Chapter 1

The Resource-Constrained Project Scheduling Problem

1.1. A combinatorial optimization problem

Informally, a resource-constrained project scheduling problem (RCPSP) considers resources of limited availability and activities of known durations and resource requests, linked by precedence relations. The problem consists of finding a schedule of minimal duration by assigning a start time to each activity such that the precedence relations and the resource availabilities are respected.

More formally, the RCPSP can be defined as a combinatorial optimization problem. A combinatorial optimization problem is defined by a solution space $\mathcal{X}$, which is discrete or which can be reduced to a discrete set, and by a subset of feasible solutions $\mathcal{Y} \subseteq \mathcal{X}$ associated with an objective function $f : \mathcal{Y} \rightarrow \mathbb{R}$. A combinatorial optimization problem aims at finding a feasible solution $y \in \mathcal{Y}$ such that $f(y)$ is minimized or maximized. A resource-constrained project scheduling problem is a combinatorial optimization problem defined by a tuple $(V, p, E, \mathcal{R}, B, b)$.

Activities constituting the project are identified by a set $V = \{A_0, \dots, A_{n+1}\}$. Activity A_0 represents by convention the start of the schedule and activity A_{n+1} symmetrically represents the end of the schedule. The set of non-dummy activities is identified by $\mathcal{A} = \{A_1, \dots, A_n\}$.

Durations are represented by a vector p in $\mathbb{N}^{n+2}$ where p_i is the duration of activity A_i, with special values $p_0 = p_{n+1} = 0$.

Chapter written by Christian ARTIGUES.

Precedence relations are given by E, a set of pairs such that $(A_i, A_j) \in E$ means that activity A_i precedes activity A_j. A precedence activity-on-node graph $G(V, E)$ is defined where nodes correspond to activities and arcs correspond to precedence relations[1]. We assume that G contains no cycle, otherwise the precedence relations are obviously inconsistent. Since precedence is a transitive binary relation, the existence of a path in G from node A_i to node A_j also means that activity A_i precedes activity A_j. Thus, all precedence graphs having the same transitive closure define the same precedence constraints. We assume that, taking account of the preceding remark, E is such that A_0 is a predecessor of all other activities and A_{n+1} is a successor of all other activities.

Renewable resources are formalized by set $\mathcal{R} = \{R_1, \dots, R_q\}$.

Availabilities of resources are represented by a vector B in $\mathbb{N}^q$ such that B_k denotes the availability of R_k. In particular, a resource R_k such that $R_k = 1$ is called a unary or disjunctive resource. Otherwise, as a resource may process several activities at a time, it is called a cumulative resource.

Demands of activities for resources are abstracted by b, a $(n+2) \times q$ integer matrix, such that b_{ik} represents the amount of resource R_k used per time period during the execution of A_i.

A *schedule* is a point S in $\mathbb{R}^{n+2}$ such that S_i represents the start time of activity A_i. C_i denotes the completion time of activity A_i, with $C_i = S_i + p_i$. S_0 is a reference point for the start of the project. Here we assume that $S_0 = 0$. A solution S is feasible if it is compatible with the precedence constraints (1.1) and the resource constraints (1.2) expressed below, where $\mathcal{A}_t = \{A_i \in \mathcal{A} \mid S_i \leq t < S_i + p_i\}$ represents the set of non-dummy activities in process at time t.

$$S_j - S_i \geq p_i \quad \forall (A_i, A_j) \in E \tag{1.1}$$

$$\sum_{A_i \in \mathcal{A}_t} b_{ik} \leq B_k \quad \forall R_k \in \mathcal{R},\ \forall t \geq 0 \tag{1.2}$$

The makespan of a schedule S is equal to S_{n+1}, the start time of the end activity. The above-defined set $\mathcal{A}_t$ and constraints state that an activity cannot be interrupted once it is started. This is referred to as not allowing *preemption*[2]. The RCPSP can then be stated as follows:

1. We will identify in the rest of the chapter each activity with the corresponding node of the precedence graph.

2. The preemptive case is presented in Chapter 8.

DEFINITION 1.1.– *The RCPSP is the problem of finding a non-preemptive schedule S of minimal makespan S_{n+1} subject to precedence constraints (1.1) and resource constraints (1.2).*

An important preliminary remark is that, since durations are integers, we can restrict ourselves to integer schedules without missing the optimal solution. A non-integer feasible schedule can be transformed into an integer feasible schedule without increasing the makespan by recursively applying the following principle. Consider a non-integer schedule S and let A_i denote the first activity in the increasing start time order such that $S_i \notin \mathbb{N}$. Then setting S_i to its nearest lower integer $\lfloor S_i \rfloor$ does not violate any precedence constraints, since the completion time of the predecessors of A_i are integers strictly lower than S_i. Left shifting an activity can violate a resource constraint only if it enters new sets $\mathcal{A}_t$ for $\lfloor S_i \rfloor \leq t < S_i$. Since we have $\mathcal{A}_t \subseteq \mathcal{A}_{S_i} \setminus \{i\}$ for $\lfloor S_i \rfloor \leq t < S_i$, no resource-constraint violation can appear by setting S_i to $\lfloor S_i \rfloor$. The set of integer schedules, containing at least one optimal solution, is said to be dominant.

1.2. A simple resource-constrained project example

In Table 1.1, a RCPSP instance is given with $n = 10$ real activities and $|\mathcal{R}| = 2$ resources with availabilities $B_1 = 7$ and $B_2 = 4$.

