d_i - p_i\}$ be the set of all possible starting times of activity i.

To formulate the problem as SAT problem, the following variables are used:

- For each $i \in V$ and each $t \in \mathcal{S}_i$ we have a start variable s_{it} which is true if and only if i starts at time t.
- For each $i \in V$ and each $t \in \{r_i, \ldots, d_i - 1\}$ we have a process variable u_{it} which is true if and only if i is processed in period $[t, t+1[$.

Additionally, we have five types of clauses:

$$\bigvee_{t \in \mathcal{S}_i} s_{it} \qquad \text{for } i \in V \tag{3.69}$$

$$\neg s_{it} \vee \neg s_{i\tau} \qquad \text{for } i \in V;\ t, \tau \in \mathcal{S}_i, t \neq \tau \tag{3.70}$$

$$\neg s_{it} \vee u_{i\tau} \qquad \text{for } i \in V; t \in \mathcal{S}_i, \tau \in \{t, \ldots, t + p_i - 1\} \tag{3.71}$$

$$\neg s_{jt} \quad \bigvee_{\tau \in \{r_i, \ldots, t - p_i\}} \quad s_{i\tau} \qquad \text{for } (i, j) \in A, t \in \mathcal{S}_j \tag{3.72}$$

$$\bigvee_{i \in F} \neg u_{it} \qquad \text{for } t \in \{0, \ldots, T-1\}, F \in \mathcal{F}. \tag{3.73}$$

Clauses (3.69) and (3.70) ensure that to each activity i exactly one starting time is assigned. Due to (3.71) activity i is processed for p_i consecutive periods when started at time t. The precedence constraints are covered by (3.72) which guarantee that if an activity j is started at time t, all its predecessors i are completed before. The last clauses take care of the resource constraints by using the set $\mathcal{F}$ of all minimal forbidden sets. Recall from Section 3.3.2 that a forbidden set F is a precedence-feasible, but resource-infeasible subset $F \subset V$ of activities. Furthermore, it is minimal if for all $i \in F$ the set $F \setminus \{i\}$ is not a forbidden set.

It is not difficult to show that a feasible schedule exists if and only if the Boolean formula with clauses (3.69) to (3.73) is satisfiable. The variables s_{it} which are set to true, define the starting times t of the activities i for such a schedule.

Unfortunately, the number of clauses (3.73) grows exponentially with the number of activities. Therefore, it is not promising to create all of them in advance. Instead, each time a satisfying assignment is found, clauses which are violated by this assignment could be created and added to the current set of clauses (similar to the introduction of cuts in integer linear programming). Then the SAT solver has to be restarted.

3.5 General Objective Functions for Problems without Resource Constraints

In this section we consider project scheduling problems with more general objective functions than the makespan, but assume that no resource constraints are given. More precisely, we study the following optimization problem (P)

$$\begin{aligned} \min \quad & f(S_0, \ldots, S_{n+1}) && (3.74) \\ \text{s.t.} \quad & S_j - S_i \geq d_{ij} \qquad ((i,j) \in A) && (3.75) \\ & S_0 = 0. && (3.76) \end{aligned}$$

As before, $V = \{0, \ldots, n+1\}$ is a set of activities (operations), where 0 and $n+1$ are a dummy starting and terminating activity, respectively. The set A is a set of directed arcs of a graph $G = (V, A)$ with vertex set V. Associated with each arc $(i,j) \in A$ is a value $d_{ij} \in \mathbb{Z}$ representing a minimal start-start distance between activities i and j. The vector $S = (S_i)_{i=0}^{n+1}$ represents the starting times of all activities $i = 0, \ldots, n+1$ in a nonpreemptive schedule (inducing completion times $C_i = S_i + p_i$), and $f(S) = f(S_0, \ldots, S_{n+1})$ may be interpreted as the costs of schedule S.

Additionally, we assume that a time interval $[0, T]$ is given, in which all activities have to be processed. As described in Section 3.1.1, the conditions $0 = S_0 \leq S_i \leq S_i + p_i \leq S_{n+1} \leq T$ may be covered by relations of the form (3.75) with minimal distances $d_{0i} = 0$, $d_{i,n+1} = p_i$ for all $i = 1, \ldots, n$ and $d_{n+1,0} = -T$.

For the remainder of this section we assume that a feasible solution S for problem (P) exists, which can be easily checked by longest path calculations. If the Floyd-Warshall algorithm detects a positive cycle in the network (V, A) according to the distances d, no feasible solution exists, i.e. we do not have to search for an optimal solution of (P).

Problem (P) with $d_{ij} = p_i$ for all $(i, j) \in A$ may be interpreted as an RCPSP in which the resource constraints are relaxed and the makespan objective function $\max\limits_{i=0}^{n}\{S_i + p_i\}$ is replaced by an arbitrary objective function $f(S_0, \ldots, S_{n+1})$. Thus, the solution of (P) provides a lower bound for such an RCPSP.

In the following we will discuss solution methods for problem (P) under the assumption that the objective function $f(S_0, \ldots, S_{n+1})$ has additional properties. In Subsection 3.5.1 regular and antiregular objective functions are considered. Afterwards we assume that $f(S_0, \ldots, S_{n+1})$ is separable, i.e. f can be written as $f(S) = \sum\limits_{i=0}^{n+1} f_i(S_i)$. While in Subsection 3.5.2 linear functions $f(S) = \sum\limits_{i=0}^{n+1} b_i S_i$ are considered, in Subsection 3.5.3 all functions $f_i(S_i)$ are assumed to be convex piecewise linear. In both cases the problem can be reduced to a maximum cost flow problem. Finally, in Subsection 3.5.4 a pseudo-polyomial algorithm for a general sum objective function is derived.

3.5.1 Regular functions

On page 12 several examples for regular objective functions of the form $f(S) = \sum f_i(S_i)$ or $f(S) = \max f_i(S_i)$ have been introduced (like the weighted flow time $\sum w_i C_i$ with non-negative weights $w_i \geq 0$, the maximum lateness $L_{\max}$ or the total weighted tardiness $\sum w_i T_i$ with $w_i \geq 0$ for all i).

If f is regular, i.e. if $f(S) = f(S_0, \ldots, S_{n+1}) \leq f(S') = f(S'_0, \ldots, S'_{n+1})$ holds for all schedules S, S' with $S_i \leq S'_i$ for $i = 0, \ldots, n+1$, then an earliest-start schedule $S^* = (S_i^*)$ is optimal. It can be obtained by calculating the lengths ℓ_i of longest paths from 0 to i in the network (V, A) with arc distances d_{ij} and setting $S_i^* := \ell_i$ for $i = 0, \ldots, n+1$.

Symmetrically, for an antiregular objective function (i.e. $f(S) \geq f(S')$ for all schedules S, S' with $S_i \leq S'_i$ for $i = 0, \ldots, n+1$) a latest-start schedule is optimal. It can be obtained by reversing all arcs in A and calculating longest path lengths from the dummy terminating activity $n+1$ to each node i in the resulting network (V, A') with $A' := \{(j, i) \mid (i, j) \in A\}$ and arc distances $d'_{ji} := d_{ij}$ for all $(i, j) \in A$. If ℓ'_i denotes the calculated longest path lengths, activity i is started at time $T - \ell'_i$.

If the arc set A is acyclic, the longest path lengths may be calculated in a topological order in $O(|A|)$ time. For the general case (i.e. a cyclic graph with arbitrary distances $d_{ij} \in \mathbb{Z}$) a label-correcting algorithm with complexity $O(|V||A|)$ may be used (cf. Section 2.2.2).

3.5.2 Linear functions

A linear objective function has the form $f(S) = \sum\limits_{i=0}^{n+1} f_i(S_i) = \sum\limits_{i=0}^{n+1} b_i S_i$, where the coefficients b_i are given integers. If some of the b_i are negative, while others are positive, then the corresponding linear objective function is neither regular nor antiregular. An example for such a function is the weighted flow time $\sum\limits_{i=0}^{n} w_iC_i = \sum\limits_{i=0}^{n} w_i(S_i + p_i)$ with arbitrary weights $w_i \in \mathbb{Z}$ which (up to the additive constant $\sum\limits_{i=0}^{n} w_ip_i$) is a linear function.

The corresponding problem (P) is equivalent to

$$\begin{aligned}
&\min && \sum_{i=0}^{n+1} b_iS_i && && (3.77)\\
&\text{s.t.} && S_j - S_i \geq d_{ij} && ((i,j) \in A) && (3.78)\\
& && S_0 = 0 && && (3.79)
\end{aligned}$$

where b_0 can be chosen arbitrarily. The dual has the form

$$\begin{aligned}
&\max && \sum_{(i,j)\in A} d_{ij}x_{ij} && && (3.80)\\
&\text{s.t.} && \sum_{\{j|(j,i)\in A\}} x_{ji} - \sum_{\{j|(i,j)\in A\}} x_{ij} = b_i && (i = 1,\dots,n+1) && (3.81)\\
& && \lambda - \sum_{j=1}^{n+1} x_{0j} = b_0 && && (3.82)\\
& && x_{ij} \geq 0 && ((i,j) \in A) && (3.83)\\
& && \lambda \in \mathbb{R}. && && (3.84)
\end{aligned}$$

If we choose b_0 such that $\sum\limits_{i=0}^{n+1} b_i = 0$ holds, then by adding all equations (3.81) for $i = 1,\dots,n+1$ and equation (3.82) we conclude $\lambda = 0$. Thus, with this choice of b_0 the dual has the form

$$\begin{aligned}
&\max && \sum_{(i,j)\in A} d_{ij}x_{ij} && && (3.85)\\
&\text{s.t.} && \sum_{\{j|(j,i)\in A\}} x_{ji} - \sum_{\{j|(i,j)\in A\}} x_{ij} = b_i && (i = 1,\dots,n+1) && (3.86)\\
& && \sum_{j=1}^{n+1} x_{0j} = -b_0 && && (3.87)\\
& && x_{ij} \geq 0 && ((i,j) \in A). && (3.88)
\end{aligned}$$

This problem is a maximum cost flow problem, which is equivalent to a minimum cost flow problem and can be polynomially solved by methods discussed in Section 2.5.6.

A special linear objective function, which is of practical interest in connection with the job-shop problem, is the function $\sum w_i(C_i - S_i)$, where $C_i - S_i$ denotes the difference between the completion time $C_{n_i,i}$ of the last operation of job i and the starting time S_{1i} of the first operation of job i. This situation can be handled as follows. For each job $i = i_1 \to i_2 \to \ldots \to i_{n_i}$ we introduce an additional starting job i^{min} and an additional finishing job i^{max} with distances $d_{i^{min},i_1} = 0$ and $d_{i_n,i^{max}} = p_{i_n}$. Then the corresponding variables $S_{i^{min}}$ and $S_{i^{max}}$ satisfy $S_{i^{min}} \leq S_i = S_{i_1}$ and $S_{i^{max}} \geq C_i = S_{i_{n_i}} + p_{n_i}$. By setting the coefficients in the objective function to $b_{i^{min}} := -w_i$, $b_{i^{max}} := w_i$ and $b_{i_j} := 0$ for all operations i_j of job i, we get the objective $\min \sum\limits_{i=1}^{n} w_i(S_{i^{max}} - S_{i^{min}})$, which is equivalent to minimize $\sum w_i(C_i - S_i)$.

Also terms involving the lateness of jobs may be added to the objective function (3.77). If the maximum lateness of jobs in a subset I (e.g. corresponding to jobs in a partial project) is important, we introduce an additional job I with $d_{iI} = p_i - d_i$ for all jobs $i \in I$. Then $S_I \geq L_i = S_i + p_i - d_i$. By setting the coefficient $b_I := 1$ we get the problem to minimize S_I subject to the conditions $S_I \geq L_i$ for all $i \in I$, which is equivalent to minimize $L_{\max}$ in the subset I.

3.5.3 Convex piecewise linear functions

In this section we generalize the preceding concepts derived for linear objective functions and study problem (P) with a sum objective function $f(S) = \sum\limits_{i=0}^{n+1} f_i(S_i)$, where all functions f_i are convex piecewise linear and bounded from below. An example for such a function is the weighted earliness-tardiness $\sum\limits_{i=1}^{n} w_i^E E_i + \sum\limits_{i=1}^{n} w_i^T T_i$ with earliness weights $w_i^E \geq 0$ and tardiness weights $w_i^T \geq 0$.

A function $f : \mathbb{R} \to \mathbb{R}$ is called **convex** if for all $\lambda \in [0,1]$ and all $y_1, y_2 \in \mathbb{R}$ the inequality

$$f(\lambda y_1 + (1-\lambda)y_2) \leq \lambda f(y_1) + (1-\lambda) f(y_2)$$

holds. Furthermore, f is **piecewise linear** if breaking points $b_0 := -\infty < b_1 < b_2 < \ldots < b_m < b_{m+1} := \infty$ exist such that f is a linear function in each interval $]b_{k-1}, b_k[$ for $k = 1, \ldots, m+1$. We denote by Θ_k the slope of f in the interval $]b_{k-1}, b_k[$ for $k = 1, \ldots, m+1$, and assume that $f(y)$ achieves its minimum in b_l. Then we have $\Theta_1 < \Theta_2 < \ldots < \Theta_l \leq 0 \leq \Theta_{l+1} < \ldots < \Theta_{m+1}$ and $f(y)$ may be written as

$$f(y) = \beta + \sum_{k=1}^{l} \gamma_k (b_k - y)^+ + \sum_{k=l+1}^{m+1} \gamma_k (y - b_{k-1})^+ \qquad (3.89)$$

where $y^+ := \max\{y, 0\}$, $\beta := f(b_l)$ and

$$\gamma_k := \Theta_{k+1} - \Theta_k \text{ for } k = 1, \ldots, l-1; \qquad \gamma_l := -\Theta_l; \qquad \gamma_{l+1} := \Theta_{l+1};$$
$$\gamma_k := \Theta_k - \Theta_{k-1} \text{ for } k = l+2, \ldots, m+1.$$

Example 3.11 : Consider the convex piecewise linear function shown in Figure 3.18. It has the representation

$$\begin{aligned} f(y) &= f(b_2) + (\Theta_2 - \Theta_1)(b_1 - y)^+ - \Theta_2(b_2 - y)^+ + \Theta_3(y - b_2)^+ \\ &\quad +(\Theta_4 - \Theta_3)(y - b_3)^+ + (\Theta_5 - \Theta_4)(y - b_4)^+. \end{aligned}$$

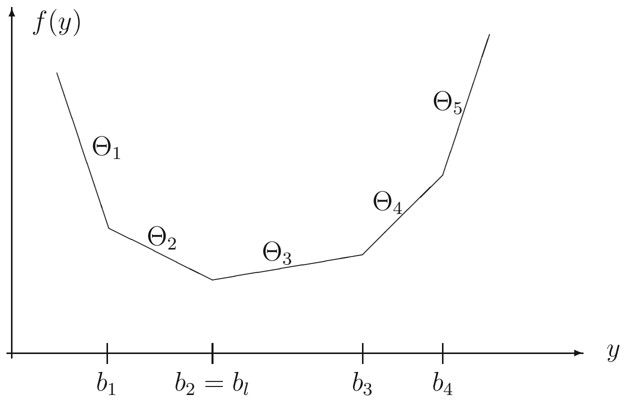


Figure 3.18: A convex piecewise linear function

For example, for $y \in]-\infty, b_1[$ we have

$$\begin{aligned} f(y) &= f(b_2) + (\Theta_2 - \Theta_1)(b_1 - y)^+ - \Theta_2(b_2 - y)^+ \\ &= f(b_2) + (\Theta_2 - \Theta_1)(b_1 - y) - \Theta_2(b_2 - b_1) - \Theta_2(b_1 - y) \\ &= f(b_2) - \Theta_2(b_2 - b_1) - \Theta_1(b_1 - y) \end{aligned}$$

and for $y \in]b_4, \infty[$ we have

$$\begin{aligned} f(y) &= f(b_2) + \Theta_3(y - b_2) + (\Theta_4 - \Theta_3)(y - b_3) + (\Theta_5 - \Theta_4)(y - b_4) \\ &= f(b_2) + \Theta_3(b_3 - b_2) + \Theta_4(b_4 - b_3) + \Theta_5(y - b_4). \end{aligned}$$

Expressions for other y-values are derived similarly. □

Since f is convex, all values γ_k are non-negative. Furthermore, $(b_k - y)^+ = \min \{\alpha_k \mid \alpha_k \geq 0, y + \alpha_k \geq b_k\}$ for $k = 1, \ldots, l$ and $(y - b_{k-1})^+ = \min \{\alpha_k \mid \alpha_k \geq 0, -y + \alpha_k \geq -b_{k-1}\}$ for $k = l+1, \ldots, m+1$. Thus, the value $f(y)$ in (3.89) may also be calulated as the optimal objective value of the following linear program

$$f(y) = \min \quad \sum_{k=1}^{m+1} \gamma_k \alpha_k + \beta \tag{3.90}$$

$$\text{s.t.} \quad y + \alpha_k \geq b_k \quad (k = 1, \ldots, l) \tag{3.91}$$

$$-y + \alpha_k \geq -b_{k-1} \quad (k = l+1, \ldots, m+1) \tag{3.92}$$

$$\alpha_k \geq 0 \quad (k = 1, \ldots, m). \tag{3.93}$$

We now consider problem (P) with a sum objective function

$$f(S_0,\ldots,S_{n+1}) = \sum_{\substack{i,j=0\\ i\neq j}}^{n+1} f_{ij}(S_j - S_i), \tag{3.94}$$

where all functions f_{ij} are convex piecewise linear and bounded from below. Let $b_1^{ij}, b_2^{ij}, \ldots, b_{m_{ij}}^{ij}$ be the breaking points of f_{ij} and assume that l_{ij} is the breaking point where f_{ij} achieves its minimum. Then problem (P) with objective function (3.94) can be written as

$$\min \sum_{\substack{i,j=0\\ i\neq j}}^{n+1} \sum_{k=1}^{m_{ij}+1} \gamma_k^{ij}\alpha_k^{ij} + \beta \tag{3.95}$$

$$\begin{aligned}
\text{s.t.}\quad S_j - S_i + \alpha_k^{ij} &\geq b_k^{ij} && (i,j\in V,\ i\neq j; k=1,\ldots,l_{ij}) && (3.96)\\
S_i - S_j + \alpha_k^{ij} &\geq -b_{k-1}^{ij} && (i,j\in V,\ i\neq j; k=l_{ij}+1,\ldots,m_{ij}+1) && (3.97)\\
S_j - S_i &\geq d_{ij} && ((i,j)\in A_0) && (3.98)\\
\alpha_k^{ij} &\geq 0 && (i,j\in V,\ i\neq j; k=1,\ldots,m_{ij}+1) && (3.99)\\
S_i &\in \mathbb{R} && (i\in V) && (3.100)
\end{aligned}$$

In connection with this problem it is convenient to introduce a multigraph, i.e. a directed graph with parallel arcs. Associated with each pair (i,j), $i,j = 0,\ldots,n+1,\ i\neq j$, is a set of parallel arcs consisting of arcs

- $(i,j)_k$ for $k=1,\ldots,l_{ij}$, corresponding to the constraints (3.96),
- $(i,j)_k$ for $k=l_{ij}+1,\ldots,m_{ij}+1$, corresponding to the constraints (3.97), and
- $(i,j)_0$ if $(i,j)\in A_0$, corresponding to the constraint (3.98).

Let A be the set of all these (possibly parallel) arcs. Then the dual (we ignore the constant β) has the form

$$\max \sum_{\substack{i,j=0\\ i\neq j}}^{n+1}\sum_{k=1}^{l_{ij}} b_k^{ij}x_{ij}^k - \sum_{\substack{i,j=0\\ i\neq j}}^{n+1}\sum_{k=l_{ij}+1}^{m_{ij}+1} b_{k-1}^{ij}x_{ij}^k + \sum_{(i,j)\in A_0} d_{ij}x_{ij}^0 \tag{3.101}$$

$$\begin{aligned}
\text{s.t.}\quad \sum_{k=0}^{m}\sum_{\{j|(j,i)_k\in A\}} x_{ji}^k - \sum_{k=0}^{m}\sum_{\{j|(i,j)_k\in A\}} x_{ij}^k &= 0 && (i\in V) && (3.102)\\
x_{ij}^k &\leq \gamma_k^{ij} && ((i,j)_k\in A\setminus A_0) && (3.103)\\
x_{ij}^k &\geq 0 && ((i,j)_k\in A) && (3.104)
\end{aligned}$$

This problem is a maximum cost circulation problem, which can be solved polynomially by network flow methods (cf. Section 2.5.6).

In problem (P) with objective function (3.94) one may again assume $S_0 = 0$ because the problem does not change if each variable S_i is replaced by $S_i - S_0$.

3.5.4 General sum functions

In the following we consider problem (P) with an arbitrary sum objective

$$\begin{aligned} \min \quad & \sum_{i=0}^{n+1} f_i(S_i) & & \qquad (3.105)\\ \text{s.t.} \quad & S_j - S_i \geq d_{ij} & ((i,j) \in A) & \qquad (3.106)\\ & S_0 = 0. & & \qquad (3.107)\end{aligned}$$

From the distances d_{ij} we may calculate for each activity $i = 0, \ldots, n+1$ a time window $[ES_i, LS_i]$ consisting of an earliest starting time ES_i with respect to $ES_0 = 0$ and a latest starting time LS_i with respect to the given time horizon $LS_{n+1} = T$ (cf. Section 3.1.1). Recall that we assume that a feasible solution S for problem (P) exists, i.e. the network (V, A) with arc distances d contains no positive cycle. In this case ES_i is the length of a longest path from 0 to i and LS_i is equal to T minus the longest path length from i to $n+1$.

W.l.o.g. all function values $f_i(t)$ with $t \in \{ES_i, \ldots, LS_i\}$ can be assumed to be positive (otherwise we may add to each $f_i(t)$ the constant

$$C := \max_{i=0}^{n+1} \max_{t=ES_i}^{LS_i} \{|f_i(t)| + 1\}$$

resulting in an additive constant $(n+2)C$ for the objective function).

We will show that problem (P) can be reduced to a minimum cut problem in an arc-capacitated network $\tilde{G} = (\tilde{V}, \tilde{A})$ with arc capacities u, which is defined as follows:

- The set of nodes $\tilde{V}$ contains for each activity $i = 1, \ldots, n$ and each integer $t \in [ES_i, LS_i + 1]$ a node v_{it}. Furthermore, it contains a dummy source a and a dummy sink b. Thus,

$$\tilde{V} = \{a, b\} \cup \bigcup_{i=1}^{n} \{v_{it} \mid t \in \{ES_i, \ldots, LS_i + 1\}\}.$$

- The set of arcs $\tilde{A}$ can be divided into three disjoint subsets $\tilde{A}_F, \tilde{A}_T, \tilde{A}_D$.

  The set $\tilde{A}_F$ of **function-value arcs** is defined by

$$\tilde{A}_F := \bigcup_{i=1}^{n} \{(v_{it}, v_{i,t+1}) \mid t \in \{ES_i, \ldots, LS_i\}\}$$

  taking care of the objective function.

  The set $\tilde{A}_T$ of **temporal arcs** guarantees that no temporal constraint is violated. This set is defined by

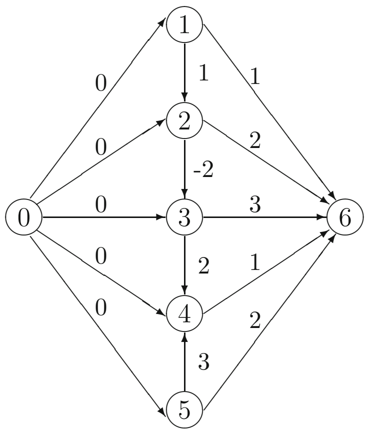
$$\begin{aligned}\tilde{A}_T := \bigcup_{i=1}^{n} \; & \{\, (v_{it}, v_{j,t+d_{ij}}) \mid (i,j) \in A;\\ & t \in \{\max(ES_i + 1, ES_j + 1 - d_{ij}), ..., \min(LS_i, LS_j - d_{ij})\}.\end{aligned}$$

The set $\tilde{A}_D$ of **dummy arcs** is defined by

$$\tilde{A}_D := \bigcup_{i=1}^{n} \{(a, v_{i,ES_i})\} \cup \bigcup_{i=1}^{n} \{(v_{i,LS_i+1}, b)\}.$$

- The arc capacities u of the function-value arcs $(v_{it}, v_{i,t+1})$ are equal to the function values $f_i(t)$. The capacities of all temporal and dummy arcs are equal to infinity.

Example 3.12: Consider a problem with $n = 5$ activities, processing times $p_1 = 1, p_2 = 2, p_3 = 3, p_4 = 1, p_5 = 2$, time horizon $T = 6$ and distances $d_{12} = 1,\ d_{23} = -2,\ d_{34} = 2,\ d_{54} = 3$ (cf. Figure 3.19). We introduce an additional dummy starting activity 0 and a dummy terminating activity $n+1 = 6$ and set $d_{0i} = 0$ and $d_{i,n+1} = d_{i6} = p_i$ for $i = 1, \dots, 5$. With the initial values $ES_0 := 0$ and $LS_{n+1} = LS_6 := T = 6$, the remaining earliest and latest starting times are obtained by calculating corresponding longest path lengths l_{0i} and $l_{i,n+1}$ for $i = 1, \dots, 5$ and setting $ES_i = l_{0i},\ LS_i = T - l_{i,n+1}$.



i	1	2	3	4	5
p_i	1	2	3	1	2
ES_i	0	1	0	3	0
$l_{i,n+1}$	3	2	3	1	4
LS_i	3	4	3	5	2

Figure 3.19: Earliest and latest starting times for Example 3.12

The corresponding network $\tilde{G} = (\tilde{V}, \tilde{A})$ is shown in Figure 3.20. The chains in the rows consisting of nodes v_{it} correspond to the activities $i = 1, \dots, 5$, the columns to the time periods $t = 0, \dots, 6$. The function-value arcs are drawn in a thick mode, all other arcs have infinite capacity. □

In the following we will show that our problem can be reduced to a minimum cut problem in $\tilde{G}$. Recall that an a,b-cut in $\tilde{G}$ is a pair $[X, \overline{X}]$ of disjoint sets $X, \overline{X} \subseteq \tilde{V}$ with $X \cup \overline{X} = \tilde{V}$ and $a \in X$, $b \in \overline{X}$ (cf. Section 2.5.4). The capacity $u[X, \overline{X}]$ of such a cut $[X, \overline{X}]$ is the sum

$$u[X, \overline{X}] := \sum_{\substack{v \in X \\ \overline{v} \in \overline{X}}} u_{v,\overline{v}}$$

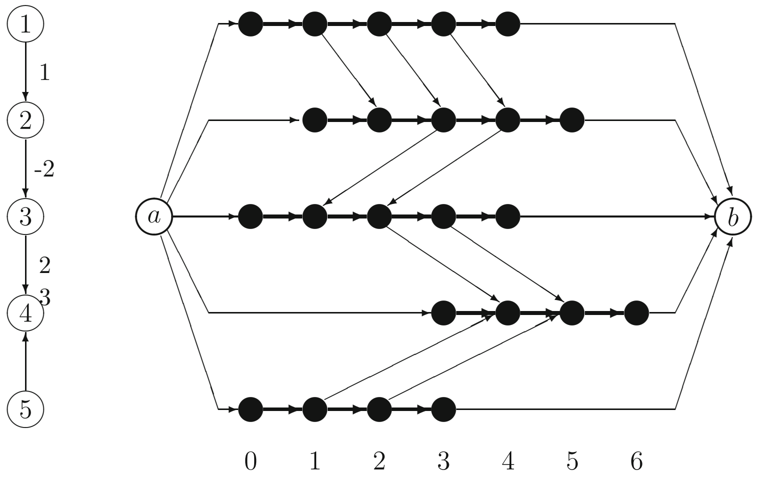


Figure 3.20: Network $\tilde{G} = (\tilde{V}, \tilde{A})$ for Example 3.12

of the capacities of forward arcs, i.e. all arcs $(v, \overline{v})$ with $v \in X$, $\overline{v} \in \overline{X}$.

At first we will show that for each feasible solution S of (P) an a,b-cut $[X, \overline{X}]$ exists such that $f(S) = u[X, \overline{X}]$. This implies that if S^* is an optimal solution for problem (P) and $[X^*, \overline{X}^*]$ is a minimum a,b-cut in $\tilde{G}$, then $f(S^*) \geq u[X^*, \overline{X}^*]$ holds. In a second step we will show that a minimum a,b-cut $[X^*, \overline{X}^*]$ provides a fesible solution S with $f(S) = u[X^*, \overline{X}^*]$. This solution must also be optimal for (P).

Constructing an a,b-cut from a feasible schedule

Given a feasible solution $S = (S_i)$, we define a cut $[X, \overline{X}]$ in $\tilde{G}$ by

$$X := \bigcup_{i=1}^{n} \{v_{it} \mid t \leq S_i\} \cup \{a\} \text{ and } \overline{X} := \tilde{V} \setminus X. \tag{3.108}$$

For example, for the earliest start schedule $S = (ES_i)$ from Example 3.12 we get the cut shown in Figure 3.21.

We now show that the capacity $u[X, \overline{X}]$ of this cut equals $f(S)$. All arcs (v_{iS_i}, v_{i,S_i+1}) with $i = 1, \ldots, n$ are forward arcs of $[X, \overline{X}]$, and the sum of the capacities of these arcs is equal to $f(S)$. Furthermore, no temporal arc is contained in $[X, \overline{X}]$ because if $v_{is} \in X$ and $v_{jt} \in \overline{X}$ then $s \leq S_i$ and $t > S_j$ due to the definition (3.108) of the cut. Since S is feasible, we have $d_{ij} \leq S_j - S_i < t - s$, i.e. $s + d_{ij} < t$. But then (v_{is}, v_{jt}) cannot be a temporal arc since for such an arc $t = s + d_{ij}$ holds. Thus, the cut contains only function-value arcs and $u[X, \overline{X}] = f(S)$.

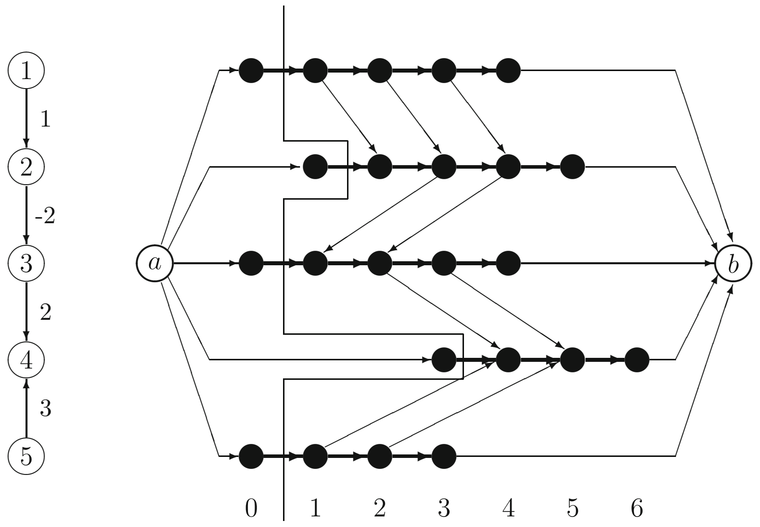


Figure 3.21: Cut in $\tilde{G} = (\tilde{V}, \tilde{A})$ corresponding to the earliest start schedule

Constructing a feasible schedule from a minimum a, b-cut

At first we will show that a minimum a, b-cut of $\tilde{G}$ contains for each activity $i = 1, \ldots, n$ exactly one function-value arc $(v_{it}, v_{i,t+1})$ as forward arc.

From the preceding arguments we know that a feasible solution S of (P) induces an a, b-cut with finite capacity (defined by (3.108)). Thus, also a minimum a, b-cut $[X, \overline{X}]$ has a finite capacity, i.e. it does not contain any of the dummy or temporal arcs as forward arcs. It follows that for each $i = 1, \ldots, n$ at least one function-value arc $(v_{it}, v_{i,t+1})$ is contained as forward arc in the cut $[X, \overline{X}]$. Assume that $[X, \overline{X}]$ for some i contains more than one of such function value arcs. Then we derive a contradiction by constructing a cut $[X', \overline{X'}]$ which has a smaller capacity than $[X, \overline{X}]$.

