Heuristic Methods for Makespan Minimization in Project Scheduling

Barneveld, T.C. van

T.C. van Barneveld

Heuristic Methods for Makespan Minimization in Project Scheduling

Master Thesis, 29 August 2012

Thesis advisor: dr. F.M. Spieksma

Mathematisch Instituut, Universiteit Leiden

Preface

This thesis has been written as the final stage of my master Applied Mathematics in Leiden. In the period august 2011 - august 2012, I have worked on heuristic methods for makespan minimization in project scheduling under supervision of dr. Floske Spieksma, who also proposed the topic of this research. This research was done simultaneously with an internship of Raoul Wols at RoserConsys BV, a company in Dordrecht developing scheduling software. It is based on problems they encountered. To test the different methods explained in this thesis, I have coded them in MATLAB. Due to the extent of this codes, I have not included them in this thesis. If you would like to obtain the codes, please send me an email to tbarneve@math.leidenuniv.nl.

Three talks have been given on this research. The first one I gave in february 2012 at RoserConSys BV. The second and the third were graduation talks given in August 2012; one for a dedicated public, treating the major results of my research and one for a wide audience treating only a minor introduction to project scheduling.

I very much enjoyed doing this research and I want to thank Floske for her suggestions, help and frequent corrections of my thesis. It was both useful and nice to have regular meetings during the whole period of doing research for this thesis.

T.C. van Barneveld
August 2012

Contents

1 Introduction

From both a theoretical as a practical point of view, scheduling problems are interesting. A project is an undertaking which has to be accomplished. A number of tasks have to be executed, all of which require an amount of time and a number of resources to complete. Moreover, one works towards a certain objective, e.g. minimization of the project duration. In addition, tasks or activities or jobs depend on each other by given time lags: some tasks can only be executed if others are finished. This influences the order in which activities have to be scheduled. Some tasks can not be executed simultaneously due to the limited availability of resources. Project scheduling then consists of fixing the start time of each activity, such that all temporal and resource constraints are fulfilled and the objective function is optimized.

Projects arises in many practical situations, such as construction work, emergency planning, production and the execution of turnarounds. For example, consider the situation in which a restaurant has just one waiter, two chefs and there is one customer, which is immediately served when he enters the restaurant. The time it takes the waiter to take the order is 1 minute. Assume the customer orders beef, with potatoes and vegetables. It takes a chef 12 minutes to prepare beef, 10 minutes for the potatoes and 6 minutes for the vegetables. Moreover, the chefs can not work simultaneously one the same part of the meal. When the beef is finished, it has to rest for 2 minutes before it can be put on the plate; for the potatoes this time is 5 minutes. After that, one chef has to finish the plate, which takes 1 minute, before the waiter can serve it, which takes 3 minutes. The owner of the restaurant wants to minimize the waiting time of the customer.

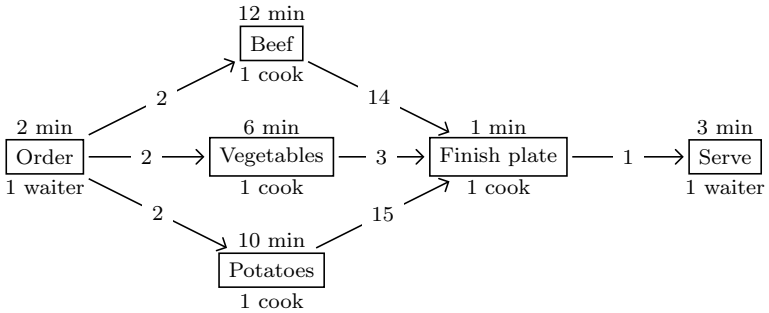

Figure 1.1: Project in restaurant

This is a project which can be represented by the *network* in Figure 1.1, in which the processing times are given above the nodes, and the resource requirements below. The numbers in the arcs represent the minimum time between the start of the activities. For example, the chefs can start preparing the meal only after the waiter has taken the order, which is after two minutes.

This problem can be easily solved: since there are only two cooks and three parts of the meal have to be prepared, just two parts can be cooked simultaneously. It is recommended to start with the potatoes and beef, since they have to wait an amount of time before the plate can be finished. Thus, one cook should prepare the beef and the other the potatoes as soon as possible, that is, when the order is taken. To minimize waiting time of the customer, the cook who is first available, that is, the cook who prepared the potatoes, can begin with the

vegetables at time 12. At time 18, all parts are ready and at time 19 the plate is finished. The customer can start dinner after 22 minutes from the moment he entered the restaurant.

This example is a project in the form that will be considered in this thesis. In addition to minimum time lags (the numbers in the arcs), maximum time lags will be introduced in chapter 2. This chapter also provides two IP-formulations for the project scheduling problem. In chapter 3, time windows, that is, earliest and latest starting times for the jobs, will be determined by two different methods. In chapter 4, two simple algorithms for computing lower bounds for the project duration are considered, and chapter 5 provides a more complex lower bound, called the Lagrangian lower bound. This lower bound is computed by transforming the project scheduling problem into a minimum cut problem. Chapter 6 serves as the 'backbone' of this thesis, in which several methods for generating schedules are discussed. In chapter 7, these heuristics are tested on benchmark instances.

2 Model Description

Theoretic scheduling problems can be denoted by a three-field notation $\alpha|\beta|\gamma$, where α indicates the *resource environment*, β the *activity characteristics* and γ the *objective function*. For this problem, we use the three-field notation proposed in [5]: $PS|temp|C_{max}$, which represents the following:

PS	Project Scheduling.
$temp$	General temporal constraints given by minimum and maximum start-start time lags between activities.
C_{max}	Minimize the total makespan, which is the finish-time of the last job.

Like in ordinary scheduling problems, we have a finite number n of activities or jobs that have to be processed, indicated by a job set $\mathcal{J}$. These jobs have fixed integral *processing times*: $p_1, p_2, \ldots, p_n$. Besides, we introduce two dummy jobs 0 and $n+1$, which are a start and a finish activity. Job 0 has to be processed *before* and job $n+1$ *after* all other jobs. Obviously, the processing times of these jobs are 0. The activity set is $\mathcal{J} = \{0, 1, 2, \ldots, n, n+1\}$.

In scheduling problems, we search for optimal schedules. Notice that a schedule is completely characterized by fixing a starting time for every job. So the goal is to search for a starting time for every activity, such that the acquired schedule is optimal with respect to the objective function, in this case minimizing the makespan. Such a schedule is denoted by a list of starting times $S = \{S_0, S_1, \ldots, S_{n+1}\}$. Evidently, fix $S_0 = 0$, which is the start time of the project. Moreover, notice that $S_{n+1} = C_{max}$, since job $n+1$ has to be processed after all other jobs and $p_{n+1} = 0$.

2.1 Temporal Constraints

In this paragraph, we take a closer look at the temp characteristic. In our model, we have general *temporal constraints* given by *minimum* and *maximum start-start time lags* between jobs. Next, we introduce minimal and maximal time lags between the start of two different jobs, as done in [19]. A *minimal time lag* $d_{ij}^{SS} \geq 0$ between the start of two jobs i and j forces the following relation between the starting times of jobs i and j: $S_j - S_i \geq d_{ij}^{SS}$. In words: job j must start at least d_{ij}^{SS} after the start of job i.

Remark 2.1. 1. All time lags are integral.

2. One could also define finish-finish (FF), start-finish (SF) or finish-start (FS) time lags, instead of start-start time lags. These are easily converted into start-start (SS) time lags:

   - $d_{ij}^{FS} + p_i = d_{ij}^{SS}$
   - $d_{ij}^{SF} - p_j = d_{ij}^{SS}$
   - $d_{ij}^{FF} + p_i - p_j = d_{ij}^{SS}$

   By this observation, all time lags in the rest of this paper are intended to be start-start time lags, so we drop the SS-index.

3. If two jobs i and j should start at the same point in time, we set both $d_{ij} = 0$ and $d_{ji} = 0$.

4. For the finish job $n+1$, there is a minimum time lag $d_{i,n+1} = p_i$ for all jobs $i \in \mathcal{J}$, since job $n+1$ is only allowed to start when all other jobs are finished.

5. Ordinary *release dates* b_j can be represented by letting $d_{0j} := b_j$.

6. Ordinary *precedence constraints* can be represented by letting $d_{ij} := p_i$ if job i must precede job j.

Maximum time lags can be modelled in the same way. Given are numbers $d_{ji}^{max} > 0$ such that $S_i - S_j \leq d_{ji}^{max}$. This means that job i must start *at most* d_{ji}^{max} time units after job j has started. Then, job j must start at least $d_{ij} := -d_{ji}^{max} < 0$ time units after job i. Hence, a maximum time lag $d_{ji}^{max} > 0$ can be converted into a minimum time lag $d_{ij} < 0$. In short, job j must start *at least* d_{ij} time unts after the start of job i, independent of whether d_{ij} is positive or not.

In such a way, time windows of the form $S_j + d_{ji} \leq S_i \leq S_j - d_{ij}$ between any two jobs j and i can be modelled. Let $L \subseteq \mathcal{J} \times \mathcal{J}$ be the set of all given time lags. Define

$$T := \sum_{i \in \mathcal{J}} \max(p_i, \max_{\langle i,j \rangle \in L} d_{ij}), \tag{2.1.1}$$

which represents an upper bound on the shortest project duration, if the given problem is feasible, i.e., if $C_{max} < \infty$. As a corollary of this definition of T, we will see that the we have a finite number of variables in the IP-formulation of this problem.

Time lags of an instance of $PS|temp|C_{max}$ can be represented in an *weighted Activity-on-Node network* N. The nodes represent the jobs and if there is a time lag between the start of two jobs i, j, an arc between nodes i and j is drawn, with weight d_{ij}. An example of such an project network N is drawn in Example 2.1.

2.2 Resource Constraints

We now take a further look at the *resource constraints*. Suppose there are m different types of resources. These are denoted by a set $\mathcal{R} = \{1, 2, \ldots, m\}$. Moreover, every type has its own *maximum capacity*, which are given by numbers $\overline{R_1}, \overline{R_2}, \ldots, \overline{R_m}$.

In addition, each job has his own *resource profile*. This is summarized in a matrix r where entry r_{jk} indicates that r_{jk} units of resource k are needed in order to process job j. These resources are needed during the whole processing time of a job. Evidently, resources that are used for processing a job are not available for other purposes until that job is completed. After that, resources are available again.

Given a schedule S, let

$$\mathcal{A}(S,t) := \{i \in \mathcal{J} | S_i \leq t < S_i + p_i\}, \quad t \geq 0 \tag{2.2.1}$$

be the set of jobs that are *active* in schedule S at time t, and

$$r_k(S,t) := \sum_{i \in \mathcal{A}(S,t)} r_{ik}, \quad k \in \mathcal{R}, t \geq 0 \tag{2.2.2}$$

the resource usage of resource k in schedule S on time t.

2.3 Two IP-formulations

Now we have introduced our problem, it can be modelled as an IP-problem using time-indexed variables, as done in [17]:

$$x_{jt} = \begin{cases} 1 & \text{if job } j \text{ starts at time } t \\ 0 & \text{otherwise} \end{cases}$$

where $j \in \mathcal{J}$ and $t = 0, 1, 2, \ldots, T$. Remember that $T < \infty$, thus there is a finite number of decision variables.