A_i	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}
p_i	0	6	1	1	2	3	5	6	3	2	4	0
b_{i1}	0	2	1	3	2	1	2	3	1	1	1	0
b_{i2}	0	1	0	1	0	1	1	0	2	2	1	0

Table 1.1. *A project with 10 real activities and 2 resources*

The precedence constraints linking the activities $A_i \in V$ are displayed in Figure 1.1 as an activity-on-node graph. A schedule of minimal makespan $S^*_{n+1} = 12$ is displayed in Figure 1.2 as a 2-dimensional Gantt chart where the x axis represents the time and the y axis represents the resource occupation.

1.3. Computational complexity

According to the computational complexity theory [GAR 79], the RCPSP is one of the most intractable combinatorial optimization problems. Indeed, the RCPSP belongs to the class of problems that are *NP-hard in the strong sense*. The complexity theory states that an optimization problem is NP-hard (in the strong sense) if its decision version is *NP-complete* (in the strong sense). Let us define the decision variant of the RCPSP. Let H denote an arbitrarily large integer.

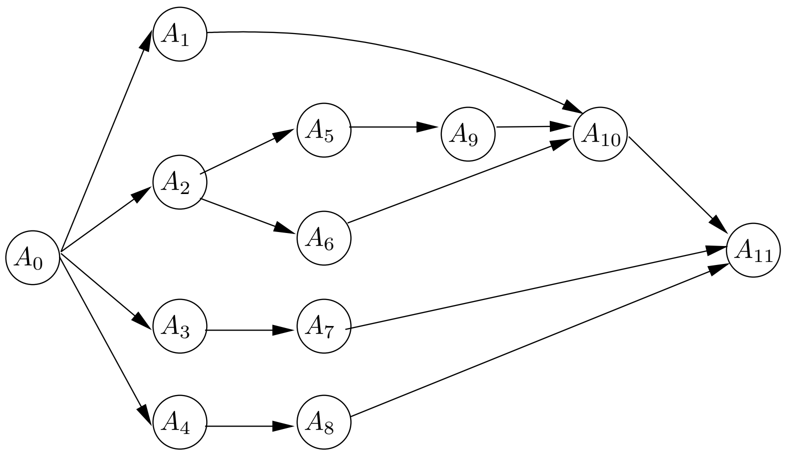

Figure 1.1. *Precedence activity-on-node graph*

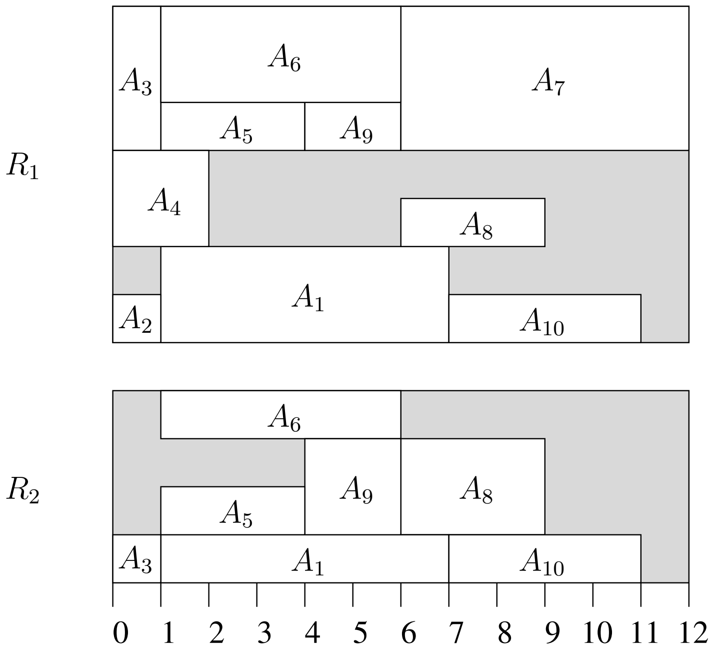

Figure 1.2. *Schedule of minimal makespan*

DEFINITION 1.2.– *The decision variant of the RCPSP is the problem of determining whether a schedule S of makespan S_{n+1} not greater than H subject to precedence constraints (1.1) and resource constraints (1.2) exists or not.*

Roughly, a decision problem is in NP if we can verify in polynomial time if a tentative solution is feasible. Let S denote a schedule. Precedence feasibility can be trivially checked in $O(|E|)$ and the makespan constraint in $O(1)$. Verifying the resource-feasibility in polynomial time is not a trivial task. Algorithm 1 verifies the feasibility according to resource-constraints in $O(n^2|\mathcal{R}|)$. The algorithm is based on the observation that each change over time of the activity total resource requirements occurs only at time $S_0 = 0$ or at the completion time of an activity. List $\mathcal{L}$ being a sorted list of the different activity completion times, the algorithm computes each set F of activities in process at each time $t \in \mathcal{L}$, and tests whether the total resource requirements of each set F exceeds the resource availabilities. Note that directly verifying that the resource availability is respected by computing set $\mathcal{A}_t$ for each time period $t \in [0, H[$ yields a pseudo-polynomial algorithm, which depends on value H.