For each i let t_i be the smallest index such that $(v_{it_i}, v_{i,t_i+1}) \in (X, \overline{X})$. Define $X' := \bigcup\limits_{i=1}^{n} \{v_{it} \mid t \leq t_i\} \cup \{a\}$ and $\overline{X'} := \tilde{V} \setminus X'$. Clearly, $X' \subset X$ and the set of function-value arcs (v_{it_i}, v_{i,t_i+1}) is a proper subset of the corresponding set of function-value arcs of $(X, \overline{X})$. Recall that the weights $f_i(t)$ of all arcs $(v_{it}, v_{i,t+1})$ are positive. Thus, it is sufficient to show that $[X', \overline{X'}]$ does not contain any of the temporal arcs as forward arc. So assume that there exists a temporal arc $(v_{is}, v_{jt}) \in (X', \overline{X'})$ with $t = s + d_{ij}$ and $s \leq t_i$, $t > t_j$. Since $(v_{is}, v_{jt}) \in \tilde{A}$, it follows by the construction of $\tilde{G}$ that all arcs $(v_{i,s-z}, v_{j,t-z}) \in \tilde{A}$ for all integers z such that $ES_i + 1 \leq s - z \leq LS_i$ and $ES_j + 1 \leq t - z \leq LS_j$. Now let $z := t - t_j - 1$. Then

$$ES_j + 1 \leq t_j + 1 = t - z \leq t \leq LS_j$$

and

$$ES_i + 1 \leq t_i + 1 \leq t_j - d_{ij} + 1 = t_j + (s - t) + 1 = s - z \leq s \leq LS_i,$$

i.e. $(t - z) \in [ES_j + 1, LS_j]$ and $(s - z) \in [ES_i + 1, LS_i]$.

Since $s \leq t_i$ we have $v_{i,s-z} \in X' \subset X$. Moreover, by definition of t_j we have $v_{j,t-z} = v_{j,t_j+1} \in \overline{X}$. Thus, we have identified a temporal arc $(v_{i,s-z}, v_{j,t-z}) \in (X, \overline{X})$, which contradicts the fact that $[X, \overline{X}]$ has a finite capacity.

Now, in a second step we will construct a feasible schedule from a minimum cut. Given a minimum cut $[X^*, \overline{X^*}]$ in $\tilde{G}$ (which has finite capacity), for each $i = 1, \ldots, n$ consider the unique function-value arc $(v_{it_i}, v_{i,t_i+1}) \in (X^*, \overline{X^*})$. If we define $S_i := t_i$ for $i = 1, \ldots, n$, then the corresponding vector $S = (S_i)$ satisfies $f(S) = u[X^*, \overline{X^*}]$. It remains to show that S is feasible, i.e. that the constraints (3.106) are satisfied. Assume that for $s := S_i$ and $t := S_j$ we have a violated constraint (3.106), i.e. $s + d_{ij} > t$. Then for the temporal arc $(v_{is}, v_{i,s+d_{ij}})$ in $\tilde{G}$ we have $v_{is} \in X^*$ and $v_{j,s+d_{ij}} \in \overline{X^*}$, which contradicts the fact that $u[X^*, \overline{X^*}]$ is finite. Thus, S is feasible. We have proven

Theorem 3.6 If problem (P) with an arbitrary objective function $f(S) = \sum f_i(S_i)$ has a feasible solution, then an a,b-cut $[X^*, \overline{X^*}]$ of $\tilde{G} = (\tilde{V}, \tilde{A})$ with minimum capacity corresponds to an optimal solution S^* of problem (P) and $u[X^*, \overline{X^*}] = f(S^*)$. Moreover, $S^* = (S_i^*)$ is given by $S_i^* = t_i$ where (v_{i,t_i}, v_{i,t_i+1}) is the unique forward arc of activity i in $(X^*, \overline{X^*})$.

A minimum cut in $\tilde{G}$ can be calculated by solving a maximum flow problem for $\tilde{G}$ (cf. Section 2.5.4). Let $m := |A|$ be the number of relations (3.106). Since $\max\limits_{i=0}^{n+1}(LS_i - ES_i) = O(T)$, the graph $\tilde{G} = (\tilde{V}, \tilde{A})$ has $|\tilde{V}| = O(nT)$ nodes and $|\tilde{A}| = O((n+m)T) = O(mT)$ arcs. Thus, by applying the FIFO preflow-push algorithm for maximum flows (cf. Section 2.5.5) the minimum cut can be computed in $O(|\tilde{V}||\tilde{A}| \log(|\tilde{V}|^2/|\tilde{A}|) = O(nmT^2 \log \frac{n^2T}{m})$ time.

If each function f_i is explicitly given by its values $f_i(t)$ for $t \in \{ES_i, \ldots, LS_i\}$, then the input length of (P) is bounded by $O(|A| + nT) = O(m + nT)$. Thus, in this case the presented algorithm is polynomial. If, on the other hand, the functions $f_i(t)$ are given in a more compact form (e.g. by formulas which can be evaluated in polynomial time), then the input length of (P) is bounded by $O(m + \log T)$ since only the value T has to be encoded. Thus, in this case the presented algorithm is only pseudo-polynomial.

3.5.5 Reference notes

Problem (P) with linear or convex piecewise linear functions was studied by Wennink [193]. He showed that in both cases the problem can be reduced to a maximum cost flow problem and developed local search algorithms for solving it. Möhring et al. [147], [148] studied problem (P) with a general sum objective function in connection with lower bound calculations for the RCPSP based on Lagrangian relaxation (see also Section 3.7.2). They showed that the problem can be reduced to a minimum cut problem and used it to provide lower and upper bounds for the RCPSP.

3.6 Constraint Propagation

As already described in Section 2.7, constraint propagation may be a very efficient technique deducing new constraints from given ones. It may be used to tighten the search space in connection with branch-and-bound methods or local search heuristics. Also infeasibility of a given upper bound UB may be detected, which is important for lower bound calculations. In connection with the RCPSP usually additional timing restrictions are derived from given temporal and resource constraints. More specifically, additional precedence constraints are deduced and the time windows $[r_i, d_i]$ of the activities are strengthened. If a solution of the RCPSP is represented by precedence-feasible activity lists (cf. Section 3.2.1), additional precedence relations reduce the search space. On the other hand, smaller time windows $[r_i, d_i]$ reduce the number of possible starting times S_i of the activities, which may, for example, be advantageous for MIP formulations (cf. Section 3.3).

3.6.1 Basic relations

Given a schedule $S = (S_i)_{i=0}^{n+1}$, for each pair (i, j) of activities i, j exactly one of the three relations $i \rightarrow j$, $j \rightarrow i$, or $i \parallel j$ holds:

- We have a so-called **conjunction** $i \rightarrow j$ if activity j does not start before i is finished, i.e. if
$$S_i + p_i \leq S_j. \tag{3.109}$$
- We have a so-called **parallelity relation** $i \parallel j$ if activities i and j are processed in parallel for at least one time unit, i.e. if
$$C_i = S_i + p_i > S_j \text{ and } C_j = S_j + p_j > S_i. \tag{3.110}$$

Furthermore, the negation of a parallelity relation $i \parallel j$ is a so-called **disjunction** $i - j$ which means that either $i \rightarrow j$ or $j \rightarrow i$ must hold.

We denote the set of all conjunctions by C, the set of all disjunctions by D and the set of all parallelity relations by N. The following relations may be immediately derived from the given data of an RCPSP-instance: An initial set C_0 of conjunctions is given by the set of all precedence relations $i \rightarrow j \in A$. An initial set D_0 of disjunctions is induced by resource constraints for pairs of activities by setting $i - j \in D_0$ if $r_{ik} + r_{jk} > R_k$ for some renewable resource $k \in \mathcal{K}^\rho$ holds.

An initial set N_0 of parallelity relations may be derived by considering two activities i, j with time windows $[r_i, d_i]$ and $[r_j, d_j]$. If
$$p_i + p_j > \max\{d_i, d_j\} - \min\{r_i, r_j\} \tag{3.111}$$
holds, then i and j overlap in any feasible schedule (cf. Figure 3.22). Thus, we may define N_0 as the set of all parallelity relations $i \parallel j$ where i and j satisfy (3.111).

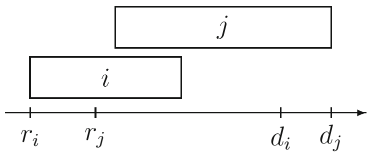


Figure 3.22: Overlapping activities

3.6.2 Start-start distance matrix

Let $S = (S_i)_{i=0}^{n+1}$ be a feasible schedule with $S_0 = 0$ and $S_{n+1} \leq UB$. Then we have $i \to j$ if and only if (3.109), i.e. $S_j - S_i \geq p_i$ holds. Furthermore, we have $i \parallel j$ if and only if (3.110) holds which is equivalent to

$$S_j - S_i \geq -(p_j - 1) \text{ and } S_i - S_j \geq -(p_i - 1) \tag{3.112}$$

because all data are integers. Additionally, for arbitrary activities i, j we have

$$S_i + p_i \leq S_{n+1} \leq UB \leq UB + S_j \text{ or equivalently } S_j - S_i \geq -(UB - p_i)$$

because $S_j \geq S_0 = 0$ and $S_{n+1} \leq UB$.

If we define

$$d_{ij} := \begin{cases} 0, & \text{if } i = j \\ p_i, & \text{if } i \to j \\ -(p_j - 1), & \text{if } i \parallel j \\ -(UB - p_i), & \text{otherwise,} \end{cases} \tag{3.113}$$

for $i, j = 0, \ldots, n+1$, then the entries d_{ij} of the $(n+2) \times (n+2)$-matrix $d = (d_{ij})$ are lower bounds for the differences $S_j - S_i$, i.e.

$$S_j - S_i \geq d_{ij} \text{ for all } i, j = 0, \ldots, n+1. \tag{3.114}$$

We call d a **start-start distance (SSD)-matrix**.

Due to (3.109), (3.112) and (3.113) we have:

- $i \to j$ if and only if $d_{ij} \geq p_i$ holds, (3.115)
- $i \parallel j$ if and only if both $d_{ij} \geq -(p_j - 1)$ and $d_{ji} \geq -(p_i - 1)$ hold. (3.116)

If additionally generalized precedence constraints (1.1) with time-lags d'_{ij} are given, we have to fulfill $S_i + d'_{ij} \leq S_j$ or equivalently $S_j - S_i \geq d'_{ij}$. Thus, we may incorporate these constraints by setting

$$d_{ij} := \max\{d_{ij}, d'_{ij}\}. \tag{3.117}$$

In this situation all results derived in the following are also valid for the RCPSP with generalized precedence constraints.

The first row of an SSD-matrix represents heads r_i because $S_i = S_i - S_0 \geq d_{0i}$ for each activity i. Similarly, the last column of a SSD-matrix contains information on tails q_i of the activities. More precisely, $d_{i,n+1} - p_i$ is a tail q_i of activity i because

$$S_{n+1} - (S_i + p_i) = (S_{n+1} - S_i) - p_i \geq d_{i,n+1} - p_i.$$

Due to the equation $d_i = UB - q_i$, also deadlines may be deduced from the matrix ($d_i = UB - d_{i,n+1} + p_i$).

The relation $S_j - S_i \geq d_{ij}$ has the following transitivity property:

$$S_j - S_i \geq d_{ij} \text{ and } S_k - S_j \geq d_{jk} \text{ imply } S_k - S_i \geq d_{ij} + d_{jk}. \tag{3.118}$$

Hence, the matrix $d = (d_{ij})$ can be replaced by its **transitive closure** $\overline{d} = (\overline{d}_{ij})$. This can be done with complexity $O(n^3)$ by applying the Floyd-Warshall algorithm (cf. Section 2.2.4 or Example 2.25) to d. For $i, j = 0, \ldots, n+1$ the recursion

$$d^k(i,j) := \max\{d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)\}$$

is evaluated for $k = 1, \ldots, n+1$, where initially $d^0(i,j) := d_{ij}$ is set.

If $d_{ii} > 0$ for some activity i holds, we get the contradiction $0 = S_i - S_i \geq d_{ii} > 0$. Thus, in this situation no feasible schedule with $C_{\max} \leq UB$ exists and we may state infeasibility. Especially, if generalized precedence relations are given, the test $d_{ii} > 0$ is important. In this case the activity-on-node network $G = (V, A)$ with arc weights d_{ij} contains a positive cycle, which implies that no feasible schedule exists even if all resource constraints are relaxed.

Calculating the transitive closure of the SSD-matrix is the simplest form of constraint propagation. One may start with a SSD-matrix containing basic information about an RCPSP-instance like the relations in C_0, D_0, N_0, heads, tails and generalized precedence constraints. Then constraint propagation methods are applied which result in additional relations and increased entries of d.

Some disjunctions may immediately be converted into conjunctions by the following observation. If we have

$$d_{ij} \geq -(p_j - 1) \text{ for activities } i, j \text{ with } i - j \in D, \tag{3.119}$$

then we get

$$S_j - S_i \geq d_{ij} \geq -(p_j - 1), \text{ i.e. } C_j = S_j + p_j \geq S_i + 1 > S_i.$$

Since the disjunctive activities i and j cannot be processed simultaneously, this implies $i \rightarrow j$. In this case we may set $d_{ij} := p_i$.

Note that the check $d_{ij} \geq -(p_j - 1)$ for a disjunction $i - j \in D$ also covers the following situation: If for two disjunctive activities i, j with time windows $[r_i, d_i], [r_j, d_j]$ the inequality

$$r_j + p_j + p_i > d_i \tag{3.120}$$

holds, obviously, in any feasible schedule j cannot be processed before i, i.e. $i \rightarrow j$ must hold. If the distance matrix d is transitively adjusted and (3.120) is satisfied, we also have

$$d_{ij} \geq d_{i,n+1} + d_{n+1,0} + d_{0j} = (UB + p_i - d_i) - UB + r_j = r_j + p_i - d_i > -p_j,$$

i.e. (3.119) is implied. Thus, it is sufficient to check condition (3.119).

More complicated constraint propagation methods will be discussed in the next subsections.

3.6.3 Symmetric triples and extensions

In this subsection we show how additional relations can be derived from the given ones and some further conditions.

A triple (i, j, k) of activities is called a **symmetric triple** if

- $k \parallel i$ and $k \parallel j$, and
- i, j, k cannot be processed simultaneously due to resource constraints.

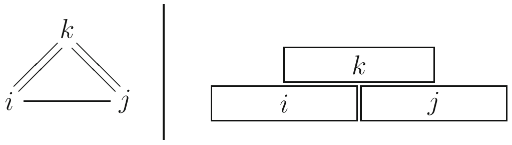


Figure 3.23: Symmetric triple

For a symmetric triple (i, j, k) we can add $i - j$ to D (see Figure 3.23), which can be seen as follows. Assume that $i - j$ does not hold. Then, we have $i \parallel j$, i.e. i and j overlap in some interval $[t, t']$ (see Figure 3.24). Since i, j, k cannot be processed simultaneously, k must be completed not later than time t or cannot be started before time t'. In both cases k cannot overlap with both i and j, which is a contradiction.

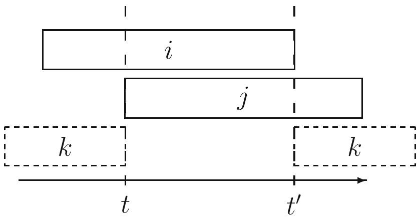


Figure 3.24: Assumption $i \parallel j$

All symmetric triples can be found in $O(nr|N|)$ time because for each $k \parallel j \in N$ we check for at most $O(n)$ activities i whether $k \parallel i$ holds. If this is the case, we check in $O(r)$ time whether a resource exists such that i, j, k cannot be processed simultaneously.

In connection with symmetric triples (i, j, k) further relations can be deduced:

(1) If $l \parallel i$ and j, k, l cannot be processed simultaneously, then $l - j$ can be added to D (see Figure 3.25). If additionally $i \to j \in C$ ($j \to i \in C$), then $l \to j \in C$ ($j \to l \in C$) may be deduced.

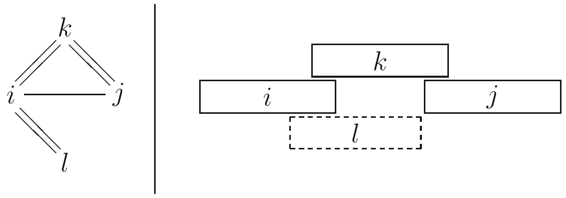


Figure 3.25: Extension of a symmetric triple – Condition (1)

Proof: If $l \parallel j$ would hold, then we have a situation as shown in Figure 3.26 (where the 'roles' of i and j may be interchanged).

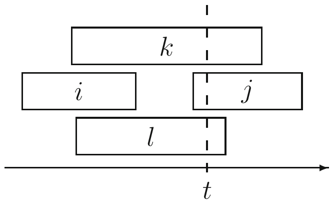


Figure 3.26: Assumption $l \parallel j$ in Condition (1)

Thus, at some time t activities j, k, l must be processed simultaneously, which is a contradiction. Furthermore, $i \to j$ implies $l \to j$ because otherwise $j \to l$ would imply $i \to l$ by transitivity, which contradicts $l \parallel i$. Similarly, the last claim can be proved. □

(2) Suppose that the conditions $p_k - 1 \leq p_i$, $p_k - 1 \leq p_j$ and $p_k - 1 \leq p_l$ hold. If i, k, l and j, k, l cannot be processed simultaneously, then $k - l \in D$ may be deduced (see Figure 3.27).

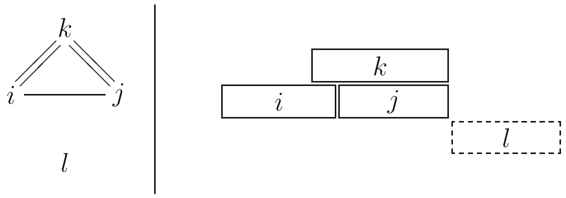


Figure 3.27: Extension of a symmetric triple – Condition (2)

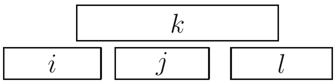


Figure 3.28: Assumption $k \parallel l$ in Condition (2)

Proof: Because $p_k - 1 \leq p_l$, it is not possible that l is processed between i and j since i and j occupy at least two time units of k.

Thus, if $k \parallel l$ holds, then we have a situation as shown in Figure 3.28 (where the roles of i and j may be interchanged or l may be processed before i and j). But this contradicts $p_k - 1 \leq p_j$ or $p_k - 1 \leq p_i$. □

Other propagation conditions are listed below. The proofs are similar to the proofs for (1) and (2). All the checks can be done in $O(rn^2|N|)$ time.

(3) Suppose that the conditions $p_k - 1 \leq p_i$ and $p_k - 1 \leq p_l$ hold. Furthermore, assume that the precedence relations $l \rightarrow j$ and $i \rightarrow j$ are given and i, k, l cannot be processed simultaneously. Then the additional conjunction $l \rightarrow k \in C$ can be fixed (see Figure 3.29).

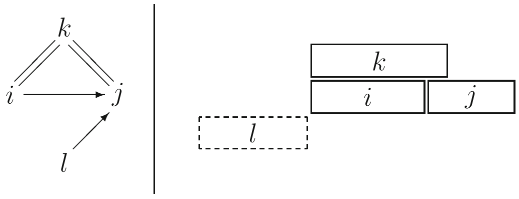


Figure 3.29: Extension of a symmetric triple – Condition (3)

(4) Suppose that the conditions $p_k - 1 \leq p_j$ and $p_k - 1 \leq p_l$ hold. Furthermore, assume that the precedence relations $i \rightarrow j$ and $i \rightarrow l$ are given and j, k, l cannot be processed simultaneously. Then symmetrically to (3) the additional conjunction $k \rightarrow l \in C$ can be fixed (see Figure 3.30).

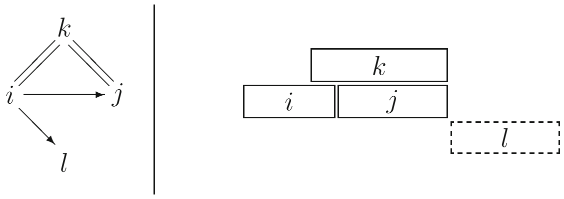


Figure 3.30: Extension of a symmetric triple – Condition (4)

(5) Suppose that the conditions $p_k - 1 \leq p_i$ and $p_k - 1 \leq p_j$ hold. Furthermore, assume that the precedence relations $l \rightarrow i$ and $l \rightarrow j$ ($i \rightarrow l$ and $j \rightarrow l$)

are given. Then we may fix the conjunction $l \to k \in C$ ($k \to l \in C$) (see Figure 3.31).

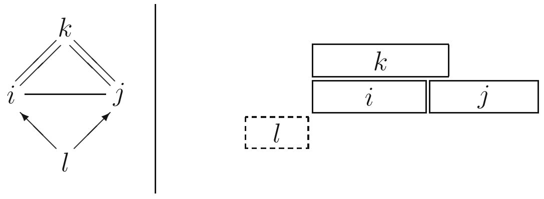


Figure 3.31: Extension of a symmetric triple – Condition (5)

(6) Let the parallelity relations $l \parallel i$ and $l \parallel j$ be given. Then the additional parallelity relation $l \parallel k$ can be fixed (see Figure 3.32).

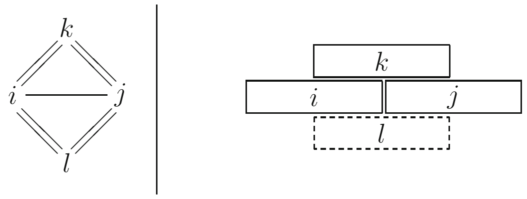


Figure 3.32: Extension of a symmetric triple – Condition (6)

3.6.4 Disjunctive sets

A subset $I \subseteq \{1, \dots, n\}$ of (non-dummy) activities with $|I| \geq 2$ is called a **disjunctive set** if $i - j \in D$, $i \to j \in C$ or $j \to i \in C$ for all $i, j \in I$ with $i \neq j$ holds. Disjunctive sets are cliques with at least two elements in the undirected graph defined by all disjunctions and conjunctions.

Examples of disjunctive sets are

- the set of all activities which need a disjunctive resource, i.e. a renewable resource k with capacity $R_k = 1$,
- the set of all jobs in a single machine scheduling problem,
- the set of all operations belonging to the same job in a general-shop problem,
- the set of operations to be processed by the same machine in a general-shop problem,
- the set of multiprocessor-tasks which need the same machine (i.e. all tasks j with $M_i \in \mu_j$ for a fixed machine M_i).

We define the total processing time of such a disjunctive set I by $P(I) := \sum\limits_{i \in I} p_i$. For each activity i let $[r_i, d_i]$ be a time window for i with $r_i + p_i \leq d_i$ (otherwise no feasible solution exists).

A first infeasibility test can be derived from the following result: If there is a subset $J \subseteq I$ with

$$\max_{\mu \in J} d_\mu - \min_{\nu \in J} r_\nu < P(J),$$

then obviously no feasible schedule exists since all jobs from the subset J have to be processed in the interval $[\min\limits_{\nu \in J} r_\nu, \max\limits_{\mu \in J} d_\mu]$, which does not have the capacity $P(J)$.

Additional conjunctions $i \rightarrow j$ and smaller time windows $[r_i', d_i'] \subseteq [r_i, d_i]$ for activities of a disjunctive set I may be derived from the following general result:

Theorem 3.7 Let I be a disjunctive set and $J', J'' \subset J \subseteq I$ with $J' \cup J'' \neq \emptyset$. If

$$\max_{\substack{\nu \in J \setminus J' \\ \mu \in J \setminus J'' \\ \nu \neq \mu}} (d_\mu - r_\nu) < P(J), \tag{3.121}$$

then in J an activity from J' must start first or an activity from J'' must end last in any feasible schedule.

Proof: If no activity in J' starts first and no activity in J'' ends last in a feasible schedule, then all activities in J must be processed in a time interval of length

$$\max_{\substack{\nu \in J \setminus J' \\ \mu \in J \setminus J'' \\ \nu \neq \mu}} (d_\mu - r_\nu),$$

which is not possible if (3.121) holds. The condition $\nu \neq \mu$ may be imposed since in a non-preemptive schedule for at least two activities an activity which starts first cannot complete also last. □

Note that Theorem 3.7 also holds if the condition $\nu \neq \mu$ is dropped and only the condition

$$\max_{\substack{\nu \in J \setminus J' \\ \mu \in J \setminus J''}} (d_\mu - r_\nu) = \max_{\mu \in J \setminus J''} d_\mu - \min_{\nu \in J \setminus J'} r_\nu < P(J) \tag{3.122}$$

is satisfied.

By choosing different subsets J' and J'', different tests may be derived. The following so-called "interval consistency tests" are usually considered in the literature:

- **Input test:** Let $J' = \{i\}$ with $i \in J$ and $J'' = \emptyset$. If we set $\Omega := J \setminus \{i\}$, then condition (3.122) can be rewritten as

$$\max_{\mu \in \Omega \cup \{i\}} d_\mu - \min_{\nu \in \Omega} r_\nu < P(\Omega \cup \{i\}).$$

If for some $\Omega \subseteq I$ and $i \in J \setminus \Omega$ this condition holds, then activity i must start first in J, i.e. we conclude that $i \to j$ for all $j \in \Omega$ (writing $i \to \Omega$). In this case i is called **input** of J.

- **Output test:** Let $J' = \emptyset$ and $J'' = \{i\}$ with $i \in J$. If we set $\Omega := J \setminus \{i\}$, then condition (3.122) can be rewritten as

$$\max_{\mu \in \Omega} d_\mu - \min_{\nu \in \Omega \cup \{i\}} r_\nu < P(\Omega \cup \{i\}).$$

  If for some $\Omega \subseteq I$ and $i \in J \setminus \Omega$ this condition holds, then activity i must end last in J, i.e. we conclude that $j \to i$ for all $j \in \Omega$ (writing $\Omega \to i$). In this case i is called **output** of J.

- **Input-or-Output test:** Let $J' = \{i\}$ and $J'' = \{j\}$ with $i, j \in J$. Then condition (3.122) can be rewritten as

$$\max_{\mu \in J \setminus \{j\}} d_\mu - \min_{\nu \in J \setminus \{i\}} r_\nu < P(J).$$

  If this condition holds, then activity i must start first in J or activity j must end last in J, i.e. i is input for J or j is output for J. If $i \neq j$, the conjunction $i \to j$ is implied.

- **Input negation test:** Let $J' = J \setminus \{i\}$ and $J'' = \{i\}$ with $i \in J$. If we set $\Omega := J \setminus \{i\}$, then condition (3.121) can be rewritten as

$$\max_{\mu \in \Omega} d_\mu - r_i < P(\Omega \cup \{i\}).$$

  If for some $\Omega \subseteq I$ and $i \in J \setminus \Omega$ this condition holds, then i cannot start first in J (**input negation**, writing $i \nrightarrow \Omega$).

- **Output negation test:** Let $J' = \{i\}$ and $J'' = J \setminus \{i\}$ with $i \in J$. If we set $\Omega := J \setminus \{i\}$, then condition (3.121) can be rewritten as

$$d_i - \min_{\nu \in \Omega} r_\nu < P(\Omega \cup \{i\}).$$

  If for some $\Omega \subseteq I$ and $i \in J \setminus \Omega$ this condition holds, then i cannot end last in J (**output negation**, writing $\Omega \nrightarrow i$).

  Note that with the input/output negation tests no additional conjunctions can be derived but time windows may be strengthened.

In the following we describe in more detail how the different consistency tests can be applied in a systematic and efficient way. For each test we describe a procedure which may be applied iteratively to the data until no more additional constraints are deduced (i.e. a so-called fixed-point is reached).

Input/Output tests

In the input/output tests we are interested in an activity i which must be processed first (or last) in a certain set J of activities, i.e. i is input (or output) of J. To derive an algorithm which performs such input and output tests in a systematic way, we consider for all $i \in I$ and subsets $\Omega \subseteq I$ with $i \notin \Omega$ the following implications:

$$\max_{\mu \in \Omega \cup \{i\}} d_\mu - \min_{\nu \in \Omega} r_\nu < P(\Omega) + p_i \Rightarrow i \to \Omega, \tag{3.123}$$

i.e. i has to be scheduled first in $\Omega \cup \{i\}$. Symmetrically,

$$\max_{\mu \in \Omega} d_\mu - \min_{\nu \in \Omega \cup \{i\}} r_\nu < P(\Omega) + p_i \Rightarrow \Omega \to i, \tag{3.124}$$

i.e. i has to be scheduled last in $\Omega \cup \{i\}$.

In the case (3.123) we may introduce the additional conjunctions $i \to j$ for all $j \in \Omega$, in the case (3.124) we may introduce the additional conjunctions $j \to i$ for all $j \in \Omega$.

Furthermore, with (3.124) the release date r_i and with (3.123) the deadline d_i of activity i may be improved. In the output situation (3.124) i cannot be started before all activities in the set Ω are completed. Obviously, for a modified release date r'_i we may require

$$r'_i \geq \max_{j \in \Omega} \{r_j + p_j\}$$

(each activity $j \in \Omega$ has to be completed before i can start), and

$$r'_i \geq \min_{\nu \in \Omega} r_\nu + P(\Omega)$$

(all activities $j \in \Omega$ together have to be completed before i can start).

A stronger lower bound for r_i is the value $\max\limits_{j \in \Omega} \{r_j + \sum\limits_{\{h \in \Omega | r_h \geq r_j\}} p_h\}$ (all activities in Ω which cannot start before a certain release date have to be completed), i.e. we may set

$$r'_i := \max\{r_i, \max_{j \in \Omega} \{r_j + \sum_{\{h \in \Omega | r_h \geq r_j\}} p_h\}\}.$$

Symmetrically, in the input situation (3.123) i cannot be completed later than the first activity in the set Ω is started. Thus, for a modified deadline we may require

$$d'_i := \min\{d_i, \min_{j \in \Omega} \{d_j - \sum_{\{h \in \Omega | d_h \leq d_j\}} p_h\}\}.$$

In the following we consider the output test (3.124), the input test (3.123) can be treated in a symmetric way. Assume that the considered disjunctive set I contains n activities and that these activities are ordered according to non-decreasing release dates $r_1 \leq r_2 \leq \ldots \leq r_n$.

For each activity i we try to find subsets $\Omega \subseteq I$ with $i \notin \Omega$ such that the inequality in (3.124) is satisfied. In the following we will show that not all possible $O(2^n)$ subsets $\Omega \subseteq I$ have to be considered as candidates in (3.124).

Assume that the inequality in (3.124) holds for some set Ω and an activity $i \notin \Omega$. Let $k \in \Omega$ be an activity with $d_k = \max\limits_{\mu \in \Omega} d_\mu$ and let $l \in \Omega \cup \{i\}$ be the activity with smallest index such that $r_l = \min\limits_{\nu \in \Omega \cup \{i\}} r_\nu$ (which implies $l \leq i$). If the inequality in (3.124) holds for Ω, then this condition is also satisfied for the (possibly larger) set $\Omega_{l,k} \setminus \{i\}$ with

$$\Omega_{l,k} := \{\mu \mid \mu \geq l \text{ and } d_\mu \leq d_k\}$$

because $\Omega \subseteq \Omega_{l,k}$ due to the definition of k and l. Thus, it is sufficient to consider only sets $\Omega_{l,k}$ as candidates (there are only $O(n^2)$ such sets).

In the following we assume that activities i and k are fixed. We search for an index $l \leq i$ such that the set $\Omega_{l,k} \setminus \{i\}$ satisfies the inequality in (3.124), i.e.

$$r_l + P(\Omega_{l,k} \setminus \{i\}) + p_i > d_k. \tag{3.125}$$

Let $P_{l,k} := P(\Omega_{l,k})$, $\Delta_{\lambda,k} := \max\limits_{\nu \in \Omega_{\lambda,k}} \{r_\nu + P_{\nu,k}\}$, and $H_{i,k} := \max\limits_{\substack{\nu < i \\ d_\nu \leq d_k}} \{r_\nu + P_{\nu,k}\}$. We distinguish two cases.