Notice that the makespan of the project, the C_{max}, is fully determined by the start time of the dummy job $n+1$. Moreover, it is easily seen that $S_{n+1} = \sum_t t x_{n+1,t}$. Hence, the following objective function is obtained:

$$\text{Minimize} \sum_t t x_{n+1,t}. \tag{2.3.1}$$

As described above, there are different types of constraints. It should be ensured that each activity has exactly one starting time. This can be modelled by introducing constraints

$$\sum_{t=0}^{T} x_{jt} = 1, \quad j \in \mathcal{J}. \tag{2.3.2}$$

The first IP-form for the temporal constraints is derived as follows: Suppose t_i is the starting time of activity i and there is a time lag d_{ij} between jobs i and j. Then job j is not allowed to start *before* time $t_i + d_{ij}$. Hence, job j can *not* start on times $0, \ldots, t_i + d_{ij} - 1$. This yields a constraint:

$$x_{it_i} + \sum_{s=0}^{t_i+d_{ij}-1} x_{js} \leq 1.$$

Summing over all possible starting times of job i, the following formulation of the temporal constraints is used:

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+d_{ij}-1} x_{js} \leq 1, \quad (i,j) \in L,\ t = 0, \ldots, T. \tag{2.3.3}$$

For the resource-constraints, notice that $\sum_{s=t-p_j+1}^{t} x_{js} = 1$ indicates that job j is active at time t. Multiplying this number by r_{jk} gives the usage of resource $k \in \mathcal{R}$ by job j at time t, i.e.,

$$r_{jk} \sum_{s=t-p_j+1}^{t} x_{js} = \begin{cases} 0 & \text{if job j is not active at time t} \\ r_{jk} & \text{if job j is active at time t} \end{cases}$$

Taking the sum over all jobs, the total resource usage of resource k on time t is computed. By requiring that this number should be equal or less than the resource capacity, the following resource constraints are obtained:

$$\sum_{j} r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) \leq \overline{R_k},\ k \in \mathcal{R}, \quad t = 0, \ldots, T. \tag{2.3.4}$$

At last we have the binary constraint

$$x_{jt} \in \{0, 1\}. \tag{2.3.5}$$

The first IP-formulation is now given by (2.3.1), subject to (2.3.2), (2.3.3), (2.3.4) and (2.3.5).

Remark 2.2. In equation (2.3.3), the upper bound of the summation is $t + d_{ij} - 1$, but it is possible that this number exceeds T. A well-defined upper bound would be $\min(t+d_{ij}-1, T)$. Moreover, the lower bound of the summation in equation (2.3.4) is $t - p_j + 1$, which could be below 0, leading to a well-defined lower bound of $\max(t - p_j + 1, 0)$. For simplicity, we omit this by treating non-defined x-variables as 0.

One could also use the following IP-formulation, introduced in [20]:

$$\begin{aligned}
\text{Minimize } & S_{n+1} && && (2.3.6)\\
\text{subject to } & S_j - S_i \geq d_{ij}, && (i,j) \in L && (2.3.7)\\
& r_k(S,t) \leq \overline{R_k}, && k \in \mathcal{R}, 0 \leq t \leq T && (2.3.8)\\
& S_j \geq 0, && j \in \mathcal{J} && (2.3.9)\\
& S_0 = 0 && && (2.3.10)
\end{aligned}$$

This formulation is intuitively more clear than the first one. However, there are no binary decision variables here. In chapters 3, 6 and 4 the second and in chapter 5 the first IP-formulations will be used in the solution methods.

2.4 Complexity

Relatively small problems already give a large number of decision variables and constraints. To illustrate this phenomenon, consider example 2.1.

Example 2.1. The project presented in Figure 2.1 and Table 2.1 has 5 jobs, which are represented as nodes. Since we have 1 type of resource, the resource profile r_{j1} is simply denoted by r_j. The capacity of the resource is 8. Notice that the arcs according to time lags $d_{02} = d_{04} = d_{06} = 0$ and $d_{36} = 4$ are omitted for clearness in the picture. Moreover, these time lags are redundant, since the longest paths do not cross these arcs. We will focus on longest paths in the next chapter.

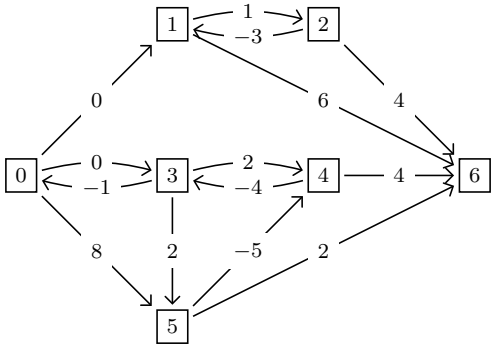

Figure 2.1: Project network N

Job	p_j	r_j
0	0	0
1	6	4
2	4	1
3	4	5
4	4	6
5	2	1
6	0	0

Table 2.1

We use (2.1.1) to obtain the initial upper bound $T = 28$.

For the first IP-formulation, this already results in $29 \times 5 = 145$ relevant decision variables for such a small problem! In chapter 3, we will see that this number can be vigorously reduced. Moreover, $|L| = 14$, so there are $14 \times 29 = 406$ temporal constraints, $1 \times 29 = 29$ resource constraints and $5 + 29 \times 5 = 150$ 'trivial' constraints, leading to a total number of 556 constraints for an instance with only 5 jobs and 1 resource type.

For the second IP-formulation the relevant decision variables are $S_1, \ldots, S_5$, so we only have 5 of them. Besides, the number of trivial (2.3.9), temporal (2.3.7) and resource constraints (2.3.8) is $5 \times 29 = 145$, 14 and $1 \times 29 = 29$, which results in only 188 constraints.

□

As described in [16], (Binary) Integer Programming problems can be solved using LP-based Branch & Bound. However, this is not a polynomial method since we have to check all $\mathcal{O}(2^{T|\mathcal{J}|})$ possible solutions in a systematic fashion. Even for the very small project in Example 2.1, it took a 1.9GHz AMD Athlon PC approximately 1698 seconds to solve the Binary Integer formulation of this problem to optimality, using an implementation of LP-based Branch & Bound in MATLAB2010a. It is known that (B)IP is $\mathcal{NP}$-complete, since there is a polynomial reduction from SAT, so question arises whether $PS|temp|C_{max}$ is $\mathcal{NP}$-complete as well. Consider the decision form of $PS|temp|C_{max}$:

Given: A set of activities $\mathcal{J}$ with processing times p_j and a set L of time lags d_{ij} between jobs i and j; a set of resources $\mathcal{R}$ with capacities $\overline{R_k}$ and r_{jk}, the amount of resources of type

k used by activity j.
Question: Does there exist a feasible schedule?

Theorem 2.1. *$PS|temp|C_{max}$ is $\mathcal{NP}$-complete.*

Proof. We first prove that $PS|temp|C_{max}$ is in $\mathcal{NP}$: Consider a yes-instance of $PS|temp|C_{max}$, that is, a schedule $S = (S_0, S_1, \ldots, S_{n+1})$. The feasibility of this schedule can be checked by computing $S_j - S_i \quad \forall (i,j) \in L$ for checking the temporal constraints and $r_k(S,t) \quad \forall k \in \mathcal{R}$, $0 \leq t < S_{n+1}$ for the resource constraints. The other types of constraints are checked easily as well. Hence, the verification of a yes-instance can be done in polynomial time.

The $\mathcal{NP}$-hardness of $PS|temp|C_{max}$ is shown in [2] by transformation from the problem PRECEDENCE CONSTRAINED SCHEDULING. □

As a consequence of Theorem 2.1, there is no polynomial algorithm to solve $PS|temp|C_{max}$, unless $\mathcal{P} = \mathcal{NP}$. In the coming chapters, we will consider some heuristics to solve the problem.

3 Time Windows

In this chapter we will establish *Time Windows* for the activities. These time windows arise as a consequence of the time lags between jobs. Let $TW(j)$ denote the time window for job j. That is, $TW(j) = \{ES_j, ES_j + 1, \ldots, LS_j - 1, LS_j\}$ with ES_j and LS_j the earliest and latest starting times of job j respectively. Strong time windows result in fewer decision variables in the first, and fewer possible values for the decision variables in the second IP-formulation, thus decreasing computation time. A simple method to compute these time windows, using a longest-path algorithm without taking care of resource constraints, is explained in paragraph 3.1, while in paragraph 3.2 we also consider resources.

3.1 Longest-path Approach

As mentioned in Example 2.1, time lags can be represented by an Activity-on-Node network. Moreover, we introduce an arc $(n+1, 0)$ with weight $d_{n+1,0} = -T$, indicating that the finish job is not allowed to start more than T time units after the start of the project, where T is as in (2.1.1). We denote this network by N. For the representation of N we use the adjacency matrix, which we denote, with a slight abuse of notation, by N as well. We set $N_{(i,i)} = 0$ for all $i \in V$ and if $(i,j) \notin A$, $N_{(i,j)} = \infty$.

By l_{ij} we denote the *longest path length* from node i to node j, where $l_{ij} = 0$ for $i = j$. Notice that $l_{ij} \geq d_{ij}$ for all $(i,j) \in A$. Moreover, the *triangle inequality* is satisfied: $l_{ij} \geq l_{ih} + l_{hj}$, $i, j, h \in V$. Since $l_{ij} \geq d_{ij}$, S does not just have to satisfy $S_j - S_i \geq d_{ij}$ for $(i,j) \in L$, but must meet $S_j - S_i \geq l_{ij}$ as well. Otherwise, in the case that $l_{ij} > d_{ij}$, at least one temporal constraint is not fulfilled. As a consequence, we replace temporal constraints (2.3.7) by

$$S_j - S_i \geq l_{ij}, \quad i, j \in \mathcal{J}. \tag{3.1.1}$$

Example 3.1. Consider the network in Example 2.1. Notice that $d_{02} = 0$ (arc not drawn in picture), while $l_{02} = 1$. Hence, it should be ensured that $S_2 - S_0 \geq 1$, since if $S_2 - S_0 = S_0 = 0$, the time lag $d_{12} = 1$ can not be fulfilled.

□

The problem can be restricted by using (3.1.1) instead of (2.3.7), which results in a modified problem, with a smaller set of time-feasible schedules. But this requires to compute longest path lengths between all pair of nodes. Since we possibly have non-negative arc lengths, which arise from maximum time lags, we can not do this by Dijkstra's Shortest Path algorithm as described in [9], so Bellman-Ford's algorithm of [3] or Floyd-Warshall's algorithm of [10] for shortest paths should be used.

Remark 3.1. 1. The longest path pairs on a network can be computed by above mentioned algorithms, by multiplying all arc lengths (or equivalently: matrix elements) by -1.

2. Due to maximum time lags N is possibly cyclic. For cycles of *positive* length, the Longest Path problem is not well defined, since we can run through that cycle infinitely often, which results in longest path lengths of ∞. However, above algorithms can detect such cycles. For cycles of negative length, there is no problem.

3. All nodes have at least one incoming and one outgoing arc: for nodes $j \in \{1, \ldots, n\}$ this is clear since both $d_{0j} \geq 0$ and $d_{j,n+1} \geq 0$. For nodes 0 and $n+1$, this is ensured by the introduction of the arc $(n+1, 0)$ with weigth $-T$.

4. As a corollary, $|l_{ij}| < \infty$ for all $i, j \in V$.

By N^{LP} we denote the complete network where an arc (i, j) has weight l_{ij}. Equivalently, with a slight abuse of notation, N^{LP} is its adjacency matrix representation, where $N^{LP}_{(i,j)} = l_{ij}$. Remark 3.1(2) can be used as a check whether there does not exist a time-feasible schedule: if $l_{ij} + l_{ji} > 0$ for $i, j \in \mathcal{J}$, the problem is infeasible.

