4

Constructive Algorithms

If the objective function to be minimized is locally regular or locally concave, the relaxation of the resource constraints does not yield a tractable problem in general. Thus, the relaxation-based approach from Chapter 3 no longer proves useful. For solving resource allocation problems with locally regular or locally concave objective function f, we have to explicitly construct the schedules from an appropriate set that contains an optimal schedule if the problem is solvable. We refer to algorithms that proceed in such a way as *constructive algorithms*. The serial schedule-generation scheme for minimizing regular objective functions is an example of a constructive algorithm. In this chapter we develop constructive algorithms that are based on the second basic representation of the set $\mathcal{S}$ of all feasible schedules as a union of disjoint equal-preorder sets (recall that the term equal-preorder set may also designate an equal-order set). As we have seen in Subsections 2.1.1 and 2.1.2, the set of all minimal points or vertices of equal-preorder sets coincides with the set of all minimal points or vertices, respectively, of schedule polytopes. In Subsection 2.2.2 we have introduced the notion of quasiactive and quasistable schedules designating those feasible schedules which represent minimal points or vertices, respectively, of their schedule polytopes. The analysis in Subsection 2.3.2 has shown that for $\mathcal{S} \neq \emptyset$, the set of all quasiactive schedules always contains some optimal schedule if the objective function f under study is locally regular. Likewise, the set of all quasistable schedules always contains an optimal schedule if f is locally concave provided that $\mathcal{S} \neq \emptyset$.

Since we again consider the general case where both renewable and cumulative resources are present, we consider relations ρ in set V of all (real and fictitious) activities. Under our assumption that the real activities use the renewable resources and that the events deplete and replenish the cumulative resources, precedence relationships need only be defined among real activities and among events of the project. More formally, instead of considering schedule-induced preorders $\theta(S) = \{(i,j) \in V \times V \mid S_j \geq S_i + p_i\}$ in set V, we may restrict ourselves to schedule-induced relations arising from the union of the respective schedule-induced strict order in set V^a and the corresponding reflexive preorder in set V^e. Since the ground sets V^a and V^e of those two preorders are disjoint, the union of both preorders is again transitive and thus represents a preorder in ground set $V = V^a \cup V^e$. Accordingly, for given schedule S we define the schedule-induced preorder to be

$$\theta(S) := \{(i,j) \in (V^a \times V^a) \cup (V^e \times V^e) \mid S_j \geq S_i + p_i\}$$

We notice that preorder $\theta(S)$ is neither irreflexive nor reflexive.

Now recall that any quasiactive schedule can be represented as a spanning outtree $G = (V, E_G)$ of its schedule network $N(\theta(S))$ rooted at node 0, where each arc (i,j) of G belongs to one temporal constraint $S_j - S_i \geq \delta_{ij}$ or one precedence constraint $S_j \geq S_i + p_i$ that is active at S (cf. Proposition 2.28). Similarly, any quasistable schedule can be assigned to a spanning tree G of its schedule network. In addition, we assign the weights $\delta_{ij}^G := d_{ij}^{\theta(S)}$ to the arcs $(i,j) \in E_G$. The active temporal and precedence constraints can then be written in the form

$$S_j - S_i \geq \delta_{ij}^G \quad ((i,j) \in E_G)$$

The constructive algorithms are based on generating such spanning outtrees and spanning trees G. The corresponding schedule S is obtained from G by computing the unique solution to the system of linear equations $S_0 = 0$ and $S_j = S_i + \delta_{ij}^G$ $((i,j) \in E_G)$, which can be achieved in linear time. By constructing a spanning outtree or spanning tree G we perform two consecutive tasks simultaneously: first, finding a feasible schedule-induced preorder in set V and second, computing some appropriate vertex (the minimal point if G is an outtree) of the corresponding relation polytope.

Resource allocation problems with locally regular or locally concave objective functions, regardless of containing explicit resource constraints or not, are much harder to solve to optimality on the average than resource-constrained project scheduling problems where some regular or convexifiable objective function is to be minimized. That is why in the present chapter we are concerned with heuristic procedures. In Section 4.1 we first discuss a generic schedule-generation scheme producing one quasiactive or one quasistable schedule. This schedule-generation scheme has been proposed by Neumann et al. (2000) and goes back to priority-rule heuristics for resource levelling problems that have been devised by Neumann and Zimmermann (1999*b*, 2000). The schedule-generation scheme provides an initial quasiactive or quasistable schedule, from which we may subsequently move stepwise towards different quasiactive or quasistable schedules by using an iterative improvement procedure. In Section 4.2 we then deal with tree-based neighborhood functions presented in Neumann et al. (2003*a*). Neighborhood functions constitute the essential building block of local search algorithms such as hill climbing, tabu search, simulated annealing, or threshold accepting (for an overview of different local search techniques, see Aarts and Lenstra 2003*b*). In particular, we show that the proposed neighborhoods allow local search algorithms to reach optimal schedules independently of the initial schedule chosen. Section 4.3 is devoted to additional notes on alternative solution procedures and an experimental performance analysis of the methods discussed.