- Case 1: $d_i \leq d_k$
  Then we have $i \in \Omega_{l,k}$ for all $l \leq i$, i.e. (3.125) can be written as $r_l + P_{l,k} > d_k$. But, if this condition holds, then no feasible schedule exists since the activities in the set $\Omega_{l,k}$ cannot be scheduled in their time window $[r_l, d_k]$. On the other hand, if $r_l + P_{l,k} > d_k$ for an index $l > i$ holds, also infeasibility can be stated. Thus, we may check the general infeasibility test

$$\Delta_{1,k} = \max_{l \in \Omega_{1,k}} \{r_l + P_{l,k}\} > d_k. \tag{3.126}$$

- Case 2: $d_i > d_k$
  Then we have $i \notin \Omega_{l,k}$ for all l. Due to $l \leq i$ two subcases for l are distinguished:

  Subcase 2.1: $l = i$
  If for $l = i$ the condition

$$r_i + P_{i,k} + p_i > d_k \tag{3.127}$$

  is satisfied, then $\Omega_{i,k} \to i$ is implied. Thus, i cannot start before all activities in the set $\Omega_{i,k}$ are completed, i.e. we may require that a modified release date r_i' satisfies

$$r_i' \geq \max_{\nu \in \Omega_{i,k}} \{r_\nu + P_{\nu,k}\} = \Delta_{i,k},$$

since due to $\Omega_{\nu,k} \subseteq \Omega_{i,k}$ for all $\nu \geq i$ the right hand side defines a lower bound for the completion time of $\Omega_{i,k}$. Thus, we may update the release date of i to

$$r_i' := \max\{r_i, \Delta_{i,k}\}. \tag{3.128}$$

Subcase 2.2: $l < i$
All cases $l < i$ are covered by checking the condition

$$H_{i,k} + p_i = \max_{\substack{l<i \\ d_l \leq d_k}} \{r_l + P_{l,k}\} + p_i > d_k. \tag{3.129}$$

If an index $l < i$ of a set $\Omega_{l,k}$ satisfying (3.129) is found, we have $\Omega_{l,k} \to i$ and may require

$$r_i' \geq \max_{\nu \in \Omega_{l,k}} \{r_\nu + P_{\nu,k}\} = \Delta_{l,k}.$$

We will show that we do not have to determine the index l explicitly, but that we can set

$$r_i' := \max\{r_i, \Delta_{1,k}\}. \tag{3.130}$$

Let $l < i$ be an index for which $H_{i,k} = \max_{\substack{\lambda<i \\ d_\lambda \leq d_k}} \{r_\lambda + P_\lambda, k\} = r_l + P_{l,k}$ holds. Hence, $r_l + P_{l,k} \geq r_\lambda + P_{\lambda,k}$ holds for all $1 \leq \lambda < l$ with $d_\lambda \leq d_k$. Thus, we have $\Delta_{l,k} = \max_{\substack{\nu \geq l \\ d_\nu \leq d_k}} \{r_\nu + P_{\nu,k}\} = \max_{\substack{\lambda \geq 1 \\ d_\lambda \leq d_k}} \{r_\lambda + P_{\lambda,k}\} = \Delta_{1,k}$.

The preceding discussions provide the algorithm shown in Figure 3.33.

In this algorithm condition (3.127) is checked in Step 14, condition (3.129) is checked in Step 16. Since the infeasibility test (3.126) is also valid in the case $d_i > d_k$, this more general check is performed in Step 10.

For fixed k the values $P_{n,k}, \ldots, P_{1,k}$; $\Delta_{n,k}, \ldots, \Delta_{1,k}$ and $H_{1,k}, \ldots, H_{n,k}$ can be computed in $O(n)$ time, since starting with the initial values $P_{n+1,k} = \Delta_{n+1,k} = H_{1k} := 0$ the values

$$P_{\lambda,k} = \sum_{\substack{\mu \geq \lambda \\ d_\mu \leq d_k}} p_\mu = \begin{cases} P_{\lambda+1,k} + p_\lambda, & \text{if } d_\lambda \leq d_k \\ P_{\lambda+1,k}, & \text{otherwise} \end{cases} \quad \text{for } \lambda = n, \ldots, 1,$$

$$\Delta_{\lambda,k} := \max_{\substack{\nu \geq \lambda \\ d_\nu \leq d_k}} \{r_\nu + P_{\nu,k}\} = \begin{cases} \max\{\Delta_{\lambda+1,k}, \, r_\lambda + P_{\lambda,k}\}, & \text{if } d_\lambda \leq d_k \\ \Delta_{\lambda+1,k}, & \text{otherwise} \end{cases}$$

for $\lambda = n, \ldots, 1$, and

$$H_{\lambda,k} := \max_{\substack{\nu < \lambda \\ d_\nu \leq d_k}} \{r_\nu + P_{\nu,k}\} = \begin{cases} \max\{H_{\lambda-1,k}, \, r_{\lambda-1} + P_{\lambda-1,k}\}, & \text{if } d_{\lambda-1} \leq d_k \\ H_{\lambda-1,k}, & \text{otherwise} \end{cases}$$

for $\lambda = 2, \ldots, n$

can be computed in constant time from the previous values $P_{\lambda+1,k}$, $\Delta_{\lambda+1,k}$ and $H_{\lambda-1,k}$, respectively. Furthermore, sorting the activities according to non-decreasing release dates can be done in $O(n \log n)$ time. Therefore, the presented implementation of the output test runs in $O(n^2)$ time.

```
Algorithm Output Test
 1.  Sort the activities such that $r_1 \leq \ldots \leq r_n$;
 2.  FOR $i := 1$ TO $n$ DO
 3.       $r'_i := r_i$;
 4.  FOR $k := 1$ TO $n$ DO
 5.       Compute $P_{n,k}, \ldots, P_{1,k}$;
 6.       Compute $\Delta_{n,k}, \ldots, \Delta_{1,k}$;
 7.       Compute $H_{1,k}, \ldots, H_{n,k}$;
 8.  ENDFOR
 9.  FOR $k := 1$ TO $n$ DO
10.       IF $\Delta_{1,k} > d_k$ THEN
11.           EXIT (no feasible schedule exists)
12.       FOR $i := 1$ TO $n$ DO
13.            IF $d_i > d_k$ THEN
14.                IF $r_i + P_{i,k} + p_i > d_k$ THEN
15.                   $r'_i := \max\{r'_i, \Delta_{i,k}\}$;
16.                IF $H_{i,k} + p_i > d_k$ THEN
17.                   $r'_i := \max\{r'_i, \Delta_{1,k}\}$;
18.            ENDIF
19.  ENDFOR
20.  FOR $i := 1$ TO $n$ DO
21.       $r_i := r'_i$;
```

Figure 3.33: Output Test

Since in Steps 14 and 16 for an activity i the set $\Omega_{l,k}$ is not explicitly determined, in this version of the algorithm the new conjunctions $j \to i$ for all $j \in \Omega_{l,k}$ cannot be directly introduced. In the following we will show that all these arcs can be derived from the updated release dates r'_i in a second step in $O(n^2)$ time.

Similarly to (3.120) we only have to check the condition

$$r'_i + p_i + p_j > d_j \tag{3.131}$$

for all $i, j \in \{1, \ldots, n\}$ with $i \neq j$. If this condition is satisfied, the conjunction $j \to i \in C$ is implied. In order to prove that all output conjunctions are derived in this way, assume that a set $\Omega_{l,k}$ and an activity $i \notin \Omega_{l,k}$ satisfy the output condition $r_l + P_{l,k} + p_i > d_k$ and the release date of i is updated to

$$r'_i = \max\{r_i, \Delta_{l,k}\} = \max\{r_i, \max_{\substack{\nu \geq l \\ d_\nu \leq d_k}}\{r_\nu + P_{\nu,k}\}\} \geq r_l + P_{l,k}.$$

Since $d_k = \max\limits_{\mu \in \Omega_{l,k}} d_\mu$, for any activity $j \in \Omega_{l,k}$ we get

$$r'_i + p_i + p_j \geq r_l + P_{l,k} + p_i + p_j > d_k + p_j > d_j,$$

i.e. (3.131) is satisfied and the conjunction $j \to i$ is derived.

Example 3.13 : Consider the following example with $n = 5$ activities:

i	1	2	3	4	5
r_i	0	1	2	4	4
p_i	2	1	3	2	1
d_i	4	5	10	7	8

Activity $i = 3$ has to be last in the set $\{1, 2, 3\}$ since for $\Omega := \Omega_{1,2} = \{1, 2\}$ we have

$$\max_{\mu\in\Omega} d_\mu - \min_{\nu\in\Omega\cup\{i\}} r_\nu = d_2 - r_1 = 5 - 0 = 5 < P(\Omega) + p_i = 3 + 3 = 6$$

(cf. also Figure 3.34).

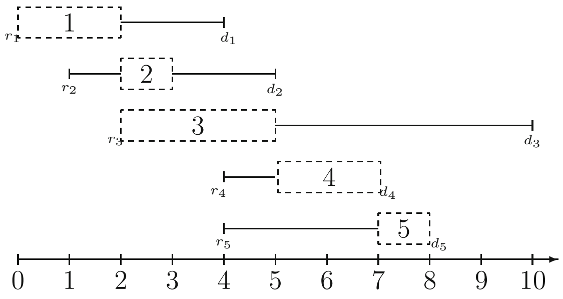


Figure 3.34: Time windows for the activities from Example 3.13

Algorithm `Output Test` detects this situation since for $i = 3$, $k = 2$ we have $d_i = 10 > d_k = 5$ and the check of condition (3.129) in Step 16 gives

$$H_{i,k}+p_i = H_{3,2}+p_3 = \max_{\substack{\nu<3 \\ d_\nu\leq d_2}} \{r_\nu+P_{\nu,2}\}+p_3 = r_1+P_{1,2}+p_3 = 0+3+3 = 6 > d_k = 5.$$

Thus, the release date of activity 3 may be updated to

$$r'_3 := \max\{r_3, \Delta_{1,k}\} = \max\{r_3, \Delta_{1,2}\} = r_1 + P_{1,2} = 3.$$

Afterwards, we may also deduce that activity $i = 3$ has to be last in the set $\{3, 4, 5\}$ since for $\Omega := \Omega_{4,5} = \{4, 5\}$ we have

$$\max_{\mu\in\Omega} d_\mu - \min_{\nu\in\Omega\cup\{i\}} r_\nu = d_5 - r'_3 = 8 - 3 = 5 < P(\Omega) + p_i = 3 + 3 = 6.$$

Algorithm `Output Test` detects this situation in a second run since for $i = 3$, $k = 5$ we have $d_i = 10 > d_k = 8$ and the check of condition (3.127) in Step 14 with the new head $r'_3 = 3$ gives

$$r_i + P_{i,k} + p_i = r'_3 + P_{3,5} + p_3 = 3 + 3 + 3 = 9 > d_k = 8.$$

Thus, the release date of activity 3 may be updated to

$$r''_3 := \max\{r'_3, \Delta_{i,k}\} = \max\{r'_3, \Delta_{3,5}\} = r_4 + P_{4,5} = 7.$$

□

Input/Output negation tests

In the preceding discussions we have focused on determining whether an activity i **must** be processed first (or last) in a certain set J of activities (i.e. i is input or output of J). In the following we consider the converse situation, i.e. we are interested in determining whether an activity i **cannot** be processed first (or last) in a certain set of activities. To derive an algorithm which performs such input negation and output negation tests in a systematic way, we consider for all $i \in I$ and subsets $\Omega \subseteq I$ with $i \notin \Omega$ the following implications:

$$\max_{\mu \in \Omega} d_\mu - r_i < P(\Omega) + p_i \Rightarrow i \nrightarrow \Omega, \tag{3.132}$$

i.e. i cannot be scheduled first in $\Omega \cup \{i\}$. Symmetrically,

$$d_i - \min_{\nu \in \Omega} r_\nu < P(\Omega) + p_i \Rightarrow \Omega \nrightarrow i, \tag{3.133}$$

i.e. i cannot be scheduled last in $\Omega \cup \{i\}$.

In the case (3.132) activity i cannot be started before the job scheduled first in Ω is completed, i.e. for its head r_i we have

$$r_i \geq \min_{\nu \in \Omega} \{r_\nu + p_\nu\}. \tag{3.134}$$

In the case (3.133) activity i cannot be completed later than the job scheduled last in Ω is started, i.e. for its deadline d_i we have

$$d_i \leq \max_{\mu \in \Omega} \{d_\mu - p_\mu\}. \tag{3.135}$$

In the following we consider the input negation test (3.132), the output negation test (3.133), can be treated in a symmetric way. Assume that the considered disjunctive set I contains n activities and that these activities are ordered according to non-decreasing deadlines $d_1 \leq d_2 \leq \ldots \leq d_n$.

For each activity i we try to find subsets $\Omega \subseteq I$ with $i \notin \Omega$ such that the inequality in (3.132) is satisfied. As in the output test we do not have to consider all possible subsets $\Omega \subseteq I$. Assume that the inequality in (3.132) holds for some set Ω with $i \notin \Omega$ and that r_i according to (3.134) may be updated to $r_j + p_j$ for an activity $j \in \Omega$ with $r_j + p_j = \min_{\nu \in \Omega}\{r_\nu + p_\nu\}$. Let $l \in \Omega$ be the activity with largest index such that $d_l = \max_{\nu \in \Omega} d_\nu$. Then the inequality in (3.132) is also satisfied for the set $\Omega_{j,l} \setminus \{i\}$ with

$$\Omega_{j,l} := \{\mu \mid \mu \leq l \text{ and } r_\mu + p_\mu \geq r_j + p_j\}$$

because $\Omega \subseteq \Omega_{j,l}$ due to the definition of j and l.

Let $P_{j,l} := P(\Omega_{j,l})$ if $j \leq l$ and $-\infty$ otherwise. Furthermore for $\lambda = 1, \ldots, n$ let $\Delta_{j,\lambda} := \min_{\mu \in \Omega_{j,\lambda}} \{d_\mu - P_{j,\mu}\}$.

In the following we assume that activities i and j are fixed. We search for an index l such that the set $\Omega_{j,l} \setminus \{i\}$ satisfies (3.132) or equivalently

$$r_i + p_i > d_l - P(\Omega_{j,l} \setminus \{i\}), \tag{3.136}$$

and the release date of i may be increased to $r_j + p_j = \min\limits_{\nu \in \Omega_{j,l}} \{r_\nu + p_\nu\}$. We distinguish two cases.

- Case 1: $r_i + p_i < r_j + p_j$

  Then $i \notin \Omega_{j,l}$ for all $l = 1, \ldots, n$ and the best we can do is to check

  $$r_i + p_i > \min_{\substack{1 \le l \le n \\ r_l + p_l \ge r_j + p_j}} \{d_l - P_{j,l}\} = \Delta_{j,n}. \tag{3.137}$$

  If (3.137) is satisfied, we can update the release date of i to

  $$r'_i := \max\{r_i, r_j + p_j\}.$$

- Case 2: $r_i + p_i \ge r_j + p_j$

  For $l < i$ we have $i \notin \Omega_{j,l}$. We cover all cases with $l < i$ by checking the condition

  $$r_i + p_i > \min_{\substack{l \le i-1 \\ r_l + p_l \ge r_j + p_j}} \{d_l - P_{j,l}\} = \Delta_{j,i-1}. \tag{3.138}$$

  For $l \ge i$ we have $i \in \Omega_{j,l}$, i.e. we have to check $r_i > d_l - P_{j,l}$. Instead of testing $r_i > \min\limits_{\substack{i \le l \\ r_l + p_l \ge r_j + p_j}} \{d_l - P_{j,l}\}$, we may check the stronger condition

  $$r_i > \min_{\substack{1 \le l \le n \\ r_l + p_l \ge r_j + p_j}} \{d_l - P_{j,l}\} = \Delta_{j,n},$$

  which is less time-consuming (since we do not have to find l explicitly). This is correct since if for some $l < i$ the condition $r_i > d_l - P_{j,l}$ holds, then also $r_i + p_i > d_l - P_{j,l}$ is valid, which implies that also condition (3.138) is satisfied.

  Thus, we may cover all cases $l < i$ and $l \ge i$ by checking whether

  $$r_i + p_i > \Delta_{j,i-1} \quad \text{or} \quad r_i > \Delta_{j,n} \tag{3.139}$$

  holds. Again, if this condition holds, we may update the release date of i to $r'_i := \max\{r_i, r_j + p_j\}$.

The preceding discussions provide the algorithm shown in Figure 3.35. In this algorithm condition (3.137) is checked in Step 8, condition (3.139) is checked in Step 12.

```
Algorithm Input Negation Test
 1.  Sort the activities such that $d_1 \leq \ldots \leq d_n$;
 2.  FOR $i := 1$ TO $n$ DO
 3.      $r'_i := r_i$;
 4.  FOR $j := 1$ TO $n$ DO
 5.      Compute $\Delta_{j,1}, \ldots, \Delta_{j,n}$;
 6.      FOR $i := 1$ TO $n$ WITH $i \neq j$ DO
 7.          IF $r_i + p_i < r_j + p_j$ THEN
 8.              IF $r_i + p_i > \Delta_{j,n}$ THEN
 9.                  $r'_i := \max\{r'_i, r_j + p_j\}$;
10.              ENDIF
11.          ELSE
12.              IF $r_i + p_i > \Delta_{j,i-1}$ OR $r_i > \Delta_{j,n}$ THEN
13.                  $r'_i := \max\{r'_i, r_j + p_j\}$;
14.          ENDIF
15.  ENDFOR
16.  FOR $i := 1$ TO $n$ DO
17.      $r_i := r'_i$;
```

Figure 3.35: Input Negation Test

For fixed j the values $\Delta_{j,1}, \ldots, \Delta_{j,n}$ can be computed in $O(n)$ time. Starting with $P'_{j0} := 0$, the values

$$P'_{j,\lambda} = \sum_{\substack{\mu \leq \lambda \\ r_\mu + p_\mu \geq r_j + p_j}} p_\mu = \begin{cases} P'_{j,\lambda-1} + p_\lambda, & \text{if } r_\lambda + p_\lambda \geq r_j + p_j \\ P'_{j,\lambda-1}, & \text{otherwise} \end{cases}$$

for $\lambda = 1, \ldots, n$ can be computed in constant time from the previous values $P'_{j,\lambda-1}$. Furthermore, with

$$P_{j,\lambda} = \begin{cases} -\infty, & \lambda < j \\ P'_{j,\lambda}, & \lambda \geq j \end{cases}$$

and the initial values $\Delta_{j,0} := \infty$ we calculate the values

$$\Delta_{j,\lambda} := \min_{\substack{\mu \leq \lambda \\ r_\mu + p_\mu \geq r_j + p_j}} \{d_\mu - P_{j,\mu}\} = \begin{cases} \min\{\Delta_{j,\lambda-1}, d_\lambda - P_{j,\lambda}\}, & \text{if } r_\lambda + p_\lambda \geq r_j + p_j \\ \Delta_{j,\lambda-1}, & \text{otherwise} \end{cases}$$

for $\lambda = 1, \ldots, n$ in constant time from the previous values $\Delta_{j,\lambda-1}$. Furthermore, sorting the activities according to non-decreasing deadlines can be done in $O(n \log n)$ time. Therefore, the presented implementation of the input negation test runs in $O(n^2)$ time.

Example 3.14: Consider the following example with $n = 3$ activities:

i	1	2	3
r_i	0	2	1
p_i	2	1	2
d_i	5	5	10

Activity $i = 3$ cannot be first in the set $\{1, 2, 3\}$ since for $\Omega := \Omega_{1,2} = \{1, 2\}$ we have

$$\max_{\mu\in\Omega} d_\mu - r_i = 5 - 1 = 4 < P(\Omega) + p_i = 3 + 2 = 5$$

(cf. also Figure 3.36).

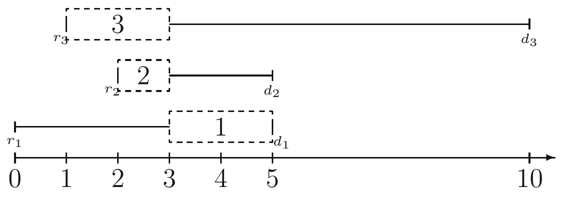


Figure 3.36: Time windows for the activities from Example 3.14

With

$$\begin{array}{ll}
\Omega_{1,1} = \{1\}, & \Delta_{1,1} = d_1 - p_1 = 3 \\
\Omega_{1,2} = \{1, 2\}, & \Delta_{1,2} = \min\{\Delta_{1,1}, d_2 - P_{1,2}\} = \min\{3, 2\} = 2 \\
\Omega_{1,3} = \{1, 2, 3\}, & \Delta_{1,3} = \min\{\Delta_{1,2}, d_3 - P_{1,3}\} = \min\{2, 5\} = 2 \\
\Omega_{2,1} = \Omega_{3,1} = \emptyset, & \Delta_{2,1} = \Delta_{3,1} = \infty \\
\Omega_{2,2} = \Omega_{3,2} = \{2\}, & \Delta_{2,2} = \Delta_{3,2} = d_2 - p_2 = 4 \\
\Omega_{2,3} = \Omega_{3,3} = \{2, 3\}, & \Delta_{2,3} = \Delta_{3,3} = \min\{\Delta_{2,2}, d_3 - P_{2,3}\} = \min\{4, 7\} = 4
\end{array}$$

Algorithm `Input Negation Test` detects this situation since for $j = 1$ and $i = 3$ we have $r_i + p_i = 3 \geq r_j + p_j = 2$ and the check of condition (3.139) in Step 12 gives

$$r_i + p_i = 3 > \Delta_{j,i-1} = \Delta_{1,2} = 2.$$

Thus, the release date of activity 3 may be updated to $r'_3 = r_1 + p_1 = 2$. □

Input-or-Output test

The input-or-output test is weaker than both the input and the output test (i.e. it is applicable more often, but its conclusion is weaker). At first we consider the test for activities $i, j \in I$ with $i \neq j$. If then a subset $\Omega \subseteq I$ with $i, j \notin \Omega$ exists and the condition

$$\max_{\mu \in \Omega \cup \{i\}} d_\mu - \min_{\nu \in \Omega \cup \{j\}} r_\nu < P(\Omega \cup \{i, j\}) \tag{3.140}$$

holds, then activity i has to be scheduled first in $\Omega \cup \{i, j\}$ or activity j has to be scheduled last in $\Omega \cup \{i, j\}$, i.e. i is input for $\Omega \cup \{i, j\}$ or j is output for $\Omega \cup \{i, j\}$. In this situation the conjunction $i \to j$ may be introduced. Furthermore, we may improve the head of j according to $r'_j := \max\{r_j, r_i + p_i\}$ and the deadline of i according to $d'_i := \min\{d_i, d_j - p_j\}$.

Assume that the activities are ordered according to non-increasing release dates $r_1 \geq r_2 \geq \ldots \geq r_n$. For each pair of activities i, j with $i \neq j$ we try to find subsets $\Omega \subseteq I$ with $i, j \notin \Omega$ such that the inequality in (3.140) is satisfied.

Assume that (3.140) holds for some set Ω and activities $i, j \notin \Omega$. Let $l \in \Omega \cup \{i\}$ be an activity with $d_l = \max\limits_{\mu \in \Omega \cup \{i\}} d_\mu$ and let $k \in \Omega \cup \{j\}$ be the activity with largest index such that $r_k = \min\limits_{\nu \in \Omega \cup \{j\}} r_\nu$.

We may assume $d_j > d_l$ since otherwise the input test condition

$$\max_{\mu \in \Omega \cup \{i,j\}} d_\mu - \min_{\nu \in \Omega \cup \{j\}} r_\nu < P(\Omega \cup \{i, j\})$$

is satisfied for the set $\Omega' := \Omega \cup \{j\}$ and the activity i. Such situations may be discovered by the input test. Symmetrically, we may assume $r_i < r_k$. Furthermore, by definition of l and k we have $d_i \leq d_l$ and $r_j \geq r_k$.

If (3.140) holds for Ω, then this condition is also satisfied for the set $\Omega_{k,l}$ with

$$\Omega_{k,l} := \{\mu \mid \mu \leq k \text{ and } d_\mu \leq d_l\}$$

because $\Omega \subseteq \Omega_{k,l}$ due to the definition of k and l. Thus, it is sufficient to consider only sets $\Omega_{k,l}$ as candidates. Note that due to $r_i < r_k$ and $d_j > d_l$ we have $i, j \notin \Omega_{k,l}$.

In the following we assume that activities i and j with $i \neq j$ are fixed. We search for an index $k < i$ with $r_j \geq r_k > r_i$ and an index l with $d_i \leq d_l < d_j$ such that the set $\Omega_{k,l}$ satisfies (3.140). Let $P_{k,l} := P(\Omega_{k,l})$. Then condition (3.140) for the set $\Omega_{k,l}$ is equivalent to

$$\delta_{k,l} := d_l - r_k - P_{k,l} < p_i + p_j. \tag{3.141}$$

We cover all cases for k and l in (3.141) by checking

$$\Delta_{i,j} := \min\{\delta_{k,l} \mid r_i < r_k \leq r_j, d_i \leq d_l < d_j\} < p_i + p_j. \tag{3.142}$$

If this condition holds, we may introduce the conjunction $i \rightarrow j$.

It remains to show how the values $\delta_{k,l}$ and $\Delta_{i,j}$ can be calculated in an efficient way. For fixed l the values $P_{1,l}, \ldots, P_{n,l}$ can be computed in $O(n)$ time, since starting with the initial values $P_{0,l} := 0$ each of the values

$$P_{k,l} = \sum_{\{\mu \mid \mu \leq k, d_\mu \leq d_l\}} p_\mu = \begin{cases} P_{k-1,l} + p_k, & \text{if } d_k \leq d_l \\ P_{k-1,l}, & \text{otherwise} \end{cases} \quad \text{for } k = 1, \ldots, n$$

can be computed in constant time from the previous value $P_{k-1,l}$. Thus, all values $P_{k,l}$ and all values $\delta_{k,l}$ for $k, l = 1, \ldots, n$ can be computed in $O(n^2)$.

The calculation of the values $\Delta_{i,j}$ is a little bit more complicated. For fixed i we consider the grid

$$G^i := \{(k,l) \mid r_i < r_k, d_i \leq d_l < d_{max}\}$$

where d_{max} denotes the largest d_μ-value. For all $j \neq i$ with $r_j > r_i$ and $d_j > d_i$ we define the rectangles

$$R_{ij} := \{(k,l) \mid r_i < r_k \leq r_j, d_i \leq d_l < d_j\} \subseteq G^i.$$

For all other j we have $R_{ij} = \emptyset$. For fixed i and j let $\alpha(i)$ be the largest index $\nu < i$ with $r_\nu > r_i$, and let $\beta(j)$ be an index μ of a largest d_μ-value with $d_\mu < d_j$. Then

$$G^i = \{(k,l) \mid r_{\alpha(i)} \leq r_k, d_i \leq d_l < d_{max}\}$$

and

$$R_{ij} = \{(k,l) \mid r_{\alpha(i)} \leq r_k \leq r_j, d_i \leq d_l \leq d_{\beta(j)}\}$$

(cf. also Figure 3.37).

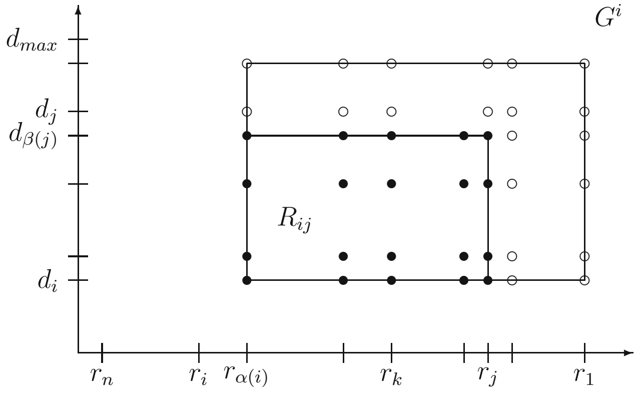


Figure 3.37: Rectangles R_{ij} in the grid G^i

In order to calculate the values $\Delta_{i,j} = \min_{(k,l) \in R_{ij}} \{\delta_{k,l}\}$ in an efficient way, we first calculate for each fixed i and every $k \in \{1, \ldots, \alpha(i)\}$ the values

$$\theta_{i,j}^k := \min \{\delta_{k,l} \mid d_i \leq d_l \leq d_{\beta(j)}\}$$

for all j with $d_j > d_i$. For fixed i and k all values $\theta_{i,j}^k$ can be determined in $O(n)$ time when the deadlines d_μ are sorted before. Thus, all values $\theta_{i,j}^k$ can be computed in $O(n^3)$ time. Since

$$\begin{aligned}\Delta_{i,j} &= \min_{(k,l)\in R_{ij}} \{\delta_{k,l}\} = \min\{\delta_{k,l} \mid r_{\alpha(i)} \le r_k \le r_j, d_i \le d_l \le d_{\beta(j)}\} \\ &= \min\,\{\theta_{i,j}^k \mid r_{\alpha(i)} \le r_k \le r_j\},\end{aligned}$$

for each pair i, j with $d_j > d_i$ the value $\Delta_{i,j}$ can be calculated in $O(n)$ time when the $\theta_{i,j}^k$-values are known. Thus, all $\Delta_{i,j}$-values can be computed in $O(n^3)$.

```
Algorithm Input-or-Output Test
 1.  Sort the activities such that $r_1 \ge \ldots \ge r_n$;
 2.  FOR $l := 1$ TO $n$ DO
 3.      FOR $k := 1$ TO $n$ DO
 4.          Compute $P_{k,l}$ and $\delta_{k,l}$;
 5.  FOR $i := 1$ TO $n$ DO
 6.      FOR $k := 1$ TO $\alpha(i)$ DO
 7.          Compute all $\theta_{i,j}^k$-values for all $j$ with $d_j > d_i$;
 8.  FOR $i := 1$ TO $n$ DO
 9.      FOR ALL $j$ with $d_j > d_i$ DO
10.          Calculate $\Delta_{i,j} := \min\{\theta_{i,j}^k \mid r_{\alpha(i)} \le r_k \le r_j\}$;
11.  FOR $i := 1$ TO $n$ DO
12.      FOR $j := 1$ TO $n$ DO
13.          IF $i \ne j$ AND $\Delta_{i,j} < p_i + p_j$ THEN
14.             Set $i \to j$;
```

Figure 3.38: Input-or-Output Test

The preceding discussions are summarized in the algorithm in Figure 3.38. This implementation of the input-or-output test runs in $O(n^3)$ time.

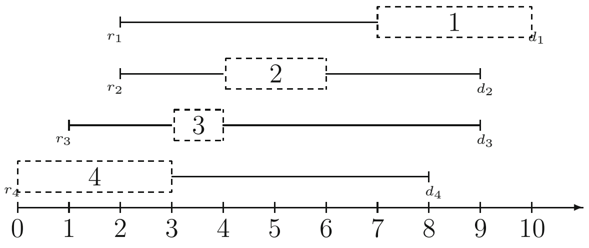


Figure 3.39: Time windows for the activities from Example 3.15

Example 3.15 : Consider the following example with $n = 4$ activities (cf. also

Figure 3.39):

i	1	2	3	4
r_i	2	2	1	0
p_i	3	2	1	3
d_i	10	9	9	8

Activity $i = 4$ has to be scheduled before activity $j = 1$ since for the set $\Omega = \{2, 3\}$ we get

$$\max_{\mu \in \Omega \cup \{i\}} d_\mu - \min_{\nu \in \Omega \cup \{j\}} r_\nu = d_2 - r_3 = 9 - 1 = 8 < P(\Omega \cup \{i, j\}) = 9.$$

We have $R_{1j} = R_{2j} = \emptyset$ for $j = 1, 2, 3, 4$ and $R_{32} = R_{33} = R_{34} = R_{44} = \emptyset$. Furthermore, $R_{31} = \{(1, 2), (2, 2), (1, 3), (2, 3)\}$ $R_{43} = \{(3, 4)\}$ $R_{42} = \{(3, 4), (1, 4), (2, 4)\}$ $R_{41} = \{(3, 2), (3, 3)\} \cup R_{42} \cup R_{31}$ (cf. also Figure 3.40).