Algorithm 1 Resource feasibility checking of a tentative schedule S

```
1: $\mathcal{L} = \{C_j | A_j \in V\}$
2: sort $\mathcal{L}$ in increasing values
3: **for** $t \in \mathcal{L}$ **do**
4:    **for** $R_k \in \mathcal{R}$ **do**
5:       $o \leftarrow 0, F \leftarrow \emptyset$
6:       **for** $A_j \in \mathcal{A}$ **do**
7:          **if** $S_j < t, C_j \geq t$ and $b_{jk} > 0$ **then**
8:             $o \leftarrow o + b_{jk}, F \leftarrow F \cup A_j$
9:          **end if**
10:         **if** $o > B_k$ **then**
11:            *$S$ is not resource-feasible because activities of $F$ are executed in parallel*
12:            **return** false
13:         **end if**
14:      **end for**
15:   **end for**
16: **end for**
17: **return** true
```

Garey and Johnson [GAR 75] have shown that the decision variant of the RCPSP with a single resource and no precedence constraints, called the resource-constrained scheduling problem, is NP-complete in the strong sense by reduction from the 3-partition problem. NP-hardness can be shown by a simpler observation made by Blazewicz *et al.* [BLA 83] yielding even worse negative results. Let us consider the famous graph coloring problem. Let $G(\mathcal{V}, \mathcal{E})$ be an undirected graph and let c be an arbitrary integer. Deciding if there is a feasible coloring of the nodes in G with c colors such that two adjacent nodes do not share the same color is NP-complete in the strong sense and appears to be a particular case of the decision variant of a RCPSP with unit durations, no precedence constraints, a disjunctive resource per arc in $\mathcal{E}$ and an activity per node in $\mathcal{V}$ with a unit requirement on each resource of its incident arcs. The coloring problem is feasible if and only if the RCPSP has a makespan not greater than c. Figure 1.3 illustrates the 3 colored solution as a RCPSP of makespan 3. Switching to optimization, the minimal number of colors needed, corresponding to the minimal makespan, is called the chromatic number of the graph.

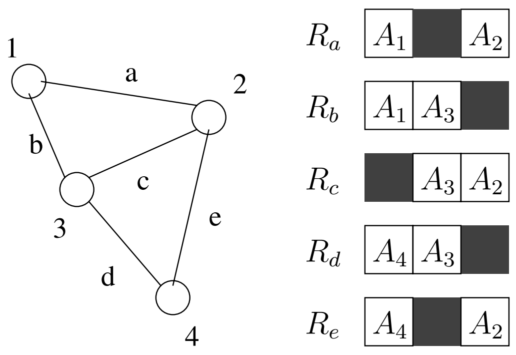

Figure 1.3. *A schedule of minimal makespan equal to the chromatic number of the displayed graph $G(\mathcal{V}, \mathcal{E})$*

When dealing with problems that are NP-hard in the strong sense, an important question is the existence of a polynomial-time approximation scheme. Uetz [UET 01] noticed that by a direct extension of non-approximability results obtained by Feige and Kilian [FEI 98] on graph coloring, it is very unlikely[3] that the minimal makespan can be approximated in polynomial time with a factor of $n^{1-\epsilon}$ for any constant $\epsilon > 0$.

1.4. Dominant and non-dominant schedule subsets

Let us consider the possibly empty set $\mathcal{S}_H$ of feasible schedules of a decision variant of the RCPSP. If we ignore the precedence constraints and considering that integer solutions are dominant, the possible values for each start time S_i can be restricted to interval $[0, H - p_i[$ yielding possibly $\Pi_{A_i \in \mathcal{A}}[0, H - p_i[$ schedules.

Fortunately, in spite of the computational complexity of the problem stated in the previous section, it is possible to avoid the complete enumeration of this possibly

3. We refer to [UET 01, FEI 98] for the precise conditions.

huge search space to find a feasible solution. It is possible to define subsets of schedules significantly smaller than $\mathcal{S}_H$ that always contain a feasible solution if it exists, i.e. dominant subsets. These sets of schedules are the sets of semi-active and active schedules, based on the concepts of local and global left shifts, respectively. Let us consider a schedule S and an activity A_i. The global left shift operator $LS(S, A_i, \Delta)$ transforms schedule S into an identical schedule S', except for $S'_i = S_i - \Delta$ with $\Delta > 0$. A left shift is local when, in addition, all schedules obtained by $LS(S, A_i, \rho)$ with $0 < \rho \leq \Delta$ are also feasible. We define the set of semi-active and active schedules as follows.

DEFINITION 1.3.– *A schedule is semi-active if it admits no feasible activity local left shift.*

From this definition, an integer schedule is semi-active if and only if for each activity $A_i \in \mathcal{A}$, there is no feasible activity left shift of 1 time unit.

DEFINITION 1.4.– *A schedule is active if it admits no feasible activity global left shift.*

From this definition, an integer schedule is active if and only if for each activity $A_i \in \mathcal{A}$, there is no feasible left shift of $\Delta \in \mathbb{N}^*$ time units. Note that the schedule displayed in Figure 1.2 is both semi-active and active.

Any solution of $\mathcal{S}_H$ can obviously be transformed into a semi-active schedule by applying a series of local left shifts. Any semi-active schedule can in turn be transformed into an active schedule by performing a series of global left shifts. It follows that the semi-active schedule set $\mathcal{S}_H^{sa}$ and the active schedule set $\mathcal{S}_H^{a}$ are dominant subsets of $\mathcal{S}_H$. More precisely we have $\mathcal{S}_H^{a} \subseteq \mathcal{S}_H^{sa} \subseteq \mathcal{S}_H$. These sets are also dominant for the search of a solution of $\mathcal{S}$ minimizing a regular objective function $f : \mathbb{R}^{n+2} \rightarrow \mathbb{R}$. A function f is regular if for all $S, S' \in \mathbb{R}^{n+2}$ such that $S \leq S'$ (where $\leq$ is meant componentwise), we have $f(S) \leq f(S')$. In particular, the makespan criterion considered in the standard RCPSP corresponds to a regular objective function.