4.1 Schedule-Generation Scheme

The schedule-generation scheme for constructing quasistable schedules expands the node set C of a subtree G of some schedule network by one node j in each iteration until $C = V$. The algorithm starts with $C = \{0\}$ and iteratively links the activities $j \in V \setminus C$ not yet scheduled with activities $i \in C$. In this way, the start times S_i of all activities $i \in C$ are fixed. They are uniquely determined by the recursion $S_0 = 0$ and $S_i = S_h + \delta_{hi}^G$ if $(h,i) \in E_G$ or $S_i = S_h - \delta_{ih}^G$ if $(i,h) \in E_G$ $(i \in C,\ i \neq 0)$, where h is the predecessor of node i on the (undirected) path from node 0 to node i in G. For a given pair (i,j) with $i \in C,\ j \in V \setminus C$, there are four alternatives of connecting nodes i and j. First, we may either introduce a *forward arc* (i,j) or a *backward arc* (j,i). Now assume that we have chosen forward arc (i,j). Then (i,j) may be weighted by $\delta_{ij}^G = \delta_{ij}$ if (i,j) is contained in project network N or weighted by $\delta_{ij}^G = p_i$ if $d_{ij} < p_i$. In the first case, we speak of a *temporal arc*, and in the second case, the arc is referred to as a *precedence arc*. Likewise, backward arc (j,i) may be weighted by $\delta_{ji}^G = \delta_{ji}$ or by $\delta_{ji}^G = p_j$. If i and j are events, then $S_i + p_i = S_i - p_j = S_i$ and thus the backward precedence arc may be omitted. The arcs have to be chosen in accordance with the temporal constraints. Let

$$ES_j^G := \max[ES_j, \max_{i \in C}(S_i + d_{ij})] \text{ and } LS_j^G := \min[LS_j, \min_{i \in C}(S_i - d_{ji})]$$

denote the earliest and latest start times of activity j given that activities $i \in C$ start at times S_i. The schedule S generated is time-feasible precisely if at any iteration it holds that $ES_j^G \le S_i + \delta_{ij}^G \le LS_j^G$ if a forward arc (i,j) is selected and $ES_j^G \le S_i - \delta_{ji}^G \le LS_j^G$ if a backward arc (j,i) is chosen.

From the viewpoint of implementation, it is expedient to allow the scheduling of activities j at their earliest or latest start times ES_j^G or LS_j^G even if the corresponding temporal arc connects activity j with an activity i that is not scheduled either. We then need not check whether or not activity j can already be linked to some activity $i \in C$, while the set of schedules which can be generated is not affected by this modification. The reason for this is that the same tree G could have been constructed by the original method just by processing the activities j in a different order. Figuratively speaking, the modification means that the directed graph G whose node set C is iteratively expanded may now be unconnected unless $C = V$. Nevertheless, the property that the start times of all scheduled activities $i \in C$ are known as soon as they are added to G is preserved.

Algorithm 4.1 shows an implementation of the procedure for the case where the availability of the resources is not limited, i.e., $R_k = \infty$ for all $k \in \mathcal{R}^\rho$ and

$\underline{R}_k = -\infty, \overline{R}_k = \infty$ for all $k \in \mathcal{R}^\gamma$. As we have mentioned in Subsection 2.3.2, this assumption is generally met in practice when dealing with resource levelling problems, where only renewable resources are taken into account and the resource capacities may be chosen according to the respective requirements. How to modify the schedule-generation scheme in the presence of resource constraints will be explained below. For simplicity, we assume that $\mathcal{S}_T \neq \emptyset$. After the computation of the earliest and latest schedules ES and LS, at each iteration some activity $j \in V \setminus C$ not yet scheduled is selected. Then, the *decision set* $\mathcal{D}_j$ of tentative start times t for j is determined. The conditions on start times t ensure that the resulting schedule S is (time-)feasible and that precedence relationships are only established between activities of the same type (i.e., among real activities and among events; recall our discussion about the proper definition of schedule-induced preorder $\theta(S)$). Finally, some $t \in \mathcal{D}_j$ is selected to be the start time of j and the time windows $[ES_h, LS_h]$ for the activities $h \in V \setminus C$ are updated. These steps are reiterated until all activities have been scheduled. The resulting tree G is a spanning tree of all relation networks $N(\rho)$ for which ρ contains the precedence arcs added and thus in particular a spanning tree of schedule network $N(\theta(S))$. The time complexity of Algorithm 4.1 equals $O(mn)$, which is the time required for calculating the initial earliest and latest schedules.

Algorithm 4.1. Schedule-generation scheme for locally concave objective functions

Input: A project without resource constraints.
Output: A quasistable schedule S.

```
   initialize set of scheduled activities $C := \{0\}$ and set $S_0 := 0$;
   compute earliest and latest schedules $ES$ and $LS$;
   while $C \neq V$ do (* not all activities $j \in V$ scheduled *)
      select an activity $j \in V \setminus C$;
      if $j \in V^a$ then put $V' := V^a$; else put $V' := V^e$;
      add $j$ to $C$ and set $\mathcal{D}_j := \{ES_j, LS_j\}$;
      for all $i \in V' \cap C$ with $ES_j < S_i + p_i < LS_j$ do add $S_i + p_i$ to $\mathcal{D}_j$;
8:    if $V' = V^a$ then
9:       for all $i \in V' \cap C$ with $ES_j < S_i - p_j < LS_j$ do add $S_i - p_j$ to $\mathcal{D}_j$;
10:   select some time $t \in \mathcal{D}_j$ and set $S_j := t$; (* schedule $j$ at time $t$ *)
      for all $h \in V \setminus C$ do (* update earliest and latest start times *)
         set $ES_h := \max(ES_h, S_j + d_{jh})$ and $LS_h := \min(LS_h, S_j - d_{hj})$;
   return $S$;
```

The following proposition (cf. Neumann et al. 2000) establishes the completeness of the schedule-generation scheme. This means that, at least in theory, a resource allocation problem with locally concave objective function can be solved by a brute-force algorithm branching over the activity $j \in V \setminus C$ to be scheduled next and the tentative start time $t \in \mathcal{D}_j$ chosen.

Proposition 4.1 (Neumann et al. 2000).

(a) *Any schedule S generated by using Algorithm 4.1 is quasistable.*
(b) *Any quasistable schedule $S \in QSS$ can be generated by using Algorithm 4.1.*

Proof.