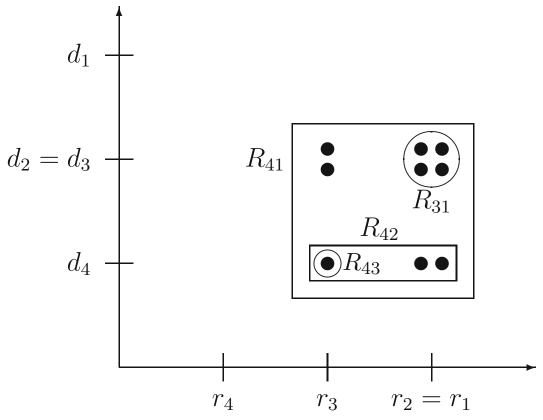


Figure 3.40: Rectangles R_{ij} in the grid G^i

With the values

$$\begin{aligned}
\delta_{1,2} &= d_2 - r_1 - P_{1,2} = 9 - 2 - 0 = 7\\
\delta_{1,3} &= d_3 - r_1 - P_{1,3} = 9 - 2 - 0 = 7\\
\delta_{1,4} &= d_4 - r_1 - P_{1,4} = 8 - 2 - 0 = 6\\
\delta_{2,2} &= d_2 - r_2 - P_{2,2} = 9 - 2 - p_2 = 5\\
\delta_{2,3} &= d_3 - r_2 - P_{2,3} = 9 - 2 - p_2 = 5\\
\delta_{2,4} &= d_4 - r_2 - P_{2,4} = 8 - 2 - 0 = 6\\
\delta_{3,2} &= d_2 - r_3 - P_{3,2} = 9 - 1 - p_2 - p_3 = 5\\
\delta_{3,3} &= d_3 - r_3 - P_{3,3} = 9 - 1 - p_2 - p_3 = 5\\
\delta_{3,4} &= d_4 - r_3 - P_{3,4} = 8 - 1 - 0 = 7
\end{aligned}$$

we get

$$\begin{aligned}
\Delta_{3,1} &= \delta_{2,2} = 5 > p_3 + p_1 = 4\\
\Delta_{4,1} &= \delta_{2,2} = 5 < p_4 + p_1 = 6\\
\Delta_{4,2} &= \delta_{1,4} = 6 > p_4 + p_2 = 5\\
\Delta_{4,3} &= \delta_{3,4} = 7 > p_4 + p_3 = 4.
\end{aligned}$$

Thus, condition (3.142) in Step 13 is satisfied for $i = 4$ and $j = 1$ implying the conjunction $4 \rightarrow 1$. □

The input-or-output test for $i = j$ can be formulated as follows. If for an activity $i \in I$ and a subset $\Omega \subseteq I$ with $i \notin \Omega$ the condition

$$\max_{\mu \in \Omega} d_\mu - \min_{\nu \in \Omega} r_\nu < P(\Omega \cup \{i\}) \tag{3.143}$$

holds, then activity i has to be scheduled first or last in $\Omega \cup \{i\}$, i.e. i is input or output for $\Omega \cup \{i\}$.

As above, it is sufficient to check (3.143) for sets $\Omega_{k,l}$ with $\Omega_{k,l} := \{\mu \mid \mu \leq k \text{ and } d_\mu \leq d_l\}$ and $r_i < r_k$ and $d_i > d_l$. Then (3.143) is equivalent to

$$\delta_{k,l} := d_l - r_k - P_{k,l} < p_i, \tag{3.144}$$

i.e. these checks can easily be integrated into the above algorithm.

The presented tests can be found with different names in the literature. Since in Theorem 3.7 for disjunctive sets the capacity of certain intervals is considered, all tests of this type are also called "disjunctive interval consistency tests". Since additional conjunctions are derived by the first three tests, they are also termed "edge finding" procedures. In branch-and-bound algorithms where in the branching step disjunctive edges $i - j$ are oriented into $i \rightarrow j$ or $j \rightarrow i$, the tests may be used to immediately select the orientation of some other disjunctive edges. Therefore, this process has also been called "immediate selection".

Test	$J \setminus J'$	$J \setminus J''$	conclusion	complexity
input	$J \setminus \{i\}$	J	$i \rightarrow J \setminus \{i\}$	$O(n \log n)$
output	J	$J \setminus \{i\}$	$J \setminus \{i\} \rightarrow i$	$O(n \log n)$
input-or-output	$J \setminus \{i\}$	$J \setminus \{j\}$	$i \rightarrow J \setminus \{i\} \vee J \setminus \{j\} \rightarrow j$	$O(n^3)$
input negation	$\{i\}$	$J \setminus \{i\}$	$i \nrightarrow J \setminus \{i\}$	$O(n \log n)$
output negation	$J \setminus \{i\}$	$\{i\}$	$J \setminus \{i\} \nrightarrow \{i\}$	$O(n \log n)$

Table 3.1: Summary of disjunctive interval consistency tests

All different interval consistency tests are summarized in Table 3.1. Furthermore, the complexities for the best known implementations are listed, where again n denotes the number of activities in the considered disjunctive set I. Note that for a test not only its complexity for a single run is important, but also the number of times it has to be applied. As mentioned before, usually a test is applied in several iterations until no more constraints can be deduced. Thus, an $O(n^2)$-algorithm which uses less iterations may be more effective than an $O(n \log n)$-algorithm which uses more iterations.

There are different ways to combine the described constraint propagation techniques in one constraint propagation procedure. In order to apply the interval consistency tests at first disjunctive sets have to be determined.

Unfortunately, the number of all possible disjunctive sets (cliques) may be quite large. Thus, usually only some (maximal) cliques are generated heuristically. A clique I is called maximal if adding an activity $i \notin I$ to I does not lead to another (larger) clique. Maximal cliques $I_1, \ldots, I_q$ which cover the whole set of activities (i.e. with $I_1 \cup I_2 \cup \ldots \cup I_q = \{1, \ldots, n\}$) may be determined as follows.

To build I_1, we start with some activity i_1 and add an activity i_2 which is incompatible to i_1 (i.e. with $i_2 - i_1 \in D$ or $i_1 \to i_2 \in C$ or $i_2 \to i_1 \in C$). Then we add some activity i_3 which is incompatible to both i_1 and i_2, etc. This process stops if no activity exists which is incompatible to all $i \in I_1$. To build I_2, we start with some activity not belonging to I_1 and apply the same procedure. Note that $i \in I_1$ may also be added to the clique I_2. If $I_1, \ldots, I_h$ are built and some activity $i \notin I_1 \cup I_2 \cup \ldots \cup I_h$ is left, then we start I_{h+1} with i. The procedure stops if all activities are covered by the constructed cliques. To all these maximal cliques the described tests may iteratively be applied, where only cliques I_ν with $|I_\nu| \geq 2$ (corresponding to disjunctive sets) are used. The tests are stopped if the time windows of the activities are no longer reduced (i.e. a fixpoint is reached).

Starting with the initial set D_0 of disjunctions and with an initial SSD-matrix $d = (d_{ij})$ defined by C_0, N_0 and given heads and tails we repeat the constrained propagation procedure to all disjunctive sets until we detect infeasibility or the SSD-matrix does not change any more. Of course, additionally also symmetric triples and its extensions may be considered in the procedure.

The interval consistency tests can also be seen as tests in which activity sequences with a special property are studied (e.g. an activity which is (or is not) scheduled first or last). Then the objective is to find a contradiction to this hypothesis implying that the reversed property must hold. A similar approach is the so-called **shaving** technique. In this method a hypothetical starting time of an activity is fixed and consequences of this decision are propagated. If afterwards inconsistency is detected, the hypothesis is falsified and the opposite constraint can be fixed. For example, if the hypothetical constraint $S_i > t_i$ with $t_i \in [r_i, d_i - p_i[$ leads to a contradiction, the time window of activity i can be reduced to $[r_i, t_i + p_i]$.

Although the presented tests may deduce several new constraints and reduce the search space, in practice one often has to find a tradeoff between the usefulness of the tests and the time spent for them. Especially, the shaving techniques are very time consuming if they are applied to every activity and every possible starting time in its time window.

3.6.5 Cumulative resources

Recall that renewable resources k for which $R_k > 1$ holds, are called **cumulative resources**. For these resources the tests of the previous section are not valid. However, some concepts may be generalized introducing the term "work" or "energy".

For this purpose we consider a fixed resource k. Let I_k be the set of activities i with $r_{ik} > 0$. Then $w_i := r_{ik}p_i$ is the **work** needed to process activity i. For $J \subseteq I_k$ we define $W(J) := \sum\limits_{i \in J} w_i$. Since in an interval $[t_1, t_2]$ with $t_1 < t_2$ the work $R_k(t_2 - t_1)$ is available, similarly to Theorem 3.7 we have

Theorem 3.8 Let $J', J'' \subset J \subseteq I_k$. If

$$R_k \cdot \max_{\substack{\nu \in J \setminus J' \\ \mu \in J \setminus J''}} (d_\mu - r_\nu) = R_k \cdot (\max_{\mu \in J \setminus J''} d_\mu - \min_{\nu \in J \setminus J'} r_\nu) < W(J), \tag{3.145}$$

then in J an activity from J' must start first or an activity from J'' must end last. $\square$

In contrast to (3.121), in (3.145) we cannot impose $\nu \neq \mu$ because in the non-disjunctive case an activity which starts first may also complete last.

This theorem can be used to derive tests similar to those listed in Table 3.1. The meaning of conclusions such as $J \setminus \{i\} \to i$ or $i \to J \setminus \{i\}$ is that i must complete after (start before) activities in $J \setminus \{i\}$. In contrast to the disjunctive case this does not imply that it must also start after (complete before) $J \setminus \{i\}$.

3.6.6 Constraint propagation for the multi-mode case

In this subsection we shortly describe how some constraint propagation techniques for the RCPSP can be extended to the multi-mode case.

At first some superfluous modes and resources may be eliminated in a preprocessing step. A mode $m \in \mathcal{M}_i$ of an activity i is called **non-executable** if its resource requirements cannot be fulfilled in any feasible schedule, i.e. if $r^\rho_{ikm} > R^\rho_k$ for a renewable resource $k \in \mathcal{K}^\rho$ or if $r^\nu_{ikm} + \sum\limits_{\substack{j=1 \\ j \neq i}}^{n} \min\limits_{\mu \in \mathcal{M}_j} \{r^\nu_{jk\mu}\} > R^\nu_k$ for a non-renewable resource $k \in \mathcal{K}^\nu$ holds.

A mode $m \in \mathcal{M}_i$ of an activity i is called **inefficient** if another mode $\mu \in \mathcal{M}_i$ for the same activity exists in which i does not need more resource units from each resource and does not have a larger duration than in mode m, i.e. $p_{i\mu} \leq p_{im}$, $r^\rho_{ik\mu} \leq r^\rho_{ikm}$ for all renewable resources $k \in \mathcal{K}^\rho$ and $r^\nu_{ik\mu} \leq r^\nu_{ikm}$ for all non-renewable resources $k \in \mathcal{K}^\nu$.

Finally, a non-renewable resource k is called **redundant** if $\sum\limits_{j=1}^{n} \max\limits_{\mu \in \mathcal{M}_j} \{r^\nu_{jk\mu}\} \leq R^\nu_k$ holds, i.e. resource k is sufficiently available independent of the assigned modes.

Obviously, non-executable and inefficient modes as well as redundant resources may be deleted from a multi-mode instance without changing the optimal solution.

Example 3.16: Consider a multi-mode project with $n = 4$ activities, where each activity may be processed in two different modes $m = 1, 2$. We are given three resources: one renewable resource 1 with capacity $R_1^\rho = 4$ and two non-renewable resources 2,3 with $R_2^\nu = 13$ and $R_3^\nu = 14$. Furthermore, the activities $i = 1, \dots, 4$ in modes $m = 1, 2$ have the following processing times p_{im} and resource requirements $r_{i1m}^\rho, r_{ikm}^\nu$ for $k = 2, 3$:

(i,m)	$(1,1)$	$(1,2)$	$(2,1)$	$(2,2)$	$(3,1)$	$(3,2)$	$(4,1)$	$(4,2)$
p_{im}	2	4	3	5	2	3	3	4
r_{i1m}^ρ	5	2	3	1	2	1	2	2
r_{i2m}^ν	2	4	3	2	8	2	3	1
r_{i3m}^ν	1	1	3	4	3	3	2	7

Due to $r_{111}^\rho = 5 > R_1^\rho = 4$ mode 1 of activity 1 is non-executable with respect to the renewable resource 1 and can therefore be deleted from the input data. This induces that mode 1 of activity 3 becomes non-executable with respect to non-renewable resource 2 since

$$r_{321}^\nu + \sum_{j \neq 3} \min_{\mu \in \mathcal{M}_j} \{r_{j2\mu}\} = 8 + 4 + 2 + 1 = 15 > R_2^\nu = 13.$$

After removing this mode, resource 2 becomes redundant and can also be deleted. Then mode 2 of activity 4 becomes inefficient and eleminating this mode causes that resource 3 also becomes redundant. Thus, the given instance may be reduced to the following instance with one renewable resource 1 with $R_1^\rho = 4$ and no non-renewable resource (the mode numbers are adjusted):

(i,m)	$(1,1)$	$(2,1)$	$(2,2)$	$(3,1)$	$(4,1)$
p_{im}	4	3	5	3	3
r_{i1m}^ρ	2	3	1	1	2

□

The example shows that removing a non-executable or inefficient mode may cause a redundant resource, and deleting a redundant resource may lead to a new inefficient mode. Thus, the elimination steps may have to be iterated as follows:

1. Eliminate all non-executable modes (several passes through the activities may be necessary).
2. Eliminate redundant resources.
3. Eliminate all inefficient modes. If a mode was eliminated, go to Step 2.

In order to reduce a multi-mode situation to the classical single-mode case, for each activity i we define its minimal processing time and minimal resource requirements by

$$p_i := \min_{m \in \mathcal{M}_i} \{p_{im}\},\ r^{\rho}_{ik} := \min_{m \in \mathcal{M}_i} \{r^{\rho}_{ikm}\}\ \forall\ k \in \mathcal{K}^{\rho},\ r^{\nu}_{ik} := \min_{m \in \mathcal{M}_i} \{r^{\nu}_{ikm}\}\ \forall\ k \in \mathcal{K}^{\nu}.$$

Since these values are lower bounds for the actual mode-dependent values in any feasible schedule, all lower bound and constraint propagation methods for the single-mode case can be applied to these data. Furthermore, we define a distance matrix $d = (d_{ij})_{i,j \in V}$, where the entries d_{ij} define lower bounds for the differences of starting times $S_j - S_i$. For example, if the precedence relation $i \to j$ exists, we set $d_{ij} := \min\limits_{m \in \mathcal{M}_i} \{p_{im}\}$. If additionally, mode-dependent start-start time-lags $d^{m\mu}_{ij}$ for activity i in mode m and activity j in mode μ are given, we may define d by

$$d_{ij} = \begin{cases} 0, & \text{if } i = j \\ \min\limits_{m \in \mathcal{M}_i} \min\limits_{\mu \in \mathcal{M}_j} \{d^{m\mu}_{ij}\}, & \text{if a time-lag between } i \text{ and } j \text{ exists} \\ p_i - T, & \text{otherwise} \end{cases} \tag{3.146}$$

where T denotes a given time horizon for the project. Let A be the set of all pairs (i, j) for which a time-lag $d^{m\mu}_{ij}$ for some modes $m \in \mathcal{M}_i, \mu \in \mathcal{M}_j$ exists.

Again, this matrix may be transitively adjusted by the Floyd-Warshall algorithm and heads and tails are given by d_{0i} and $d_{i,n+1}$, respectively. In order to strengthen these heads and tails, the modes may be taken into account more carefully.

For this purpose let r^m_i be a lower bound for the starting time of activity i if it is processed in mode $m \in \mathcal{M}_i$ (independent of the modes of the other activities). We must have $r^m_i \geq \min\limits_{\mu \in \mathcal{M}_j} \{r^{\mu}_j + d^{\mu m}_{ji}\}$ for all other activities j since each j has to be processed in some mode $\mu \in \mathcal{M}_j$. Thus, the values r^m_i can be calculated by the recursion

$$r^m_i = \max_{\{j \mid (j,i) \in A\}} \left[\min_{\mu \in \mathcal{M}_j} \{r^{\mu}_j + d^{\mu m}_{ji}\}\right]. \tag{3.147}$$

Initially, we set $r^m_i := r_i = d_{0i}$ for all activities $i \in V$ and all modes $m \in \mathcal{M}_i$. Iteratively we scan all activities $i \in V$ in the order $1, 2, \ldots, n, n+1, 1, 2, \ldots$ and update the values r^m_i for all $m \in \mathcal{M}_i$ according to (3.147). This label-correcting procedure (cf. Section 2.2.2) stops if during one scan of all activities $i \in V$ no value is changed any more. Then $r_i := \min\limits_{m \in \mathcal{M}_i} \{r^m_i\}$ provides a new head for activity i.

Calculating all these values can be done in pseudo-polynomial time. One scan through all activities needs $O(\sum\limits_{i \in V} \sum\limits_{\{j \mid (j,i) \in A\}} |\mathcal{M}_i||\mathcal{M}_j|)$ time and it can be shown that at most $O(\sum\limits_{i \in V} |\mathcal{M}_i| T)$ iterations are needed.

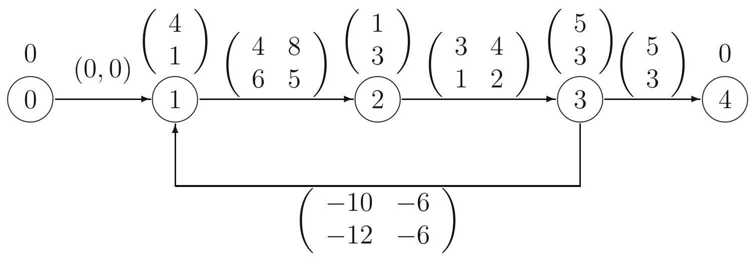
Symmetrically, let q^m_i be a lower bound for the time we need after starting activity i if it is processed in mode m. These values can be calculated by the recursion

$$q_i^m = \max_{\{j|(i,j)\in A\}} [\min_{\mu\in\mathcal{M}_j} \{d_{ij}^{m\mu} + q_j^\mu\}], \tag{3.148}$$

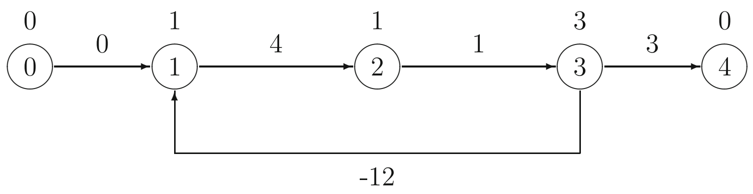
where initially $q_i^m := q_i = d_{i,n+1}$ for all $i \in V$ and $m \in \mathcal{M}_i$. After no value is changed any more, $q_i := \min\limits_{m\in\mathcal{M}_i} \{q_i^m\}$ defines a new tail for activity i. The values r_i and q_i may be used to adjust the corresponding entries d_{0i} and $d_{i,n+1}$ in the distance matrix d. Furthermore, for a given value T a deadline d_i for the completion time of activity i is given by $d_i := \max\limits_{m\in\mathcal{M}_i} \{T - q_i^m + p_{im}\}$.

After the calculation of time windows $[r_i, d_i]$ additional methods of constraint propagation may be applied like in the single-mode case (e.g. by using the concept of symmetric triples or the disjunctive set tests).

Example 3.17: Consider the multi-mode instance with time-lags $d_{ij}^{m\mu}$ and $n = 3$ activities shown in Figure 3.41(a), where each activity can be processed in two different modes. In the figure for each activity i the corresponding processing time vector $(p_{im}) = \begin{pmatrix} p_{i1} \\ p_{i2} \end{pmatrix}$ is given, furthermore, each arc $i \to j$ is weighted with the matrix $(d_{ij}^{m\mu})$ of start-start time-lags $\begin{pmatrix} d_{ij}^{11} & d_{ij}^{12} \\ d_{ij}^{21} & d_{ij}^{22} \end{pmatrix}$.



(a) Multi-mode instance with time-lags and $n = 3$ activities



(b) Corresponding single-mode instance

Figure 3.41: Multi-mode instance and the corresponding single-mode instance

Associated with this multi-mode instance is the single-mode instance shown in Figure 3.41(b), where each node i is weighted with the minimal processing time p_i and each arc $i \to j$ is weighted with the minimal distance d_{ij}.

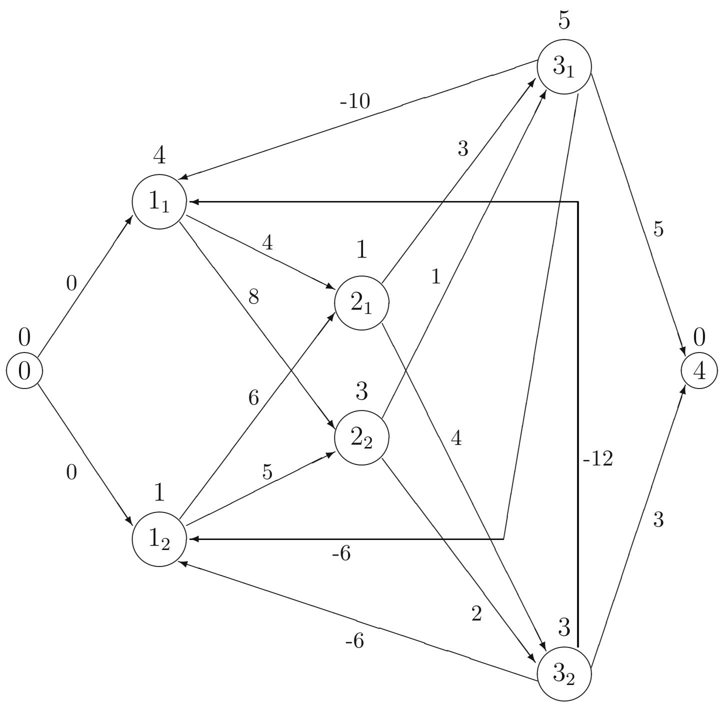


Figure 3.42: Expanded network for the multi-mode instance

In order to calculate the values r_i^m according to (3.147) we consider the network shown in Figure 3.42 with nodes i_m for all activity-mode combinations (i, m) and arcs $i_m \xrightarrow{d_{ij}^{m\mu}} j_\mu$.

During the first scan of the activities 1, 2, 3 we calculate the values r_1^1, r_1^2, r_2^1, r_2^2, r_3^1, r_3^2. For the combinations $(1, m)$ with $m = 1, 2$ we get $r_1^m = r_0 + 0 = 0$, for $(2, m)$ we get

$$\begin{aligned} r_2^1 &= \min\{r_1^1 + d_{12}^{11}, r_1^2 + d_{12}^{21}\} = \min\{4, 6\} = 4, \\ r_2^2 &= \min\{r_1^1 + d_{12}^{12}, r_1^2 + d_{12}^{22}\} = \min\{8, 5\} = 5, \end{aligned}$$

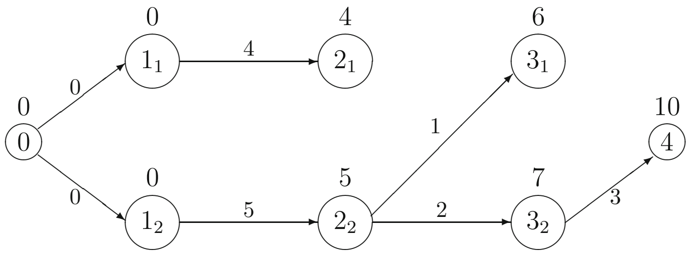
and for $(3, m)$ we get

$$\begin{aligned} r_3^1 &= \min\{r_2^1 + d_{23}^{11}, r_2^2 + d_{23}^{21}\} = \min\{4+3, 5+1\} = 6, \\ r_3^2 &= \min\{r_2^1 + d_{23}^{12}, r_2^2 + d_{23}^{22}\} = \min\{4+4, 5+2\} = 7. \end{aligned}$$

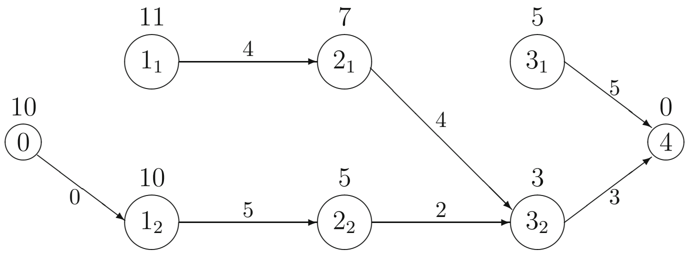
During the second scan of the activities we get

$$\begin{aligned} r_1^1 &= \max\{r_0 + 0, \min\{r_3^1 + d_{31}^{11}, r_3^2 + d_{31}^{21}\}\} = \max\{0, \min\{-4, -5\}\} = 0, \\ r_1^2 &= \max\{r_0 + 0, \min\{r_3^1 + d_{31}^{12}, r_3^2 + d_{31}^{22}\}\} = \max\{0, \min\{0, 1\}\} = 0, \end{aligned}$$

i.e. the values r_1^m are not changed in the second iteration. Also the other values are not changed any more and we get the heads r_i^m as shown in Figure 3.43(a) and the tails q_i^m as in Figure 3.43(b).



(a) Calculated heads r_i^m



(b) Calculated tails q_i^m

Figure 3.43: Calculation of heads r_i^m and tails q_i^m

Thus, improved heads r_i and tails q_i for the activities are given by

$$r_1 = 0, r_2 = 4, r_3 = 6, r_4 = 10, q_3 = 3, q_2 = 5, q_1 = 10, q_0 = 10.$$

$\square$

3.6.7 Reference notes

The book of Baptiste et al. [10] and the survey on disjunctive scheduling problems by Dorndorf et al. [67] contain the most important constraint propagation techniques in connection with scheduling problems.

A survey on the different disjunctive interval consistency tests can be found in Dorndorf et al. [66], Dorndorf [65], and Phan Huy [164]. In connection with a branch-and-bound algorithm for the job-shop problem, Carlier and Pinson [42] were the first to present an algorithm which finds all inputs and outputs for all subsets $J \subseteq I$. It is based on calculating Jackson's preemptive schedule for the considered set and updates all release dates and deadlines in $O(n^2)$. The $O(n^2)$-implementation of the input/output test presented in Section 3.6.4 is due to Nuijten [158]. Improved algorithms with complexity $O(n \log n)$ were later proposed by Carlier and Pinson [43] and Brucker et al. [31] using more complex data structures.

The $O(n^2)$-implementation of the input/output negation test presented in Section 3.6.4 is due to Baptiste and Le Pape [9]. An algorithm where one iteration needs $O(n \log n)$ time, but in general more iterations are necessary, was proposed by Vilim [189]. The input-or-output test has been suggested by Dorndorf et al. [66] and an $O(n^3)$-algorithm was given by Phan Huy [164].

Usually, in the literature only implementations for the weaker tests (3.122), in which the condition $\nu \neq \mu$ is dropped, are considered. Phan Huy [164] developed an $O(n^3)$-algorithm for the stronger input/output tests based on (3.121) and an $O(n^4)$-algorithm for the stronger input-or-output test, where additionally $\nu \neq \mu$ is imposed.

Interval consistency tests for cumulative resources are also called "energetic reasoning" and are summarized in Baptiste et al. [10]. The concept of shaving has been proposed by Martin and Shmoys [139].

The concept of symmetric triples and its extensions was introduced by Brucker et al. [33] in connection with a branch-and-bound algorithm for the RCPSP.

The preprocessing procedures reducing modes and resources in the multi-mode case are presented in Sprecher et al. [180], the calculation of improved heads and tails taking into account the modes can be found in Heilmann [96].

3.7 Lower Bounds

In this section methods to calculate lower bounds for the makespan of the RCPSP are discussed. If LB is a lower bound for an instance of the RCPSP and UB is the solution value given by some heuristic, then $UB - LB$ is an upper bound for the distance between the optimal solution value and UB. Furthermore, $\frac{UB-LB}{LB}$ is an upper bound for the relative error. Thus, good lower bounds may be used to estimate the quality of heuristic solutions. They are also needed in connection with branch-and-bound algorithms.

Usually, two types of lower bounds can be distinguished for a combinatorial optimization problem: constructive and destructive bounds. **Constructive** lower bounds are usually provided by solving relaxations of the problem, which are less complex than the original problem. In a **relaxation** of an optimization problem certain restrictions (which make the problem hard) are eliminated. For example, if we relax the resource constraints of the RCPSP, the optimal makespan of the relaxed problem is equal to the length of a longest path in the activity-on-node network, which can be calculated efficiently.

Destructive lower bounds are based on a different idea. To derive lower bounds with this technique we consider the decision version (feasibility problem) of the optimization problem: Given a threshold value T, does a feasible schedule exist with an objective value smaller than or equal to T? If we can prove that such a schedule does not exist, then $T + 1$ is a valid lower bound for the optimization problem supposing that all data are integral. To contradict (destruct) a threshold value T, again relaxations may be used. If we can state infeasibility for a relaxed problem, obviously the original problem is also infeasible. To find the best lower bound we search for the largest T, where infeasibility can be proved.

This can be organized in incremental steps or by binary search. An incremental procedure starts with a valid lower bound value $T = LB$ and increases T in each step by an appropriate value $\Delta \geq 1$ until it is not possible to state infeasibility. When applying binary search, in each step we consider an interval $[L, U]$ in which we search for a valid lower bound value. Initially, we start with a valid upper bound U^0 and a valid lower bound L^0. In each iteration we solve the feasibility problem for $T = \lfloor \frac{U+L}{2} \rfloor$. If we can prove that no feasible solution with an objective value smaller than or equal to T exists, we increase the lower bound L to the value $T+1$ and repeat the procedure with the interval $[T+1, U]$. Otherwise, if we do not succeed in proving infeasibility for the threshold T, we replace U by T and repeat the procedure in the interval $[L, T]$. The binary search procedure terminates as soon as $L \geq U$ holds after at most $O(\log(U^0 - L^0))$ steps. Then L equals the largest lower bound value which can be calculated in this way.

In the next subsections we discuss several methods to calculate lower bounds for the optimal makespan $C_{\max}$. While in Sections 3.7.1 to 3.7.3 constructive lower bounds are described, a destructive lower bound can be found in Section 3.7.4. This approach is generalized to the multi-mode case in Section 3.7.5.

3.7.1 Combinatorial constructive lower bounds

In Section 3.1.1 we already considered a simple constructive lower bound: if we relax all resource constraints and only take into account the precedence constraints, the length of a critical path in the activity-on-node network defines a lower bound for the optimal makespan, denoted by LB_0. If no generalized precedence relations are given, the graph is acyclic, and a critical path can be calculated in $O(n^2)$ time.

Another simple lower bound can be determined in $O(nr)$ time by considering each resource separately. For each renewable resource $k = 1, \dots, r$ the value $\lceil \sum\limits_{i=1}^{n} r_{ik}\, p_i / R_k \rceil$ defines a lower bound for the optimal makespan because the total work $\sum\limits_{i=1}^{n} r_{ik}\, p_i$ for all activities cannot be greater than the work $R_k \cdot C_{\max}$ which is available for resource k in the interval $[0, C_{\max}]$. Thus, the maximum among all resources gives the resource-based lower bound

$$LB_1 := \max_{k=1}^{r} \left\lceil \sum_{i=1}^{n} r_{ik}\, p_i / R_k \right\rceil .$$

The longest path length LB_0 of a critical path CP_0 can be strengthened by partially taking into account some resource conflicts as follows. If a schedule with $C_{\max} = LB_0$ exists, each activity i has to be completed before the deadline $d_i := LB_0 - q_i$ (recall that the tail q_i is a lower bound for the length of the time period between the completion time of i and the optimal makespan).

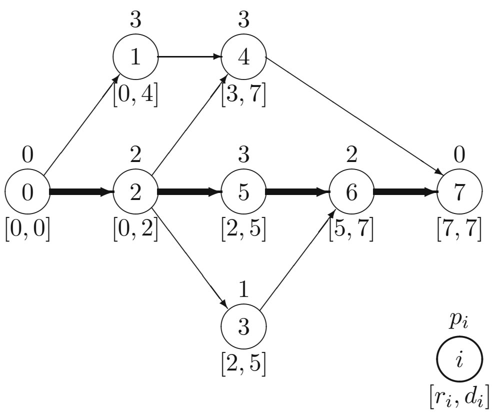
For each activity i not belonging to the critical path CP_0, let e_i be the maximal length of an interval contained in $[r_i, d_i]$ in which i can be processed with the activities from CP_0 simultaneously without violating the resource constraints. If $e_i < p_i$ holds, no feasible schedule with $C_{\max} = LB_0$ exists and in order to get a feasible schedule the time window of activity i has to be enlarged by at least $p_i^+ := \max \{p_i - e_i, 0\}$ time units. Thus,

$$LB_S := LB_0 + \max_{i \notin CP_0} \{p_i^+\}$$

defines a valid lower bound value. For each activity not in CP_0 we have to check, whether the resources left by the activities from CP_0 are sufficient or not. This can be done in $O(nr|CP_0|)$ time, where $|CP_0|$ denotes the number of activities on CP_0.