Example 3.2. Consider the network in Figure 3.1 (compare with Figure 2.1). We compute N^{LP} as in Table 3.1

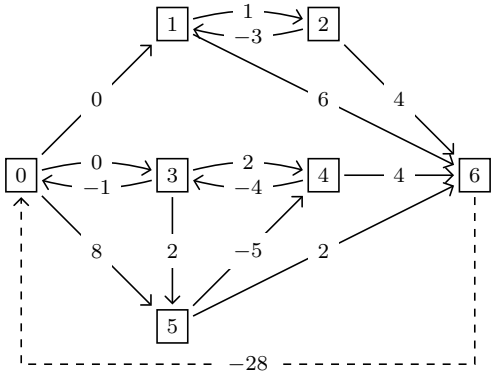

Figure 3.1: N with added backward arc

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 3 & 8 & 10 \\ -22 & 0 & 1 & -22 & -19 & -14 & 6 \\ -24 & -3 & 0 & -24 & -21 & -16 & 4 \\ -1 & -1 & 0 & 0 & 2 & 7 & 9 \\ -24 & -24 & -23 & -4 & 0 & 3 & 5 \\ -26 & -26 & -25 & -9 & -5 & 0 & 2 \\ -28 & -28 & -27 & -28 & -25 & -20 & 0 \end{pmatrix}$$

Table 3.1: Matrix of longest paths

□

The first row of N^{LP} plays an important role, as it denotes the longest path length of node 0 to every other node. All these entries are non-negative, because node 0 only has outgoing arcs with non-negative weight and $l_{0j} \geq d_{0j}$. In terms of activities and time lags: $N^{LP}_{(0,j)}$ marks the

earliest time job j may start, since it satisfies all temporal constraints. Hence, $ES_j = N^{LP}_{(0,j)}$. Besides, the first column denotes the longest path lengths of node $i \neq 0$ to node 0. Since node 0 does not have any incoming arcs with non-positive weight, all these numbers are non-positive. As a consequence, these numbers correspond to maximum time lags: $-N^{LP}_{(i,0)}$ designates the *latest* time job i may start: $LS_i = -N^{LP}_{(i,0)}$. Hence, $TW(j) = \{N^{LP}_{(0,j)}, \ldots, -N^{LP}_{(j,0)}\}$.

Remark 3.2.
1. If there are no resources involved in the project scheduling problem, computing the earliest starting times would solve the problem to optimality, because $S_{opt} = ES$, the schedule in which each job is scheduled on its earliest starting time. Moreover, the earliest starting time of job $n+1$, which is $N^{LP}_{(0,n+1)}$, is a lower bound for the makespan of the problem.

2. One is not restricted to (2.1.1) for the upper bound of the makespan. Also a prescribed deadline $\overline{d}$ could be used. In that case, the arc from node $n+1$ to node 0 must have weight $-\overline{d}$. This will only influence the latest starting times. Introducing a prescribed deadline which is not feasible however, e.g., $\overline{d} < N^{LP}_{(0,n+1)}$, yields a cycle of positive length.

Due to Remark 3.2(2), it can be easily tested wheter a prescribed deadline $\overline{d}$ is infeasible. However, this says nothing about feasibility of the problem: a deadline $\overline{d}$ may be feasible, but the problem inclusive resource-constraints may be not.

3.2 Forbidden Set Approach (FSP)

In this paragraph, we turn to a procedure that strengthens the time windows found by the longest-path approach, also considering the resource constraints. For instance, if jobs i and j have overlapping time windows, and $r_{ik} + r_{jk} > \overline{R_k}$ for at least one resource type $k \in \mathcal{R}$, then these two jobs can not be processed simultaneously. (E.g., jobs 1 and 3 of Example 2.1). This leads to the definition of a *2-fold forbidden set*:

Definition 3.1. The *2-fold Forbidden Set* $\mathcal{F}$ is the set

$$\mathcal{F} := \{(i,j) : i,j \in \mathcal{J}, \exists k \in \mathcal{R} : r_{ik} + r_{jk} > \overline{R_k}\}.$$

If in a time-feasible schedule for an $(i,j) \in \mathcal{F}$ it holds that both $l_{ij} < p_i$ and $l_{ji} < p_j$, then jobs i and j may overlap. Notice that always one of both conditions is fulfilled, since either $l_{ij} \leq 0$ or $l_{ji} \leq 0$, with equality only when $l_{ij} = l_{ji} = 0$. Without loss of generality, suppose that $l_{ij} > 0$ and $l_{ji} < 0$. Then $l_{ji} < p_j$ is always satisfied. If this is the only one that holds, then $l_{ij} \geq p_i$ and there is no resource conflict at all, since job j is already forced to start after the finish of job i. Hence, it suffices to consider just

$$\mathcal{F}' := \{(i,j) : (i,j) \in \mathcal{F}, l_{ij} < p_i, l_{ji} < p_j\}. \tag{3.2.1}$$

Example 3.3. In Example 2.1, $\mathcal{F} = \mathcal{F}' = \{(1,3),(1,4),(3,4)\}$. Since $TW(1) = \{0,\ldots,22\}$ and $TW(3) = \{0,1\}$ and jobs 1 and 3 can not be processed simultaneously, there is an additional restriction on the earliest start of job 1, namely $ES_1 = 1$. As a consequence a constraint $S_1 \geq S_3 + 4$ has to be introduced. Moreover, we have to make sure that either $S_1 \geq S_4 + 4$ or $S_4 \geq S_1 + 6$. The *forbidden set approach* consists of introducing such time lags.

□

We denote the time lags introduced due to resolving a resource conflict by l^*_{ij}. As a consequence, $d_{ij} \leq l_{ij} \leq l^*_{ij}$. This leads to the following theorem, as in [6], [8].

Theorem 3.1. *Suppose that for $(i,j) \in \mathcal{F}'$ with $l_{ij} < p_i$ it holds that $l_{ij} > -p_j$. Then $S_j \geq S_i + p_i$ for any feasible schedule S with $S_{n+1} \leq T$.*

Proof. Suppose that for $(i,j) \in \mathcal{F}'$ with $l_{ij} < p_i$ it holds that $l_{ij} > -p_j$. This means that $S_j - S_i \geq l_{ij} > -p_j$, which reduces to $S_j + p_j > S_i$. We have two cases: $l_{ij} \geq 0$ or $0 > l_{ij} > -p_j$.

In case I, job j starts possibly earlier than job i, with $S_i - p_j + 1$ as earliest starting time. But then the earliest completion time of job j is $S_i - p_j + 1 + p_j = S_i + 1$, so i and j overlap for sure if job j starts earlier than job i. This has to be resolved. Since $l_{ij} \geq 0$, it is clear that job i must precede job j in a feasible schedule. That is, $S_j \geq S_i + p_i$ is satisfied.

In case II, the earliest possible starting time of job j is $S_i + l_{ij}$, yielding an earliest possible completion time of $S_i + l_{ij} + p_j > S_i$. Hence, jobs i and j overlap surely if job j starts earlier than job i, which has to be resolved. A left-shift of job j or a right-shift of job i is not possible, since $S_i + l_{ij} \leq S_j$ is not necessarily fulfilled then. Thus the only thing possible is a right-shift of job j or a left-shift of job i such that job i precedes job j in a feasible schedule: $S_j \geq S_i + p_i$ is satisfied. □

Corollary 3.2. *If for $(i,j) \in \mathcal{F}'$ with $l_{ij} < p_i$ it holds that $LS_i < ES_j + p_j$, then $S_j \geq S_i + p_i$ for any feasible schedule S with $S_{n+1} \leq T$.*

Proof. In terms of longest-path lengths, $LS_i < ES_j + p_j$ reads as $-l_{i0} < l_{0j} + p_j$, in other words: $l_{i0} + l_{0j} > -p_j$. By the triangle inequality, $l_{ij} > l_{i0} + l_{0j}$. Hence, $l_{ij} > -p_j$ and Theorem 3.1 can be applied. □

If for $(i,j) \in \mathcal{F}'$ it holds that $l_{ji} < p_j$ but $l_{ji} \leq -p_i$, we can again apply Theorem 3.1, but now with the roles of i and j interchanged. Hence, $S_i \geq S_j + p_j$ in any feasible schedule S with $S_{n+1} \leq T$. Notice that the conditions $l_{ij} < p_i$ and $l_{ji} < p_j$ are always fulfilled since $(i,j) \in \mathcal{F}'$.

But if neither the additional condition of Theorem 3.1, i.e., $l_{ij} \leq -p_j$, nor the additional condition of this 'reversed' theorem, i.e., $l_{ij} \geq p_i$, holds, then it is not clear which job has to be scheduled first. However, we know that either job i must precede job j or vice versa. Consider two jobs $h, k \in \mathcal{J}$, as done in [20], for which we will introduce a time lag as follows, using a 2-fold forbidden set (i,j) with $l_{ij} < p_i$. Let S_h be the (fixed) starting time of job h.

Consider the situation where job i precedes job j. By Remark 3.1(4), both $|l_{hi}| < \infty$ and $|l_{jk}| < \infty$ hold. That is, longest paths $h \leftrightarrow i$ and $j \leftrightarrow k$ exist and are finite, yielding a temporal constraint of type (3.1.1). Hence, the earliest possible starting time of job i is $S_h + l_{hi}$ and the earliest possible completion time of job i is $S_h + l_{hi} + p_i$. Moreover, it is the earliest possible starting time of activity j, yielding an earliest possible starting time of job k of $S_h + l_{hi} + p_i + l_{jk}$.

In the second case, where activity j precedes activity i, a similar analysis can be done, which results in a earliest possible starting time of job k of $S_h + l_{hj} + p_j + l_{ik}$.

Beforehand we do not know in which situation we are, but one of the two is valid, so it suffices to consider the minimum of both earliest possible starting times of job k. In other words, $S_k \geq S_h + \min(l_{hi} + p_i + l_{jk}, l_{hj} + p_j + l_{ik})$. Since $S_k \geq S_h + l_{hk}$ must be fulfilled as well, we define for $(i,j) \in \mathcal{F}'$:

$$l^*_{hk}(i,j) := \max(l_{hk}, \min(l_{hi} + p_i + l_{jk}, l_{hj} + p_j + l_{ik})). \tag{3.2.2}$$

Hence, $l^*_{hk}(i,j) \geq l_{hk}$. For all forbidden pairs (i,j) of $\mathcal{F}'$ such an modified time lag can be computed. Of all these modified time lags $l^*_{hk}(i,j)$, the maximum is taken to ensure that all induced time lags $l^*_{hk}(i,j)$ are satisfied:

$$l^*_{hk} := \max_{(i,j)\in\mathcal{F}'} l^*_{hk}(i,j). \tag{3.2.3}$$

Then, the new temporal constraint

$$S_k - S_h \geq l^*_{hk}, \quad h,k \in \mathcal{J} \tag{3.2.4}$$

is introduced. Using (3.2.4) instead of (3.1.1) results (again) in a modified problem, in which the set of time-feasible schedules further reduces, since $d_{ij} \leq l_{ij} \leq l^*_{ij}$ for $i,j \in \mathcal{J}$. This leads to a new complete activity-on-node network, denoted by N^{FS}, with corresponding adjacency matrix $N^{FS}_{(i,j)}$. At the end we once again have to perform a longest-path computation on N^{FS}. In pseudo-code, this procedure is as follows, as proposed in [20].

Algorithm 3.1 Forbidden Set Procedure (FSP)

Input: Instance of $PS|temp|C_{max}$ and an upper bound (or deadline) UB of the makespan.
Output: $\mathcal{F}'$, distances l^*_{ij} such that $l^*_{ij} \geq l_{ij} \geq d_{ij},\ (i,j) \in \mathcal{J} \times \mathcal{J}$, infeasibility (if cycle of length > 0).