(a) Since the earliest and latest start times are updated in the course of the algorithm, schedule S is (time-)feasible if at each iteration $\mathcal{D}_j \subseteq [ES_j, LS_j]$. It follows from the definition of decision set $\mathcal{D}_j$ that we only need to show that $ES_j \leq LS_j$. Now assume that we have scheduled some activity j' and before the update of the earliest and latest start times it holds that $ES_h \leq LS_h$ for all $h \in V \setminus C$. Then $S_{j'} \leq LS_{j'} \leq LS_h - d_{j'h}$ and $S_{j'} \geq ES_{j'} \geq ES_h + d_{hj'}$ for all $h \in V \setminus C$ and in particular $S_{j'} + d_{j'j} \leq LS_j$ and $S_{j'} - d_{jj'} \geq ES_j$. Consequently, $ES_j > LS_j$ after the update would imply $S_{j'} + d_{j'j} > S_{j'} - d_{jj'}$, i.e., $d_{j'j} + d_{jj'} > 0$, which contradicts $\mathcal{S}_T \neq \emptyset$. Thus, S is feasible. The quasistableness of S now follows from Proposition 2.28b.
(b) We consider some quasistable schedule S and show how to generate S by using Algorithm 4.1. Let G be a spanning tree of schedule network $N(\theta(S))$. The existence of such a spanning tree is guaranteed by Proposition 2.28b. Since G is a tree and the procedure starts with $C = \{0\}$, at each iteration there is some activity $j \in V \setminus C$ whose predecessor i on the path from node 0 to node j in G has already been scheduled. We may then connect j with i by selecting $t = S_i + \delta^G_{ij} \in \mathcal{D}_j$ if $S_j = S_i + \delta^G_{ij}$ and $t = S_i - \delta^G_{ji} \in \mathcal{D}_j$, otherwise. Thus, when $C = V$, the schedule generated coincides with schedule S. □

Now recall that any quasiactive schedule S can be associated with a spanning outtree G of its schedule network $N(\theta(S))$ with root node 0. Such a spanning outtree is obtained if each activity j to be scheduled is linked with some activity $i \in C$ by a forward arc (i, j). Accordingly, a schedule-generation scheme for quasiactive schedules is readily obtained from Algorithm 4.1 by initializing decision set $\mathcal{D}_j$ with $\{ES_j\}$ instead of $\{ES_j, LS_j\}$ and deleting lines 8 and 9. The statements of Proposition 4.1 with "quasistable" and QSS replaced by "quasiactive" and QAS immediately carry over to this modification of Algorithm 4.1.

For what follows, we drop our assumption of infinite renewable-resource capacities. To take account of renewable-resource constraints, in line 10 of Algorithm 4.1 we only select *feasible start times* t from decision set $\mathcal{D}_j$ such that the residual resource capacities suffice to execute activity j in time interval $[t, t + p_j[$, i.e.,

$$\sum_{\substack{i \in C \cap V^a: \\ S_i \leq t' < S_i + p_i}} r_{ik} + r_{jk} \leq R_k \quad (k \in \mathcal{R}^\rho,\ t \leq t' < t + p_j) \tag{4.1}$$

Of course, inequality (4.1) only needs to be evaluated for real activities $j \in V^a$. By using a support-point representation of the resource demand over time that results from the real activities $i \in C \cap V^a$ scheduled, testing (4.1) for given $t \in \mathcal{D}_j$ takes $\mathcal{O}(|\mathcal{R}^\rho|n)$ time. If times t in decision set $\mathcal{D}_j$ are iterated in increasing order, the amortized time complexity for eliminating all infeasible start times from $\mathcal{D}_j$ is $\mathcal{O}(n \log n + |\mathcal{R}^\rho|n)$. By keeping a sorted list of all start and completion times of scheduled activities $i \in C \cap V^a$, the amortized time complexity for checking (4.1) is decreased to $\mathcal{O}(|\mathcal{R}^\rho|n)$ per iteration. Thus, the time complexity of Algorithm 4.1 including the test of inequality (4.1) is $\mathcal{O}(mn + |\mathcal{R}^\rho|n^2)$.

It may happen that no tentative start time $t \in \mathcal{D}_j$ is resource-feasible, which means that the current *partial schedule* $(S_i)_{i \in C}$ cannot be extended to a feasible schedule $S \in \mathcal{S}$. In that case, either the schedule generation is stopped or an unscheduling step is performed. Different unscheduling techniques are known from literature. The method by Franck (1999), Ch. 4, tailored to the case of regular objective functions, has been sketched in Subsection 3.1.4. Further unscheduling procedures have been devised by Neumann and Zimmermann (1999*b*, 2000) (see also Zimmermann 2001*a*, Sect. 3.2, and Neumann et al. 2003*b*, Sect. 3.7), where those activities $i \in C$ are unscheduled whose start at a different time frees capacity for processing activity j. Alternatively, one may also generate a time-feasible schedule S using Algorithm 4.1 first and then resolve resource conflicts by left- or right-shifting certain activities. This approach corresponds to *schedule-repair methods* described by Neumann and Zimmermann (2000). We finally notice that the proof of Proposition 4.1 remains valid for the case of renewable-resource constraints, which means that even without unscheduling, any quasistable schedule may still be generated using Algorithm 4.1 with reduced decision sets $\mathcal{D}_j$.

For certain choices of tentative start times $t \in \mathcal{D}_j$ one obtains specific types of schedules. If at each iteration we select $t = \min \mathcal{D}_j$ and no unscheduling step is performed, the resulting schedule is active because each activity is scheduled at its earliest feasible start time. Likewise, by always choosing $t = \min \mathcal{D}_j$ or $t = \max \mathcal{D}_j$ we obtain a stable schedule. The following example, however, shows that due to the presence of maximum time lags, not all active or stable schedules can be generated in this way.

Example 4.2. We consider a project with one renewable resource of capacity $R = 1$ and four real activities $i = 1, 2, 3, 4$ with durations $p_i = 1$ and resource requirements $r_i = 1$ $(i = 1, \ldots, 4)$. The project network N is depicted in Figure 4.1a. Clearly, there is precisely one feasible schedule $S = (0, 1, 2, 3, 4, 5)$, which, as a consequence, is active and stable. Schedule S is illustrated by the Gantt chart shown in Figure 4.1b, where each real activity $i \in V^a$ is represented as a box of length p_i and height r_i over the time axis from S_i to C_i.