Example 3.18 : Consider the instance with $n = 6$ activities and $r = 2$ resources with capacities $R_1 = 3, R_2 = 1$ shown in Figure 3.44.

The critical path $CP_0 = (0 \to 2 \to 5 \to 6 \to 7)$ has the length $LB_0 = 7$. In Figure 3.45 the partial schedule for all critical activities from CP_0 is drawn, furthermore, the time windows for the remaining activities $i \notin CP_0$ are shown. For these activities we have $e_1 = 2$ since activity 1 can be processed in parallel



i	1	2	3	4	5	6
p_i	3	2	1	3	3	2
r_{i1}	2	2	0	2	1	2
r_{i2}	1	1	1	1	0	1
r_i	0	0	2	3	2	5
q_i	3	5	2	0	2	0
d_i	4	2	5	7	5	7

Figure 3.44: A project with $n = 6, r = 2$ and time windows for $LB_0 = 7$

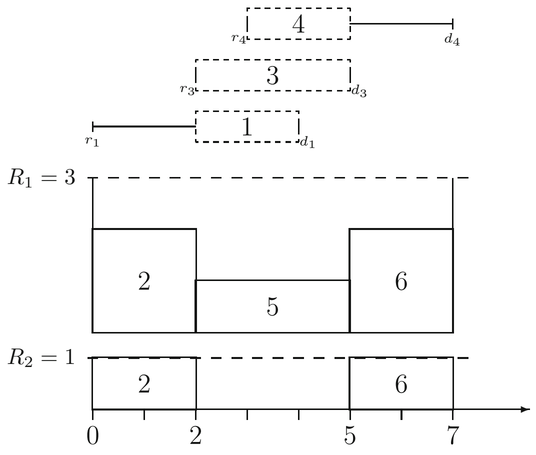


Figure 3.45: Partial schedule for CP_0

with CP_0 in the interval $[2, 4]$, $e_3 = 3$ since 3 can be processed in the interval $[2, 5]$ and $e_4 = 2$.

Thus, $p_1^+ = 3 - 2 = 1$, $p_3^+ = \max\{1 - 3, 0\} = 0$, $p_4^+ = 3 - 2 = 1$ and

$$LB_S = LB_0 + \max_{i \notin CP_0} \{p_i^+\} = 7 + 1 = 8.$$

□

Note that in LB_S only conflicts between activities from CP_0 and activities $i \notin CP_0$ are taken into account, conflicts among activities $i \notin CP_0$ are not considered. More complicated extensions of the critical path lower bound LB_0 may be obtained by considering a critical path together with a second node-disjoint path in the network. Taking into account the fixed positions of the activities from the critical path, the minimum completion time of the activities in the second path provides a lower bound.

3.7.2 Lower bounds based on Lagrangian relaxation

As already mentioned in Section 3.3, lower bounds can also be obtained by solving the continuous relaxations of IP formulations. In this subsection we describe a lower bound approach based on **Lagrangian relaxation** (cf. Section 2.4) where the resource constraints are relaxed, but violations of them are penalized in the objective function. We use the disaggregated discrete time (DDT) formulation from Section 3.3.1 with variables $x_{it} \in \{0,1\}$ where $x_{it} = 1$ if and only if activity i starts at time t:

$$\min \quad \sum_{t=0}^{T} t x_{n+1,t} \tag{3.149}$$

$$\text{s.t.} \quad \sum_{t=0}^{T} x_{it} = 1 \qquad (i \in V) \tag{3.150}$$

$$\sum_{\tau=t}^{T} x_{i\tau} + \sum_{\tau=0}^{\min\{t+p_i-1,T\}} x_{j\tau} \leq 1 \qquad ((i,j) \in A;\ t = 0,\ldots,T-1) \tag{3.151}$$

$$\sum_{i=1}^{n} r_{ik} \sum_{\tau=\max\{0,t-p_i+1\}}^{t} x_{i\tau} \leq R_k \qquad (k = 1,\ldots,r;\ t = 0,\ldots,T) \tag{3.152}$$

$$x_{it} \in \{0,1\} \quad (i \in V;\ t = 0,\ldots,T) \tag{3.153}$$

If we relax the resource constraints (3.152) and introduce Lagrangian multipliers $\lambda_{tk} \geq 0$ for $t = 0,\ldots,T; k = 1,\ldots,r$, we get the objective function

$$\min \quad \sum_{t=0}^{T} t x_{n+1,t} + \sum_{t=0}^{T}\sum_{k=1}^{r} \lambda_{tk} \left(\sum_{i=1}^{n} r_{ik} \sum_{\tau=\max\{0,t-p_i+1\}}^{t} x_{i\tau} \right) - \sum_{t=0}^{T}\sum_{k=1}^{r} \lambda_{tk} R_k.$$

After reordering the second sum term and due to

$$\sum_{t=0}^{T} \lambda_{tk} \sum_{\tau=\max\{0,t-p_i+1\}}^{t} x_{i\tau} = \sum_{t=0}^{T} x_{it} \sum_{\tau=t}^{\min\{t+p_i-1,T\}} \lambda_{\tau k} \text{ for all } i,k$$

we obtain the Lagrangian subproblem

$$\min \quad \sum_{t=0}^{T} t x_{n+1,t} + \sum_{i=1}^{n}\sum_{t=0}^{T} \left(\sum_{k=1}^{r} r_{ik} \sum_{\tau=t}^{\min\{t+p_i-1,T\}} \lambda_{\tau k} \right) x_{it} - \sum_{t=0}^{T}\sum_{k=1}^{r} \lambda_{tk} R_k$$

$$\text{s.t.} \quad (3.150), (3.151), (3.153).$$

By omitting the constant term $\sum_{t=0}^{T}\sum_{k=1}^{r} \lambda_{tk} R_k$ and introducing weights

$$w_{it} := \begin{cases} \sum_{k=1}^{r} r_{ik} \sum_{\tau=t}^{\min\{t+p_i-1,T\}} \lambda_{\tau k}, & \text{if } i \neq n+1 \\ t, & \text{if } i = n+1 \end{cases}$$

for $i = 1, \dots, n+1; t = 0, \dots, T$ the problem can be rewritten as

$$\min \quad \sum_{i=1}^{n+1} \sum_{t=0}^{T} w_{it} x_{it} \tag{3.154}$$

$$\text{s.t.} \quad \sum_{t=0}^{T} x_{it} = 1 \qquad (i \in V) \tag{3.155}$$

$$\sum_{\tau=t}^{T} x_{i\tau} + \sum_{\tau=0}^{\min\{t+p_i-1,T\}} x_{j\tau} \leq 1 \qquad ((i,j) \in A;\ t = 0, \dots, T-1) \tag{3.156}$$

$$x_{it} \in \{0,1\} \quad (i \in V;\ t = 0, \dots, T) \tag{3.157}$$

In this problem we have to schedule $n+1$ activities subject to precedence but without resource constraints such that $\sum\limits_{i=1}^{n+1} f_i(S_i)$ with $f_i(t) = w_{it}$ is minimized. Thus, the Lagrangian subproblem is a special case of problem (P) introduced in Section 3.5.4 where $d_{ij} = p_i$ and $w_{it} \geq 0$ due to $\lambda \geq 0$. Hence, it can efficiently be solved as a minimum cut problem.

As discussed in Section 2.4, for any vector $\lambda = (\lambda_{tk})$ the optimal solution value $L(\lambda)$ of the Lagrangian subproblem is a lower bound for the optimal makespan of the RCPSP. The best lower bound achievable in this way can be obtained by solving the Lagrangian dual problem

$$\max_{\lambda \geq 0} \{L(\lambda)\}.$$

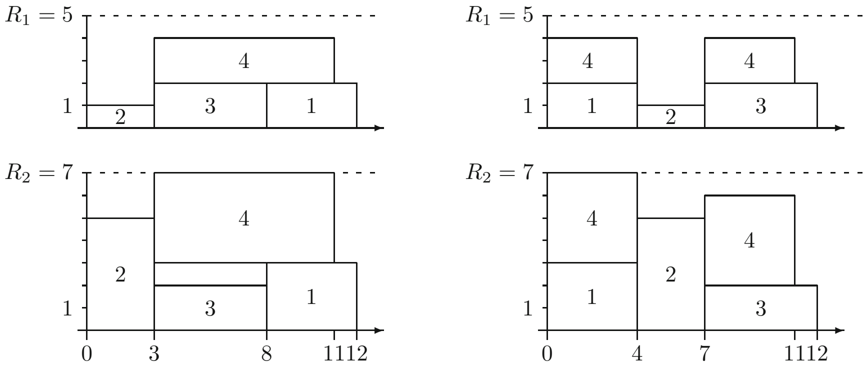
It can be shown that the Lagrangian subproblem has the integrality property, i.e. the optimal value $L(\lambda^*)$ of the Lagrangian dual problem is equal to the optimal value of the continuous LP-relaxation of (3.149)-(3.153). However, the Lagrangian dual can often be solved in a shorter amount of time (e.g. with the subgradient method).

3.7.3 An LP-based constructive lower bound

In the following we consider a relaxation of the RCPSP partially relaxing the precedence constraints and allowing preemption. This relaxation can be formulated as a linear program where the columns correspond to so-called feasible subsets.

Example 3.19: Let us again consider the instance from Example 1.1 and its optimal schedule shown in Figure 3.46(a). If we cut this schedule into slices in which the same activities are scheduled, we get the set $\{2\}$ in $[0,3[$, the set $\{3,4\}$ in $[3,8[$, the set $\{1,4\}$ in $[8,11[$, and the set $\{1\}$ in $[11,12[$.

In Figure 3.46(b) a preemptive schedule is shown consisting of the sets $\{1,4\}$, $\{2\}$, $\{3,4\}$, and $\{3\}$. $\square$



(a) A non-preemptive schedule (b) A preemptive schedule

Figure 3.46: Two schedules for Example 1.1

Feasible subsets X are subsets of activities which may be processed simultaneously, i.e. there are no conjunctions or disjunctions between any pair of activities $i, j \in X$ and all resource constraints are respected (i.e. $\sum\limits_{i \in X} r_{ik} \leq R_k$ for $k = 1, \ldots, r$). This means that the conjunctions are treated as disjunctions (instead of $i \rightarrow j \in C$ we only impose $i - j \in D$, i.e. in the relaxation also j may be scheduled before i).

Let us consider all feasible subsets and denote them by $X_1, X_2, \ldots, X_g$. With each subset X_j we associate an incidence vector $a^j \in \{0,1\}^n$ defined by

$$a_i^j := \begin{cases} 1, & \text{if } i \in X_j \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, let $x_j \geq 0$ be a variable denoting the total number of time units when the subset X_j is processed. Then the preemptive relaxation can be formulated as the following linear program:

$$\min \quad \sum_{j=1}^{g} x_j \tag{3.158}$$

$$\text{s.t.} \quad \sum_{j=1}^{g} a_i^j x_j = p_i \qquad (i = 1, \ldots, n) \tag{3.159}$$

$$x_j \geq 0 \qquad (j = 1, \ldots, g). \tag{3.160}$$

In (3.158) we minimize the makespan $C_{\max} = \sum\limits_{j=1}^{g} x_j$ which is equal to the sum of all processing times of the subsets. The constraints (3.159) ensure that all activities i are processed for exactly p_i time units.

Unfortunately, the number of all feasible subsets is very large since it grows exponentially with the number n of activities. In order to reduce this large number, we may reduce the set of all feasible subsets to a smaller subset of so-called non-dominated ones. We call a set X **dominated** if it is a proper subset $X \subset Y$ of another feasible subset Y, otherwise it is called **non-dominated**. We consider all non-dominated feasible subsets and denote them by $X_1, X_2, \ldots, X_f$. Then the linear program (3.158)-(3.160) is changed to

$$\min \quad \sum_{j=1}^{f} x_j \tag{3.161}$$

$$\text{s.t.} \quad \sum_{j=1}^{f} a_i^j x_j \geq p_i \qquad (i = 1, \ldots, n) \tag{3.162}$$

$$x_j \geq 0 \qquad (j = 1, \ldots, f). \tag{3.163}$$

This formulation differs from the previous one only in constraints (3.162) where we must allow that an activity i may be processed longer than its processing time p_i since only non-dominated subsets are considered. It can easily be shown that formulations (3.158)-(3.160) and (3.161)-(3.163) are equivalent, i.e. the optimal makespans are the same. If in the first formulation a dominated subset is used, we fill it with other activities until it becomes non-dominated (obviously this does not change the makespan). If on the other hand, in the second formulation an activity i is processed longer than its processing time p_i, we eliminate parts of it from the used subsets until exactly p_i time units are scheduled.

Furthermore, the optimal makespan of the preemptive relaxation cannot be less than LB_0 since due to the precedence constraints all activities belonging to a critical path must be contained in different feasible subsets.

Example 3.20: Consider the instance in Figure 3.47 with $n = 6$ activities, one resource with capacity $R_1 = 4$ and $LB_0 = 4$.

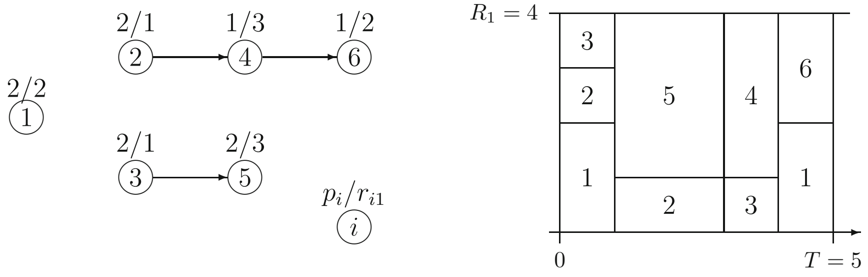


Figure 3.47: A project and a corresponding optimal preemptive schedule

For this instance we have 14 feasible subsets $\{1\}$, $\{1,2\}$, $\{1,2,3\}$, $\{1,3\}$, $\{1,6\}$, $\{2\}$, $\{2,3\}$, $\{2,5\}$, $\{3\}$, $\{3,4\}$, $\{3,6\}$, $\{4\}$, $\{5\}$, $\{6\}$. Among these sets we have 5 non-dominated subsets: $X_1 = \{1,2,3\}$, $X_2 = \{1,6\}$, $X_3 = \{2,5\}$, $X_4 = \{3,4\}$, $X_5 = \{3,6\}$.

An optimal preemptive schedule for the relaxation with makespan 5 is shown in Figure 3.47. Note that all activities $i \neq 2$ are processed for exactly p_i time units, only activity 2 is processed longer for $p_2 + 1 = 3$ time units. This schedule corresponds to the LP solution $x_1 = 1, x_2 = 1, x_3 = 2, x_4 = 1, x_5 = 0$. □

Unfortunately, the number of all non-dominated feasible subsets still grows exponentially with the number n of activities. For $n = 60$ we have approximately $f = 300\,000$, for $n = 90$ even $f = 8\,000\,000$ columns. However, it is not necessary to generate and store all these columns.

As described in Section 2.3.4, we can work with only a few of them at a time and only generate new feasible sets when they are really needed. In the following we apply such a **delayed column generation approach** to (3.161) to (3.163) and describe the problem-specific details. As shown in the generic column generation algorithm in Figure 2.11, besides the problem-independent subprocedure `Optimize` we have the problem-specific subprocedures `Initialize` providing a basic starting solution, `InsertDeleteColumns` organizing the insertion and deletion of columns, and the pricing procedure `CalculateColumns`.

`Initialize`: As a starting basis for the first iteration of the simplex method we may use the one-element sets $X_i := \{i\}$ for $i = 1, \ldots, n$ and set $x_i := p_i$ for the corresponding variables x_i (corresponding to the unit-vectors $e^i \in \mathbb{R}^n$). These sets may be dominated, but we use them anyway. Furthermore, we add the columns corresponding to the slack variables in (3.162) to the working set.

`InsertDeleteColumns`: We use a delete-strategy which never deletes the slack variables from the working set, i.e. $-e^i$ is always in the working set.

`CalculateColumns`: After solving the corresponding LP to optimality by the procedure `Optimize`, we search for feasible columns which are able to improve the objective value when entering the basis. If y_i $(i = 1, \ldots, n)$ are the values of the dual variables associated with the current basic solution, we have to find a feasible column $a \in \{0, 1\}^n$ with $ya > 1$, i.e. a has to satisfy

$$\sum_{i=1}^{n} y_i \cdot a_i \quad > \quad 1 \tag{3.164}$$

$$\sum_{i=1}^{n} r_{ik} \cdot a_i \quad \leq \quad R_k \quad (k = 1, \ldots, r) \tag{3.165}$$

$$a_i \cdot a_j \quad = \quad 0 \qquad (i - j \in D, i \rightarrow j \in C \text{ or } j \rightarrow i \in C). \tag{3.166}$$

Conditions (3.165) and (3.166) ensure that a column a corresponds to a feasible subset (respecting the resource constraints, the disjunctions D, and the conjunctions C).

Columns which satisfy conditions (3.164) to (3.166) are generated by an enumeration procedure, which works as follows: Since we have solved the current LP to optimality and the slack-variables are kept in the working set, we have for all dual variables $y_i \geq 0$ because

- $-y_i = -ye^i = c_i = 0$, if $-e^i$ is in the basis, or
- $-y_i = -ye^i \leq c_i = 0$, since $-e^i$ is in the working set.

We sort the activities such that

$$y_1 \geq y_2 \geq \ldots \geq y_{n_0} > 0, \; y_{n_0+1} = \ldots = y_n = 0,$$

where n_0 denotes the last index of a positive dual variable.

Due to (3.164) we only have to consider activities i with $y_i > 0$, i.e. it is sufficient to generate non-dominated vectors $(a_1, \ldots, a_{n_0})$ of length n_0. They can be extended to non-dominated vectors $(a_1, \ldots, a_n)$ with length n by adding some activities $i \in \{n_0 + 1, \ldots, n\}$ which maintain feasibility.

To calculate incidence vectors of relevant non-dominated feasible sets, we define an enumeration tree in which the vertices represent partial vectors $(a_1, \ldots, a_l)$ with $1 \leq l \leq n_0$. A vector $(a_1, \ldots, a_l)$ has at most two sons: $(a_1, \ldots, a_l, 0)$ is the right son, the left son $(a_1, \ldots, a_l, 1)$ exists only if $(a_1, \ldots, a_l, 1, 0, \ldots, 0)$ corresponds to a feasible set.

We apply a depth-first search in the enumeration tree where leaves correspond to feasible subsets. The sets are generated in a lexicographic order.

Example 3.21 : Consider a small example with $n = 4$ activities, $r = 2$ resources with capacities $R_1 = 2$, $R_2 = 3$, and a precedence relation $2 \rightarrow 3$. The resource requirements r_{ik} of the activities are given by

i	1	2	3	4
r_{i1}	1	1	2	0
r_{i2}	1	1	2	2

i.e. the pairs $1 - 3$ and $3 - 4$ are incompatible with respect to the resources.

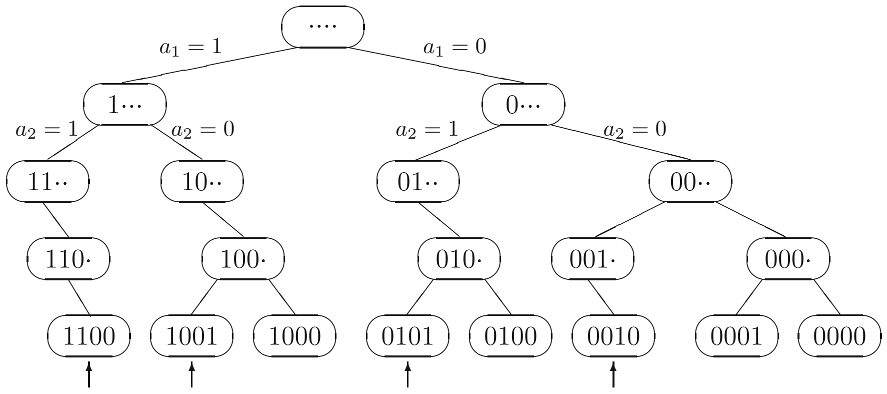


Figure 3.48: Enumeration tree for the calculation of non-dominated sets

The calculation of the non-dominated sets is shown in Figure 3.48, where each partial set is characterized by a 4-tuple $(a_1, a_2, a_3, a_4) \in \{0,1\}^4$. We obtain 8 feasible subsets, 4 of them are non-dominated (indicated by an arrow under the corresponding leaf). □

Instead of checking for each subset whether it is non-dominated or not, we only generate a subset of feasible columns satisfying (3.164) to (3.166), which are automatically non-dominated. Each time a column a^* satisfying (3.164) to (3.166) has been found, we only search for further columns a with

$$\sum_{i=1}^{n_0} y_i a_i > \sum_{i=1}^{n_0} y_i a_i^*. \tag{3.167}$$

This ensures that all further sets during the computation are non-dominated for the following reason: An incidence vector a is dominated if at least one component a_k can be changed from 0 to 1 without violating feasibility. But then a dominating column a' with $a'_k = 1$ has already been considered (either generated as a leaf or eliminated due to its objective value), since we scan the columns in lexicographic order. Let a^* be the best column found when we consider a. Since a' has been considered before, we have

$$\sum_{i=1}^{n_0} y_i a_i^* \geq \sum_{i=1}^{n_0} y_i a'_i \geq \sum_{i=1}^{n_0} y_i a_i + y_k \geq \sum_{i=1}^{n_0} y_i a_i,$$

due to $y \geq 0$, i.e. column a is not generated by our procedure. Thus, we do not have to check dominance explicitly.

When we have found a suitable leaf a^*, we fill this column successively with all activities $n_0+1, \ldots, n$ which do not violate feasibility. Then the corresponding column is also non-dominated and we backtrack to depth n_0. Note that if we search the enumeration tree completely, then a non-dominated column a which maximizes $\sum\limits_{i=1}^{n} y_i a_i$ is found.

In order to reduce the enumeration tree we may include the following simple dominance rule into our enumeration procedure: If we set $a_k := 0$ in step k and if

$$\sum_{\lambda=1}^{k-1} y_\lambda \cdot a_\lambda + \sum_{\lambda=k+1}^{n_0} y_\lambda \leq \sum_{i=1}^{n_0} y_i \cdot a_i^*$$

holds, the best column a^* found so far cannot be improved and therefore we can backtrack.

We stop the search process if one of the following conditions holds:

- we have found a given number of columns, or
- we have found at least one column and the ratio between the number μ of generated leaves and the number μ^u of leaves satisfying (3.167) exceeds a certain limit s (i.e. $\mu \geq \mu^u \cdot s$).

After applying the procedure `CalculateColumns`, the generated columns are added to the working set by the procedure `InsertDeleteColumns`. Furthermore, if the number of columns in the current working set exceeds a given size, some arbitrary non-basic columns are deleted.

Unfortunately, for problems with a larger number of activities the quality of the presented lower bound worsens. The reason is that the possibility of preemption allows us to split the activities into many pieces and to put them appropriately into different feasible subsets yielding low objective values. Therefore, in the next subsection we strengthen the lower bound by additionally taking into account time windows. Furthermore, we use a destructive approach.

3.7.4 An LP-based destructive method

In this section we consider an extension of the previous linear programming formulation where additionally time windows for the activities are taken into account. We use a destructive approach which combines constraint propagation and linear programming techniques.

Given a hypothetical upper bound T, one first tries to prove infeasibility by constraint propagation. Constraint propagation also provides time windows $[r_i, d_i]$ for the activities $i = 1, \ldots, n$. In case that we cannot prove infeasibility, we use the time windows and try to prove that no preemptive schedule with $C_{\max} \leq T$ exists such that all activities are processed within their time windows and all resource constraints are respected.

For the LP-formulation let $z_0 < z_1 < \ldots < z_\tau$ be the ordered sequence of all different r_i- and d_i-values. For $t = 1, \ldots, \tau$ we consider the intervals $I_t := [z_{t-1}, z_t]$ of length $z_t - z_{t-1}$. With each interval I_t we associate a set A_t of all activities i which can partially be scheduled in this interval, i.e. with $r_i \leq z_{t-1} < z_t \leq d_i$. Let X_{jt} $(j = 1, \ldots, f_t)$ be the feasible subsets of A_t and denote by a^{jt} the incidence vector corresponding to X_{jt}. Furthermore, let x_{jt} be a variable denoting the number of time units when the subset X_{jt} is processed. Then the preemptive feasibility problem may be written as follows:

$$\sum_{t=1}^{\tau} \sum_{j=1}^{f_t} a_i^{jt} x_{jt} \geq p_i \qquad (i = 1, \ldots, n) \tag{3.168}$$

$$\sum_{j=1}^{f_t} x_{jt} \leq z_t - z_{t-1} \qquad (t = 1, \ldots, \tau) \tag{3.169}$$

$$x_{jt} \geq 0 \qquad (t = 1, \ldots, \tau;\ j = 1, \ldots, f_t) \tag{3.170}$$

Due to restrictions (3.168) all activities i are processed for at least p_i time units. Conditions (3.169) ensure that the number of time units scheduled in interval I_t does not exceed the length $z_t - z_{t-1}$ of this interval.

By introducing artificial variables u_t for $t = 1, \ldots, \tau$ in conditions (3.169) the feasibility problem can be formulated as a linear program:

$$\min \quad \sum_{t=1}^{\tau} u_t \tag{3.171}$$

$$\text{s.t.} \quad \sum_{t=1}^{\tau} \sum_{j=1}^{f_t} a_i^{jt} x_{jt} \geq p_i \qquad (i = 1, \ldots, n) \tag{3.172}$$

$$-\sum_{j=1}^{f_t} x_{jt} + u_t \geq -z_t + z_{t-1} \qquad (t = 1, \ldots, \tau) \tag{3.173}$$

$$x_{jt} \geq 0 \qquad (t = 1, \ldots, \tau;\ j = 1, \ldots, f_t) \tag{3.174}$$

$$u_t \geq 0 \qquad (t = 1, \ldots, \tau) \tag{3.175}$$

A solution exists for the preemptive feasibility problem if and only if the linear program has the optimal solution value zero, i.e. if all values of the artifical variables become zero.

Example 3.22 : Consider again Example 3.20 from the previous subsection. In Figure 3.49 additionally the time windows $[r_i, d_i]$ for $T = 6$ are shown.

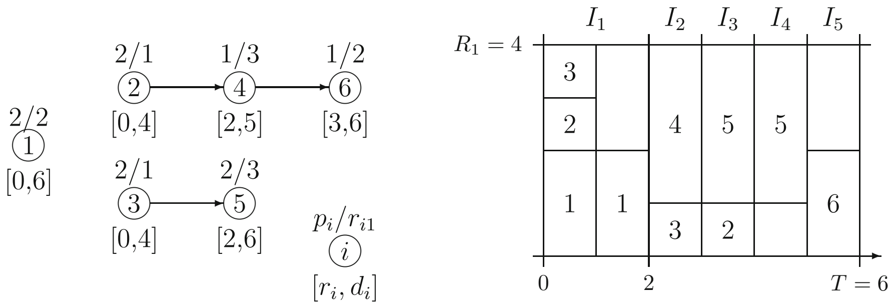


Figure 3.49: A feasible preemptive schedule for $T = 6$

For this instance we have $\tau = 5$ intervals I_t with the following sets A_t:

$$\begin{aligned} I_1 &= [0,2]: & A_1 &= \{1,2,3\}, \\ I_2 &= [2,3]: & A_2 &= \{1,2,3,4,5\}, \\ I_3 &= [3,4]: & A_3 &= \{1,2,3,4,5,6\}, \\ I_4 &= [4,5]: & A_4 &= \{1,4,5,6\}, \\ I_5 &= [5,6]: & A_5 &= \{1,5,6\}. \end{aligned}$$

A feasible preemptive schedule corresponding to the solution

$$x_{\{1,2,3\},1} = 1,\ x_{\{1\},1} = 1,\ x_{\{3,4\},2} = 1,\ x_{\{2,5\},3} = 1,\ x_{\{5\},4} = 1,\ x_{\{6\},5} = 1,$$

(where $x_{X,t}$ denotes that subset X is processed in interval I_t) can be found in Figure 3.49. Furthermore, it is easy to see that for $T = 5$ no feasible schedule respecting the time windows exists. Thus, we get a lower bound value of 6, which improves the lower bound from the previous subsection by one unit. □

Again the linear programming formulation contains an exponential number of variables, but again it can be solved efficiently with column generation techniques. In the following we describe how the procedures from the previous subsection have to be modified:

- `Initialize`: For each activity $i \in \{1\dots,n\}$ let $I_{t(i)} \subseteq [r_i, d_i]$ be the first interval in which i can be processed. Then the starting working set consists of the columns corresponding to the one-element sets $\{i\}$ $(i = 1,\dots,n)$ in the interval $I_{t(i)}$, the artificial variables u_t $(t = 1,\dots,\tau)$ and the slack variables according to (3.172) and (3.173).
- `InsertDeleteColumns`: We use a delete-strategy which never deletes the artificial variables and the slack variables according to (3.172), (3.173) from the working set.
- `CalculateColumns`: We have to calculate new entering columns or to show that no improving column exists. If we denote the dual variables corresponding to conditions (3.172) by y_i $(i = 1,\dots,n)$ and the dual variables belonging to (3.173) by w_t $(t = 1,\dots,\tau)$, then the constraints in the associated dual problem have the form:

$$\sum_{i=1}^{n} a_i^{jt} y_i - w_t \le 0 \quad (t = 1,\dots,\tau;\ j = 1,\dots,f_t) \tag{3.176}$$

$$w_t \le 1 \quad (t = 1,\dots,\tau) \tag{3.177}$$

$$y_i \ge 0 \quad (i = 1,\dots,n) \tag{3.178}$$

$$w_t \ge 0 \quad (t = 1,\dots,\tau) \tag{3.179}$$

  Since the artificial and slack variables are kept in the working set, conditions (3.177) to (3.179) are always satisfied. Thus, we have to find a column $a^{jt} \in \{0,1\}^n$ violating (3.176), i.e. satisfying

$$\sum_{i=1}^{n} a_i^{jt} y_i - w_t > 0$$

  for an index $t \in \{1,\dots,\tau\}$. This can be done by an enumeration procedure as in Section 3.7.3, where we restrict the search for improving columns to activities from the set A_t.

The LP-formulation may be strengthened by adding additional valid inequalities. In the following three different types of inequalities are described. Since these inequalities “cut off” solutions for the relaxation which are infeasible for the original problem, they are also called **cuts**.

Energetic cuts

For an interval $[a, b] \subseteq [0, T]$ and an activity i we may calculate a lower bound $P_i(a, b)$ for the amount of time where i has to be processed in $[a, b]$ in any feasible schedule. The value $P_i(a, b)$ is given by the minimum of

- the interval length $b - a$,
- $p_i^+ := \max\{0, p_i - \max\{0, a - r_i\}\}$, which equals the required processing time in $[a, d_i]$ if i is started at time r_i, and
- $p_i^- := \max\{0, p_i - \max\{0, d_i - b\}\}$, which equals the required processing time in $[r_i, b]$ if i is completed at time d_i.

Obviously, the minimum of these three values is a lower bound for the processing time of activity i in the interval $[a, b] \cap [r_i, d_i]$.

For the LP-formulation we may consider each interval $[z_{t_1}, z_{t_2}]$ with $0 \leq t_1 < t_2 \leq \tau$ and add the inequalities

$$\sum_{t=t_1+1}^{t_2} \sum_{j=1}^{f_t} a_i^{jt} x_{jt} \geq P_i(z_{t_1}, z_{t_2}) \quad (i = 1, \ldots, n;\ 0 \leq t_1 < t_2 \leq \tau). \tag{3.180}$$

Example 3.23 : Consider the example with $n = 2$ activities and a single resource with capacity $R_1 = 1$ shown in Figure 3.50. A feasible preemptive schedule for $T = 3$ respecting the time windows is also shown in the figure.