For practical reasons it can be relevant to consider non-dominant subsets of schedules, i.e. that may exclude the optimal solution. The non-delay scheduling concept is linked to the requirement that no resource is left idle while it could process an activity. Although it may appear intuitively as efficient, this scheduling policy may lead only to sub-optimal solutions. The set of non-delay schedules can be defined by considering partial left shifts. A partial left shift consists of allowing preemption for the left-shifted activity. An activity is preemptive if it can be interrupted at any (not necessarily integer) time-point (see Chapter 8). Thus, a partial left shift $PLS(S, A_i, \Delta, \Gamma)$ of an originally non-interrupted activity consists of a left shift of $\Delta > 0$ time units of the first $\Gamma > 0$ units of the activity.

DEFINITION 1.5.– *A schedule is non-delay if it admits no feasible activity partial left shift.*

For an integer schedule S, whenever a resource is left idle while it could start processing an activity, it is possible to globally left shift the first unit of this activity into the idle period. From the above definition, the set $\mathcal{S}_H^{nd}$ of semi-active schedules of makespan not greater than H verifies $\mathcal{S}_H^{nd} \subseteq \mathcal{S}_H^{a}$. The schedule displayed in Figure 1.2 is also a non-delay schedule .

Introduced for the job-shop problem by Baker [BAK 74], similar definitions of semi-active, active and non-delay schedule sets for the RCPSP can be found in [SPR 95, NEU 00a].

1.5. Order-based representation of schedules and related dominant schedule sets

From the dominance property of the semi-active schedule set, we may derive a relevant representation of schedules. Let S be a feasible schedule. For any activity A_j, if $S_j > 0$ and if there is no other activity A_i such that $S_i + p_i = S_j$, then activity A_j can be left shifted and the schedule is not semi-active. Hence, in any semi-active schedule S and for any activity A_j such that $S_j > 0$, there is an activity A_i such that $S_j = S_i + p_i$. It follows that it is intuitively relevant to represent a left-shifted schedule by additional precedence constraints. Bartusch *et al.* [BAR 88] propose to consider the decomposition of set $\mathcal{S}$ of feasible solutions into finitely many subsets, each one including schedules satisfying the precedence constraints induced by a different strict order. Formally, let P denote a strict order[4] on the set of activities $\mathcal{A}$ and let $\mathcal{S}(P) = \{S | S_j - S_i \geq p_i \quad \forall (A_i, A_j) \in E \cup P\}$ denote the set of (not necessarily feasible) schedules that verify the precedence constraints given by E and the ones given by P. $\mathcal{S}(P)$ is a polyhedron and so is $\mathcal{S}(\emptyset)$ which is the set of time-feasible schedules. By definition, a strict order P is time-feasible if $\mathcal{S}(P) \neq \emptyset$ and is (resource- and time-) feasible if, in addition, $\mathcal{S}(P) \subseteq \mathcal{S}$. A result established by Bartusch *et al.* [BAR 88] states that the set of schedules $\mathcal{S}$ is the union of the polyhedra $\mathcal{S}(P)$ for all inclusion-minimal feasible strict orders P on the set of activities $\mathcal{A}$. In the case of the search for a feasible solution in $\mathcal{S}_H$, or the minimization of a regular objective function in $\mathcal{S}$, the minimal element of $\mathcal{S}(P)$ dominates the other elements of this set. This is the point $ES(P)$ (earliest-start schedule) of $\mathcal{S}(P)$ such that $\forall S' \in \mathcal{S}(P)$, $ES(P) \leq S'$. By definition, $ES(P)_i$ is equal to the length of the longest path from A_0 to A_i in graph $G(V, E \cup P)$, each arc $(A_i, A_j) \in E \cup P$ being valuated by p_i. It follows that $\mathcal{S}(P) \neq \emptyset$ if and only if $G(V, E \cup P)$ is acyclic.

The RCPSP can be defined as the search for a feasible strict order P on $\mathcal{A}$ such that $ES(P)$ is of minimal makespan. Note that P is said to be feasible not only if $ES(P)$

4. A strict order is an irreflexive and transitive binary relation.

is feasible, but if and only if all schedules in $\mathcal{S}(P)$ are feasible. We will see how this type of feasibility can be checked in the next section to specify the strict order-based RCPSP formulation.