1: Compute the longest path lengths l_{ij} in network N with $d_{n+1,0} = -UB,\ (i,j) \in \mathcal{J} \times \mathcal{J}$
2: Determine $\mathcal{F}' := \{(i,j) : (i,j) \in \mathcal{F}, l_{ij} < p_i, l_{ji} < p_j\}$
3: $stop := 0$
4: **while** $\mathcal{F}' \neq \emptyset$ and $stop = 0$ **do**
5: $\quad stop := 1$
6: $\quad$ **for** all $(i,j) \in \mathcal{F}'$ **do**
7: $\quad\quad$ **for** all $h,k \in \mathcal{J}$ **do**
8: $\quad\quad\quad$ **if** $l_{ij} > \min(l_{hi} + p_i + l_{jk}, l_{hj} + p_j + l_{ik})$ **then**
9: $\quad\quad\quad\quad l^*_{hk}(i,j) := l_{hk}$
10: $\quad\quad\quad$ **else**
11: $\quad\quad\quad\quad$ **if** $l_{ij} \leq -p_j$ and $l_{ij} \geq p_i$ **then**
12: $\quad\quad\quad\quad\quad l^*_{hk}(i,j) := \min(l_{hi} + p_i + l_{jk}, l_{hj} + p_j + l_{ik})$
13: $\quad\quad\quad\quad$ **else**
14: $\quad\quad\quad\quad\quad$ **if** $l_{ij} > -p_j$ **then**
15: $\quad\quad\quad\quad\quad\quad l^*_{hk}(i,j) := l_{hi} + p_i + l_{jk}$
16: $\quad\quad\quad\quad\quad$ **else**
17: $\quad\quad\quad\quad\quad\quad l^*_{hk}(i,j) := l_{hj} + p_j + l_{ik}$
18: $\quad\quad\quad\quad\quad$ **end if**
19: $\quad\quad\quad\quad$ **end if**
20: $\quad\quad\quad$ **end if**
21: $\quad\quad\quad l^*_{hk} := \max_{(i,j) \in \mathcal{F}'} l^*_{hk}(i,j)$
22: $\quad\quad\quad$ **if** $h = k$ and $l^*_{hk}(i,j) > 0$ **then return** cycle of positive length!
23: $\quad\quad\quad$ **end if**
24: $\quad\quad$ **end for**
25: $\quad\quad$ **if** $l^*_{ij}(i,j) \geq p_i$ **then**
26: $\quad\quad\quad stop := 0,\ \mathcal{F}' := \mathcal{F}' \backslash \{(i,j)\}$
27: $\quad\quad$ **end if**
28: $\quad$ **end for**
29: **end while**
30: Correct the distances $l^*_{hk}(i,j)$ in network $N^{FS},\ (h,k) \in \mathcal{J} \times \mathcal{J}, (i,j) \in \mathcal{F}'$

Line 1: Can be done as described in paragraph 3.1. This results in network N^{LP}, with longest path lengths l_{ij}.

Line 8+9: Then, in equation (3.2.2), the maximum is attained for l_{hk}.

Line 10: The maximum in (3.2.2) is attained for $\min(l_{hi} + p_i + l_{jk}, l_{hj} + p_j + l_{ik})$.

Line 11+12: Here the condition for Theorem 3.1 is *not* met for $(i,j) \in \mathcal{F}'$. So here it is not known whether job i precedes job j or vice versa.

Line 13: $l_{ij} > -p_j$ *or* $l_{ji} > -p_i$

Line 14+15: Precisely as the condition in Theorem 3.1. Hence, job i has to precede job j: the order of the jobs must be h, i, j, k.

Line 16+17: The roles of i and j are interchanged: 'reversed' Theorem 3.1 holds, leading to an order of h, j, i, k.

Line 21: The problem is not feasible, since there is a cycle of positive length.

Line 24+25: If this is the case, job i and job j can not be processed simultaneously anymore. Hence, we can delete (i, j) from $\mathcal{F}'$.

Line 29: Can be done in the same way as before. The corrected distances are called $l^*_{hk}(i, j)$ as well.

Example 3.4. Consider once again the project of Figure 2.1 and Table 2.1. Now we give this project a prescribed deadline of $\overline{d} = UB = 15$. This yields the adjacency matrix N:

$$N = \begin{pmatrix} 0 & 0 & -\infty & 0 & -\infty & 8 & -\infty \\ -\infty & 0 & 1 & -\infty & -\infty & -\infty & 6 \\ -\infty & -3 & 0 & -\infty & -\infty & -\infty & 4 \\ -1 & -\infty & -\infty & 0 & 2 & 2 & -\infty \\ -\infty & -\infty & -\infty & -4 & 0 & -\infty & 4 \\ -\infty & -\infty & -\infty & -\infty & -5 & 0 & 2 \\ -15 & -\infty & -\infty & -\infty & -\infty & -\infty & 0 \end{pmatrix}$$

Computing the longest-path-matrix, we find:

$$N^{LP} = \begin{pmatrix} 0 & 0 & 1 & 0 & 3 & 8 & 10 \\ -9 & 0 & 1 & -9 & -6 & -1 & 6 \\ -11 & -3 & 0 & -11 & -8 & -3 & 4 \\ -1 & -1 & 0 & 0 & 2 & 7 & 9 \\ -5 & -5 & -4 & -4 & 0 & 3 & 5 \\ -10 & -10 & -9 & -9 & -5 & 0 & 2 \\ -15 & -15 & -14 & -15 & -12 & -7 & 0 \end{pmatrix}$$

Moreover, $\mathcal{F}' = \{(1, 3), (1, 4), (3, 4)\}$.

For $(i, j) = (1, 3)$ the following distance matrix with distances l^*_{hk}, $h, k \in \mathcal{J}$, is obtained. For the uncolored numbers (h, k) it holds that $l^*_{hk} = l_{hk}$, while the colored ones are strictly increased: $l^*_{hk} > l_{hk}$, i.e., in equation (3.2.2) the second term was attained as maximum.

$$\begin{pmatrix} 0 & 4 & 5 & 0 & 3 & 8 & 10 \\ -9 & 0 & 1 & -9 & -6 & -1 & 6 \\ -11 & -3 & 0 & -11 & -8 & -3 & 4 \\ -1 & 4 & 5 & 0 & 2 & 7 & 10 \\ -5 & 0 & 1 & -4 & 0 & 3 & 6 \\ -10 & -5 & -4 & -9 & -5 & 0 & 2 \\ -15 & -11 & -10 & -15 & -12 & -7 & 0 \end{pmatrix}$$

It is seen that $l^*_{13} = -9 \not\geq p_3 = 4$, so $(1, 3)$ is not deleted from $\mathcal{F}'$. After considering $(i, j) = (1, 4)$, we find the following distances:

$$\begin{pmatrix} 0 & 7 & 8 & 0 & 3 & 8 & 13 \\ -9 & 0 & 1 & -9 & -6 & -1 & 6 \\ -11 & -3 & 0 & -11 & -8 & -3 & 4 \\ -1 & 6 & 7 & 0 & 2 & 7 & 12 \\ -5 & 4 & 5 & -4 & 0 & 3 & 10 \\ -10 & -1 & 0 & -9 & -5 & 0 & 5 \\ -15 & -8 & -7 & -15 & -12 & -7 & 0 \end{pmatrix}$$

Now, $l^*_{14} = -6 \not\geq p_4 = 4$, so $(1,3)$ is neither deleted from $\mathcal{F}'$. We now consider the last element of $\mathcal{F}' : (3,4)$.

$$\begin{pmatrix} 0 & 8 & 9 & 0 & 4 & 8 & 14 \\ -9 & 0 & 1 & -9 & -5 & -1 & 6 \\ -11 & -3 & 0 & -11 & -7 & -3 & 4 \\ -1 & 8 & 9 & 0 & 4 & 7 & 14 \\ -5 & 4 & 5 & -4 & 0 & 3 & 10 \\ -10 & -1 & 0 & -9 & -5 & 0 & 5 \\ -15 & -7 & -6 & -15 & -11 & -7 & 0 \end{pmatrix}$$

Moreover, $l^*_{34} = 4 \geq p_4 = 4$. Hence, it is ensured that jobs 3 and 4 do not overlap, since job 4 is restricted to start at/after the finish time of job 3. Thus we can delete $(3,4)$ from $\mathcal{F}'$, and $stop = 0$ again. Now, we repeat the algorithm, since $\mathcal{F}' \neq \emptyset$ and $stop = 0$. However, no difference occurs for considering $(1,3)$ and $(1,4)$, so $stop = 1$. Besides, it turns out that the distances l^*_{ij}, $i,j \in \mathcal{J}$ all satisfy the triangle-inequality, so the algorithm terminates: the above matrix is N^{FS}, with distances l^*_{ij}. Again, time windows can be obtained by $TW(j) = \{N^{FS}_{(0,j)}, \ldots, -N^{FS}_{(j,0)}\}$. Notice that $N^{FS}_{(0,6)} = 14$ is a lower bound for the makespan of the project, which is larger than the lower bound of the makespan found by the longest-path approach, which was 10.

It should be noticed that in considering $(1,3)$ the columns corresponding to jobs 1,2 and 6 are increased. This is due to the fact that jobs 1 and 3 can not be processed simultaneously and since job 3 can only start on time 0 or 1, the earliest starting time of job 1 should be later. Since job 2 and 6 depend on job 1, the earliest starting times of job 2 and 6 are increased as well. Moreover, the starting time of job 4 is also very restricted and related to job 3, so a similar argument holds for considering $(1,4)$ and $(3,4)$.

□

Remark 3.3. Algorithm 3.1 can test whether a given deadline is feasible: if it is not, the algorithm terminates since there is a cycle of positive length. Hence, lower bounds on the makespan of a project can be computed by iteratively performing the Forbidden Set Procedure. This will be the subject of paragraph 4.1.

4 Simple Lower Bounds

Lower bounds on the makespan of a project are important for pratical purposes. For instance, think of a construction project, in which the contractor has to make an estimate of the duration of the project. But also for theoretical reasons, lower bounds are important: if by some heuristics a coinciding lower and upper bound are found, then the schedule corresponding to

the lower (or upper) bound is an optimal one. In paragraph 4.1 we will consider so-called *destructive lower bounds*, which can be computed using the Forbidden Set Procedure (algorithm 3.1) of paragraph 3.2. An alternative lower bound, the *workload based lower bound*, is the subject of paragraph 4.2, as in [20]. A more complex lower bound, the *Lagrangian lower bound* introduced in [17], will be discussed in chapter 5.

It should be noticed that we already found a valid lower bound for the makespan of an instance of $PS|temp|C_{max}$: it is ES_{n+1}, the earliest starting time of finish-job $n+1$, which can be computed by the longest-path approach of paragraph 3.1. For simplicity, we will denote this lower bound by TLB. This is the lower bound if we relax the problem by omitting the resource constraints.

Instead of ignoring the resource constraints, the temporal constraints can be discarted to obtain an alternative lower bound, which we will call RLB. In this relaxation, we split a job j with resource requirement r_{jk} in jobs $j_{k1}, \ldots, j_{kr_{jk}}$ for each $k \in \mathcal{R}$. So we split job $j \in \{1, \ldots, n\}$ in $\sum_{k \in \mathcal{R}} r_{jk}$ jobs. For $k, i \in \mathcal{R}$ and $l \in \{1, \ldots, r_{jk}\}$ the resource requirement of job j_{kl} is as follows:

$$r_{j_{kl}i} = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{otherwise} \end{cases}$$

So each job now requires *exactly* one resource, except the artificial start and finish job. In addition, $p_{j_{kl}} = p_j$: all obtained jobs have processing time which is equal to the original processing time. Moreover, we allow preemption of jobs. We define

$$w_k := \sum_{j \in \mathcal{J}} r_{jk} p_j \tag{4.0.5}$$

to be the *workload* of resource $k \in \mathcal{R}$. In addition:

$$W_k := \left\lceil \frac{w_k}{\overline{R_k}} \right\rceil \tag{4.0.6}$$

which represents the amount of time which is needed to process all jobs j_{kl} which require resource $k \in \mathcal{R},\ l \in \{1, \ldots, r_{jk}\}$. Since resources can simultaneously work on jobs, it is easily seen that in order to process all $\sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{R}} r_{jk} + 2$ jobs, we have to take the maximum. This will be a valid lower bound of the makespan of a project:

$$RLB := \max_{k \in \mathcal{R}}(W_k) \tag{4.0.7}$$

We want a lower bound which is as tight as possible. Thus, our initial lower bound LB_{init}, which will be improved in the upon chapters, is defined as $LB_{init} := \max(TLB, RLB)$.

Example 4.1. Consider the $PS|temp|C_{max}$-instance in Figure 4.1 and Table 4.1, for which we already know that $TLB = 10$, as computed in Example 3.2. However, we now have 3 types of resources, with capacities $\overline{R_1} = 8,\ \overline{R_2} = 7$ and $\overline{R_3} = 10$.