The start times of activities 0, 1, 4, and 5 are fixed by the prescribed time lags because the corresponding nodes form a cycle of length zero in N. If in

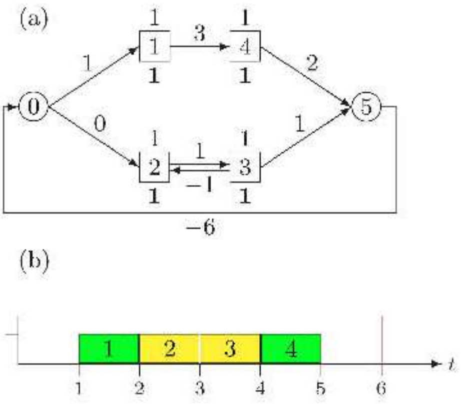

Fig. 4.1. Incompleteness of schedule-generation scheme for active or stable schedules: (**a**) project network N; (**b**) Gantt chart for unique feasible schedule S

the course of the algorithm, activity 2 is scheduled before activity 3, we obtain $\min \mathcal{D}_2 = 0$ and $\max \mathcal{D}_2 \geq 3$. Symmetrically, if activity 3 is scheduled before activity 2, we have $\min \mathcal{D}_3 \leq 2$ and $\max \mathcal{D}_3 = 5$. Hence, the unique feasible schedule S cannot be generated if solely scheduling at earliest or latest feasible start times is considered.

We obtain the schedule-generation scheme of the priority-rule methods for resource levelling proposed by Neumann and Zimmermann (1999*b*, 2000) if start time $t \in \mathcal{D}_j$ is always chosen to be the greatest minimizer of an *additional-cost function* f_j^a on $\mathcal{D}_j$, i.e., $t = \max \arg\min_{t' \in \mathcal{D}_j} f_j^a(t')$. For given $t' \in \mathcal{D}_j$, $f_j^a(t')$ is the increase in the objective function value if activity j is scheduled at time t' given partial schedule $(S_i)_{i \in C}$, where we put $r_{hk} := 0$ for all activities $h \in V \setminus C$ not yet scheduled and all $k \in \mathcal{R}^\rho$. Similarly to Example 4.2 it can be shown that the restriction to locally optimal tentative start times $t \in \arg\min_{t' \in \mathcal{D}_j} f_j^a(t')$ generally implies that the schedule-generation scheme is no longer complete, which means that one may miss the optimum even if all sequences in which activities j are scheduled are enumerated.

In the case where the availability of cumulative resources is limited as well, the feasibility of the generated schedule can no longer be ensured by iterating partial schedules which observe the resource constraints. The reason for this is that a partial schedule leading to a shortage or a surplus in some cumulative resource may be extended to a feasible schedule. Nevertheless, we may still exclude certain tentative start times from further consideration by computing, for given partial schedule, lower and upper bounds on the inventory in cumulative resources.

Let $(S_i)_{i \in C}$ be the partial schedule under consideration and assume that we want to test whether event $j \in V^e \setminus C$ can be scheduled at time $S_j = t \in \mathcal{D}_j$. By D' we denote the distance matrix for the expanded project network N' where for each $h \in C \cup \{j\}$ we add the two arcs $(0, h)$ and $(h, 0)$ weighted by

$\delta_{0h} = S_h$ and $\delta_{h0} = -S_h$ to project network N. The set $\mathcal{S}'_T \subseteq \mathcal{S}_T$ of all time-feasible schedules belonging to project network N' coincides with the set of all schedules that can be obtained by extending $(S_h)_{h \in C \cup \{j\}}$ to a time-feasible schedule. If for all schedules $S \in \mathcal{S}'_T$, the inventory level at time S_j either falls below the safety stock or exceeds the storage capacity, i.e.,

$$r_k(S, S_j) < \underline{R}_k \text{ or } r_k(S, S_j) > \overline{R}_k \text{ for some } k \in \mathcal{R}^\gamma \tag{4.2}$$

then event j cannot be scheduled at time $t = S_j$ because $\mathcal{S}'_T \cap \mathcal{S} = \emptyset$. In this case, tentative start time t can be deleted from decision set $\mathcal{D}_j$. (4.2) holds true for any schedule $S \in \mathcal{S}'_T$ precisely if for some cumulative resource $k \in \mathcal{R}^\rho$, the maximum inventory $\max_{S \in \mathcal{S}'_T} r_k(S, S_j)$ at time S_j is less than safety stock $\underline{R}_k$ or the minimum inventory $\min_{S \in \mathcal{S}'_T} r_k(S, S_j)$ at time S_j exceeds storage capacity $\overline{R}_k$. The problems of computing the maximum and minimum inventories have been addressed in Subsection 2.1.2, where we have been concerned with checking the feasibility of a given relation ρ in set V^e. In the latter context, we have shown that maximizing or minimizing $r_k(\cdot, S_j)$ on a relation polytope $\mathcal{S}_T(\rho)$ can be stated as a binary program with totally unimodular coefficient matrix, the dual of whose continuous relaxation is a minimum-flow problem (see (2.2) and (2.4)). We obtain analogous formulations of our present problems if we choose reflexive preorder θ in (2.2) to be the preorder $\theta = \Theta(D')$ induced by distance matrix D'. Since solving a minimum-flow problem takes $\mathcal{O}(n^3)$ time, the computational effort for testing the feasibility of a tentative start time $t \in \mathcal{D}_j$ is $\mathcal{O}(|\mathcal{R}^\gamma| n^3)$. Hence, the time complexity of the variant of Algorithm 4.1 coping with renewable-resource and cumulative-resource constraints is $\mathcal{O}(|\mathcal{R}^\rho| n^2 + |\mathcal{R}^\gamma| n^5)$.