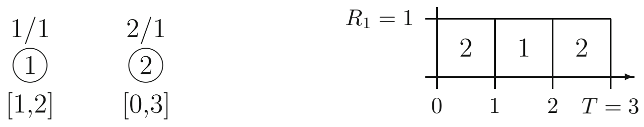


Figure 3.50: A feasible preemptive schedule for $T = 3$

If the inequalities (3.180) are added, this schedule is no longer feasible since for activity 2 in the interval $[1, 2]$ we have $P_2(1, 2) = 1$ but in the schedule no part of activity 2 is scheduled in $[1, 2]$. Thus, adding the inequalities (3.180) may lead to stronger lower bounds. $\square$

Non-preemptive cuts

Some other valid inequalities my be derived from the observation that in a non-preemptive schedule (where additionally all starting times are integral), an activity cannot be processed simultaneously in two intervals $[z_{t_\nu - 1}, z_{t_\nu}[$ $(\nu = 1, 2)$ which have a distance of at least $p_i - 1$ time units. For example, an activity i with processing time $p_i = 5$ cannot overlap with the intervals $[1, 4[$ and $[8, 10[$

simultaneously. Furthermore, we may state that activity i can be processed for at most $\max\{4-1, 10-8\} = 3$ time units in $[1, 4[\cup[8, 10[$.

More generally, for each activity i we consider subsets of the given intervals $I_t = [z_{t-1}, z_t[$ in which i can be processed and which have a distance of at least $p_i - 1$ time units. For activity i let $\Psi_i \subseteq \{1, \ldots, \tau\}$ be a subset of interval indices with $I_t = [z_{t-1}, z_t[\subseteq [r_i, d_i]$ for all $t \in \Psi_i$ and $z_{t'-1} - z_t \geq p_i - 1$ for all indices $t < t' \in \Psi_i$. Then we may state that i can be processed in at most one interval of Ψ_i, i.e. the maximal length of an interval in Ψ_i is an upper bound for the total processing time of i in all intervals belonging to Ψ_i.

Thus, we may add the inequalities

$$\sum_{t\in\Psi_i}\sum_{j=1}^{f_t} a_i^{jt} x_{jt} \leq \max_{t\in\Psi_i}\{z_t - z_{t-1}\} \quad (i = 1, \ldots, n;\ \Psi_i \subseteq \{1, \ldots, \tau\}). \tag{3.181}$$

There are many possible subsets $\Psi_i \subseteq \{1, \ldots, \tau\}$ for an activity i representing non-overlapping intervals with distance at least $p_i - 1$. For each activity i and each $t \in \{1, \ldots, \tau\}$ with $I_t \subseteq [r_i, d_i]$ we construct a set $\Psi_{i,t}$ containing t where I_t has the maximal length in $\Psi_{i,t}$, i.e. $z_t - z_{t-1} = \max\limits_{\psi\in\Psi_{i,t}}\{z_\psi - z_{\psi-1}\}$.

Such a set $\Psi_{i,t}$ may be constructed as follows.

1. Set $\Psi_{i,t} := \{t\}$.
2. Let ψ be the maximal index in the current set $\Psi_{i,t}$. Determine the smallest $\psi' > \psi$ with $z_{\psi'-1} - z_\psi \geq p_i - 1$ and $z_{\psi'} - z_{\psi'-1} \leq z_t - z_{t-1}$. If no such ψ' exists, go to Step 3. Otherwise, add ψ' to $\Psi_{i,t}$ and iterate Step 2.
3. Let ψ be the minimal index in the current set $\Psi_{i,t}$. Determine the largest $\psi' < \psi$ with $z_{\psi-1} - z_{\psi'} \geq p_i - 1$ and $z_{\psi'} - z_{\psi'-1} \leq z_t - z_{t-1}$. Add ψ' to $\Psi_{i,t}$ and iterate Step 3 until no further index ψ' exists.

Example 3.24: Consider one activity i with $p_i = 3$, $[r_i, d_i] = [0, 27]$ and time points $z_0 = 0$, $z_1 = 4$, $z_2 = 6$, $z_3 = 8$, $z_4 = 11$, $z_5 = 14$, $z_6 = 16$, $z_7 = 20$, $z_8 = 23$, $z_9 = 25$, $z_{10} = 27$ (cf. also Figure 3.51).

For $t = 5$ the construction of $\Psi_{i,5}$ proceeds as follows.

- Step 1: We set $\Psi_{i,5} := \{5\}$ and have $z_t - z_{t-1} = z_5 - z_4 = 14 - 11 = 3$.
- Step 2: Starting with the maximal index $\psi = 5$, the first $\psi' > \psi = 5$ with the required properties is $\psi' = 8$ (for $\psi' = 6$ we have $z_{\psi'-1} - z_\psi = z_5 - z_5 = 0 < 2 = p_i - 1$, for $\psi' = 7$ we have $z_{\psi'} - z_{\psi'-1} = z_7 - z_6 = 4 > 3 = z_t - z_{t-1}$, and for $\psi' = 8$ we have $z_{\psi'-1} - z_\psi = z_7 - z_5 = 6 \geq 2 = p_i - 1$ and $z_{\psi'} - z_{\psi'-1} = z_8 - z_7 = 3 \leq 3 = z_t - z_{t-1}$). After adding $\psi' = 8$ to $\Psi_{i,5}$ the new maximal index is $\psi = 8$. Since for $\psi' = 10$ we get $z_{\psi'-1} - z_\psi = z_9 - z_8 = 2 \geq 2 = p_i - 1$ and $z_{\psi'} - z_{\psi'-1} = z_{10} - z_9 = 2 \leq 3 = z_t - z_{t-1}$, we add $\psi' = 10$ to $\Psi_{i,5}$.

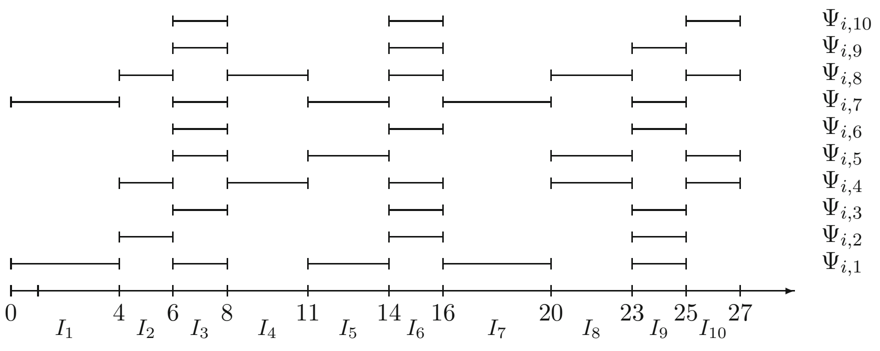


Figure 3.51: Construction of the sets $\Psi_{i,t}$

- Step 3: Starting with the minimal index $\psi = 5$, the first $\psi' < \psi = 5$ with the required properties is $\psi' = 3$ (for $\psi' = 4$ we have $z_{\psi-1} - z_{\psi'} = z_4 - z_4 = 0 < 2 = p_i - 1$, and for $\psi' = 3$ we have $z_{\psi-1} - z_{\psi'} = z_4 - z_3 = 3 \geq 2 = p_i - 1$ and $z_{\psi'} - z_{\psi'-1} = z_3 - z_2 = 2 \leq 3 = z_t - z_{t-1}$). After adding $\psi' = 3$ to $\Psi_{i,5}$ the new minimal index is $\psi = 3$. Since for $\psi' = 2$ we have $z_{\psi-1} - z_{\psi'} = 0 < 2 = p_i - 1$ and for $\psi' = 1$ we have $z_{\psi'} - z_{\psi'-1} = 4$, no further suitable index ψ' exists. Thus, the final set is $\Psi_{i,5} = \{3, 5, 8, 10\}$.

The other sets $\Psi_{i,t}$ are constructed similarly (cf. Figure 3.51). As a result we get $\Psi_{i,1} = \Psi_{i,7} = \{1, 3, 5, 7, 9\}$, $\Psi_{i,2} = \{2, 6, 9\}$, $\Psi_{i,3} = \Psi_{i,6} = \Psi_{i,9} = \{3, 6, 9\}$, $\Psi_{i,4} = \Psi_{i,8} = \{2, 4, 6, 8, 10\}$, $\Psi_{i,10} = \{3, 6, 10\}$. □

Precedence cuts

Finally, the third type of inequalities tries to take into account the precedence constraints a little bit more. For each activity $i = 1, \ldots, n$ we introduce an additional variable M_i representing the **midpoint** of activity i (i.e. the average of its starting and its completion time, $M_i = \frac{S_i + C_i}{2}$). Then the precedence relations $i \to j \in C$ may be expressed as $S_j \geq C_i$, which implies $C_j - C_i \geq p_j$ and $S_j - S_i \geq p_i$. By adding the last two inequalities and dividing the result by 2 we get

$$M_j - M_i \geq \frac{p_j + p_i}{2} \quad \text{for all } i \to j \in C. \tag{3.182}$$

For a given schedule if activity i is processed in interval I_t, let S_{it} be the starting time, C_{it} be the completion time and $p_{it} := C_{it} - S_{it} > 0$ be the processing time of the part of i processed in interval I_t. In a non-preemptive schedule each activity i starts in some interval I_{t_1}, is continuously processed in some intervals I_t and completes in some interval I_{t_2} with $t_1 \leq t \leq t_2$. Thus, we have

$$S_i = S_{it_1} \geq z_{t_1-1}, \; C_i = C_{it_2} \leq z_{t_2} \text{ and } z_t = C_{it} = S_{i,t+1} \text{ for } t_1 \leq t \leq t_2 - 1.$$

This implies

$$\begin{aligned} M_i\, p_i &= \frac{C_i + S_i}{2}(C_i - S_i) = \frac{C_i^2 - S_i^2}{2} = \sum_{t=t_1}^{t_2}(C_{it}^2 - S_{it}^2)/2 \\ &= \sum_{t=t_1}^{t_2}(C_{it} - S_{it})(C_{it} + S_{it})/2 = \sum_{t=t_1}^{t_2} p_{it} M_{it} = \sum_{t=1}^{\tau} p_{it} M_{it}, \end{aligned}$$

where $M_{it} := (C_{it} + S_{it})/2$ defines the midpoint of i in the interval I_t.

For each interval I_t with $p_{it} > 0$ we have $p_{it} \geq 1$, i.e.

$$M_{it} = S_{it} + \frac{p_{it}}{2} \geq z_{t-1} + \frac{1}{2} \;\text{ and }\; M_{it} = C_{it} - \frac{p_{it}}{2} \leq z_t - \frac{1}{2}.$$

This implies

$$\sum_{t=1}^{\tau}(z_{t-1} + \frac{1}{2})\, p_{it} \leq \sum_{t=1}^{\tau} M_{it} p_{it} \leq \sum_{t=1}^{\tau}(z_t - \frac{1}{2})\, p_{it}.$$

Using the variables x_{jt}, the processing time of i in I_t my be expressed as $p_{it} := \sum_{j=1}^{f_t} a_i^{jt} x_{jt}$. Thus, the midpoint variables M_i may be linked to the x_{jt}-variables by the additional inequalities

$$\sum_{t=1}^{\tau}(z_{t-1} + \frac{1}{2}) \sum_{j=1}^{f_t} a_i^{jt} x_{jt} \leq M_i p_i \leq \sum_{t=1}^{\tau}(z_t - \frac{1}{2}) \sum_{j=1}^{f_t} a_i^{jt} x_{jt} \quad (i = 1, \ldots, n). \tag{3.183}$$

If these inequalities as well as conditions (3.182) are added to the LP, additional (infeasible) solutions are excluded, i.e. the lower bound is strengthened.

Example 3.25 : Consider again Example 3.22. The schedule shown in Figure 3.49 is no longer feasible if the inequalities (3.182) and (3.183) for the precedence relation $2 \to 4$ are added to the LP. For this conjunction (3.182) becomes $M_4 - M_2 \geq \frac{p_4 + p_2}{2} = 1.5$. In the schedule activity 2 is processed in the intervals $[0, 1]$ and $[3, 4]$. Thus, (3.183) for $i = 2$ reads

$$4 = (0 + \frac{1}{2}) + (3 + \frac{1}{2}) \leq M_2\, p_2 \leq (1 - \frac{1}{2}) + (4 - \frac{1}{2}) = 4,$$

which implies $M_2 = 2$. Furthermore, activity 4 is processed in the interval $[2, 3]$, i.e. (3.183) for $i = 4$ reads

$$2.5 = 2 + \frac{1}{2} \leq M_4\, p_4 \leq 3 - \frac{1}{2} = 2.5$$

implying $M_4 = 2.5$. But then $M_4 - M_2 = 0.5 < 1.5$, i.e. (3.182) is violated for the conjunction $2 \to 4$. $\square$

```
Algorithm Destructive LB (incremental search)
 1.  Calculate an upper bound $UB$ and set $T := UB - 1$;
 2.  infeasible := ConstraintPropagation($T$);
 3.  WHILE infeasible = FALSE DO
 4.         infeasible := LP($T$);
 5.         IF infeasible = FALSE THEN
 6.            $T := T - 1$;
 7.            infeasible := ConstraintPropagation($T$);
 8.         ENDIF
 9.  ENDWHILE
10.  $LB := T + 1$;
```

Figure 3.52: Incremental destructive lower bound procedure

To finish this subsection, the previously described methods are summarized in an incremental destructive lower bound procedure combining constraint propagation techniques and linear programming (cf. Figure 3.52). In Steps 2 and 7 the subprocedure `ConstraintPropagation`(T) is called, which applies certain reduction techniques to the restricted problem where $C_{\max} \leq T$ is imposed. It may find additional constraints which strengthen the problem data (i.e. additional precedence relations and reduced time windows) before the LP is solved. Furthermore, it returns a boolean variable `infeasible` which is `TRUE` if infeasibility could be detected for the threshold value T. If infeasibility could not be proved by constraint propagation (i.e. `ConstraintPropagation`(T) returns `FALSE`), in Step 4 another feasibility check is performed by solving the LP.
Since in each step of the procedure T is reduced by one, the first value T for which the boolean flag `infeasible` equals `TRUE` corresponds to the largest infeasible T, i.e. $LB := T + 1$ is the best possible lower bound value calculated in this way.

```
Algorithm Destructive LB (binary search)
 1.  Calculate an upper bound $U$ and a lower bound $L$;
 2.  WHILE $L < U$ DO
 3.         $T := \lfloor \frac{U+L}{2} \rfloor$;
 4.         infeasible := CheckInfeasibility($T$);
 5.         IF infeasible = TRUE THEN
 6.            $L := T + 1$;
 7.         ELSE
 8.            $U := T$;
 9.  ENDWHILE
```

Figure 3.53: Binary search destructive lower bound procedure

As already mentioned at the beginning of Section 3.7, a binary search procedure is more efficient. Such a procedure can be found in Figure 3.53, where the procedure `CheckInfeasibility`(T) does all infeasibility tests for a threshold value T. Iteratively we consider an interval $[L, U]$ with $L < U$ in which we search for a valid lower bound value. If we can prove that no feasible solution with an objective value smaller than or equal to $T = \lfloor \frac{U+L}{2} \rfloor$ exists, we increase the lower bound L to the value $T+1$ and repeat the procedure with the interval $[T+1, U]$. Otherwise, we replace U by T and repeat the procedure in the interval $[L, T]$. As soon as $L \geq U$ holds, the procedure terminates. Then L equals the largest lower bound value which can be calculated in this way.

3.7.5 A destructive method for the multi-mode case

In this subsection we describe how the linear programming formulation from the previous subsection can be extended to the multi-mode case. In the corresponding preemptive relaxation we also allow that an activity may be processed in different modes. As described in Section 3.6.6, for each activity i and each mode $m \in \mathcal{M}_i$ we have a mode-dependent head r_i^m and a deadline $d_i^m := T - q_i^m + p_{im}$. Furthermore, mode-dependent time-lags $d_{ij}^{m\mu}$ may be given.

The time windows $[r_i^m, d_i^m]$ for activity-mode combinations (i, m) are first sharpened by constraint propagation as described in Section 3.6.6 and then used in the LP.

In a preprocessing step we eliminate all time windows $[r_i^m, d_i^m]$ with $d_i^m - r_i^m < p_{im}$ which are too small to process p_{im} time units in it. Let $z_0 < z_1 < \ldots < z_\tau$ be the ordered sequence of all other different r_i^m- and d_i^m-values with $d_i^m - r_i^m \geq p_{im}$. For $t = 1, \ldots, \tau$ we consider the intervals $I_t := [z_{t-1}, z_t]$ of length $z_t - z_{t-1}$. With each interval I_t we associate a set A_t of feasible activity-mode combinations (i, m) with $i = 1, \ldots, n$ and $m \in \mathcal{M}_i$ which can partially be executed in this interval, i.e. with $r_i^m \leq z_{t-1}$ and $z_t \leq d_i^m$.

A subset X of activity-mode combinations $(i, m) \in A_t$ is called **feasible** for the interval I_t if

- $(i, m) \in X$ implies $(i, m') \notin X$ for all $m' \in \mathcal{M}_i$ with $m' \neq m$, i.e. activity i is not processed in two different modes in X,
- for all pairs $(i, m), (j, \mu) \in X$ no precedence relation exists between activity i and activity j (or $d_{ij}^{m\mu} < p_{im}$ in the case that additionally mode-dependent time-lags are given),
- the renewable resource constraints are respected for the activities in X, i.e. we have
$$\sum_{(i,m) \in X} r_{ikm}^{\rho} \leq R_k^{\rho} \text{ for all } k \in \mathcal{K}^{\rho},$$

- the non-renewable resource constraints are respected for the activities in X if the remaining activities not contained in X only need their minimal resource requirements, i.e. we have

$$\sum_{(i,m)\in X} r^{\nu}_{ikm} + \sum_{j\in\overline{X}} \min_{\mu\in\mathcal{M}_j} \{r^{\nu}_{jk\mu}\} \leq R^{\nu}_k \text{ for all } k \in \mathcal{K}^{\nu}$$

  where $\overline{X} := \{\, j \mid (j,\mu) \notin X \text{ for all } \mu \in \mathcal{M}_j\}$ denotes the set of activities which are not contained in X.

Let X_{lt} for $l = 1,\dots,f_t$ be all these feasible subsets for the interval I_t and denote the set of all feasible subsets in I_t by F_t. With each set X_{lt} we associate an incidence vector a^{lt} defined by

$$a^{lt}_{im} = \begin{cases} 1, & \text{if } (i,m) \in X_{lt} \\ 0, & \text{otherwise} \end{cases} \quad \text{for activities } i = 1,\dots,n \text{ and modes } m \in \mathcal{M}_i.$$

Let ξ_{im} $(i = 1,\dots,n; m \in \mathcal{M}_i)$ be a 0-1 variable with

$$\xi_{im} = \begin{cases} 1, & \text{if activity } i \text{ is processed in mode } m \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, let x_{lt} be a variable denoting the total number of time units when all activity-mode combinations in X_{lt} are processed. Then the preemptive feasibility problem may be written as follows:

$$\sum_{m\in\mathcal{M}_i} \xi_{im} = 1 \qquad (i = 1,\dots,n) \tag{3.184}$$

$$\sum_{i=1}^{n}\sum_{m\in\mathcal{M}_i} r^{\nu}_{ikm}\xi_{im} \leq R^{\nu}_k \qquad (k \in \mathcal{K}^{\nu}) \tag{3.185}$$

$$\sum_{t=1}^{\tau}\sum_{l=1}^{f_t} a^{lt}_{im} x_{lt} \geq p_{im}\xi_{im} \qquad (i = 1,\dots,n;\ m \in \mathcal{M}_i) \tag{3.186}$$

$$\sum_{l=1}^{f_t} x_{lt} \leq z_t - z_{t-1} \qquad (t = 1,\dots,\tau) \tag{3.187}$$

$$x_{lt} \geq 0 \qquad (t = 1,\dots,\tau;\ l = 1,\dots,f_t) \tag{3.188}$$

$$\xi_{im} \in \{0,1\} \qquad (i = 1,\dots,n;\ m \in \mathcal{M}_i) \tag{3.189}$$

Conditions (3.184) ensure that each activity i is assigned to exactly one mode $m \in \mathcal{M}_i$ and conditions (3.185) take care of the non-renewable resource constraints. In (3.186) the variables ξ_{im} are linked with the x_{lt}-variables by requiring that all activities i are processed in their assigned modes m for at least p_{im} time units. Finally, conditions (3.187) ensure that the number of time units scheduled in interval I_t does not exceed the length $z_t - z_{t-1}$ of this interval.

Since the integrality constraints (3.189) make the problem hard, we relax them and replace them by $\xi_{im} \geq 0$ (i.e. in this relaxation an activity may be processed in different modes). Furthermore, we introduce artificial variables v_k for $k \in \mathcal{K}^\nu$ in conditions (3.185) which allow that the non-renewable resource contraints may be violated. We also introduce artificial variables u_t for $t = 1, \ldots, \tau$ in conditions (3.187) which allow that the capacities of the time intervals I_t may be exceeded. Then the feasibility problem can be formulated as the following linear program:

$$\min \quad \sum_{k \in \mathcal{K}^\nu} v_k + \sum_{t=1}^{\tau} u_t \tag{3.190}$$

$$\text{s.t.} \quad \sum_{m \in \mathcal{M}_i} \xi_{im} = 1 \qquad (i = 1, \ldots, n) \tag{3.191}$$

$$-\sum_{i=1}^{n} \sum_{m \in \mathcal{M}_i} r_{ikm}^\nu \xi_{im} + v_k \geq -R_k^\nu \qquad (k \in \mathcal{K}^\nu) \tag{3.192}$$

$$\sum_{t=1}^{\tau} \sum_{l=1}^{f_t} a_{im}^{lt} x_{lt} - p_{im}\xi_{im} \geq 0 \qquad (i = 1, \ldots, n;\ m \in \mathcal{M}_i) \tag{3.193}$$

$$-\sum_{l=1}^{f_t} x_{lt} + u_t \geq -z_t + z_{t-1} \qquad (t = 1, \ldots, \tau) \tag{3.194}$$

$$x_{lt} \geq 0 \qquad (t = 1, \ldots, \tau;\ l = 1, \ldots, f_t) \tag{3.195}$$

$$\xi_{im} \geq 0 \qquad (i = 1, \ldots, n;\ m \in \mathcal{M}_i) \tag{3.196}$$

$$v_k \geq 0 \qquad (k \in \mathcal{K}^\nu) \tag{3.197}$$

$$u_t \geq 0 \qquad (t = 1, \ldots, \tau) \tag{3.198}$$

A solution exists for the continuous relaxation of the preemptive feasibility problem if and only if the linear program (3.190) to (3.198) has the optimal solution value zero, i.e. if all values of the artifical variables become zero.

Example 3.26 : Consider again Example 3.17. We assume that no renewable resources but one non-renewable resource k with $R_k^\nu = 5$ is given and that $r_{ikm}^\nu = m$ for all activity-mode combinations (i, m) holds. The longest path length $d_{04} = r_4 = 10$ (after improving heads and tails) provides a first simple lower bound for the given instance. For the threshold value $T = 11$ we get the following time windows $[r_i^m, d_i^m]$ with deadlines $d_i^m = T - q_i^m + p_{im}$:

$$r_1^1 = 0, \quad r_1^2 = 0, \quad r_2^1 = 4, \quad r_2^2 = 5, \quad r_3^1 = 6, \quad r_3^2 = 7,$$
$$d_1^1 = 4, \quad d_1^2 = 2, \quad d_2^1 = 5, \quad d_2^2 = 9, \quad d_3^1 = 11, \quad d_3^2 = 11,$$

which lead to the feasible activity-mode combinations A_t in the time intervals I_t for $t = 1, \ldots, \tau = 7$ as shown in Figure 3.54.

Since a feasible subset X cannot contain the combinations $(1, 1)$ and $(1, 2)$ or $(3, 1)$ and $(3, 2)$ simultaneously (because an activity may not be processed in two different modes in X), we obtain 2+1+1+1+3+5+2=15 feasible subsets X_{lt}.

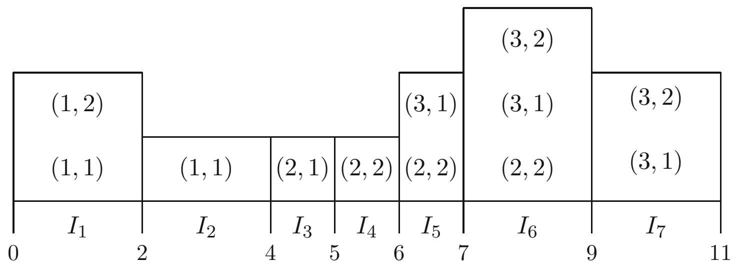


Figure 3.54: Feasible activity-mode combinations A_t in the intervals I_t

For example, in the interval $I_6 = [7, 9]$ we have 5 feasible subsets: $\{(2, 2), (3, 1)\}$, $\{(2, 2), (3, 2)\}$, $\{(2, 2)\}$, $\{(3, 1)\}$ and $\{(3, 2)\}$. The subset $\{(2, 2), (3, 1)\}$ is feasible since $d_{23}^{21} = 1 < p_{22} = 3$ (i.e. no precedence relation $(2, 2) \rightarrow (3, 1)$ is induced by the time-lags) and $r_{2k2}^{\nu} + r_{3k1}^{\nu} + \min\limits_{m \in \mathcal{M}_1} \{r_{1km}^{\nu}\} = 2 + 1 + 1 = 4 \leq R_k^{\nu} = 5$ holds.

The LP-formulation (3.184) to (3.189) contains 6 variables ξ_{im} $(i = 1, 2, 3; m = 1, 2)$ and 15 variables x_{lt} corresponding to the feasible subsets X_{lt}. It is easy to check that we get a feasible solution for (3.184) to (3.189) by setting $\xi_{12} = \xi_{21} = \xi_{32} = 1$ and processing the activity-mode combination $(1, 2)$ in $[0, 2]$ for one time unit, the combination $(2, 1)$ in $[4, 5]$ for one time unit and the combination $(3, 2)$ in $[7, 9]$ for two and in $[9, 11]$ for one time unit.

For the threshold value $\tilde{T} = 10$ we get the deadlines $\tilde{d}_i^m = d_i^m - 1$, i.e.

$$\tilde{d}_1^1 = 3, \tilde{d}_1^2 = 1, \tilde{d}_2^1 = 4, \tilde{d}_2^2 = 8, \tilde{d}_3^1 = 10, \tilde{d}_3^2 = 10.$$

Due to $\tilde{d}_1^1 - r_1^1 = 3 < p_{11} = 4$, $\tilde{d}_2^1 - r_2^1 = 0 < p_{21} = 1$ and $\tilde{d}_3^1 - r_3^1 = 4 < p_{31} = 5$ only the activity-mode combinations $(1, 2), (2, 2), (3, 2)$ remain feasible. But due to $r_{1k2}^{\nu} + r_{2k2}^{\nu} + r_{3k2}^{\nu} = 6 > R_k^{\nu} = 5$, no feasible mode assignment satisfying (3.185) exists. Thus, we may state infeasibility for $\tilde{T} = 10$, which leads to the improved lower bound $\tilde{T} + 1 = 11$. □

Again the linear program can be solved by column generation techniques. Since in general the number of columns corresponding to the variables ξ_{im} is not very large, all these columns may be generated at the beginning and kept in the working set during the whole procedure. Thus, only columns corresponding to the variables x_{lt} have to be generated by column generation techniques.

We solve the linear program with a two-phase method. At first we try to get a feasible fractional mode assignment (i.e. variables $0 \leq \xi_{im} \leq 1$ satisfying (3.191) and (3.192) with $v_k = 0$ for all $k \in \mathcal{K}^{\nu}$). If in the set of all variables ξ_{im} no such feasible fractional mode assignment exists, we can state infeasibility. Otherwise, we proceed with the second phase in which we perform column generation for the variables x_{lt} and try to get rid of the artificial variables u_t.

In the following we describe how the procedures from the previous subsection have to be modified:

`Initialize`: In order to get a feasible basic starting solution we determine for each activity $i = 1, \ldots, n$ an index $t(i)$ of an interval $I_{t(i)}$ with $I_{t(i)} \subseteq [r_i, d_i]$, where again $r_i := \min\limits_{m \in \mathcal{M}_i} r_i^m$ and $d_i := \max\limits_{m \in \mathcal{M}_i} d_i^m$. Besides all columns corresponding to the variables ξ_{im}, all artifical variables u_t, and all slack variables according to (3.192), (3.193) and (3.194) we add all columns corresponding to the one-element sets $\{(i, m)\}$ with $m \in \mathcal{M}_i$ to the initial working set of columns. Note that modes $m \in \mathcal{M}_i$ with $r_i^m > z_{t(i)-1}$ or $d_i^m < z_{t(i)}$ may not be executed in $I_{t(i)}$. For this reason, variables corresponding to infeasible activity-mode combinations $(i, m) \notin A_{t(i)}$ are treated as additional artificial variables and penalized with a positive coefficient in the objective function (3.190). Then in a basic starting solution we may process activity i in the interval $I_{t(i)}$ according to the fractional mode assignment ξ for $\max\limits_{m \in \mathcal{M}_i} \{p_{im}\xi_{im}\}$ time units. If these processing times exceed the capacities of the intervals I_t, this is compensated by the artificial variables u_t in (3.194).

`InsertDeleteColumns`: We work iteratively with a restricted working set of columns which contains a basis, the columns corresponding to all variables ξ_{im}, all artifical variables u_t, and all slack variables according to (3.192), (3.193) and (3.194). We again use a delete-strategy which never deletes slack variables from the working set. Thus, only columns corresponding to the variables x_{lt} have to be calculated by column generation techniques.

`CalculateColumns`: We have to calculate new entering columns a^{lt} or to show that no improving column exists. If we denote the dual variables corresponding to conditions (3.193) by y_{im} $(i = 1, \ldots, n;\ m \in \mathcal{M}_i)$ and the dual variables belonging to (3.194) by w_t $(t = 1, \ldots, \tau)$, then the dual constraints corresponding to the variables x_{lt} have the form

$$\sum_{i=1}^{n} \sum_{m \in \mathcal{M}_i} a_{im}^{lt} y_{im} - w_t \leq 0 \qquad (t = 1, \ldots, \tau;\ l = 1, \ldots, f_t).$$

Thus, in order to find entering columns we have to determine columns a^{lt} with

$$\sum_{i=1}^{n} \sum_{m \in \mathcal{M}_i} a_{im}^{lt} y_{im} > w_t \tag{3.199}$$

for an index $t \in \{1, \ldots, \tau\}$. Feasible columns satisfying (3.199) are calculated by scanning iteratively through the intervals I_t $(t = 1, \ldots, \tau)$ and searching in each interval I_t for sets in F_t which correspond to improving columns. This can be done by an enumeration procedure as in Section 3.7.4, where the nodes of the enumeration tree correspond to feasible activity-mode combination $(i, m_i) \in A_t$.

3.7.6 Reference notes

The lower bound LB_S was proposed by Stinson et al. [182], the LP-based constructive lower bound by Mingozzi et al. [145]. To calculate the LP lower bound values, in [145] the dual linear program was considered and the corresponding weighted node packing problem was solved heuristically (which gives weaker bounds than the original LP formulation). The column generation procedure for the constructive LP bound is due to Baar et al. [8]. Brucker and Knust [34] strengthened the LP by taking into account time windows and solved the resulting LP by column generation in a destructive approach combined with constraint propagation. This approach was later improved by Baptiste and Demassey [11] who considered further constraint propagation techniques and introduced additional valid inequalities in the LP formulation. Also in Demassey et al. [53] constraint propagation and linear programming were combined in order to calculate lower bounds. The extension of the destructive linear programming lower bound to the situation with multiple modes is due to Brucker and Knust [37].

Other constructive and destructive lower bounds for the RCPSP are described in Klein and Scholl [114] who compared many bounds in computational experiments. A recent survey on lower bounds for the RCPSP can also be found in Néron et al. [150]. Fisher [75] was the first who derived a lower bound based on Lagrangian relaxation of the resource constraints. A similar bound was introduced by Christofides et al. [44]. In this paper also two other lower bounds were proposed: the first is based on the LP relaxation of the DDT-formulation with several additional valid inequalities to strengthen the LP relaxation. The second is based on a disjunctive graph by adding disjunctive arcs (representing resource conflicts) to the precedence constraints of the activity-on-node network. Improved lower bound calculations based on Lagrangian relaxation can be found in Möhring et al. [148].

The idea of Stinson et al. was improved by Demeulemeester [54] who augmented a critical path by a second node-disjoint path and solved the resulting two-path relaxation by dynamic programming.