Notice that we have, once the splitting is done, there are 51 'real' jobs, plus an artificial start and finish one. Workloads of resources can be computed by (4.0.5), which results in $W_1 = \lceil 9.25 \rceil = 10,\ W_2 = \lceil 10.286 \rceil = 11,\ W_3 = \lceil 5.6 \rceil = 6$. Hence, by (4.0.7), $RLB = 11$. Consequently, $LB_{init} = \max(10, 11) = 11$.

□

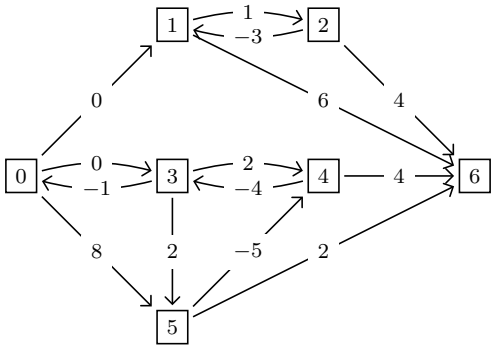

Figure 4.1: Project network N

Job	p_j	r_{j1}	r_{j2}	r_{j3}
0	0	0	0	0
1	6	4	2	4
2	4	1	1	2
3	4	5	7	0
4	4	6	5	3
5	2	1	4	6
6	0	0	0	0

Table 4.1

4.1 Destructive Lower Bound (DLB)

As the name suggests, a lower bound will not be derived in a *constructive* way, but the procedure consists of testing and proving whether a prescribed upper bound is infeasible. If it is infeasible, this upper bound is *destructed* and has to be increased. In the same way as was done in paragraph 3.1, TLB can be increased using the Forbidden Set Procedure of Algorithm 3.1 in addition to the longest-path approach. In order to perform this procedure, we need an upper bound UB_{init}, which can be taken equal either to T of equation (2.1.1) or to a prescribed deadline $\overline{d}$. If $LB_{init} > UB_{init}$, it is clear that there is no feasible schedule.

The Destructive Lower Bound approach consist of generating monotonically increasing and decreasing sequences LB_i and UB_i respectively of lower and upper bounds in iteration $i \in \mathbb{N}$, as done in [20]. Here upper bounds are no upper bounds on the makespan of the problem, but just an estimate of an upper bound for the lower bound. We know that the best lower bound is in the integral interval $[\max_i(LB_i), \min_i(UB_i)]$. So we try some values g_i as upper bound in this interval in a systematic fashion by performing the Forbidden Set Procedure. That is, we give weight $-g_i$ to the arc from $n+1$ tot 0 in network N. We have two possibilities:

- If in iteration i the Forbidden Set Procedure finds a cycle of positive length, we know that the best lower bound of the problem is in the integral interval $[g_i + 1, UB_i]$, since g_i can not be attained as makespan. That is: the lower bound LB_i is rejected. So in the next iteration $i+1$ we take $LB_{i+1} = g_i + 1,\ UB_{i+1} = UB_i$.

- If in iteration i the Forbidden Set Procedure terminates, then it is possibly probable to do the project within g_i time units. Hence, UB_i is a valid upper bound on LB_i, so it suffices to investigate the integral interval $[LB_i, g_i - 1]$. Because the Forbidden Set Procedure terminated, a value $N^{FS}_{(0,n+1)}$ is found, which is also a lower bound on the project makespan, since it is the earliest starting time of the finish job $n+1$. The strongest one of these is taken. Thus, in iteration $i+1$ we take $LB_{i+1} = \max(LB_i, N^{FS}_{(0,n+1)}),\ UB_{i+1} = g_i - 1$.

We choose g_i as the rounded mean of LB_i and UB_i in iteration i. The algorithm succesfully terminates if $LB_i > UB_i$: the lower bound can not be decreased any further. However, if UB_{init} already yielded a cycle of positive length in algorithm 3.1, then there is no feasible schedule. The Destructive Lower Bound algorithm is summarized as follows:

Algorithm 4.1 Destructive Lower Bound Algorithm (DLB)

Input: Instance of $PS|temp|C_{max}$, bounds $LB_{init},\ UB_{init}$ on the project makespan.
Output: Destructive Lower Bound DLB.

```
1: LB_1 := LB_init, UB_1 := UB_init
2: i := 1
3: while LB_i ≤ UB_i do
4:     g_i := ⌈(LB_i + UB_i)/2⌉
5:     Perform the Forbidden Set Procedure with upper bound g_i.
6:     if FSP finds cycle of positive length then
7:         if g_i = UB_1 then
8:             There is no feasible schedule!
9:         else
10:            LB_{i+1} := g_i + 1
11:            UB_{i+1} := UB_i
12:        end if
13:    else
14:        UB_{i+1} := g_i − 1
15:        LB_{i+1} := max(N^FS_{0,n+1}, LB_i)
16:    end if
17:    i := i + 1
18: end while
19: DLB := LB_i
```

Example 4.2. For the problem of Example 4.1, starting with $LB_{init} = 11,\ UB_{init} = T = 28$, the algorithm produces the following outcomes:

$i = 1$**:** $LB_1 = 11,\ UB_1 = 28$: FSP accepts $g_1 = 20$ and $N^{FS}_{0,n+1} = 14$.
$i = 2$**:** $LB_2 = \max(11, 14) = 14,\ UB_2 = 19$: FSP accepts $g_2 = 17$ and $N^{FS}_{0,n+1} = 14$.
$i = 3$**:** $LB_3 = 14,\ UB_3 = 16$: FSP accepts $g_3 = 15$ and $N^{FS}_{0,n+1} = 14$.
$i = 4$**:** $LB_4 = 14,\ UB_4 = 14$: FSP accepts $g_4 = 14$ and $N^{FS}_{0,n+1} = 14$.
$i = 5$**:** $LB_5 = 14,\ UB_5 = 13$: Algorithm terminates with $DLB = LB_5 = 14$.

In this example, each g_i is accepted, but in general that is certainly not the case.

□

4.2 Workload based Lower Bound (WLB)

The more activities or resources an instance of $PS|temp|C_{max}$ has, the more computation time it takes to perform the Forbidden Set Procedure, since the 2-fold forbidden set $\mathcal{F}$ is larger. Hence, the Destructive Lower Bound procedure is less effective. Then, an alternative lower bound can be used which will be called the *workload based lower bound*, as in [20]. The idea of this lower bound is similar to that of RLB. We define:

$$p_j(t) := (\min(p_j, ES_j + p_j - t))^+, \tag{4.2.1}$$

where $j \in \mathcal{J},\ t \in \mathbb{Z}_{\geq 0}$ and ES_j is the earliest starting time of job j. This number clearly represents a lower bound on the processing time of job j and can be interpreted as the amount of time job j still has to be processed in the real interval $[t, \infty]$. Hence,

$$w_k(t) := \sum_{j \in \mathcal{J}} r_{jk} p_j(t) \tag{4.2.2}$$

represents a lower bound on the workload w_k of resource $k \in \mathcal{R}$ at time t. Similar by equation (4.0.6),

$$W_k(t) := \left\lceil \frac{w_k(t)}{R_k} \right\rceil \tag{4.2.3}$$

can be defined, which is a lower bound on the time resource $k \in \mathcal{R}$ is needed for processing the remainder of all jobs. Moreover, $ES_j + \max_{k \in \mathcal{R}}(W_k(ES_j))$ represents the earliest finish time of job j. To obtain the workload-based lower bound on the makespan of the project, we have to consider the maximum of all earliest finish times. Hence,

$$WLB := \max_{j \in \mathcal{J}}(ES_j + \max_{k \in \mathcal{R}} W_k(ES_j)) \tag{4.2.4}$$

Remark 4.1. The workload-based lower bound is a stronger bound than the simple lower bounds TLB and RLB: for the artificial start job $j = 0$, we have $ES_0 = 0$ and $W_k(ES_0) = W_k$. Hence, by equation (4.0.7), $WLB \geq RLB$. For the artificial finish job $j = n + 1$ with ES_{n+1} it holds that $W_k(ES_{n+1}) = 0$. Thus, $WLB \geq ES_{n+1} = TLB$.

Example 4.3. Consider the problem of Example 4.1. The earliest starting times can be obtained by the longest-path approach in paragraph 3.1. Equations (4.2.1)-(4.2.3) are used to obtain the W_k-values. These are listed in Table 4.2. By equation (4.2.4): $WLB =$

Job	ES_j	$k = 1$	$k = 2$	$k = 3$
0	0	10	11	6
1	0	10	11	6
2	1	9	9	6
3	0	10	11	6
4	3	6	7	4
5	8	1	2	2
6	10	0	0	0

Table 4.2: W_k-table

$\max(0+11, 0+11, 1+9, 0+11, 3+7, 8+2, 10+0) = 11$. Here it turns out that WLB equals RLB and is less tight than DLB, but, as mentioned before, for instances with more jobs and resources this could be different.

□

5 Lagrangian Lower Bound (LLB)

In this chapter a more complex lower bound will be derived, the *Lagrangian Lower Bound*. In addition to the simple lower bounds derived in chapter 4, also a, possibly resource-infeasible,

schedule attaining this lower bound will be generated. However if the generated schedule is infeasible, not that many resource constraints are expected to be harmed. In this chapter, the first IP-formulation will be used, defined by constraints (2.3.1), (2.3.2), (2.3.3), (2.3.4) and (2.3.5).

5.1 Lagrange Relaxation

In [7], it is proposed to relax the resource constraints (2.3.4)

$$\sum_{j\in\mathcal{J}} r_{jk}\Big(\sum_{s=t-p_j+1}^{t} x_{js}\Big) \leq \overline{R_k},\ k\in\mathcal{R},\quad t=0,\ldots,T.$$

by introducing non-negative Lagrangian multipliers λ_{tk}, $t=0,\ldots,T$, $k\in\mathcal{R}$. As argued in [17], by relaxing the resource-constraints, the polytope described by the remaining constraints (2.3.2), (2.3.3) and (2.3.5) is integral. This results in a contribution to the objective function of equation (2.3.1) by

$$\sum_{t=0}^{T}\sum_{k\in\mathcal{R}} \lambda_{tk}\left(\sum_{j\in\mathcal{J}} r_{jk}\Big(\sum_{s=t+p_j-1}^{t} x_{js}\Big) - \overline{R_k}\right). \tag{5.1.1}$$

This can be further simplified and the following Lagrangian subproblem (LS) is obtained, where the non-defined λ-values are considered to be 0, as in Remark 2.2:

$$\text{Minimize}\quad \sum_{t=0}^{T} t x_{n+1,t} + \sum_{j\in\mathcal{J}}\sum_{t=0}^{T}\left(\sum_{k\in\mathcal{R}} r_{jk}\sum_{s=t}^{t+p_j-1}\lambda_{sk}\right)x_{jt} - \sum_{t=0}^{T}\sum_{k\in\mathcal{R}}\lambda_{tk}\overline{R_k} \tag{5.1.2}$$

subject to constraints (2.3.2), (2.3.3) and (2.3.5). To simplify the LS-problem, numbers w_{jt} are introduced as follows:

$$w_{jt} := \begin{cases} \displaystyle\sum_{k\in\mathcal{R}} r_{jk}\sum_{s=t}^{t+p_j-1}\lambda_{sk} & \text{if } j\neq n+1 \\ t & \text{if } j=n+1. \end{cases} \tag{5.1.3}$$

Equation (5.1.2) then simplifies to

$$w_\lambda(x) := \min\quad \sum_{j\in\mathcal{J}}\sum_{t=0}^{T} w_{jt}x_{jt} - \sum_{t=0}^{T}\sum_{k\in\mathcal{R}}\lambda_{tk}\overline{R_k}. \tag{5.1.4}$$

This form of the Lagrange subproblem will be used in paragraph 5.3 to compute $w_\lambda(x)$, which is a lower bound for this $PS|temp|C_{max}$-instance, since resource-constraints were relaxed. As mentioned before, there are $n\times(T+1)$ decision variables. However, this number can be reduced by performing the Forbidden Set Procedure. Once ES_j and LS_j are computed for job $j\in\mathcal{J}$, it suffices to consider only the decision variables x_{jt} with $ES_j\leq t\leq LS_j$, leading to a new Lagrangian subproblem with less decision variables, and thus less Lagrangian multipliers, but with the same set of feasible solutions. This new Lagrangian subproblem, which we from now on will just call Lagrangian subproblem, will become useful in the next paragraph.