Alternatively, resource constraints can be taken into account by combining the relaxation-based and constructive approaches into a *two-phase method*. In phase 1, we determine a feasible relation ϱ in set V. In phase 2, we generate a vertex of relation polytope $\mathcal{S}_T(\varrho) \subseteq \mathcal{S}$ (i.e., a quasistable schedule) by using a variant of Algorithm 4.1 where the original project network N is replaced with relation network $N(\varrho)$. A feasible relation ϱ in set V can be generated using a modification of the enumeration scheme given by Algorithm 3.3. In the modified version, forbidden sets F are given by antichains U and unions of predecessor sets U in preorders $\theta = \Theta(D(\rho))$ rather than by active sets $\mathcal{A}(S, t)$ for minimizers S on relation polytopes $\mathcal{S}_T(\rho)$. For given relation ρ, those sets U can be determined by solving the minimum-flow problems discussed in Subsections 2.1.1 and 2.1.2 for the restrictions of ρ to sets V^a and V^e, respectively. The solutions to the dual problems, i.e., the maximum (s,t)-cuts in the respective flow networks, then provide the activity sets U sought (for details we refer to Section 5.2, where we shall use a similar technique for computing forbidden active sets when sequence-dependent changeover times arise between the execution of activities that are executed at different locations). If no set U is forbidden any longer, we have obtained a feasible relation $\varrho = \rho$.

4.2 Local Search

The schedule constructed by using the schedule-generation scheme may be improved by performing a local search in the set $\mathcal{QAS}$ of all quasiactive schedules if objective function f is locally regular or in the set $\mathcal{QSS}$ of all quasistable schedules if f is locally concave. Starting with some initial solution, local search algorithms try to find better solutions by exploring neighborhoods (cf. Aarts and Lenstra 2003a). The neighborhoods are given by a *neighborhood function* $\mathcal{N} : \Sigma \to \mathbb{P}(\Sigma)$ mapping the *set of solutions* Σ into the power set of Σ. For each solution $s \in \Sigma$, $\mathcal{N}$ defines a set $\mathcal{N}(s)$ of *neighboring solutions* s'. $\mathcal{N}(s)$ is called the *neighborhood* of s, and neighboring solutions $s' \in \mathcal{N}(s)$ are referred to as *neighbors* of s.

A neighborhood function $\mathcal{N}$ can be represented by its (directed) *neighborhood graph* $\mathcal{G}$ with node set Σ. Two nodes s and s' are linked by an arc (s, s') in $\mathcal{G}$ precisely if s' is a neighbor of s, where it may happen that s' is a neighbor of s but not vice versa. Local search can be regarded as a directed walk in neighborhood graph $\mathcal{G}$. Graph $\mathcal{G}$ is called *weakly optimally connected* if from any node s of $\mathcal{G}$, there is a directed path from s to some optimal solution s^*. If $\mathcal{G}$ is weakly optimally connected, an optimal solution s^* can be reached from any initial solution just by iteratively moving from solutions s to appropriate neighboring solutions s'. Obviously, $\mathcal{G}$ is weakly optimally connected if it is strongly connected.

In this section we review neighborhoods for resource allocation problems with locally regular or locally concave objective functions f that have been proposed by Neumann et al. (2003a). We first deal with the case of locally concave objective functions and then explain how to adapt the neighborhood to locally regular objective functions. Recall that each quasistable schedule S can be represented by a spanning tree $G = (V, E_G, \delta^G)$ of its schedule network $N(\theta(S))$ such that S is the unique solution to the system of linear equations $S_0 = 0$ and $S_j - S_i = \delta_{ij}^G$ $((i,j) \in E_G)$. That is why we identify the set of solutions Σ with the set Σ^{st} of all spanning trees of schedule networks $N(\theta)$ where $\theta \in \mathcal{SIP}$ is some schedule-induced preorder in set V.

The starting point for constructing a neighborhood function $\mathcal{N}^{st}$ on set Σ^{st} is the observation that first, two spanning trees in set Σ^{st} differing in only one arc always belong to either coinciding or adjacent vertices of some schedule polytope and that second, for any two adjacent vertices of a schedule polytope, there exist two corresponding spanning trees in set Σ^{st} which differ in exactly one arc. Roughly speaking, we determine neighbors G' of a spanning tree G by removing a *leaving arc* (i, j) from G and adding a different *entering arc* (i', j') to G such that the resulting directed graph G' is again a tree. By deleting arc (i, j), G decomposes into two subtrees with node sets $C \supseteq \{0\}$ and $C' = V \setminus C$. Let S and S' be the quasistable schedules that are represented by spanning trees G and G', respectively. Obviously, $S'_h = S_h$ for all $h \in C$ and $S'_h = S_h + \sigma$ for all $h \in C'$ or $S'_h = S_h - \sigma$ for all $h \in C'$ and some stepsize $\sigma \geq 0$. In other words, when moving from S to S' we uniformly shift all activities h of C' by some $\sigma \geq 0$ until a new inequality $S_{j'} - S_{i'} \geq \delta_{i'j'}^{G'}$ with $i' \in C'$, $j' \in C$ or $i' \in C$, $j' \in C'$ becomes active. Arc (i', j') may be a temporal arc or a precedence arc. Similarly to the steepest descent and flattest ascent methods discussed in Section 3.2, stepsize σ may be equal to 0 and thus $S' = S$ if S is a degenerate vertex of its schedule polytope $\mathcal{S}_T(\theta(S))$.