3.8 Branch-and-Bound Algorithms

In this section we will discuss different exact methods to solve the RCPSP with constant resource profiles. They are all based on the general branch-and-bound principle, i.e. they partition the problem into subproblems and try to eliminate subproblems by calculating lower bounds or using dominance rules.

A branch-and-bound algorithm based on so-called precedence trees is presented in Section 3.8.1. In Sections 3.8.2 and 3.8.3 algorithms based on extension and delaying alternatives can be found. In Section 3.8.4 so-called schedule schemes are introduced, in Section 3.8.5 briefly the multi-mode case is reviewed.

3.8.1 An algorithm based on precedence trees

In this subsection we present a branch-and-bound algorithm which is based on the list scheduling techniques introduced in Section 3.2.1. The nodes of the enumeration tree correspond to precedence-feasible subsequences of activities which are extended by an unscheduled activity in each branching step. Associated with each node is the earliest start schedule for the corresponding subsequence in which all activities are scheduled according to the subsequence and no activity i is started before the starting time of the previously scheduled activity (i.e. for a list $(i_1, \ldots, i_n)$ we have $S_{i_1} \leq S_{i_2} \leq \ldots \leq S_{i_n}$). As mentioned in Section 3.2.1, for a regular objective function always an optimal schedule exists in the set of such schedules. Since in the enumeration tree all sequences which are compatible with the precedence relations are enumerated, the resulting tree is also called **precedence tree**. This tree has $n + 2$ levels because in each stage one activity is scheduled (including the dummy activities 0 and $n + 1$) and the leafs correspond to complete schedules.

With each node of the enumeration tree in stage $\lambda \in \{1, \ldots, n + 1\}$ two disjoint activity sets are associated: the set of scheduled activities F_λ and the set E_λ of all eligible activities (i.e. all unscheduled activities for which all predecessors are scheduled). Then an eligible activity $j \in E_\lambda$ is chosen and scheduled at the earliest precedence- and resource-feasible time t which is not smaller than the starting time of the previously scheduled activity in stage $\lambda - 1$. Afterwards, the resource profiles of the partial schedule are updated and the set of eligible activities $E_{\lambda+1}$ is determined. Then the next activity is chosen in stage $\lambda + 1$.

If stage $\lambda = n + 1$ is reached, all activities are scheduled and the makespan of the corresponding complete schedule is compared to the best upper bound value found so far. Afterwards, backtracking to the previous stage occurs where the next unconsidered eligible activity is chosen. If all eligible activities have been considered as branching candidates in a stage, a further backtracking step to the previous stage is performed.

Example 3.27 : Consider the instance with $n = 4$ activities and one resource with capacity $R_1 = 3$ shown in Figure 3.55. Due to the precedence relation

$2 \rightarrow 4$ only sequences in which 2 is placed before 4 are feasible. The resulting precedence tree can be found in Figure 3.56, where the level of the dummy activity $n + 1 = 5$ is not drawn. In the leafs of this tree the six different schedules $S_1, \ldots, S_6$ shown in Figure 3.57 are obtained.

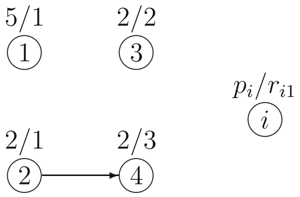


Figure 3.55: An instance with $n = 4$ activities

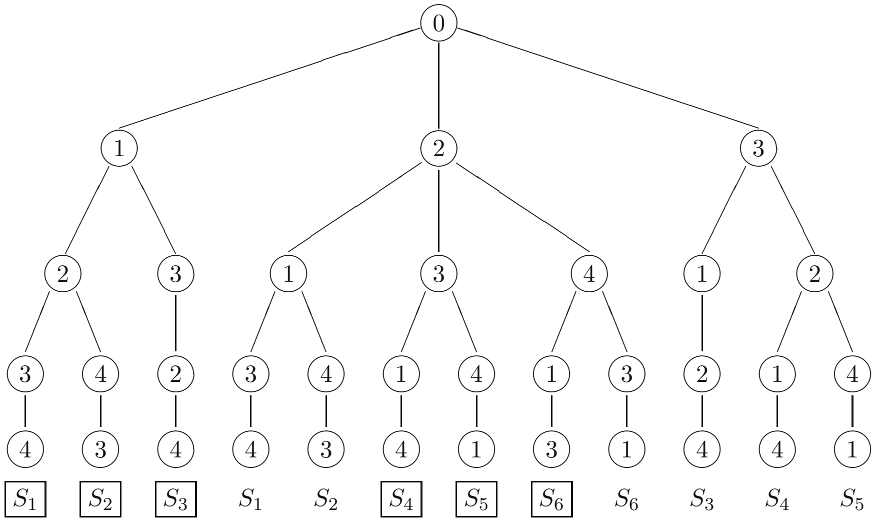


Figure 3.56: Precedence tree for the instance in Example 3.27

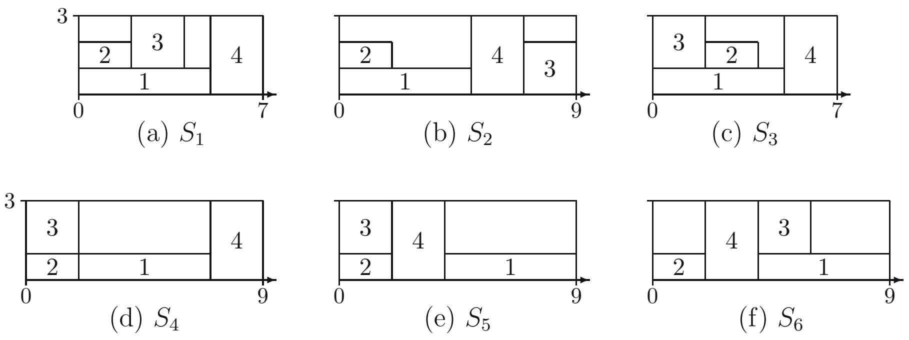


Figure 3.57: Schedules obtained by the precedence tree algorithm

□

The preceding discussions are summarized in the algorithm shown in Figure 3.58.

```
Algorithm Branch-and-Bound Precedence Tree
 1.  Calculate an initial upper bound $UB$ for the instance;
 2.  $S_0 := 0$;  $\lambda := 1$;  $F_1 := \{0\}$;  $s_0 := 0$;
 3.  Let $E_1$ be the set of all activities without predecessor;
 4.  B&B $(\lambda, F_1, E_1, S, s_0)$;

Procedure B&B $(\lambda, F_\lambda, E_\lambda, S, s_{\lambda-1})$
 1.  $B_\lambda := E_\lambda$;
 2.  WHILE $B_\lambda \neq \emptyset$ DO
 3.     Choose an activity $j \in B_\lambda$;   $B_\lambda := B_\lambda \setminus \{j\}$;
 4.     Calculate the smallest time $t \geq s_{\lambda-1}$ such that $j$
        can be scheduled in the interval $[t, t+p_j[$ without
        violating resource or precedence constraints;
 5.     Schedule $j$ in the interval $[t, t+p_j[$ and set $S_j := t$;
 6.     $F_\lambda := F_\lambda \cup \{j\}$;   $E_\lambda := E_\lambda \setminus \{j\}$;   $s_\lambda := t$;
 7.     IF $\lambda = n+1$ THEN
 8.        $UB := \min\{UB, S_{n+1}\}$; unschedule $n+1$;
 9.        RETURN
10.     ELSE
11.        Let $E_{\lambda+1} := E_\lambda$ and add to $E_{\lambda+1}$ all successors
           $i \notin E_\lambda$ of $j$ for which all predecessors are in $F_\lambda$;
12.        Calculate a lower bound $LB(S)$ for all extensions
           of the current partial schedule $S$;
13.        IF $LB(S) < UB$ THEN
14.           B&B $(\lambda+1, F_\lambda, E_{\lambda+1}, S, s_\lambda)$;
15.        Unschedule $j$ and set $F_\lambda := F_\lambda \setminus \{j\}$;   $E_\lambda := E_\lambda \cup \{j\}$;
16.     ENDIF
17.  ENDWHILE
```

Figure 3.58: A branch-and-bound algorithm based on precedence trees

In this algorithm the recursive procedure B&B $(\lambda, F_\lambda, E_\lambda, S, s_{\lambda-1})$ is called, where λ denotes the current stage, and F_λ, E_λ are the sets of scheduled and eligible activities, respectively. The set B_λ contains all activities which are branching candidates in the current stage. S is the starting time vector of the current partial schedule which is successively filled by setting one component S_j to some time period t in Step 4 and by removing the S_j-value again in Step 7 and 15 during backtracking. Initially, we set $S_0 := 0$. Finally, $s_{\lambda-1}$ denotes the starting time of the previously scheduled activity in stage $\lambda - 1$.

In Step 12 additionally a lower bound $LB(S)$ for the makespan of all extensions of the current partial schedule S is calculated. If this value is not smaller than the best makespan UB found so far, the corresponding branch in the tree does not have to be inspected and backtracking can be performed. Such a lower bound may for example be obtained by adding for the currently scheduled activity j its processing time p_j and its tail q_j to its starting time S_j, i.e. $LB(S) = S_j + p_j + q_j$. Better lower bounds may additionally take into account unscheduled activities. Since each unscheduled activity $i \notin F_\lambda$ cannot start before time S_j, it must satisfy $S_i \geq S_j$, i.e. a simple lower bound is $LB(S) = \max\limits_{i \notin F_\lambda} \{S_j + p_i + q_i\}$. More complicated lower bounds may be obtained by calculating lower bounds for the whole set of unscheduled activities (e.g. LB_S).

As observed in Example 3.3 in Section 3.2.1, the list scheduling procedure may generate the same earliest start schedule for different lists (cf. also the precedence tree in Figure 3.56 where only six different schedules are generated in the 12 leafs of the tree). Thus, in order to reduce the size of the enumeration tree, it is desirable to eliminate sequences leading to the same earliest start schedule. The following dominance rule, which can be applied for regular objective functions, ensures that all generated schedules are different.

Swapping rule: If two activities j, i are started at the same time, they can be swapped in the sequence without resulting in a worse schedule. Thus, if we have considered all extensions of $\ldots, i, j$ we do not have to consider the extensions of $\ldots, j, i$. Hence, it is sufficient to consider only extensions $\ldots, i, j$ with $i < j$.

If the swapping rule is applied in the precedence tree algorithm for Example 3.27, the sequences $(2, 1, ., .), (3, 1, ., .), (3, 2, ., .)$ and $(2, 4, 3, 1)$ are eliminated. Then, all generated schedules in the six remaining leafs are different.

In Theorem 3.1 in Section 3.1.2 we proved that for a regular objective function always an optimal active schedule exists. Thus, additionally the following **local left shift rule** can be applied: If an activity which is scheduled in the current stage of the branch-and-bound algorithm can be locally shifted to the left, the corresponding partial schedule does not have to be completed. This rule may be extended to the **global left shift rule**, where additionally global left shifts of the current activity are considered. While using the first rule the search space is reduced to the set of semi-active schedules, with the second rule only active schedules are enumerated. If the global left shift rule is applied in the precedence tree algorithm for Example 3.27, the schedules S_2 and S_6 are eliminated (since activity 3 can be globally shifted to the left). On the other hand, the local left shift rule is not applicable in this example.

If dominance rules are integrated into the algorithm, additionally the lower bound values may be improved. For example, if the swapping rule is applied, we know that each unscheduled activity $i \notin F_\lambda$ with a smaller number than the currently scheduled activity j cannot start before time $S_j + 1$, i.e. we may set

$$LB(S) = \max_{\{i \notin F_\lambda, i < j\}} \{S_j + 1 + p_i + q_i\}.$$

Since all additional dominance criteria need more computation time, in general a tradeoff between their effort and their effectiveness has to be found.

3.8.2 An algorithm based on extension alternatives

In this subsection we present a branch-and-bound algorithm which is based on a different idea. Contrary to the activity-oriented precedence tree algorithm, in this algorithm time points are enumerated at which activities may be started (like in the parallel schedule generation scheme, cf. Section 3.2.1). The nodes of the enumeration tree correspond to partial schedules which are precedence- and resource-feasible. In each branching step a partial schedule is extended by enumerating different subsets of unscheduled activities which can be scheduled at the current decision point.

The tree has at most $n+1$ levels and the leafs correspond to complete schedules. With each node of the enumeration tree in stage $1 \leq \lambda \leq n+1$ a decision point t_λ and three disjoint activity sets are associated: the set of finished activities F_λ which do not complete later than time t_λ, the set A_λ of all active activities (i.e. scheduled activities which complete after time t_λ) and the set E_λ of all eligible activities (i.e. all unscheduled activities for which all predecessors are completed up to time t_λ).

It is easy to see that due to the constant resource availabilities only completion times of activities have to be considered as decision points for starting unscheduled activities. Thus, in each stage λ of the branch-and-bound procedure the current decision point t_λ is calculated as the minimal completion time of all active activities in $A_{\lambda-1}$. Then the set F_λ of completed activities for the new decision point t_λ contains all activities from $F_{\lambda-1}$ and all activities in $A_{\lambda-1}$ which complete at time t_λ. Furthermore, the set E_λ of eligible activities contains all unscheduled activities $j \notin F_\lambda \cup A_{\lambda-1}$ for which all predecessors i are completed up to time t_λ, i.e. which are contained in F_λ. Finally, the new set A_λ of active activities is given by $A_{\lambda-1} \setminus F_\lambda$.

In each branching step the current partial schedule is extended by scheduling a subset of eligible activities at the corresponding decision point t_λ without violating the resource constraints. More precisely, a so-called **extension alternative** $\gamma_\lambda \subseteq E_\lambda$ is chosen, which is a resource-feasible subset of the eligible activities in E_λ with $\sum\limits_{i \in A_\lambda \cup \gamma_\lambda} r_{ik} \leq R_k$ for all resources k, i.e. all activities from the set γ_λ can be processed simultaneously with the current active activities in A_λ. After choosing such an extension alternative γ_λ, all its activities are added to A_λ.

In order to ensure that the algorithm terminates, we must also consider the empty extension alternative $\gamma_\lambda = \emptyset$, but impose $\gamma_\lambda \neq \emptyset$ when no activities are active at the current decision point t_λ (i.e. when $A_\lambda = \emptyset$ holds).

An extension alternative is called **maximal** if it is not contained in another feasible extension alternative. Unfortunately, in order to find an optimal schedule it is not sufficient to consider only maximal extension alternatives. This can be seen from Example 3.2. If only maximal extension alternatives are considered, in the first iteration the extension alternative $\{1, 4\}$ has to be used. All extensions of the corresponding partial schedule provide a non-delay schedule (cf. Figure 3.3(d)), which is not optimal.

However, the following weaker form of dominance may be applied. An extension alternative γ_λ is called **dominated** if an unscheduled eligible activity $i \notin \gamma_\lambda$ exists which can be scheduled simultaneously with the activities in $A_\lambda \cup \gamma_\lambda$ and which does not finish later than the first completed activity in $A_\lambda \cup \gamma_\lambda$ (which defines the next decision point $t_{\lambda+1}$). In this situation, $\gamma_\lambda \cup \{i\}$ dominates γ_λ since scheduling i together with γ_λ does not worsen the situation. For this reason (which may also be seen as a special application of the left shift rule) dominated sets may be excluded from the search process. Note that this concept also applies for the empty extension alternative.

```
Algorithm Branch-and-Bound based on Extension Alternatives
 1.  Calculate an initial upper bound $UB$ for the instance;
 2.  $S_0 := 0$; $F_0 := \emptyset$; $A_0 := \{0\}$; $\lambda := 1$;
 3.  B&B $(\lambda, F_0, A_0, S)$;

Procedure B&B $(\lambda, F_{\lambda-1}, A_{\lambda-1}, S)$
 1.  $t_\lambda := \min\limits_{i \in A_{\lambda-1}} \{S_i + p_i\}$;
 2.  $F_\lambda := F_{\lambda-1} \cup \{i \in A_{\lambda-1} \mid S_i + p_i = t_\lambda\}$;
 3.  Calculate the set $E_\lambda$ as the set of all activities
     $j \notin F_\lambda \cup A_{\lambda-1}$ for which all predecessors are in $F_\lambda$;
 4.  $A_\lambda := A_{\lambda-1} \setminus F_\lambda$;
 5.  Determine the set $\Gamma_\lambda$ of all non-dominated extension
     alternatives for the sets $A_\lambda, E_\lambda$;
 6.  WHILE $\Gamma_\lambda \neq \emptyset$ DO
 7.     Choose an extension alternative $\gamma_\lambda \in \Gamma_\lambda$;
 8.     $\Gamma_\lambda := \Gamma_\lambda \setminus \{\gamma_\lambda\}$;
 9.     Schedule all activities $j \in \gamma_\lambda$ at time $S_j := t_\lambda$;
10.     $A_\lambda := A_\lambda \cup \gamma_\lambda$;
11.     IF $n+1 \in \gamma_\lambda$ THEN
12.        $UB := \min\{UB, S_{n+1}\}$; unschedule $n+1$;
13.        RETURN
14.     ELSE
15.        Calculate a lower bound $LB(S)$ for all extensions
           of the current partial schedule $S$;
16.        IF $LB(S) < UB$ THEN
17.           B&B $(\lambda+1, F_\lambda, A_\lambda, S)$;
18.        Unschedule all $j \in \gamma_\lambda$ and set $A_\lambda := A_\lambda \setminus \gamma_\lambda$;
19.     ENDIF
20.  ENDWHILE
```

Figure 3.59: A branch-and-bound algorithm based on extension alternatives

Using the current sets A_λ and E_λ the set Γ_λ of all non-dominated extension alternatives can easily be calculated for the current partial schedule. After choosing an extension alternative $\gamma_\lambda \in \Gamma_\lambda$ and scheduling all activities from this set γ_λ, the next branching stage $\lambda + 1$ is considered.

If all activities are scheduled, the makespan of the corresponding schedule is compared to the best upper bound value found so far. Afterwards, backtracking to the previous stage occurs where the next unconsidered extension alternative is chosen. If all non-dominated extension alternatives have been considered as branching candidates in a stage, a further backtracking step to the previous stage is performed.

The preceding discussions are summarized in the algorithm shown in Figure 3.59. In this algorithm the recursive procedure `B&B` $(\lambda, F_{\lambda-1}, A_{\lambda-1}, S)$ is called, where λ denotes the current stage, $F_{\lambda-1}$ and $A_{\lambda-1}$ are the sets of finished and active activities from the previous stage $\lambda - 1$ and S is the starting time vector of the current partial schedule. In order to have a unique description of the recursive procedure initially again a dummy starting activity 0 is started at time $S_0 := 0$.

In Step 15 a lower bound $LB(S)$ for the makespan of all extensions of the current partial schedule S can be calculated similarly as in Step 11 of the precedence tree algorithm, for example as

$$LB(S) = \max_{j \in \gamma_\lambda} \{S_j + p_j + q_j\} \text{ or } LB(S) = \max_{i \notin (F_\lambda \cup A_\lambda)} \{\min_{j \in A_\lambda} (S_j + p_j) + p_i + q_i\}.$$

Furthermore, as in the precedence tree algorithm dominance rules like the local or global left shift rule can be integrated. Then the search space is reduced to the subset of semi-active and active schedules, respectively.

Example 3.28 : Consider again the instance from Example 3.27 shown in Figure 3.55. The resulting enumeration tree for the algorithm based on extension alternatives can be found in Figure 3.60. For each node the current decision point t_λ, the set of active activities A_λ and the set Γ_λ of non-dominated extension alternatives are given.

In the first braching step for $t_1 = 0$ and $A_1 = \{0\}$ only the three extension alternatives $\{1, 2\}, \{1, 3\}$ and $\{2, 3\}$ have to be considered since the extension alternative $\{1\}$ is dominated by the alternative $\{1, 2\}$ and the sets $\{2\}, \{3\}$ are both dominated by $\{2, 3\}$.

For example, in the third stage for the left branch $\{1, 2\} - \{3\}$ the empty extension alternative has to be used since at the current decision point $t_3 = C_3 = 4$ we have $A_3 = \{1\} \neq \emptyset$ and no activity can be processed in parallel with the active activity 1 at time 4. On the other hand, in the second stage for the branch $\{2, 3\}$ at the decision point $t_2 = C_2 = 2$ the empty extension alternative may not be used since the set of active activities A_2 is empty. □

Another approach which extends partial schedules by subsets of activities is the so-called **block extension procedure** using the concept of feasible subsets X introduced in Section 3.7.3. It is based on the observation that an optimal schedule defines time points $t_0 = 0 < t_1 < \ldots < t_\tau$ and corresponding feasible subsets $X_1, \ldots, X_\tau \subset \{1, \ldots, n\}$ such that

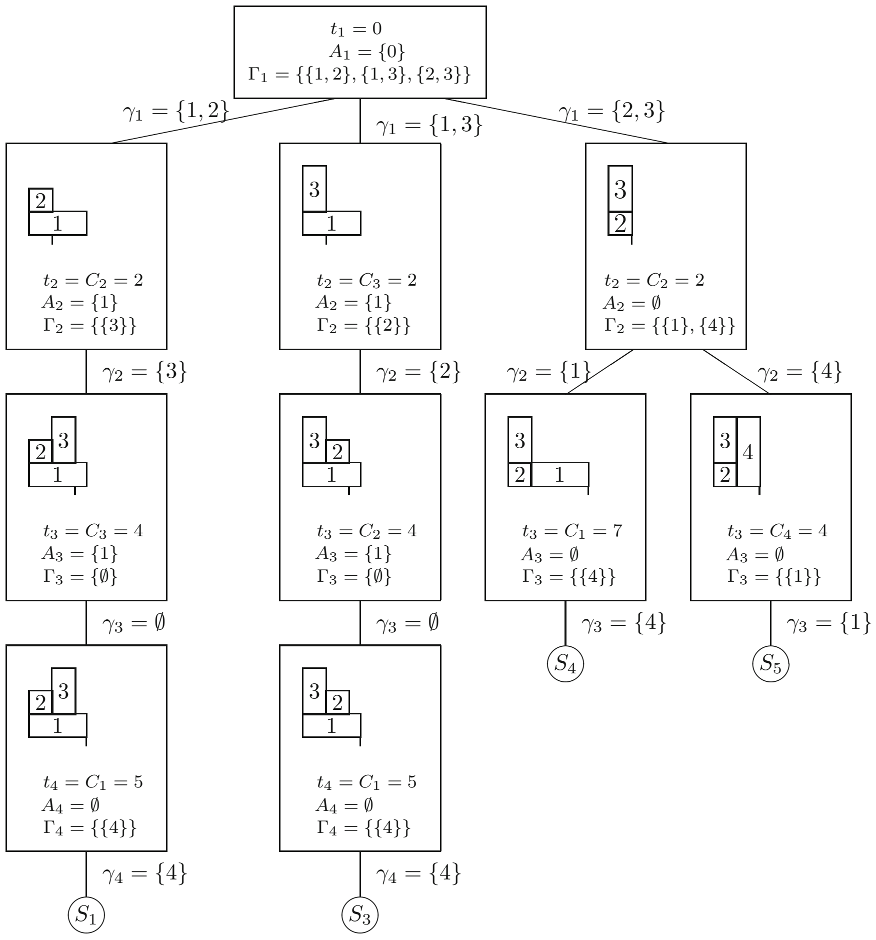


Figure 3.60: Search tree for the algorithm based on extension alternatives

- each time point t_j with $j \in \{1, \ldots, \tau\}$ is the completion time of an activity,
- all activities in X_j are processed simultaneously in the whole interval $[t_{j-1}, t_j[$ for $j = 1, \ldots, \tau$,
- if an activity $i \in X_j$ is not finished in the interval $[t_{j-1}, t_j[$, it is continued in the interval $[t_j, t_{j+1}[$,
- all predecessors of an activity starting at time t_j are finished before t_j.

An interval $[t_{j-1}, t_j[$ with a corresponding feasible subset X_j of activities satisfying the last two properties is called a feasible block. Obviously, a schedule can be built by successively adding a feasible block to the end of a partial schedule. Thus, in a branch-and-bound algorithm all possible block extensions of a given partial schedule may be enumerated.

3.8.3 An algorithm based on delaying alternatives

In this subsection we present a branch-and-bound algorithm which is based on scheduling activities until a resource conflict occurs. As in the algorithm based on extension alternatives time points are enumerated at which activities may be started. The nodes of the enumeration tree correspond to partial schedules which are precedence- and resource-feasible and are extended by starting all eligible activities. If this causes a resource conflict, in a branching step this conflict is resolved by delaying different subsets of activities.

As in the algorithm based on extension alternatives, the enumeration tree has at most $n+1$ levels and the leafs correspond to complete schedules. With each node of the enumeration tree in stage $1 \leq \lambda \leq n+1$ a decision point t_λ and three disjoint activity sets are associated: the set of finished activities F_λ which do not complete later than time t_λ, the set A_λ of all active activities (i.e. scheduled activities which complete after time t_λ) and the set E_λ of all eligible activities (i.e. all unscheduled activities for which all predecessors are completed up to time t_λ).

As in the algorithm based on extension alternatives, in each stage λ of the branch-and-bound procedure the current decision point t_λ is calculated as the minimal completion time of all active activities in $A_{\lambda-1}$ (again only completion times of activities have to be considered as decision points for starting unscheduled activities). Then the set F_λ of finished activities for the new decision point t_λ contains all activities from $F_{\lambda-1}$ and all activities in $A_{\lambda-1}$ which complete at time t_λ. Furthermore, the set E_λ of eligible activities contains all unscheduled activities $j \notin F_\lambda \cup A_{\lambda-1}$ for which all predecessors i are completed up to time t_λ, i.e. which are contained in F_λ.

Then, contrary to the algorithm based on extension alternatives, all eligible unscheduled activities are temporarily started, i.e. afterwards the new set of active activities is given by $A_\lambda = (A_{\lambda-1} \setminus F_\lambda) \cup E_\lambda$. If this set causes a resource conflict (i.e. $\sum\limits_{i \in A_\lambda} r_{ik} > R_k$ for some resource k), some activities are delayed, i.e. they have to be started later.

In each branching step a resource conflict in the current partial schedule is resolved by delaying a subset of the active activities A_λ at the corresponding decision point t_λ such that the resources are sufficient for the non-delayed activities. More precisely, a so-called **delaying alternative** $\delta_\lambda \subset A_\lambda$ is chosen, such that the remaining set $A_\lambda \setminus \delta_\lambda$ satisfies $\sum\limits_{i \in A_\lambda \setminus \delta_\lambda} r_{ik} \leq R_k$ for all resources k. Note that in this algorithm only activities from the set F_λ are permanently scheduled in the current partial schedule, activities from the set A_λ may be moved to the right in subsequent iterations.

A delaying alternative is called **minimal**, if it does not contain a proper subset which is also a delaying alternative. Contrary to the algorithm based on extension alternatives where also non-maximal extension alternatives have to be considered, in the algorithm based on delaying alternatives it is sufficient to consider only minimal delaying alternatives. This is based on the fact that activities which are not delayed in the current stage of the branch-and-bound tree can still be delayed in later stages of the algorithm (in the algorithm based on extension alternatives previous decisions cannot be changed in later iterations). A proof of this property will be given after the description of the whole algorithm.

Using the current set A_λ the set Δ_λ of all minimal delaying alternatives can be calculated for the current partial schedule. After choosing a delaying alternative $\delta_\lambda \in \Delta_\lambda$ all activities from the set δ_λ are unscheduled, i.e. removed from the current partial schedule. Note that if no resource conflict occurs, the empty set is the only minimal delaying alternative. After storing the old starting time $S_j^\lambda := t_\lambda$ for all delayed activities $j \in \delta_\lambda$ (which are used in the backtracking step), the next branching stage $\lambda + 1$ is considered.

If all activities are scheduled, the makespan of the corresponding schedule is compared to the best upper bound value found so far. Afterwards, backtracking to the previous stage occurs where the next unconsidered minimal delaying alternative is chosen. If all minimal delaying alternatives have been considered as branching candidates in a stage, a further backtracking step to the previous stage is performed.

The preceding discussions are summarized in the algorithm shown in Figure 3.61. In this algorithm the recursive procedure `B&B` $(\lambda, F_{\lambda-1}, A_{\lambda-1}, \delta_{\lambda-1}, S)$ is called, where λ denotes the current stage, $F_{\lambda-1}$ and $A_{\lambda-1}$ are the sets of finished and active activities from the previous stage $\lambda - 1$, the set $\delta_{\lambda-1}$ is the chosen delaying alternative in stage $\lambda - 1$, and S is the starting time vector of the current partial schedule.

In Step 14 a lower bound $LB(\delta_\lambda)$ for the makespan of all extensions induced by the current delaying alternative δ_λ can be calculated using the tails q_j of the delayed activities $j \in \delta_\lambda$. For example, we may set

$$LB(\delta_\lambda) := \min_{i \in A_\lambda \setminus \delta_\lambda} \{S_i + p_i\} + \max_{j \in \delta_\lambda} \{p_j + q_j\},$$

since no delayed activity can be started before the first currently scheduled activity is completed.

Theorem 3.9 In the branch-and-bound algorithm based on delaying alternatives it is sufficient to consider only minimal delaying alternatives.

Proof: Assume to the contrary that an instance exists for which an optimal schedule can only be constructed by the algorithm if non-minimal delaying alternatives are used. Let $(\delta_1, \ldots, \delta_\lambda, \ldots, \delta_k)$ be a sequence of delaying alternatives where λ is the largest index such that $(\delta_1, \ldots, \delta_\lambda, \ldots, \delta_k)$ leads to an optimal schedule S^* and $\delta_1, \ldots, \delta_{\lambda-1}$ are minimal delaying alternatives. Since δ_λ is not minimal, an activity $x \in \delta_\lambda$ exists which can be eliminated such that $\delta'_\lambda := \delta_\lambda \setminus \{x\}$ is again a delaying alternative. Let S and S' be the partial schedules corresponding to the subsequences $(\delta_1, \ldots, \delta_\lambda)$ and $(\delta_1, \ldots, \delta_{\lambda-1}, \delta'_\lambda)$,

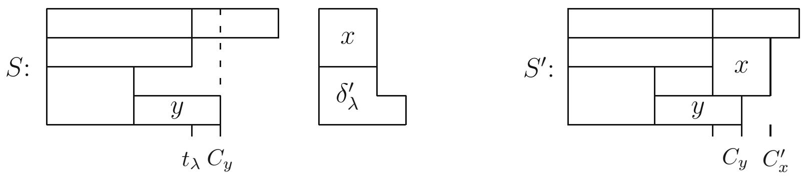
```
Algorithm Branch-and-Bound based on Delaying Alternatives
 1.  Calculate an initial upper bound $UB$ for the instance;
 2.  $S_0 := 0$; $F_0 := \emptyset$; $E_0 := \emptyset$; $A_0 := \{0\}$; $\delta_0 := \emptyset$; $\lambda := 1$;
 3.  B&B $(\lambda, F_0, A_0, \delta_0, S)$;

Procedure B&B $(\lambda, F_{\lambda-1}, A_{\lambda-1}, \delta_{\lambda-1}, S)$
 1.  $t_\lambda := \min\limits_{i \in A_{\lambda-1}} \{S_i + p_i\}$;
 2.  $F_\lambda := F_{\lambda-1} \cup \{i \in A_{\lambda-1} \mid S_i + p_i = t_\lambda\}$;
 3.  Calculate the set $E_\lambda$ as the set of all activities
     $j \notin F_\lambda \cup A_{\lambda-1}$ for which all predecessors are in $F_\lambda$;
 4.  Schedule all $j \in E_\lambda$ temporarily at time $S_j := t_\lambda$;
 5.  $A_\lambda := (A_{\lambda-1} \setminus F_\lambda) \cup E_\lambda$;
 6.  IF $n+1 \in E_\lambda$ THEN
 7.     $UB := \min\{UB, S_{n+1}\}$; unschedule $n+1$;
 8.     RETURN
 9.  ELSE
10.     Determine the set $\Delta_\lambda$ of all minimal delaying
        alternatives for the set $A_\lambda$;
11.     WHILE $\Delta_\lambda \neq \emptyset$ DO
12.        Choose a delaying alternative $\delta_\lambda \in \Delta_\lambda$;
13.        $\Delta_\lambda := \Delta_\lambda \setminus \{\delta_\lambda\}$;
14.        Calculate a lower bound $LB(\delta_\lambda)$ for all extensions
           of the delaying alternative $\delta_\lambda$;
15.        IF $LB(\delta_\lambda) < UB$ THEN
16.           $A_\lambda := A_\lambda \setminus \delta_\lambda$;
17.           FOR ALL $j \in \delta_\lambda$ DO $S_j^\lambda := S_j$;
18.           Unschedule all $j \in \delta_\lambda$;
19.           B&B $(\lambda+1, F_\lambda, A_\lambda, \delta_\lambda, S)$;
20.           FOR ALL $j \in \delta_\lambda$ DO $S_j := S_j^\lambda$;
21.           $A_\lambda := A_\lambda \cup \delta_\lambda$;
22.        ENDIF
23.     ENDWHILE
24.  ENDIF
```

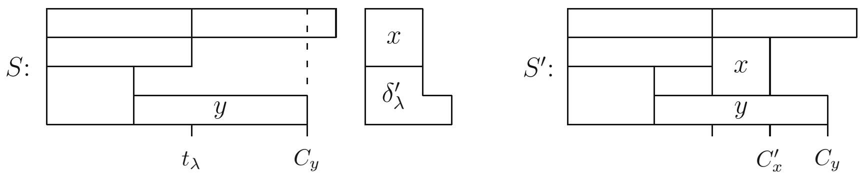
Figure 3.61: A branch-and-bound algorithm based on delaying alternatives

respectively. Denote by C_j and C'_j the completion times of the scheduled activities after delaying the activities in the sets δ_λ and δ'_λ. Furthermore, let y be an activity in $A_\lambda \setminus \delta_\lambda$ which completes first in S.