5.2 Minimum Cut Approach

As in [17], the Lagrangian subproblem (5.1.4) subject to constraints (2.3.2), (2.3.3) and (2.3.5), can be transformed to a *minimum cut problem*. The $-\sum_{t=0}^{T}\sum_{k\in\mathcal{R}}\lambda_{tk}\overline{R_k}$ term of equation (5.1.4) will be omitted in this paragraph, since there are no decision variables x_{jt} in it. As a consequence, this term is constant. Hence, the objective function for this paragraph is:

$$\text{Minimize} \quad \sum_{j\in\mathcal{J}}\sum_{t=0}^{T} w_{jt}x_{jt}. \tag{5.2.1}$$

A network $D = (V, A)$ is constructed as follows: for time t for which $ES_j \le t \le LS_j + 1$ for $j \in \mathcal{J}$ holds, a node v_{jt} is introduced. Moreover, two nodes a and b are added, which represent the source and sink of V:

$$V := \Big\{v_{jt} : j \in \mathcal{J}, t \in \{ES_j, \ldots, LS_j, LS_j + 1\}\Big\} \cup \{a, b\}. \tag{5.2.2}$$

The arc set A consists of three types of arcs.

Assignment arcs. For each job $j \in \mathcal{J}$, assignment arcs are introduced between all nodes v_{jt} and $v_{j,t+1}$, where $ES_j \le t \le LS_j$. The capacity of each assignment arc $(v_{jt}, v_{j,t+1})$ is given by w_{jt} as defined in equation (5.1.3).

Temporal arcs are corresponding to the temporal constraints. If there is a time lag d_{ij} between jobs i and j (minimal or maximal), then there is a temporal arc between all pairs of starting times, due to this time lag. That is: an arc (v_{it_i}, v_{jt_j}) exists between all pairs $ES_i \le t_i \le LS_i + 1$, $ES_j \le t_j \le LS_j + 1$, where $t_j = t_i + d_{ij}$. The capacity of each temporal arc is infinite. This is a slight modification of the approach of [17]. There, temporal arcs between all pairs satisfying $ES_i + 1 \le t_i \le LS_i$, $ES_j + 1 \le t_j \le LS_j$ are introduced. We think our approach is more intuitive, but it does not matter for the minimum cut, due the auxiliary arcs which will be defined next.

Auxiliary arcs are added to connect the source a and sink b with the remaining network. Arcs (a, v_{j,ES_j}) and (v_{j,LS_j+1}, b) are introduced for each $j \in \mathcal{J}$, all with infinite capacity.

Example 5.1. Consider the project in Figure 5.1 with one resource type and $\overline{R} = 4$. The processing times and resource requirements are listed in Table 5.1, as well as the earliest and latest starting times, as computed by the Forbidden Set Approach of Algorithm 3.1. We use the prescribed upper bound $\overline{d} := 5$. Instead of using N^{FS}, the original network N will be

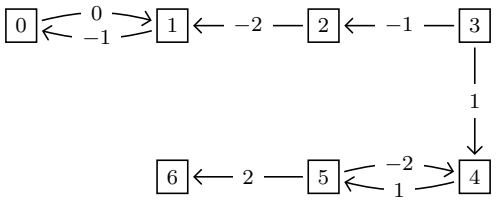

Figure 5.1: Project network N

Job	p_j	r_j	ES_j	LS_j
0	0	0	0	0
1	2	2	0	1
2	1	2	0	2
3	1	2	0	1
4	1	1	1	2
5	2	3	2	3
6	0	0	4	5

Table 5.1

used in this example for the construction of network D. Otherwise D would become very

messy, since FSP adds an arc between all pair of nodes. Notice that due to the construction of this instance, job 5 always determines the makespan of the project, so in D the time lags $d_{j6} := p_j$ are omitted, for $j \neq 5$. Moreover, for simplicity the time lags $d_{0j} := 0$ for $j \neq 1$ are neglected. Assume $\lambda_{jt} := 1$ is chosen as Lagrange multiplier for all $j \in \mathcal{J}$ and $0 \leq t \leq 5$. Then $w_{jt} := r_j p_j$ for $j \neq 6$, $0 \leq t \leq 5$ and $w_{6t} := t$, $0 \leq t \leq 5$. This results in the network in Figure 5.2. In this picture, the capacities of the arcs are written in the arcs, while the arcs with a white arrowhead have infinite capacity.

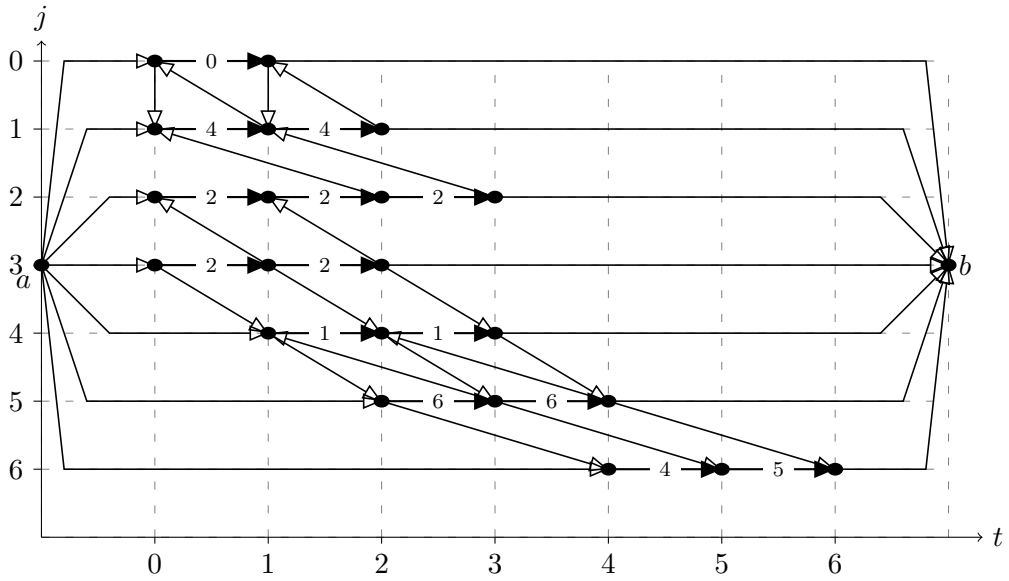

Figure 5.2: Transformation to network D

□

A cut is an ordered pair $(C, \overline{C})$ of disjoint sets $C, \overline{C} \subset V$ with $C \cup \overline{C} = V$ and $a \in C$, $b \in \overline{C}$. The capacity is the sum of the weights of the arcs with tail in C and head in $\overline{C}$. A minimum cut is a cut with minimal capacity. In the remainder, only cuts with exactly one assignment arc for each job $j \in \mathcal{J}$ are important, but that is not covered by our definition of a cut. This drawback for minimum cuts is cured by Lemma 5.1.

Lemma 5.1. *A minimum cut $(C, \overline{C})$ can be transformed into a cut $(C^*, \overline{C^*})$ with the same capacity, but with exactly one assignment arc for each job $j \in \mathcal{J}$.*

Proof. Let $(C, \overline{C})$ be a minimum cut of D. Suppose its capacity is finite. Otherwise, the claim is trivial. The auxiliary arcs have infinite capacity, so for each job $j \in \mathcal{J}$, there is at least one assignment arc $(v_{jt}, v_{j,t+1})$ in $(C, \overline{C})$.

Now, suppose that there is more than one assignment arc $(v_{jt}, v_{j,t+1})$ in $(C, \overline{C})$ for some jobs $j \in \mathcal{J}$. A cut $(C^*, \overline{C^*})$ with the same capacity can be constructed as follows:

- For $j \in \mathcal{J}$, let t_j be the smallest time index for which $v_{j,t_j} \in C$, $v_{j,t_j+1} \in \overline{C}$: the 'earliest' assignment arc.
- Define $C^* := \bigcup_{j \in \mathcal{J}} \{v_{jt} : t \leq t_j\} \cup \{a\}$.

- Define $\overline{C^*} := V \backslash C^*$.

It is easily seen that $C^* \subset C$ and the set of assignment arcs of $(C^*, \overline{C^*})$ is included in the set of assignment arcs of $(C, \overline{C})$. Because $(C, \overline{C})$ is a minimum cut, the capacity of $(C^*, \overline{C^*})$ is as least as big. Hence, to prove that the capacity of $(C^*, \overline{C^*})$ equals the capacity of $(C, \overline{C})$, it suffices to prove that $(C^*, \overline{C^*})$ does not contain any of the temporal arcs. Remember that all arc weights w_{jt} are nonnegative.

To show this, suppose that there exists such a temporal arc in $(C^*, \overline{C^*})$, say (v_{is}, v_{jt}) with $s \leq t_i$, $t > t_j$. We are going to deduce a contradiction. Let k be the number of assignment arcs between t and $t_j + 1$. Formally, $k := t - (t_j + 1)$. It is easily seen that $k \geq 0$, since $t > t_j$. Rearranging some terms, we find that $t - k = t_j + 1$. Moreover, all times $t_j, t_j + 1, \ldots, t - 1$ are feasible starting times for job j. Thus, $ES_j + 1 \leq t - k \leq LS_j$.

Because $s - k \leq s \leq t_i \leq LS_i$, we find that $s - k \leq LS_i$. Assume that in addition $s - k < ES_i + 1$. Then $t - k = s - k + (t - s) < ES_i + 1 + (t - s)$. Moreover, since there is a time lag between jobs i and j of $t - s$ time units, we find $ES_i + (t - s) \leq ES_j$. Combining the last two inequalities, we find $t - k < ES_j + 1$. This is a contradiction with $t - k = t_j + 1$, since $t_j \geq ES_j$. So $s - k \geq ES_i + 1$. That is, we have $v_{i,s-k} \in C^* \subset C$, since $v_{i,s} \in C^* \subset C$, $v_{j,t-k} = v_{j,t_j+1} \in \overline{C}$ and a temporal arc $(v_{i,s-k}, v_{j,t-k})$ is contained in $(C, \overline{C})$. Hence, the minimum cut $(C, \overline{C})$ contains a temporal arc. Since temporal arcs have infinite capacity, this is a contradiction to the fact that $(C, \overline{C})$ is a minimum cut. Hence, $(C^*, \overline{C^*})$ does not contain any temporal arc. □

Lemma 5.2. *For each feasible solution x of the relaxed problem described by equations* (5.2.1), (2.3.2), (2.3.3) *and* (2.3.5), *there exists a finite capacity cut $(C, \overline{C})$ of D such that x corresponds to $(C, \overline{C})$ via the transformation*

$$x_{jt} := \begin{cases} 1 & \text{if arc } (v_{jt}, v_{j,t+1}) \text{ is in the cut,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.2.3}$$

Proof. Let x be a feasible solution of the relaxed problem described by equations (5.2.1), (2.3.2), (2.3.3) and (2.3.5). For each job $j \in \mathcal{J}$, there is exactly one $x_{j,t_j} = 1$, $0 \leq t_j \leq T$ by constraint (2.3.2). The others are 0. Define a cut $(C, \overline{C})$ by setting
$C := \bigcup_{j \in \mathcal{J}} \{v_{jt} : t \leq t_j\} \cup \{a\}$ and $\overline{C} := V \backslash C$. Due to the construction of D, all arcs (v_{j,t_j}, v_{j,t_j+1}), $j \in \mathcal{J}$ are arcs in the cut $(C, \overline{C})$. Thus, x is transformed into $(C, \overline{C})$ under the transformation of (5.2.3). Finite capacity of $(C, \overline{C})$ is ensured by the (temporal) feasibility of x. □

Corollary 5.3. *If a minimum cut has infinite capacity, then there is no feasible solution to the relaxed problem described by* (5.2.1), (2.3.2), (2.3.3) *and* (2.3.5).