Now recall that we refer to (g, h) as a forward arc of G if g is the predecessor of h on the (undirected) path from 0 to h in G, and as a backward arc of G, otherwise. If leaving arc (i, j) is a forward arc of G, then $i \in C$ and $j \in C'$, and if (i, j) is a backward arc of G, then $i \in C'$ and $j \in C$. For what follows we associate a direction z with leaving arc (i, j) with $z_h = 0$ for all $h \in C$ and $z_h = 1$ for all $h \in C'$ if (i, j) is a forward arc and $z_h = -1$ for all $h \in C'$ if (i, j) is a backward arc. Let (g, h) be an arc in some schedule network. We say that set C' is *shifted along* arc (g, h) if $z_h - z_g = 1$. If $z_h - z_g = -1$, we speak of a *shift against* arc (g, h). Clearly, a shift of C' against leaving arc (i, j) is only meaningful if (i, j) is a precedence arc. In that case, the precedence relationship between activities i and j is deleted when passing from S to neighboring schedule S'. Symmetrically, a shift along an entering temporal arc is not possible. If we shift C' along leaving arc (i, j), then $S' = S + \sigma z$, and for a shift against leaving arc (i, j) we have $S' = S - \sigma z$. Before we describe neighborhood function $\mathcal{N}^{st}$ in more detail, we consider the four cases that may occur when shifting set C'. For illustration, we consider the spanning tree G and the corresponding Gantt chart displayed on the top of Figure 4.2, where for simplicity we have omitted the arc weights. We assume that the underlying project has one renewable resource and that the real activities $i = 1, 2, 3$ are unrelated and can be started at the project beginning. Thus, all arcs $(g, h) \in E_G$ are precedence arcs.

(a) We shift C' along leaving arc (i, j) and against entering arc (i', j'). This means that the schedule-induced preorder remains unchanged when passing from S to S', i.e., $\theta(S') = \theta(S)$, or, in other words, $S' \in \mathcal{S}_T^{=}(\theta(S))$. This case is shown in Figure 4.2a, where $(i, j) = (0, 1)$ and $(i', j') = (4, 0)$. In the resulting spanning tree G', the activities $h \in C'$ shifted are drawn in bold.

(b) We shift C' along leaving arc (i, j) and along entering precedence arc (i', j'). This means that the schedule-induced preorder is augmented when passing from S to $S' \neq S$, i.e., $\theta(S') \supset \theta(S)$. This case is shown in Figure 4.2b, where $(i, j) = (1, 3)$ and $(i', j') = (2, 3)$.

(c) We shift C' against leaving precedence arc (i, j) and against entering arc (i', j'). This means that the schedule-induced preorder is reduced when passing from S to $S' \neq S$, i.e., $\theta(S') \subset \theta(S)$. Such a shift is always opposite to a shift augmenting the schedule-induced preorder (case (b)). This case is shown in Figure 4.2c, where $(i, j) = (1, 3)$ and $(i', j') = (0, 3)$.

(d) We shift C' against leaving precedence arc (i, j) and along entering precedence arc (i', j'). This means that $\theta(S') \not\supseteq \theta(S)$ and $\theta(S') \not\subseteq \theta(S)$ if $S' \neq S$. This case is shown in Figure 4.2d, where $(i, j) = (1, 2)$ and $(i', j') = (2, 3)$.

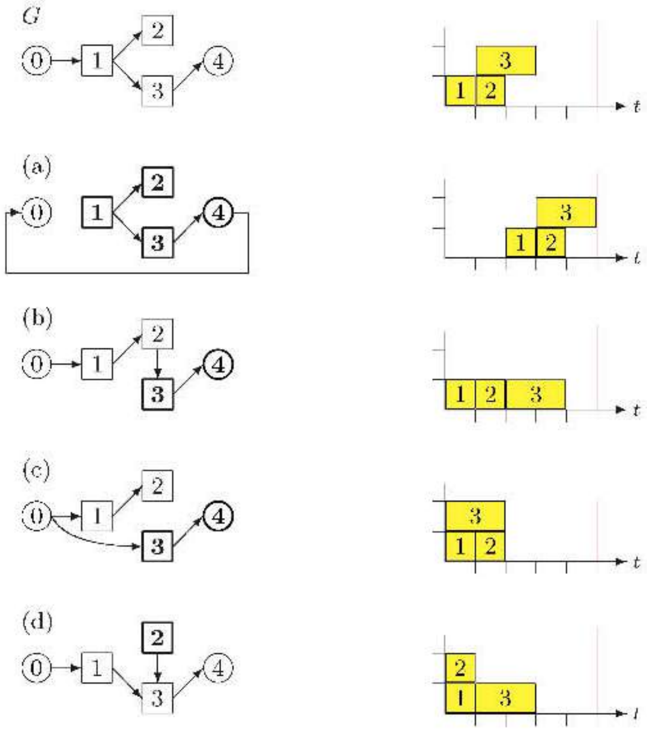

Fig. 4.2. Cases occurring when shifting set C': (**a**) shift along leaving and against entering arc; (**b**) shift along leaving and along entering arc; (**c**) shift against leaving and against entering arc; (**d**) shift against leaving and along entering arc

In all four cases, the resulting schedule S' either coincides with S (which may happen when S is a degenerate vertex of its schedule polytope) or S' is a vertex adjacent to S in the closure of equal-order set $\mathcal{S}_T^{=}(\theta(S''))$ of some schedule S''. In cases (a) and (b), S and S' are adjacent vertices of the closure of $\mathcal{S}_T^{=}(\theta(S))$. In cases (a) and (c), S and S' are adjacent vertices of the closure of $\mathcal{S}_T^{=}(\theta(S'))$. In case (d), S and S' are adjacent vertices of the closure of $\mathcal{S}_T^{=}(\theta(S''))$ with $S'' := \frac{1}{2}(S + S')$.

A neighbor $G' \in \mathcal{N}^{st}(G)$ can be determined in two steps. First, we delete an arc (i,j) from G. Then, we shift set C' until a temporal or precedence constraint corresponding to some arc (i',j') becomes active. If (i,j) is a temporal arc, C' can only be shifted along (i,j). C' can be shifted along or against (i,j) if (i,j) is a precedence arc. Finally, we add arc (i',j') to G and obtain spanning tree G'. Since G contains $n+1$ arcs (i,j), which all may leave G, and because we may shift either along or against (i,j), the size of neighborhood $\mathcal{N}^{st}(G)$ is of order $\mathcal{O}(n)$.