To illustrate these concepts, let us consider schedule S displayed in Figure 1.2 and strict orders

$$P_1 = \{(A_2, A_1), (A_3, A_1), (A_3, A_6), (A_6, A_7), (A_3, A_7), (A_9, A_7), (A_9, A_8)\},$$
$$P_2 = P_1 \cup \{(A_3, A_8), (A_6, A_8)\} \quad \text{and} \quad P_3 = P_2 \setminus \{A_9, A_8\}.$$

Note that we have $S = ES(P_1) = ES(P_2) = ES(P_3)$. P_1 and P_2 are both feasible strict orders since any schedule in $\mathcal{S}(P_1)$ or $\mathcal{S}(P_2)$ is feasible. However, P_3 is not a feasible strict order although $ES(P_3)$ is feasible. Indeed, consider schedule $S' = S$, except that S_9 and S_7 are both right shifted by 1 time unit. S' is in $\mathcal{S}(P_3)$ but is not resource-feasible since R_2 is oversubscribed at time C_9. As stated above, it is not trivial to check the feasibility of a strict order, which corresponds here to see that $\mathcal{S}(P_1) \subseteq \mathcal{S}$ and $\mathcal{S}(P_2) \subseteq \mathcal{S}$. This is linked to the forbidden set concept described in section 1.6. However, this example shows that the above-defined feasibility condition of a strict order is not a necessary condition for obtaining a feasible schedule. We can also remark that the *important* arcs of a feasible strict order are the ones belonging to its transitive reduction. Let us consider $P_1' = P_1 \setminus (A_3, A_7)$, the transitive reduction of P_1. We have $\mathcal{S}(P_1) = \mathcal{S}(P_1')$. Furthermore, P_1 is an inclusion-minimal feasible strict order: if we remove any arc belonging to the transitive reduction of P_1 (i.e. all arcs except (A_3, A_7)), $ES(P_1)$ becomes unfeasible. P_2 is not an inclusion-minimal feasible strict order since we have $P_2 \subset P_1$. This shows that several strict orders may lead to the same earliest-start schedule.

Another important remark is that despite the fact that the semi-active concept is intuitively related to the strict order concept, each earliest-start schedule with respect to a given strict order does not necessarily correspond to a semi-active schedule in the sense of Definition 1.3 as noted by Sprecher *et al.* [SPR 95]. For instance, let us define strict order $P_4 = P_1 \cup \{(A_{10}, A_8)\}$. The earliest start schedule $ES(P_4)$ is identical to the one displayed in Figure 1.2 except that the start time of activity 8 is delayed up to the completion time of activity A_{10}. Since a local left shift of activity A_8 is made feasible, schedule $ES(P_4)$ is not semi-active. For that reason, Neumann *et al.* [NEU 00a] propose additional schedule sets related to the strict order representation: the quasi-active and the pseudo-active schedule sets. These sets are based on the order-preserving and order-monotonic left shifts, respectively. These left shifts are defined by reference to the interval order $P(S)$ induced by a schedule S where $P(S) = \{(A_i, A_j) \in V^2 | S_j \geq S_i + p_i\}$. By definition, $P(S)$ is the inclusion maximal strict order P such that $S \in \mathcal{S}(P)$. Since $P(S)$ is inclusion maximal, each antichain in graph $G(V, E \cup P(S))$ represents a set of activities simultaneously in process.

An order-preserving left shift of an activity on S yields a feasible schedule S' such that $P(S) \subseteq P(S')$. An order-monotonic left shift of an activity on S yields a feasible schedule S' such that $P(S) \subseteq P(S')$ or $P(S') \subseteq P(S)$. From this definition it follows that in an order-monotonic left shift from schedule S, either all chains of $G(V, E \cup P(S))$ or all antichains are preserved. Consequently, an order-preserving left shift is also order-monotonic. Moreover, since the initial and final schedules belong to the same feasible order polyhedron $\mathcal{S}(P(S))$ or $\mathcal{S}(P(S'))$, an order-monotonic left shift is also a local left shift.

DEFINITION 1.6.– *A schedule is quasi-active if it admits no feasible order-preserving left shift.*

From a polyhedral point of view, a schedule S is quasi-active if for each activity $A_i \in V$, $S' = LS(S, A_i, \Delta)$ does not belong to $\mathcal{S}(P(S))$, for all $\Delta > 0$, i.e. if it is the minimal point of $\mathcal{S}(P(S))$.

We can show that schedule $ES(P_4)$ defined above is quasi-active since any left shift violates its interval order $P(ES(P_4)) = \{(A_1, A_8)$, (A_1, A_{10}), (A_2, A_1), (A_2, A_5), (A_2, A_6), (A_2, A_7), (A_2, A_8), (A_2, A_9),(A_2, A_{10}), (A_3, A_1), (A_3, A_5), (A_3, A_6), (A_3, A_7), (A_3, A_8), (A_3, A_9), (A_3, A_{10}), (A_4, A_7), (A_4, A_8), (A_4, A_9), (A_4, A_{10}), (A_5, A_7), (A_5, A_8), (A_5, A_9), (A_5, A_{10}), (A_6, A_7), (A_6, A_8), (A_6, A_{10}), (A_9, A_7), (A_9, A_8), (A_9, A_{10}), $(A_{10}, A_8)\}$.

Since each semi-active schedule is also a quasi-active schedule, we have the following relations $\mathcal{S}_H^{sa} \subseteq \mathcal{S}_H^{qa} \subseteq \mathcal{S}_H$ where $\mathcal{S}_H^{qa}$ denote the set of quasi-active schedules.

DEFINITION 1.7.– *A schedule is pseudo-active if it admits no feasible order-monotonic left shift.*

A schedule S is pseudo-active if for each activity $A_i \in V$, $S' = LS(S, A_i, \Delta)$ does not belong to $\mathcal{S}(P)$, for all $\Delta > 0$ and for all strict orders P such that $S \in \mathcal{S}(P)$, i.e. if it is the minimal point of all schedule sets $\mathcal{S}(P)$ containing S.