Lemma 5.4. *If a solution x of the relaxed problem described by* (5.2.1), (2.3.2), (2.3.3) *and* (2.3.5) *is infeasible, the cut $(C, \overline{C})$ corresponding to x via* (5.2.3) *has infinite capacity.*

Proof. If x is as in the statement, then it harms at least one temporal constraint, say d_{ij}. As a consequence, a temporal arc corresponding to d_{ij} is in $(C, \overline{C})$. Since this arc has infinite capacity, the cut has infinite capacity. □

Lemma 5.5. *For each finite capacity cut $(C, \overline{C})$ of D with the property that there is exactly one assignment arc per job, there exists a feasible solution x to the relaxed problem described by equations* (5.2.1)*,* (2.3.2)*,* (2.3.3) *and* (2.3.5) *such that $(C, \overline{C})$ corresponds to x under the reversed transformation of* (5.2.3)*.*

Proof. Let $(C, \overline{C})$ be a cut of finite capacity of D. By assumption, this cut contains exactly one assignment arc for each job $j \in \mathcal{J}$. Therefore, the corresponding solution under the transformation of (5.2.3) satisfies constraint (2.3.2). Since $(C, \overline{C})$ has finite capacity, none of the temporal arcs are in the cut, and thus the temporal constraints (2.3.3) are fulfilled. Hence, x is a feasible solution of the relaxed problem described by equations (5.2.1), (2.3.2), (2.3.3) and (2.3.5). □

Now, we are ready to state the main Theorem about why the Minimum Cut Approach works:

Theorem 5.6. *There is a one-to-one correspondence between finite capacity cuts $(C, \overline{C})$ of D with the property that there is exactly one assignment arc per job and feasible solutions x of the relaxed problem described by equations* (5.2.1)*,* (2.3.2)*,* (2.3.3) *and* (2.3.5)*, via the transformation* (5.2.3)*. Moreover, the capacity of $(C, \overline{C})$ equals the value of the objective function $\sum_{j \in \mathcal{J}} \sum_{t=0}^{T} w_{jt} x_{jt}$, under constraints* (2.3.2)*,* (2.3.3) *and* (2.3.5)*.*

Proof. The first statement is covered by lemmas 5.2 and 5.5. The second statement is fulfilled by the construction of D. □

By Theorem 5.6, job j starts at time t if and only if $v_{jt} \in C$, $v_{j,t+1} \in \overline{C}$. As a consequence of Theorem 5.6, the capacity of a minimum cut $(C_{min}, \overline{C}_{min})$ equals the value of the objective function of equation (5.2.1), under the same constraints as above. We can solve the Minimum-Cut problem for the constructed network D, e.g. by performing the Ford-Fulkerson algorithm for maximum flows (cf. [11]) and applying the well-known MaxFlow MinCut theorem, proposed in [22]. This method always finds a cut which satisfies the property that there is exactly one assignment for each job, unless the problem is infeasible. Other methods to find a minimum cut may not have this property, but then Lemma 5.1 can be applied.

Example 5.2. Consider Example 5.1 again. It is easily seen that a minimum cut is prescribed by $C_{min} := \{a, v_{00}, v_{10}, v_{20}, v_{30}, v_{41}, v_{52}, v_{64}\}$, $\overline{C}_{min} = V \backslash C$. The capacity of this cut is 19. This yields the following schedule, in the notation of the second IP-formulation: $S_0 = S_1 = S_2 = S_3 := 0, S_4 := 1, S_5 := 2, C_{max} = S_6 := 4$. This schedule is time-feasible, by construction, but not resource-feasible, since the total resource-usage at time 0 is 6, while there is only 4 available. Notice that none of the schedules corresponding to a minimum cut yield a resource-feasible schedule.

There is a cut which yields a resource-feasible schedule, but it is not a minimum one, since we have to cross the arc (v_{65}, v_{66}) which contributes 5. By modifying λ_t for some $0 \leq t \leq T$, the capacities w_{jt} change. For instance, if $\lambda_0 = \lambda_1 = \lambda_2 := 1$, $\lambda_3 = \lambda_4 = \lambda_5 = \lambda_6 := 0$, then w_{53} is changed to 0. Then, a minimum cut crosses arc (v_{65}, v_{66}). The capacity of this minimum cut is 14.

□

5.3 Subgradient Optimization

Due to the Minimum Cut Approach, a time-feasible schedule with presumably least violation of the resource constraints is obtained, minimizing the makespan. By modification of the Lagrangian multipliers, the capacities w_{jt} can be changed to ensure that some schedules that lead to a large violation of resource constraints, are excluded. A very simple modification is done in Example 5.2. In this paragraph, a *subgradient optimization method* as described in [4], will be used to compute near-optimal values of Lagrange multipliers (cf. [17]). Equation (5.1.4) is used as the objective function for the Lagrangian Subproblem. Since this is a relaxation, w_λ is a lower bound for the minimal makespan of the original problem. The aim is to maximize this lower bound.

Example 5.3. Consider Example 5.2. For all Lagrange multipliers equal to 1, $w_\lambda = 19 - 4 \times 7 = -9$ is computed, which trivially is a lower bound on the makespan. However, since it is negative, it is rather useless. For $\lambda_0 = \lambda_1 = \lambda_2 := 1$, $\lambda_3 = \lambda_4 = \lambda_5 = \lambda_6 := 0$ we derive $w_\lambda = 14 - 4 \times 3 = 2$, which is already much larger than the lower bound found before. However, it is still trivial, since the largest processing time equals 2. But it can be further increased, using subgradient optimization.

□

The problem of maximizing w_λ over $\lambda = (\lambda)_{tk}$, $0 \leq t \leq T$, $k \in \mathcal{R}$ is known as the *Lagrangian Dual*, with value $LD := \max_\lambda w_\lambda$. It is clear that $LS \leq LD \leq C^*_{max}$, where C^*_{max} denotes the optimal (minimal) makespan. As a consequence of the fact that the polytope described by constraints (2.3.2), (2.3.3) and (2.3.5) has integral vertices, the optimal solution of the Linear Programming relaxation of problem (2.3.1) subject to constraints (2.3.2), (2.3.3), (2.3.4) and (2.3.5) equals the optimal solution of the Lagrangian Dual. Although there are efficient methods for solving LP-problems, it is not recommended to do this, since the computation time is very large and probably no integral schedules are obtained. Hence, subgradient optimization is used here.

By an iterative procedure, which is subgradient optimization, the Lagrangian multipliers are updated. As in [17], the following update formula for the matrix λ^i of Lagrangian multipliers in iteration i is used:

$$\lambda^{i+1} := [\lambda^i + \delta^i g^i]^+, \tag{5.3.1}$$

where $g^i = (g^i)_{kt}$, $0 \leq t \leq T$, $k \in \mathcal{R}$ is the *subgradient* at λ^i defined as the the contribution of the Lagrange relaxation of the resource constraints (2.3.4) to the objective function for an optimal solution x^i to the Lagrangian subproblem of equation (5.1.2):

$$g^i_{kt} := \sum_{j \in \mathcal{J}} r_{jk} \Big(\sum_{s=t-p_j+1}^{t} x^i_{js} \Big) - \overline{R_k}. \tag{5.3.2}$$

Moreover, δ^i denotes the step size of the subgradient method, and determines the convergence speed of the subgradient optimization method. If δ^i is large, the difference between the new and old Lagrangian multipliers is large as well, while if δ^i is small, the Lagrangian multipliers are modified slightly. The step size is updated as follows:

$$\delta^i := \frac{\delta\Big(w^* - w_{\lambda^i}(x^i)\Big)}{\|g^i\|_2}. \tag{5.3.3}$$

Here, $\|\cdot\|_2$ is the Euclidean 2-norm and w^* is an upper bound on the optimal value of the Lagrangian Dual: $w^* \geq \max_\lambda w_\lambda$. For instance, $w^* := T$, since T is an upper bound on the project makespan, thus for the lower bound of the project makespan as well, although it is quite rough. One could also take the makespan of a feasible schedule as w^*, but then such a schedule have to be computed first. The parameter δ is a scalar which is updated according to the improvement of the lower bound. For instance, if no improvement of the lower bound has been found within I_1 iterations, $I_1 \in \mathbb{N}_{\geq 1}$, then we reduce δ by a fraction of $0 < q < 1$. Moreover, if no improvement is found within I_2 iterations, $I_2 \in \{n \in \mathbb{N}_{\geq 1} : n > I_1\}$, the algorithm terminates. It turns out that $\delta = 2$ is a good starting point, as in [4].

Summarizing, in order to start the subgradient optimization machinery, the parameters δ, I_1, I_2 and q have to be specified, as indicated above. Moreover, an initial λ^0 has to be chosen. There is no recipe for computing such an initial λ^0, so we will use $\lambda^0_{kt} := 1$, for every $0 \leq t \leq T$ and every $k \in \mathcal{R}$. The algorithm to find a Lagrangian lower bound using the Minimum Cut Approach and subgradient optimization, is listed in Algorithm 5.1.

Algorithm 5.1 Lagrangian lower bound (LLB)

Input: $PS|temp|C_{max}$-instance, $I_1, I_2 \in \mathbb{N}, q \in (0,1), \delta, \lambda^0$
Output: Lagrangian lower boud LLB

1: Compute T by equation (2.1.1)
2: Perform FSP to obtain N^{FS}, thus ES_j, LS_j for $j \in \mathcal{J}$
3: Construct network D as done in paragraph 5.2
4: $terminate := 0$, $i := 0$, $c_1 = c_2 := 0, LLB := 0$

5: **while** $terminate = 0$ **do**
6: $\quad i := i + 1$
7: $\quad$ Compute w_{jt} as in equation (5.1.3)
8: $\quad$ Solve Minimum Cut problem on D with weights w_{jt} to obtain solution x^i
9: $\quad$ Compute $w_{\lambda^i}(x^i)$ by equation (5.1.4)
10: $\quad$ **if** $w_{\lambda^i}(x^i) > LLB$ **then**
11: $\quad\quad c_1 := 0$, $c_2 := 0$
12: $\quad$ **else**
13: $\quad\quad c_1 := c_1 + 1$, $c_2 := c_2 + 1$
14: $\quad$ **end if**
15: $\quad$ **if** $c_2 = I_2$ **then**
16: $\quad\quad terminate := 1$
17: $\quad$ **else**
18: $\quad\quad$ **if** $c_1 = I_1$ **then**
19: $\quad\quad\quad \delta := q \times \delta$
20: $\quad\quad$ **end if**
21: $\quad\quad$ Compute g^i using equation (5.3.2)
22: $\quad\quad$ Compute δ^i by equation (5.3.3)
23: $\quad\quad$ Compute λ^{i+1} using equation (5.3.1)
24: $\quad$ **end if**
25: $\quad LLB := \max_i w_{\lambda^i}(x^i)$
26: **end while**

Remark 5.1. Line 10+11: If an improvement is found, the counters are set equal to 0 again.

Line 15+16: If c_2 reaches its upper bound I_2, then no improvement was found in the last I_2 iterations, so the algorithm terminates.

Line 18+19: If no improvement was found in the last I_1 iteraions, the search space is restricted by reducing δ by a fraction q.

Line 25: LLB takes the value of the largest lower bound ever found. It is non-decreasing by construction.

In addition to the Lagrangian lower bound, the solution x corresponding to this lower bound can be determined as well. This schedule is time-feasible, but it is highely unlikely to be resource-feasible. This solution can be written as schedule S, as in the second IP-formulation,

by setting

$$S_j := \begin{cases} t & \text{if } x_{jt} = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{5.3.4}$$

However, presumably only few resource constraints are violated, so it is almost resource-feasible. We can use this schedule as a *priority-list*, which is the subject of the next chapter.