Next, we define a neighborhood function $\mathcal{N}^{ot}$ on the set $\Sigma^{ot} \subseteq \Sigma^{st}$ of spanning outtrees G of schedule networks $N(\theta)$ where $\theta \in \mathcal{SIP}$. Those spanning outtrees represent minimal points of schedule polytopes $S_T(\theta)$. A tree G is an outtree with root node 0 precisely if all arcs $(g, h) \in E_G$ are forward arcs. Hence, to obtain a spanning outtree $G' \in \Sigma^{ot}$ from a spanning outtree $G \in \Sigma^{ot}$ such that G and G' differ in exactly one arc, the leaving arc (i, j) must be replaced by an entering arc $(i', j') \neq (i, j)$ such that G' is a tree and (i', j') is a forward arc in G'. Clearly, both conditions are satisfied precisely if $j' = j$. Since (i, j) is a forward arc in G, in addition we have $z_j \in \{0, 1\}$. This implies that if we shift along leaving arc (i, j), we necessarily shift along entering precedence arc (i', j), and if we shift against leaving precedence arc (i, j), we necessarily shift against entering arc (i', j) (see Figures 4.2b and 4.2c). We obtain a neighbor $G' \in \Sigma^{ot}$ of G as follows. An arc (i, j) can be chosen to be the leaving arc if the first constraint that becomes active when shifting set C' corresponds to an arc (i', j) with terminal node j. After the selection of an appropriate leaving arc (i, j) we proceed analogously as for neighborhood function $\mathcal{N}^{st}$. We first delete (i, j) from G, then shift set C' until the constraint corresponding to entering arc (i', j) becomes active, and finally add arc (i', j) to G. The size of neighborhood $\mathcal{N}^{ot}(G)$ is again of order $\mathcal{O}(n)$.

Proposition 4.3 (Neumann et al. 2003*a*). *The neighborhood graphs $\mathcal{G}^{st}$ and $\mathcal{G}^{ot}$ of (a) neighborhood function $\mathcal{N}^{st}$ and (b) neighborhood function $\mathcal{N}^{ot}$ are strongly connected.*

Proof.

(a) Clearly, each spanning outtree G representing the earliest schedule ES can be reached from any other spanning tree $G \in \Sigma^{st}$ by performing a sequence of (left-)shifts along a backward leaving arc or against a forward leaving arc. This proves $\mathcal{G}^{st}$ to be weakly connected. Moreover, each shift of type (a), (b), (c), or (d) transforming some spanning tree G into a different neighboring spanning tree G' is reversible because the opposite shift is of type (a), (c), (b), or (d), respectively. Consequently, any two adjacent nodes in $\mathcal{G}^{st}$ are linked by a pair of oppositely directed arcs, i.e., $\mathcal{G}^{st}$ is symmetric (see, e.g., Bang-Jensen and Gutin 2002, Sect. 1.6). From the weak connectivity and the symmetry of $\mathcal{G}^{st}$ it follows that $\mathcal{G}^{st}$ is strongly connected.

(b) $\mathcal{G}^{ot}$ is the subgraph of $\mathcal{G}^{st}$ that is induced by set Σ^{ot} and thus $\mathcal{G}^{ot}$ is symmetric as well. The weak connectivity of $\mathcal{G}^{ot}$ follows from the fact that the spanning outtrees representing schedule ES can be obtained from any outtree $G \in \Sigma^{ot}$ by successively shifting against leaving precedence arcs (i, j). □

4.3 Additional Notes and References

Locally regular and locally concave objective functions have essentially been studied in the context of resource levelling problems, where one strives at smoothing the utilization of renewable resources over time. Resource levelling problems have been investigated since the very beginning of algorithmic project planning in the early 1960s. An overview of different problem settings and solution procedures can be found in Zimmermann (2001*a*), Ch. 5, and Kimms (2001*a*), Sect. 11.3. Resource levelling procedures for the case of general temporal constraints have first been proposed by Brinkmann and Neumann (1996), who have devised simple priority-rule methods where the activities are scheduled one after the other according to a *quasi-topological ordering* $\prec$ of the nodes in project network N. Strict order $\prec$ arises from arc set E by deleting all arcs with nonpositive weight (i.e., the maximum time lags) and taking the transitive hull of the resulting relation. An activity h becomes eligible for scheduling as soon as all its predecessors i with respect to strict order $\prec$ have been processed, i.e., $Pred^{\prec}(h) \subseteq C$. Among the eligible activities h, an activity j is selected by using a priority rule and j is scheduled at a minimizer t of additional-cost function f^a on set $[ES_j, LS_j] \cap \mathbb{Z}$. Since f^a is evaluated on set $[ES_j, LS_j] \cap \mathbb{Z}$ by complete enumeration, the heuristic shows a pseudo-polynomial time complexity.

Neumann and Zimmermann (1999*b*, 2000) have streamlined this approach in different respects. First, instead of scanning all integral times $t \in [ES_j, LS_j]$, only the relevant tentative start times t from decision set $\mathcal{D}_j$ are investigated (see Section 4.1). Second, the concept of *core loading profiles* r_k^c (see Subsection 1.2.4) is used to anticipate (a part of) the unavoidable resource usage by activities $h \in V^a \setminus C$ not yet scheduled. In doing so, deadlocks where $\mathcal{D}_j = \emptyset$ can more likely be avoided when resource constraints have to be taken into account. The definition of additional-cost function f_j^a is based on the core loading profiles, which means that the cost $f_j^a(t)$ of starting activity j at time $t \in \mathcal{D}_j$ arises from comparing the costs associated with the core loading profiles before and after putting $S_j := t$. A third improvement on Brinkmann and Neumann's procedure is the use of different unscheduling techniques invoked when no feasible start time can be assigned to activity j (for details see Section 4.1).