Schedule $ES(P_4)$ is not pseudo-active since a local left shift of 1 unit on activity A_{10} is order-monotonic: it removes $(10, 8)$ and $(1, 8)$ from $P(ES(P_4))$ yielding a necessarily feasible and non-worse schedule. In the terms of Definition 1.7, we can also remark alternatively that $ES(P_4)$ belongs to the polyhedron $\mathcal{S}(P_1)$ and that activity A_{10} can be locally left shifted while obtaining a schedule still in set $\mathcal{S}(P_1)$.

Based on these definitions, the following theorem can be established.

THEOREM 1.1.– *The set of semi-active schedules and the set of pseudo-active schedules are identical for the RCPSP.*

Proof. Since there is no feasible local left shift from a semi-active schedule, the set of semi-active schedules is included in the set of pseudo-active schedules. We show that, conversely, there is no feasible local left shift from a pseudo-active schedule. Suppose that there is a feasible local non-order-monotonic left shift $LS(S, A_i, \Delta)$ from a pseudo-active schedule S yielding schedule S'. Since $LS(S, A_i, \Delta)$ is not order-monotonic, let $(A_j, A_i) \in P(S)$, $(A_j, A_i) \notin P(S')$, $(A_i, A_k) \notin P(S)$ and $(A_i, A_k) \in P(S')$. Let $\Delta' = (S_i + p_i - S_k)/2$. We have $\Delta > \Delta'$ and then $LS(S, A_i, \Delta')$ is feasible and does not create (A_i, A_k) while (A_j, A_i) is either removed from or kept in $P(S')$. From S', we can recursively apply this principle until obtaining a feasible order-monotonic left shift from S, which contradicts the hypothesis. $\square$

To sum up, we have the following relations between the presented dominant schedule sets: $\mathcal{S}_H^a \subseteq \mathcal{S}_H^{sa} = \mathcal{S}_H^{pa} \subseteq \mathcal{S}_H^{qa} \subseteq \mathcal{S}_H$, where $\mathcal{S}_H^{pa}$ denotes the set of pseudo-active schedules. It should be mentioned that for the RCPSP with minimal and maximal time lags, an extension of the problem considered in Chapter 11, it generally holds that $\mathcal{S}_H^{sa} \subset \mathcal{S}_H^{pa}$ [NEU 00a].

1.6. Forbidden sets and resource flow network formulations of the RCPSP

From the order-based representation of dominant schedules, we can derive two alternative formulations of the RCPSP restricting the search space to the dominant set of quasi-active schedules.

A forbidden set is a set F of activities that cannot be scheduled in parallel in a feasible solution because of resource limitations due to a resource R_k such that $\sum_{A_i \in F} b_{ik} > B_k$. A forbidden set is minimal if any of its subsets of activities F' verifies, $\forall R_k \in \mathcal{R}$, $\sum_{A_i \in F'} b_{ik} \leq B_k$. Let $\mathcal{F}$ denote the set of minimal forbidden sets. No element of $\mathcal{F}$ can be an antichain of $G(V, E \cup P(S))$ for any feasible schedule S. Consequently, a necessary condition for feasibility of a schedule S is that $\forall F \in \mathcal{F}$, $F^2 \cap P(S) \neq \emptyset$. A forbidden set F such that $F^2 \cap P(S) \neq \emptyset$ is said to be broken up. If, in addition, $S \in \mathcal{S}(\emptyset)$, we obtain a necessary and sufficient condition. Consequently, the RCPSP can be reformulated as follows.

DEFINITION 1.8.– *The RCPSP aims at finding a strict order P such that $G(V, E \cup P)$ is acyclic, $\forall F \in \mathcal{F}$, $F^2 \cap P \neq \emptyset$ and $ES(P)_{n+1}$ is minimal.*

The drawback of the preceding formulation is that the number of minimal forbidden sets $|\mathcal{F}|$ can be extremely large. A significant number of forbidden sets are implicitly broken up due to precedence relations. Indeed as soon as there is a path from A_i to A_j in the precedence graph $G(V, E)$, any precedence-feasible schedule breaks up any forbidden set including both A_i and A_j. Thus, we do not have to consider such forbidden sets in $\mathcal{F}$.

However, it is not trivial to efficiently enumerate all the minimal forbidden sets. Stork and Uetz propose in [STO 05] an algorithm based on the enumeration of all the subsets of $\mathcal{A}$ through a tree, in which each node, except the root node, is associated with an activity. In the tree, each branch corresponds to a candidate minimal forbidden set. The root node has n children corresponding to $A_1,\ldots,A_n$. Then, each node A_i has potentially $n-i$ children $A_{i+1},\ldots,A_n$. A node corresponds to a minimal forbidden set as soon as the sum of the resource demands over the node itself and all its ascendants exceeds the capacity of one resource. A node is fathomed as soon as it does not lead to a minimal forbidden set. Figure 1.4 displays the 24 minimal forbidden sets, not broken up by precedence relations, obtained through the tree structure for the illustrative example. Note that because of precedence relations, activity A_{10} does not belong to any forbidden set and, consequently, can be ignored for the computation of the strict orders. If we ignore the precedence relations, the number of minimal forbidden sets grows up to 97. The reader may verify that both strict orders P_1 and P_2 break up the 24 forbidden sets whereas P_3 does not (see for instance forbidden set $\{A_1, A_8, A_9\}$). Chapter 7 discusses computational experiments implementing the Stork and Uetz algorithm to evaluate the hardness of RCPSP instances.