- Case 1: $C'_x \geq C_y$, i.e. when x is not delayed in stage λ, it does not complete earlier than y in S' (cf. Figure 3.62(a)). If we proceed the algorithm with the subsequence $(\delta_1, \ldots, \delta_{\lambda-1}, \delta'_\lambda)$, the next considered decision point is



(a) Case 1: $C'_x \geq C_y$



(b) Case 2: $C'_x < C_y$

Figure 3.62: The two cases in the proof of Theorem 3.9

$t'_{\lambda+1} = C_y$ which is the same as for the subsequence $(\delta_1, \ldots, \delta_\lambda)$. Since the schedule S^* can be generated by the second subsequence and in S' activity x may still be delayed at time $t'_{\lambda+1} = C_y$, it can also be generated by the first subsequence. But this contradicts our assumption that in order to construct an optimal schedule S^* in stage λ the non-minimal delaying alternative δ_λ has to be used.

- Case 2: $C'_x < C_y$ (cf. Figure 3.62(b)). In S and S' up to stage λ the same activities are scheduled, i.e. we have $C_j = C'_j$ for all activities $j \in F_\lambda$, and C'_x is smaller than the completion time of x in any extension of schedule S since in S activity x cannot be started before time $t_{\lambda+1} = C_y$. Consider the schedule $\tilde{S}$ with completion times

$$\tilde{C}_j := \begin{cases} C'_j & \text{for } j \in F_\lambda \cup \{x\} \\ C^*_j & \text{otherwise,} \end{cases}$$

which is an extension of the partial schedule S'. This schedule is feasible since in S' all activities $j \in F_\lambda \cup \{x\}$ are completed no later than time $t'_{\lambda+1} = C'_x < C_y = t_{\lambda+1}$ and all other activities do not overlap with them since they do not overlap in S^*. Furthermore, $\tilde{S}$ has the same makespan as the optimal schedule S^*.

Additionally, a schedule $\tilde{S}'$ with the same makespan as $\tilde{S}$ exists which is equal to $\tilde{S}$ or can be obtained from $\tilde{S}$ by some additional left shifts of activities due to the left shift of x. It can be shown that this optimal schedule $\tilde{S}'$ can be constructed by the subsequence $(\delta_1, \ldots, \delta_{\lambda-1}, \delta'_\lambda)$ since the next decision point $t'_{\lambda+1} = C'_x$ is smaller than the next decision point $t_{\lambda+1} = C_y$ for the subsequence $(\delta_1, \ldots, \delta_\lambda)$ and S^* can be constructed by this subsequence. This again contradicts the fact that in stage λ a non-minimal delaying alternative δ_λ has to be used in order to construct an optimal schedule. □

Example 3.29 : Consider again the instance from Example 3.27 shown in Figure 3.55. The resulting enumeration tree for the algorithm based on delaying alternatives can be found in Figure 3.63. For each node the current decision point t_λ, the set of eligible activities E_λ and the set Δ_λ of minimal delaying alternatives are given.

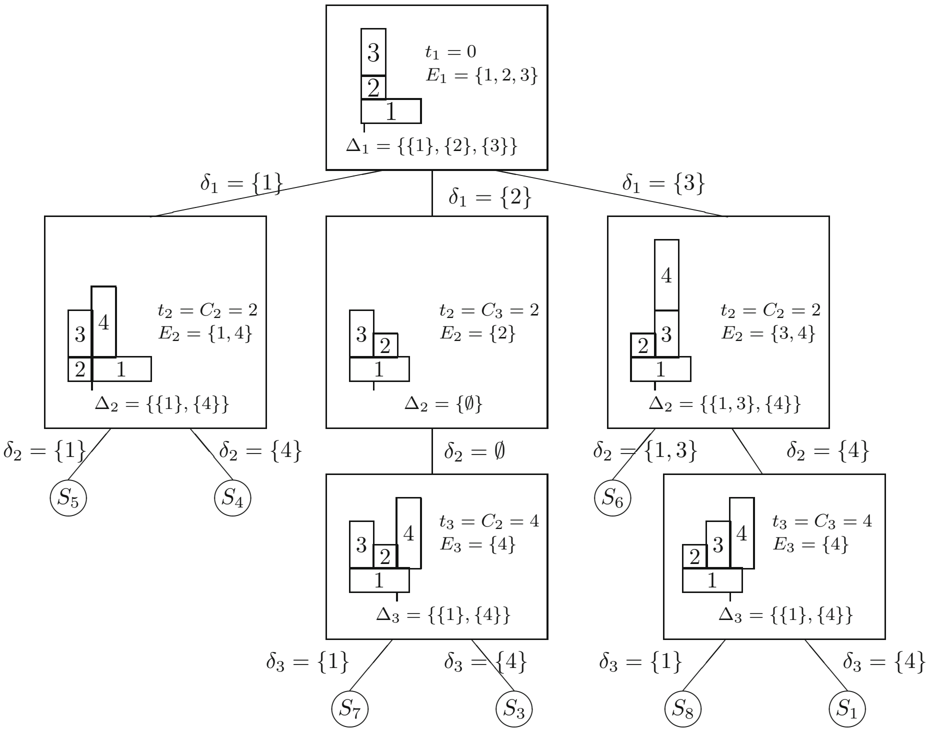


Figure 3.63: Search tree for the algorithm based on delaying alternatives

Compared with the enumeration trees of the previous two algorithms two new schedules are generated (cf. Figure 3.64).

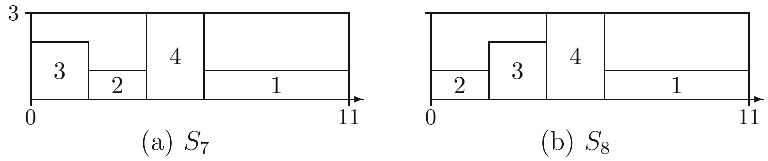


Figure 3.64: Additional schedules obtained in Example 3.29 □

A main difference between the the two previous algorithms (based on precedence trees and extension alternatives) and the algorithm based on delaying alternatives is the treatment of scheduling decisions from previous stages. While in the first two algorithms scheduling decisions from earlier stages are not withdrawn in later stages, in the third algorithm activities which have been scheduled in previous stages may be delayed in later stages.

For this reason, also the left shift rules behave differently. Usually, in connection with delaying alternatives the local left shift rule is formulated as follows: If at the current decision point t_λ an activity i is scheduled which has been delayed in the previous stage (i.e. $i \in \delta_{\lambda-1} \setminus \delta_\lambda$) and delaying the activities from the current delaying alternative δ_λ allows that i can be locally shifted to the left, then the current partial schedule can be eliminated. However, delaying an activity in a later stage can make a local left shift applicable which is not feasible at the current stage. Thus, this version of the local left shift rule does not guarantee that only semi-active schedules are generated.

As an example, consider the schedule S_8 from the example above which is not semi-active. It is also generated if the local left shift rule is used. When in the third stage the delaying alternative $\delta_3 = \{1\}$ is chosen, only the activity $4 \in \delta_2 = \{4\}$ from the previous delaying alternative is tested for a local left shift. However, due to the delay of activity 1, now activity 3 can be shifted to the left, which is not detected.

This example shows that in order to guarantee that only semi-active schedules are generated, additionally some other activities have to be tested for possible left shifts. More precisely, all scheduled activities which do not start before the first currently delayed activity $j \in \delta_\lambda$ have to be tested:

Extended local left shift rule: Let $t_{\min} := \min\limits_{j \in \delta_\lambda} \{S_j\}$ be the minimal starting time of all activities belonging to the current delaying alternative δ_λ in stage λ. If a scheduled activity $i \notin \delta_\lambda$ with $S_i \geq t_{\min}$ exists which can be locally shifted to the left after delaying δ_λ, the current partial schedule can be eliminated.

For the above example, with this modification the possible left shift of activity 3 in the schedule S_8 is detected since $t_{\min} = S_1 = 0 < S_3 = 2$.

Additionally to the left shift rule the following dominance rule was proposed:

Cutset rule: For each decision point t_λ let the cutset $C(t_\lambda)$ be defined as the set of all unscheduled activities for which all predecessors are already scheduled (i.e. belong to the set $F_{\lambda-1} \cup A_{\lambda-1}$). Furthermore, for each scheduled activity $j \in F_{\lambda-1} \cup A_{\lambda-1}$ denote by C_j^λ its completion time in the partial schedule belonging to t_λ. Assume that a cutset $C(t_\lambda)$ equals a cutset $C(t_\nu)$ which has previously been saved in a node of the search tree. If $t_\nu \leq t_\lambda$ and each activity j which is active at time t_ν in the partial schedule belonging to t_ν satisfies $C_j^\nu \leq \max\{t_\lambda, C_j^\lambda\}$, then the partial schedule belonging to t_λ is dominated.

In order to prove the correctness of this dominance rule, it can be shown that an optimal extension of the partial schedule C^λ cannot be better than an optimal extension of the partial schedule C^ν. Using this rule, the computation times can often be reduced since large parts of the enumeration tree can be pruned off. On the other hand, this rule needs a large amount of storage since the completion times in all considered partial schedules have to be stored.

Two further dominance rules are:

Immediate scheduling of one activity: If in stage λ no active activity exists at time t_λ (i.e. $A_{\lambda-1} \setminus F_\lambda = \emptyset$) and an unscheduled activity $i \in E_\lambda$ cannot be scheduled simultaneously with any other unscheduled activity at any time point $t \geq t_\lambda$, then activity i can immediately be scheduled at time t_λ.

Immediate scheduling of two activities: If in stage λ no active activity exists at time t_λ (i.e. $A_{\lambda-1} \setminus F_\lambda = \emptyset$), an unscheduled activity $i \in E_\lambda$ can only be scheduled simultaneously with an other unscheduled activity j at any time point $t \geq t_\lambda$ and $p_j \leq p_i$, then activities i, j can immediately be scheduled at time t_λ.

3.8.4 An algorithm based on schedule schemes

In this subsection we present a branch-and-bound algorithm which is based on the basic relations introduced in Section 3.6.1. The nodes of the enumeration tree represent sets of feasible schedules which are defined by conjunctions, disjunctions, and parallelity relations. In each branching step the set of schedules belonging to a node is split into two subsets. This is accomplished by a binary branching scheme where for two activities i, j in one branch the parallelity relation $i \parallel j$ and in the other branch the disjunction $i - j$ is added.

First we will introduce schedule schemes, which are used to represent the nodes of the enumeration tree mathematically. Let $C, D, N \subseteq V \times V$ be disjoint relations where C is a set of conjunctions, D is a set of disjunctions, and N is a set of parallelity relations. The relations D and N are symmetric, i.e. with $i - j \in D$ ($i \parallel j \in N$) also $j - i \in D$ ($j \parallel i \in N$) holds. On the other hand, C is antisymmetric, i.e. if $i \rightarrow j \in C$, then $j \rightarrow i \notin C$. If $i - j \in D$, then it is not specified whether $i \rightarrow j$ or $j \rightarrow i$ holds. Based on C, D, N we define the set of remaining relations as

$$F = \{(i, j) \mid i, j \in V;\ i \neq j;\ i \rightarrow j \notin C,\ j \rightarrow i \notin C,\ i - j \notin D,\ i \parallel j \notin N\}.$$

The relations in F are called **flexibility relations** and are denoted by $i \sim j$. Note, that F is symmetric. The tuple (C, D, N, F) with the property that for all $i, j \in V$ with $i \neq j$ exactly one of the relations $i \rightarrow j \in C$, $j \rightarrow i \in C$, $i - j \in D$, $i \parallel j \in N$ or $i \sim j \in F$ holds, is called a **schedule scheme**.

A schedule scheme (C, D, N, F) defines a (possibly empty) set $\mathcal{S}(C, D, N, F)$ of schedules. More precisely, $\mathcal{S}(C, D, N, F)$ is the set of all schedules with the following properties:

- if $i \rightarrow j \in C$, then i is finished when j starts,
- if $i \parallel j \in N$, then i and j are processed in parallel for at least one time unit, and
- if $i - j \in D$, then i and j are not processed in parallel.

$\mathcal{S}_f(C, D, N, F)$ denotes the corresponding set of feasible schedules, i.e. all schedules in $\mathcal{S}(C, D, N, F)$ which additionally satisfy the resource constraints. If for an RCPSP instance the sets C_0, D_0, N_0 are defined as in Section 3.6.1 and F_0 are the corresponding remaining flexibility relations, then $\mathcal{S}_f(C_0, D_0, N_0, F_0)$ is the set of all feasible schedules for this instance.

Starting with the initial schedule scheme (C_0, D_0, N_0, F_0) as root of the enumeration tree, the branch-and-bound algorithm is based on a binary branching scheme where in each step for one flexibility relation $i \sim j \in F$ the two branches $i - j \in D$ and $i \parallel j \in N$ are created. This branching process is repeated until the set F becomes empty. We will show that for such a schedule scheme $(C, D, N, \emptyset)$

- we can either detect that $\mathcal{S}_f(C, D, N, \emptyset) = \emptyset$ (i.e. no feasible schedule corresponding to this scheme exists), or
- we can calculate a schedule which dominates all feasible schedules in the set $\mathcal{S}_f(C, D, N, \emptyset)$.

Thus, a schedule scheme $(C, D, N, \emptyset)$ with no flexibility relations can be treated as leaf in the enumeration tree.

Consider a schedule scheme $(C, D, N, \emptyset)$ with no flexibility relations. Then the schedule scheme $(C', \emptyset, N, \emptyset)$ is called a **transitive orientation** of $(C, D, N, \emptyset)$ if

- $C \subseteq C'$ and $(C', \emptyset, N, \emptyset)$ is derived from $(C, D, N, \emptyset)$ by changing each disjunction $i - j \in D$ into either $i \rightarrow j \in C'$ or $j \rightarrow i \in C'$,
- C' is transitive, i.e. $i \rightarrow j \in C'$ and $j \rightarrow k \in C'$ imply $i \rightarrow k \in C'$.

It is easy to see that each feasible schedule can be represented by a transitive orientation $(C', \emptyset, N, \emptyset)$ of some schedule scheme $(C, D, N, \emptyset)$ (if for a disjunction $i - j \in D$ activity i is started before j in the schedule, set $i \rightarrow j \in C'$). In the following we will show how a dominating schedule can be calculated for a transitive orientation $(C', \emptyset, N, \emptyset)$.

Note that if $(C', \emptyset, N, \emptyset)$ is a transitive orientation, then the directed graph (V, C') is acyclic because C' is antisymmetric. Let $(C', \emptyset, N, \emptyset)$ be a transitive orientation of a schedule scheme. Ignoring the parallelity relations we calculate a corresponding (not necessarily feasible) schedule as follows. Consider the acyclic directed graph (V, C'). For each activity $i \in V$ we calculate the length r_i of a longest path from 0 to i and start i at time r_i. We denote this **earliest start schedule** (which only depends on C'), by $S_{ES}(C')$. Note that $S_{ES}(C')$ does not have to belong to the set $\mathcal{S}_f(C', \emptyset, N, \emptyset)$ since the parallelity relations are not explicitly taken into account and resource constraints may be violated. However, since obviously $C_{\max}(S_{ES}(C')) \leq C_{\max}(S)$ for all schedules $S \in \mathcal{S}_f(C', \emptyset, N, \emptyset)$ holds and the following theorem is valid, this causes no problem.

Theorem 3.10 Let $(C', \emptyset, N, \emptyset)$ be an arbitrary transitive orientation of a schedule scheme $(C, D, N, \emptyset)$ and let $S_{ES}(C')$ be the corresponding earliest start schedule. If $S_{ES}(C')$ is feasible, then $S_{ES}(C')$ dominates all schedules in the set $\mathcal{S}_f(C, D, N, \emptyset)$. Otherwise, the set $\mathcal{S}_f(C, D, N, \emptyset)$ of feasible schedules is empty.

Proof: Assume that $S_{ES}(C')$ is infeasible, i.e. there is a time period t such that activities of a set H are processed in parallel during t and for some resource type k we have $\sum\limits_{i \in H} r_{ik} > R_k$. If there exists a feasible schedule $S \in \mathcal{S}_f(C, D, N, \emptyset)$, then for at least two activities $i, j \in H$ one of the relations $i \to j \in C$ or $j \to i \in C$ or $i - j \in D$, i.e. $i \parallel j \notin N$ holds. Otherwise, if we have $i \parallel j \in N$ for all $i, j \in H$ with $i \neq j$, this implies that also in S all activities are processed in parallel during some time period t. This can be seen as follows. Let $H^0 \subseteq H$ be a maximal set of activities which are processed in parallel in S during a time interval I and assume that $k \in H \setminus H^0$, i.e. for the processing interval $I_k := [S_k, C_k[$ we have $I \cap I_k = \emptyset$. Assume w.l.o.g. that I_k is to the right of I. Then an activity $l \in H^0$ exists which does not complete later than k starts in S. Thus, $I_k \cap I_l = \emptyset$, which is a contradiction to $k \parallel l \in N$. However, if all activities in H are processed in parallel, then S cannot be a feasible schedule. Hence, for at least two activities $i, j \in H$ we have $i \parallel j \notin N$ implying $i \to j \in C \subseteq C'$ or $j \to i \in C \subseteq C'$, which contradicts the definition of H. Thus, if $S_{ES}(C')$ is infeasible, the set $\mathcal{S}_f(C, D, N, \emptyset)$ of feasible schedules must be empty.

Now consider an arbitrary feasible schedule $S \in \mathcal{S}_f(C, D, N, \emptyset)$. This schedule S defines a transitive orientation $(C'', \emptyset, N, \emptyset)$ of $(C, D, N, \emptyset)$ and the corresponding earliest start schedule $S_{ES}(C'')$ does not have a larger makespan than S.

Due to graph theoretical results in connection with so-called comparability graphs an earliest start schedule $S_{ES}(C')$ associated with a transitive orientation $(C', \emptyset, N, \emptyset)$ of a schedule scheme $(C, D, N, \emptyset)$ has always the same $C_{\max}$-value, regardless which transitive orientation is chosen. Thus, $S_{ES}(C')$ and $S_{ES}(C'')$ have the same $C_{\max}$-value, i.e. $S_{ES}(C')$ dominates S. □

Thus, due to Theorem 3.10 we may proceed as follows. Whenever the set of flexibility relations becomes empty, we first check whether a transitive orientation of the corresponding schedule scheme $(C, D, N, \emptyset)$ exists (not every schedule scheme $(C, D, N, \emptyset)$ can be transitively oriented). This can be done by a polynomial-time algorithm (cf. the reference notes). If no transitive orientation exists, the schedule scheme does not represent a feasible schedule and backtracking may be performed. Otherwise, for the transitive orientation the corresponding earliest start schedule is calculated. If it is feasible, its makespan is compared to the best upper bound value found so far. Afterwards, backtracking occurs.

The preceding discussions are summarized in the algorithm shown in Figure 3.65. In this algorithm the recursive procedure `B&B` (C, D, N, F) is called. This procedure applies at first some constraint propagation techniques to the given schedule scheme in order to reduce the search space. It introduces additional conjunctions, disjunctions and parallelity relations as described in Section 3.6.4 and returns the new schedule scheme.

```
Algorithm Branch-and-Bound based on Schedule Schemes
 1.  Calculate an initial upper bound $UB$ for the instance;
 2.  B&B $(C_0, D_0, N_0, F_0)$

Procedure B&B $(C, D, N, F)$
 1.  ConstraintPropagation $(C, D, N, F)$;
 2.  IF $F = \emptyset$ THEN
 3.     Check if a transitive orientation exists for
        $(C, D, N, \emptyset)$;
 4.     IF no transitive orientation exists THEN
 5.        RETURN
 6.     ELSE
 7.        Calculate a transitive orientation $(C', \emptyset, N, \emptyset)$ of
           $(C, D, N, \emptyset)$, the earliest start schedule $S_{ES}(C')$
           and its makespan $C_{\max}(S_{ES}(C'))$;
 8.        IF $S_{ES}(C')$ is feasible THEN
 9.           $UB := \min\{UB, C_{\max}(S_{ES}(C'))\}$;
10.        ELSE RETURN
11.     ENDIF
12.  ELSE
13.     Calculate a lower bound $LB(C, D, N, F)$;
14.     IF $LB(C, D, N, F) < UB$ THEN
15.        Choose $i \sim j \in F$;
16.        B&B $(C, D \cup \{i - j\}, N, F \setminus \{i \sim j\})$;
17.        B&B $(C, D, N \cup \{i \parallel j\}, F \setminus \{i \sim j\})$;
18.     ENDIF
19.  ENDIF
```

Figure 3.65: A branch-and-bound algorithm based on schedule schemes

In Step 13 additionally a lower bound $LB(C, D, N, F)$ for all schedules in the set $\mathcal{S}_f(C, D, N, F)$ is calculated. Such a lower bound may for example be obtained by considering the distance matrix based on the schedule scheme (cf. Section 3.6.2) and taking the distance $d_{0,n+1}$ as lower bound value after constraint propagation has been applied. If this value is not smaller than the best makespan UB found so far, the corresponding branch in the tree does not have to be inspected and backtracking can be performed.

In Step 15 a flexibility relation $i \sim j \in F$ has to be chosen creating the two branches $i - j \in D$ and $i \parallel j \in N$. This choice may be guided by a priority rule. For example, weights w_{i-j} and $w_{i\|j}$ may be calculated which are approximations of lower bounds for the schedules in the two branches. Then a relation $i \sim j \in F$ is chosen such that the sum $w_{i-j} + w_{i\|j}$ is maximal (in order to prune off the branch with a higher probability).

Example 3.30 : Consider again the instance from Example 3.27 shown in Figure 3.55. Initially we have

$$C_0 = \{2 \rightarrow 4\},\ D_0 = \{1 - 4, 3 - 4\},\ N_0 = \emptyset,\ F_0 = \{1 \sim 2, 1 \sim 3, 2 \sim 3\}.$$

A possible enumeration tree for the algorithm based on schedule schemes can be found in Figure 3.66. In the first stage the relation $1 \sim 2$ is chosen, in the second $1 \sim 3$ and in the third $2 \sim 3$. For all resulting schedule schemes $(C, D, N, \emptyset)$ in the 8 leafs transitive orientations exist. In Figure 3.67 possible orientations for the schedule schemes in the 8 leafs are shown together with the corresponding earliest-start schedules.

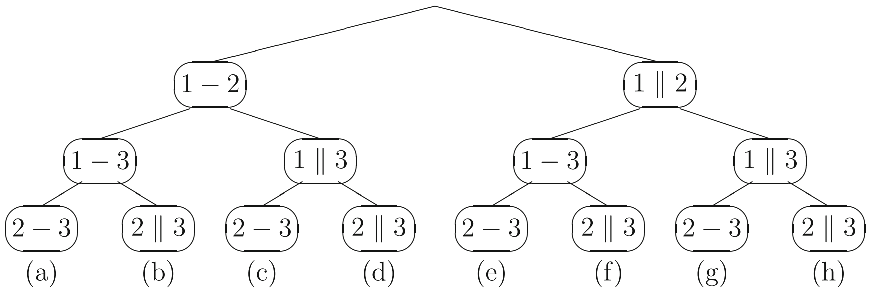


Figure 3.66: Search tree for the algorithm based on schedule schemes

Note that in (d) and (f) the earliest start schedules do not belong to the set of feasible schedules represented by the corresponding schedule schemes $(C, D, N, \emptyset)$ since in (d) the parallelity relation $2 \parallel 3$ and in (f) the parallelity relation $1 \parallel 2$ is violated. Furthermore, for the orientation in (h) the earliest start schedule is not feasible since processing the activities $1, 2, 3$ in parallel violates the resource constraints. Thus, in this case the set of feasible schedules belonging to the corresponding schedule scheme is empty. □

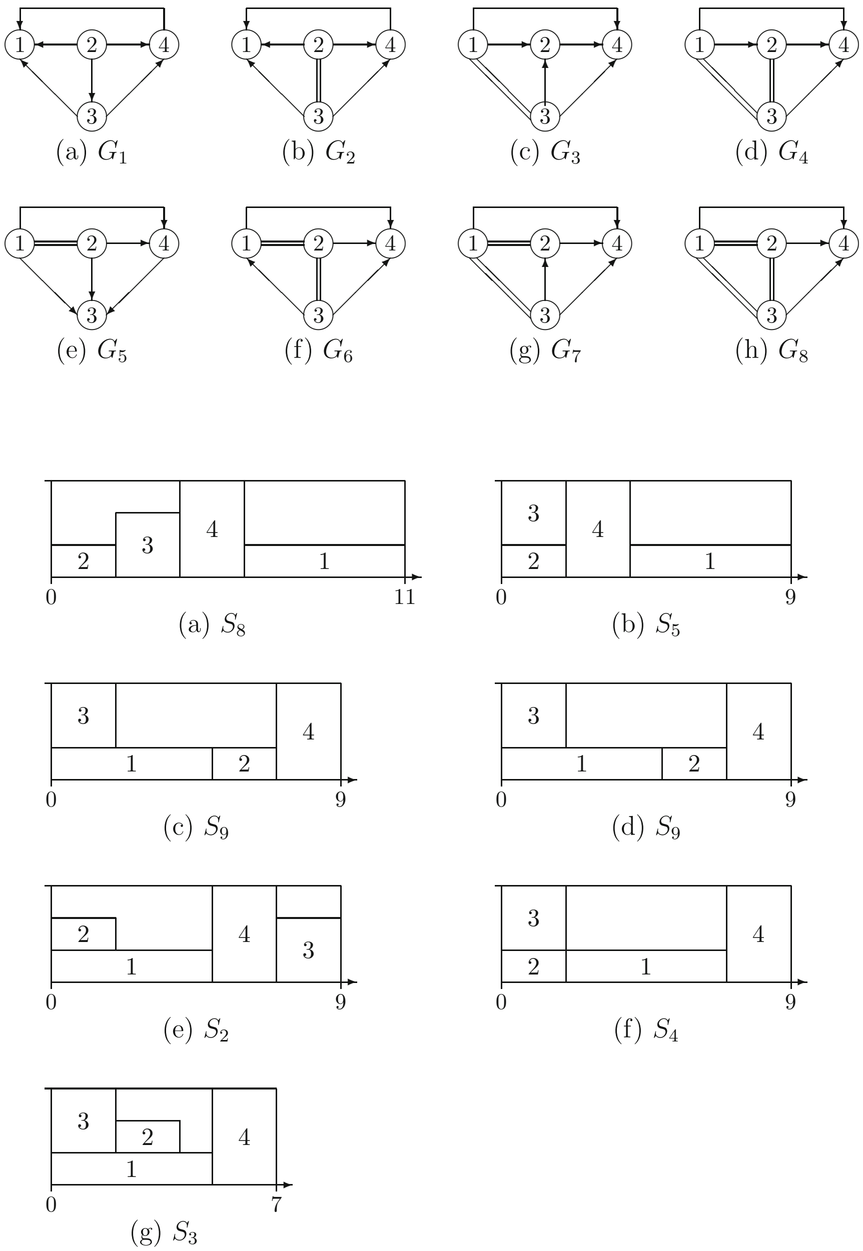


Figure 3.67: Enumerated graphs and schedules for Example 3.30

3.8.5 Algorithms for the multi-mode case

In order to solve the multi-mode RCPSP by a branch-and-bound algorithm additionally modes have to be enumerated. The algorithms from Sections 3.8.1 to 3.8.3 can be extended to the multi-mode case as follows.

In the precedence tree algorithm in each stage λ besides an eligible activity $j \in E_\lambda$ also a mode $m \in \mathcal{M}_j$ for the activity has to be chosen. Each combination of an eligible activity and a mode for it corresponds to a successor of the current node in the enumeration tree.

In the algorithm based on extension alternatives additionally modes have to be determined before the activities in an extension alternative can be started. Thus, in this step the modes are enumerated by so-called mode alternatives which consist of all feasible mode combinations for the eligible activities. Enumerating all possible mode alternatives for these activities leads to a refined branching step where each combination of a mode alternative and a corresponding feasible extension alternative leads to a successor of the current search tree node.

The same idea can also be used in the algorithm based on delaying alternatives. Here, in stage λ modes have to be assigned to all eligible activities in $E_\lambda \setminus E_{\lambda-1}$ before they are temporarily started. Enumerating all possible mode alternatives for these activities leads to a refined branching step where each combination of a mode alternative and a corresponding feasible delaying alternative corresponds to a successor of the current search tree node.

For all these algorithms also dominance rules have been generalized. For example, in all algorithms which enumerate partial schedules the following rule may be applied: If an eligible unscheduled activity cannot be feasibly scheduled in any mode in the current partial schedule, then the current schedule can be eliminated.

Furthermore, in addition to local left shifts where the mode of the shifted activity remains the same, so-called **multi-mode left shifts** may be applied. For a given schedule a multi-mode left shift of an activity i reduces the completion time of i without changing the modes or completion times of the other activities (but allowing that the mode of activity i may be changed).

3.8.6 Reference notes

The precedence tree algorithm was proposed by Patterson et al. [162], [163] and improved by additional dominance rules in Sprecher [177]. Furthermore, in Sprecher and Drexl [178] it was generalized to the multi-mode case.

The branch-and-bound algorithm based on extension alternatives is due to Stinson et al. [182]. It was generalized to the multi-mode case by Hartmann and Drexl [94]. A branch-and-bound algorithm based on block extensions was proposed by Mingozzi et al. [145].

The concept of delaying alternatives was first used by Christofides et al. [44] and enhanced by Demeulemeester and Herroelen [56], [57]. In Sprecher and Drexl [179] it was shown that the local left shift rule in connection with the branch-and-bound algorithm based on delaying alternatives does not guarantee that only semi-active schedules are enumerated. The branch-and-bound algorithm based on delaying alternatives was generalized to the multi-mode case by Sprecher et al. [180], to the RCPSP with generalized precedence relations by De Reyck and Herroelen [60] and to the multi-mode RCPSP with generalized precedence relations by De Reyck and Herroelen [61]. Other time-oriented branch-and-bound algorithms for the RCPSP with generalized precedence constraints can be found in Dorndorf et al. [68] and Fest et al. [74].

The branch-and-bound algorithm based on schedule schemes was proposed by Brucker et al. [33] using concepts from Bartusch et al. [15] and Krämer [128]. The mentioned graph theoretical results in connection with comparability graphs can be found in Golumbic [85] and Möhring [146]. Checking whether a transitive orientation exists, can be done by a polynomial-time algorithm of Korte and Möhring [125].

A comparison of exact algorithms for the multi-mode RCPSP was performed by Hartmann and Drexl [94]. Besides computational results for different algorithms a theoretical comparison of the enumerated search spaces (with and without different dominance criteria) is given.

Almost all solution algorithms (exact or heuristic procedures) for the RCPSP, multi-mode RCPSP or the RCPSP with generalized precedence relations were tested on benchmark instances which can be found in the so-called PSPLIB (project scheduling problem library) on the website [169]. These instances and their generation process are described in Kolisch et al. [121], [122] and Kolisch and Sprecher [123].