Neumann and Zimmermann (2000) have also proposed a *tabu search procedure* for resource levelling, which in principle is as follows (for an introduction to tabu search we refer to Glover and Laguna 1997 or Hertz et al. 2003). Given some time-feasible schedule S, a neighboring schedule S' is constructed by selecting a real activity $j \in V^a$ such that $r_k(S, S_j) \geq \lambda \max_{0 \leq t \leq \bar{d}} r_k(S, t)$, where λ with $0 < \lambda \leq 1$ is a control parameter. Then activity j is shifted behind or in front of some activity $i \in V^a$, i.e., $S_j' = S_i + p_i$ or $S_j' = S_i - p_j$. Subsequently, the time-feasibility of resulting schedule S' is restored. The move from S to S' is only accepted if $r_k(S', t) < r_k(S, t)$ for some resource $k \in \mathcal{R}^\rho$ and some "peak time" $t \in \arg\max_{0 \leq t' \leq \bar{d}} r_k(S, t')$. In general, the schedules S iterated are not resource-feasible. That is why they are evaluated on the basis of a cost function including a penalty term for violations of the resource constraints. The penalty term is similar to that used by Schwindt (2000*b*) in the local search algorithm for the earliness-tardiness problem with renewable resources (see Subsection 3.2.5).

For solving the resource investment problem, Nübel (1999) has proposed a branch-and-bound algorithm that (implicitly) makes use of the property that the total procurement cost represents a preorder-decreasing objective function (see Subsection 2.3.3). The principle of this branch-and-bound algorithm is to introduce fictitious resource capacities that are stepwise decreased at certain enumeration nodes. Starting with the earliest schedule $S = ES$ at the root node, the capacity R_k of some resource $k \in \mathcal{R}^\rho$ is put to $\max_{0 \le t \le \overline{d}} r_k(S,t) - 1$ and a new schedule S is sought with $r_k(S,t) \le R_k$ for all $0 \le t \le \overline{d}$ by using the enumeration scheme for regular objective functions given by Algorithm 3.1. Reduction of fictitious resource capacities and computation of quasiactive schedules with lower maximum resource requirements are reiterated until no feasible schedule with $S_{n+1} \le \overline{d}$ can be found any more. Each time a new feasible schedule has been found, one branches over the resource k whose capacity is decreased next.

Based on an enumeration scheme by Patterson et al. (1989) for project scheduling subject to precedence and renewable-resource constraints, Neumann and Zimmermann (2000) have developed a time-based branch-and-bound procedure. The algorithm is capable of solving arbitrary resource allocation problems for which an optimal schedule can be chosen to be integer-valued. The latter condition is obviously always satisfied if the objective function is locally regular or locally concave because any quasistable schedule is integral. The nodes of the enumeration tree are associated with partial schedules $(S_i)_{i \in C}$ satisfying the temporal and resource constraints. Starting with $C = \{0\}$ and $S_0 = 0$, at each level an activity j from set $V \setminus C$ with minimum total float $TF_j = LS_j - ES_j$ is added to C. For each integral start time t in the current time window $[ES_j, LS_j]$ of j, a corresponding child node with $S_j = t$ is generated and the time windows of the activities $h \in V \setminus C$ not yet scheduled are updated. Leaves of the enumeration node correspond to feasible, not necessarily quasistable schedules.

Next, we discuss the results of an experimental performance analysis for the time-constrained resource investment and total squared utilization cost problems. We compare a tabu search implementation of Neumann et al.'s local search principle discussed in Section 4.2 to some of the alternative solution procedures. The test set has been created using project generator ProGen/max and contains 90 projects with 500 activities and 1, 3, or 5 resources each. For all algorithms a time limit of 100 seconds has been imposed, which refers to a Pentium personal computer with 200 MHz clock pulse. The results were communicated by Zimmermann (2001*b*).

Table 4.1 shows the results obtained for the **resource investment problem**, where besides the tree-based tabu search procedure ("TS") we have tested truncated versions (filtered beam search, "FBS") of the branch-and-bound algorithms of Nübel (1999) and Neumann and Zimmermann (2000). Since tight lower bounds for large resource levelling problems are not available, we give the mean deviation Δ_{best} from the objective function value of a best solution found by the three procedures. p_{best} denotes the percentage of instances for which the respective method has found a best solution (the values sum to more than 100 % because for some instances, a best solution was found by more than one procedure).

Table 4.1. Performance of algorithms for the resource investment problem

Algorithm	Δ_{best}	p_{best}
Nübel (1999) FBS	23 %	6.7 %
Neumann and Zimmermann (2000) FBS	18 %	11.1 %
Neumann et al. (2003*a*) TS	3 %	90.0 %

The data from Table 4.1 suggest that the tabu search heuristic provides markedly better schedules on the average than the truncated exact algorithms. For 81 out of the 90 projects, the tree-based approach yields a best solution. In addition, the mean deviation from the best solution found is considerably smaller than for the two other algorithms.

The results for the **total squared resource utilization cost problem** are given in Table 4.2. The tree-based tabu search procedure has been compared to the priority-rule ("PR") and tabu search ("TS") methods by Neumann and Zimmermann (2000). The priority-rule method has been run as a multi-pass procedure with ten priority rules. Again, we give the mean deviation Δ_{best} from the best objective function value and the percentage p_{best} of best solutions found.

Table 4.2. Performance of algorithms for the total squared utilization cost problem

Algorithm	Δ_{best}	p_{best}
Neumann and Zimmermann (2000) PR	10 %	4.4 %
Neumann and Zimmermann (2000) TS	3 %	43.3 %
Neumann et al. (2003*a*) TS	1 %	68.9 %

Not surprisingly, the schedule-improvement procedures outperform the priority-rule method. As for the resource investment problem, the tree-based approach again shows the best performance among the tested algorithms. Compared to the tabu search of Neumann and Zimmermann (2000), the favorable behavior is probably due to the small size of the neighborhoods to be explored and the little time needed for moving from one schedule to another.