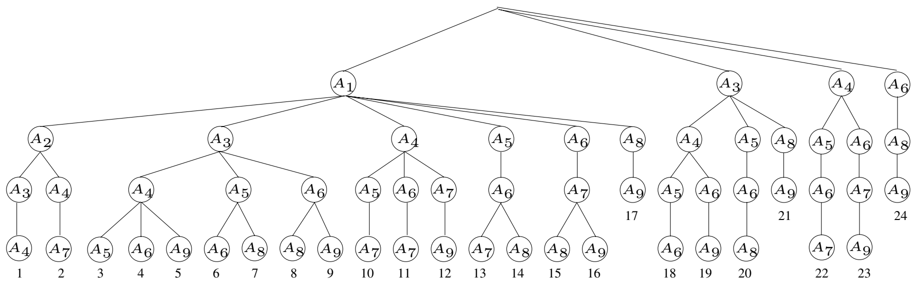

Figure 1.4. *Minimal forbidden sets of the illustrative example*

An intermediate order-based representation can be used to avoid the use of $\mathcal{F}$ in the RCPSP formulation. This representation involves the concept of resource flow decision variables f_{ij}^k for $A_i \in \mathcal{A} \cup \{A_0\}$, for $A_j \in \mathcal{A} \cup \{A_{n+1}\}$ and for $R_k \in \mathcal{R}$. f is a resource flow if it verifies the following constraints:

$$f_{ij}^k \geq 0 \quad \forall A_i \in \mathcal{A} \cup \{A_0\},\ \forall A_j \in \mathcal{A} \cup \{A_{n+1}\},\ \forall R_k \in \mathcal{R} \tag{1.3}$$

$$\sum_{A_i \in \mathcal{A} \cup \{A_{n+1}\}} f_{0i}^k = \sum_{A_i \in \mathcal{A} \cup \{A_0\}} f_{i(n+1)}^k = B_k \quad \forall R_k \in \mathcal{R} \tag{1.4}$$

$$\sum_{A_j \in \mathcal{A} \cup \{A_{n+1}\}} f_{ij}^k = \sum_{A_j \in \mathcal{A} \cup \{A_0\}} f_{ji}^k = b_{ik} \quad \forall A_i \in \mathcal{A},\ \forall R_k \in \mathcal{R} \tag{1.5}$$

A resource flow f induces the strict order $P(f)$, the transitive closure of $\{(A_i, A_j) \in V^2 \mid f_{ij} > 0\}$. The following lemmas establish the link between flows and strict orders.

LEMMA 1.1.– *If f is a flow and $G(V, E \cup P(f))$ is acyclic, then $P(f)$ is a feasible strict order.*

Proof. First, if $G(V, E \cup P(f))$ is acyclic, then $ES(P(f))$ is time feasible. Second, suppose that a forbidden set F is not broken up by $ES(P(f))$ for a resource k. Since all activities of F are simultaneously in process during at least one time period, there is a total incoming flow of F equal to $\sum_{i \in F} b_{ik} > B_k$ which violates the flow conservation conditions. $\square$

Consequently, a resource flow such that $G(V, E \cup P(f))$ is acyclic is said to be feasible.

LEMMA 1.2.– *For each feasible strict order P, there is a feasible resource flow f such that $ES(P(f)) \leq ES(P)$.*

Proof. Let us consider the feasible schedule $S = ES(P)$ and strict order $P(S) \supseteq P$ with $ES(P(S)) = S$. By setting additional constraints $f_{ij}^k = 0$ for all $(A_i, A_j) \notin P(S)$ we show that there exists a flow verifying equations (1.3)-(1.5). This amounts to verifying that there exists a feasible flow in each network $G_k(V, E \cup P(S))$, for each resource $R_k \in \mathcal{R}$ with minimal and maximal node capacities b_{ik} for all activities $A_i \in \mathcal{A}$ and node capacities B_k for nodes A_0 and A_{n+1}. Let us transform this graph with bounds on node flow into a graph with bounds on arc flows. We split each node $A_i \in V$ into two nodes A_i and A_i' linked by an arc of minimal and maximal capacities equal to the node capacity. All other arcs have null minimal capacities and infinite maximal capacities. If we now ignore the maximal capacities, since $P(S)$ is feasible, there is no $A_0 - A_{n+1}'$ cut of (minimal) capacity greater than B_k. Indeed suppose that such a cut exists. Then, it includes a set of arcs (i, i') that represents a non-broken up forbidden set. Furthermore there is a $A_0 - A_{n+1}'$ cut (reduced to arc (A_0, A_0')) of minimal capacity equal to B_k. Therefore, according to the min-flow max-cut theorem (see [NEU 03, LEU 04]), we have a minimal flow equal to B_k. $\square$

It follows from Lemmas 1.1 and 1.2 that the RCPSP can be defined as follows.

DEFINITION 1.9.– *The RCPSP aims at finding a feasible flow f such that $ES(P(f))_{(n+1)}$ is minimal.*

From a feasible flow f, a feasible schedule $ES(P(f))$ can be computed by longest path computations in graph $G(V, E \cup P(f))$ where all arcs $(A_i, A_j) \in E \cup P(f)$ are valuated by p_j.

Conversely, Algorithm 2 allows the computation of a feasible flow f from any feasible schedule S in $O(n^2|\mathcal{R}|)$. The algorithm assumes activities are sorted in increasing order of their completion times. For each activity A_i, the incoming flow is simply taken from the activities A_j completed before the start time of A_i. At each step of the algorithm, β_{ik} denotes the number of resource R_k units that remain to be sent to A_i.

Algorithm 2 BUILDFLOWFROMSCHEDULE(f, S): generates flow f from schedule S

```
1: $\beta_{ik} \leftarrow b_{ik}, \forall A_i \in \mathcal{A}, \forall R_k \in \mathcal{R}$
2: $f^k_{i(n+1)} \leftarrow b_{ik}, \forall A_i \in \mathcal{A}, \forall R_k \in \mathcal{R}$
3: $f^k_{0(n+1)} \leftarrow B_k, \forall R_k \in \mathcal{R}$
4: for $A_i \in \mathcal{A}$ (in increasing order of completion times $C_i$) do
5:     for $A_j \in \mathcal{A}$, $A_j \neq A_i$ do
6:         for $R_k \in \mathcal{R}$ do
7:             if $C_j \leq S_i$ then
8:                 $\rho \leftarrow \min(f^k_{j(n+1)}, \beta_{ik})$
9:                 $\beta_{ik} \leftarrow \beta_{ik} - \rho$
10:                $f^k_{j(n+1)} \leftarrow f^k_{i(n+1)} - \rho$
11:                $f^k_{ji} \leftarrow f^k_{ji} + \rho$
12:            end if
13:        end for
14:    end for
15: end for
```

Figure 1.5 gives a flow representing the schedule displayed in Figure 1.2. The flow values for both resources are displayed near each arc. Dotted arcs are the arcs representing the additional precedence constraints induced by the flow and used to define $P(f)$. We observe here that $P_1 \subset P(f)$, P_1 being the inclusion-minimal strict order defined in section 1.5. Consequently, we can verify on this example that $P(f)$ breaks up all the forbidden sets.

1.7. A simple method for enumerating a dominant set of quasi-active schedules

A very simple algorithm can be designed to find an optimal solution of a RCPSP without enumerating the forbidden sets nor using the concept of flows. Conceptually, it consists of starting with an empty strict order $P = \emptyset$. If the time feasible earliest-start schedule $ES(P)$ is feasible, which can be checked by Algorithm 1, then it is the optimal schedule. Otherwise, Algorithm 1 returns a non-broken up minimal forbidden set F. The optimal solution can be found by recursively applying this procedure for each $(A_i, A_j) \in F$ after adding (A_i, A_j) to P. The recursive algorithm OPTFS is given as Algorithm 3.

Algorithm 3 can also be used as a one pass method to build a single feasible solution to the RCPSP. It suffices to select at each step a single arc $(A_i, A_j) \in F^2$ such that

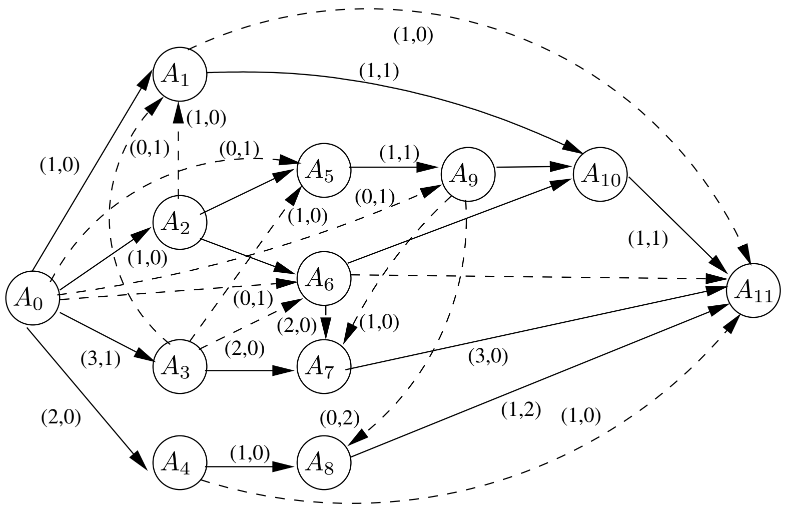

Figure 1.5. *A resource flow for the schedule of Figure 1.2*

Algorithm 3 OPTFS(P): search for an optimal solution to the RCPCP with an initial set of additional arcs P

```
1: if $\mathcal{S}(P) = \emptyset$ then
2:     return $+\infty$
3: else if $ES(P)$ is resource-feasible (checked by Algorithm 1) then
4:     return $ES(P)_{n+1}$
5: else
6:     (Algorithm 1 has returned a violated forbidden set $F$)
7:     return $\min_{(A_i,A_j)\in F\times F}$ OPTFS$(P \cup \{(A_i, A_j)\})$
8: end if
```

(A_i, A_j) does not induce a cycle in $G(V, E \cup P)$. An $O(|E \cup P|)$ algorithm can be used to perform a cycle check. Such an arc always exists because if $(A_i, A_j) \in F^2$ induces a cycle, (A_j, A_i) does not induce a cycle and also breaks up the forbidden set. Since an arc is added at each iteration, the algorithms gives a feasible solution in $n(n-1)/2$ iterations. The obtained schedule is quasi-active since it is equal to $ES(P)$. It follows that Algorithm 3 computes a set of dominant quasi-active schedules. In the algorithm we refer to P as a set of additional arcs and not as a strict order for practical reasons. As stated earlier, the relevant elements of a strict order are the arcs belonging to the transitive closure. When adding arc (A_i, A_j) to P at step 7, computing the transitive closure of P to keep the transitivity property would be an unnecessary computational effort.