6 Priority-rule methods

In this chapter we will turn to solution methods to find optimal schedules, by means of the tools developed in the previous chapters. Remember that $PS|temp|C_{max}$ is $\mathcal{NP}$-complete, so we have to use heuristics: to obtain an order to schedule the jobs, we will consider some *priority rules*, which will be the subject of paragraph 6.1. In paragraphs 6.2 and 6.3 the basic versions of the *serial* and *parallel* generation scheme will be considered, which are the basis of priority-rule methods. They will be used in paragraphs 6.6 and 6.7 for the *direct*, *decomposition* and *regret biased sampling* methods discussed there.

6.1 Priority-lists

In priority-rule methods, jobs are scheduled one by one, by fixing their starting times, which are denoted by $S_0, \ldots, S_{n+1}$. We denote the set of scheduled jobs by $\mathcal{C}$, where $\mathcal{J}\backslash\mathcal{C}$ is the set of jobs that still have to be scheduled, as in [20]. Initially, we start with the artificial job 0 so we state $S_0 := 0$ and $\mathcal{C} := \{0\}$. At each moment in time, not each job is allowed to start, due to minimal time lags. Only taking the temporal constraints into account, the set of jobs which are allowed to start at time t is denoted by $\mathcal{E}_t$, the *eligible* set:

$$\mathcal{E}_t := \{j \in \mathcal{J}\backslash\mathcal{C} : \forall i \in \mathcal{C} : t \geq S_i + l^*_{ij}\}. \tag{6.1.1}$$

Notice that the temporal constraints resulting from the Forbidden Set Procedure of paragraph 3.2 are used. Thus, algorithm 3.1 has to be executed first. Moreover, this definition induces that job $j \in \mathcal{E}_t$ only if $t \geq ES_j$, since $ES_j = l^*_{0,j}$ and $S_0 = 0$.

The big question in using priority-rule methods is: at time t, if there is more than one candidate, which job $j \in \mathcal{E}_t$ has to be picked? For this purpose, *priority-lists* $\pi = (\pi(0), \ldots, \pi(n+1))$ are used, where $\pi(j) \in \mathbb{N}$ is the *priority* of job j. At time t, the next job j^* to be scheduled is the activity of $\mathcal{E}_t$ with highest priority. That is,

$$j^* = \{j \in \mathcal{E}_t : \pi(j) = \min_{i \in \mathcal{E}_t} \pi(i)\}. \tag{6.1.2}$$

If there is again a tie, i.e., $\pi(h) = \pi(k)$ for some $h, k \in \mathcal{E}_t$, we random pick a job with highest priority, all with equal probability. Another method is to consider a second priority-list for such cases. When a job $j^* \in \mathcal{E}_t$ is selected, it is scheduled as early as possible, taking the resource constraints into account. Optimality of this procedure will be considered in paragraph 6.9.

Let Π be the set of different priority-lists π, with $\pi(0) \leq \min_{j \in \mathcal{J}} \pi(j)$ and $\pi(n+1) \geq \max_{j \in \mathcal{J}} \pi(j)$. It is known that the cardinality of this set is countable. We call two priority-lists π_1, π_2 equivalent if they yield the same order of jobs. That is: $\pi_1 \sim \pi_2$ if and only if

for all job pairs $i, j \in \mathcal{J}$ with $\pi_1(i) \leq \pi_1(j)$ it holds that $\pi_2(i) \leq \pi_2(j)$. It can easily be checked that this definition really yields an equivalence relation. Thus, $\mathcal{O} := \Pi/\sim$ is the set of different orders on $\mathcal{J}$. It is easily seen that $|\mathcal{O}| = (|\mathcal{J}| - 2)!$, which already for the very small example of 5 jobs leads to 120 different orders. Hence, it is not recommended to check each order whether it yields a schedule in which the makespan is minimized. We can however reduce the cardinality of $\mathcal{O}$, taking the temporal constraints into account. Obviously, if the temporal constraint $S_j \geq S_i + l_{ij}^*$ holds, then it should be ensured that $\pi(i) \leq \pi(j)$ for a representant π of order $o \in \mathcal{O}$. This can be done by the following algorithm:

Algorithm 6.1 Priority-list with respect to Temporal Constraints

Input: Instance of $PS|temp|C_{max}$, initial priority-list π_{init}, distance matrix N^{FS}.
Output: Priority-list π with priorities respecting the Temporal Constraints.

```
1: for all i, j ∈ J do
2:     if N^FS_(i,j) > 0 then
3:         if π_init(i) > π_init(j) then
4:             π(i) := π_init(j)
5:             π(j) := π_init(i)
6:         end if
7:     end if
8: end for
```

A selection of well known priority-rules is *Shortest/Longest Processing Time First*, *Random Priority-lists* and *Earliest/Latest Starting Time First*. More priority rules are considered in [1], [12], [14] and [21]. In addition, also priority-lists arising from solutions of the, possibly relaxed, problem can be considered, such as the schedule corresponding to the Lagrangian relaxation. This is the case in paragraph 6.7 and chapter 5.

It turns out (cf. [20]) that the priority-rule *Latest Starting Time First*, which we will call *LST*, provides 'good' schedules. However, we can distinguish between a *static* version and a *dynamic* version of *LST*. In the static version, the latest starting times arising from the time windows found by FSP, is a *global* priority-list: the latest starting times are not changed during the generation of the schedules. In the *dynamic* version, this is not the case. Suppose there is a (modified) *maximum* time lag $l_{ji}^* < 0$ between job i and j and job i is picked for scheduling, i.e., S_i is fixed. Then

$$LS_j := \min(LS_j, S_i - l_{ji}^*) \tag{6.1.3}$$

possibly changes. Moreover, the earliest starting time may increase due to (modified) *minimum* time lags $l_{ij}^* > 0$ between job i and job j. Then

$$ES_j := \max(ES_j, S_i + l_{ij}^*). \tag{6.1.4}$$

In the dynamic version, which will be denoted by *LSTd*, every time a job is scheduled, the earliest and latest starting times are updated, which is the priority-list for the new iteration.

In the upcoming paragraphs two generation schemes will be considered: the *serial* and *parallel* generation scheme. The main difference between these two is that in the serial generation

scheme an *activity-incrementation*, while in the parallel one a *time-incrementation* is performed, as in [15]. As a consequence, in the serial generation scheme only one job will be scheduled, while in the parallel generation scheme multiple jobs can be scheduled during each iteration.

6.2 Serial Scheduling

Recall that $\mathcal{C}$ is defined to be the set of already scheduled jobs. In addition, we define $S^{\mathcal{C}}$ to be the corresponding *partial* schedule. We want to schedule a chosen job $j^* \in \mathcal{E}_t$ as early as possible. That is, at the earliest time where it is both temporal- and resource-feasible, which we will call t^*. The temporal-feasibility is always ensured due to the definition of $\mathcal{E}_t$. So we need to know the *resource usage* for each resource $k \in \mathcal{R}$ of partial schedule $S^{\mathcal{C}}$ at time t: the amount of resources of type k which is being used by the already processed jobs $\mathcal{C}$ at time t. In order to determine the resource usage, we need to know which jobs of partial schedule $S^{\mathcal{C}}$ are active at time t. This *active set* will be called $\mathcal{A}(S^{\mathcal{C}}, t)$, as in [20]. Then

$$\mathcal{A}(S^{\mathcal{C}}, t) := \{j \in S^{\mathcal{C}} : S_j^{\mathcal{C}} \leq t < S_j^{\mathcal{C}} + p_j\}. \tag{6.2.1}$$

Moreover, we define by $r_k(S^{\mathcal{C}}, t)$ the resource usage of resource $k \in \mathcal{R}$ at time t in partial schedule $S^{\mathcal{C}}$. That is:

$$r_k(S^{\mathcal{C}}, t) := \sum_{j \in \mathcal{A}(S^{\mathcal{C}}, t)} r_{jk} \tag{6.2.2}$$

Such a partial schedule can be represented by $m = |\mathcal{R}|$ Gantt charts in which at a glance the active set and resource usage of resource $k \in \mathcal{R}$ can be determined. As usual, the horizontal axis represents time, while the vertical axis is the resource usage of type k. The jobs are represented by blocks of size $p_j \times r_{jk}$, for $j \in \mathcal{J}$ and $k \in \mathcal{R}$. By 'fitting' a block in the Gantt chart, t^* can easily be determined, as shown in the next example.

Example 6.1. Consider once again the instance of Figure 4.1 and Table 4.1, with $\overline{R_1} = 8$, $\overline{R_2} = 7$, but without the third resource. The optimal schedule is represented in Figure 6.1, one chart for each resource-type. Moreover, partial schedules can be represented like this. The active set and the resource usage is easily seen, e.g., $\mathcal{A}(S^{\mathcal{C}}, 11) = \{1, 2\}$, $r_1(S^{\mathcal{C}}, 11) = 5$ and $r_2(S^{\mathcal{C}}, 11) = 3$. Considering the Gantt charts, t^* can easily be determined. For instance, the block which represents job 4 can not be scheduled simultaneously with job 3, since placing this block at time 2 would result in a crossing of the line of the resource-capacity for both resource 1 and 2. Hence, for scheduling job $j^* = 4$, we see that $t^* = 4$.

Notice that $\overline{R_1} = 6$ would yield the same schedule, since the maximum resource usage of type 1 is 6. Moreover, job 2 can also be scheduled at time 10. Optimality is proven by the fact that $\mathcal{F}' = \{(1, 3), (3, 4), (1, 4), (4, 5)\}$. Hence, $\max(p_1 + p_3 + p_4, p_4 + p_5) = 14$ is a lower bound for this instance. Besides, $DLB = 14$, so the upper bound coincides with the lower bound and the schedule is optimal.

□

Now, the procedure of 'fitting' block $j^* \in \mathcal{E}_t$ into partial schedule $S^{\mathcal{C}}$ and thus determining the earliest starting time t^* of job j^* which is both resource- en temporal-feasible, is done using the following formula:

$$t^* = \min(s \geq t : r_k(S^{\mathcal{C}}, \tau) + r_{j^*k} \leq \overline{R_k} \text{ for } s \leq \tau < s + p_{j^*} \text{ and all } k \in \mathcal{R},\ j^* \in \mathcal{E}_t). \tag{6.2.3}$$

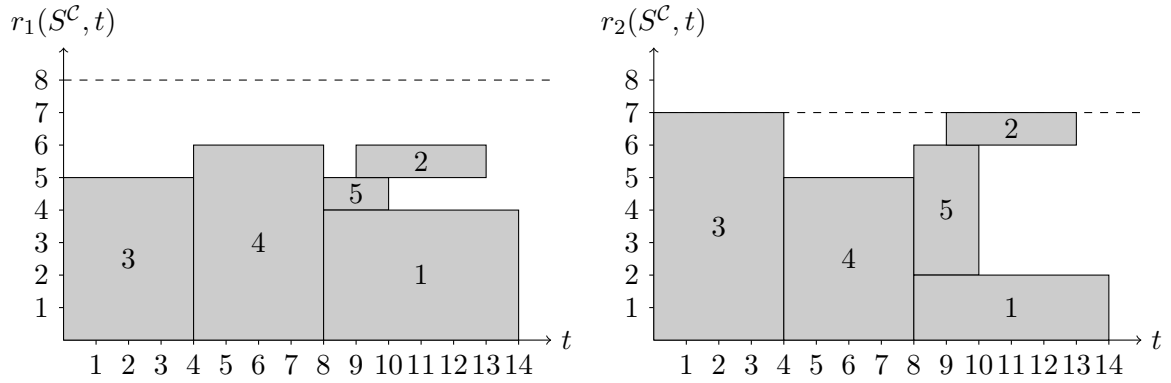

Figure 6.1: Gantt-charts of schedule S for both resource types

That is, during the whole time needed to process j^*, the resource constraints should not be harmed. It is guaranteed for job j^* that $t^* \geq ES_{j^*}$, but in addition it should be ensured that $t^* \leq LS_{j^*}$: job j^* can not start later than its latest starting time, due to maximum time lags. If this is not the case, we perform a so-called *unscheduling step*, which will be discussed in paragraph 6.4.

After fixing the starting time $t^* = S_{j^*}$, the new t has to be computed. This is done by searching through all times on which jobs of which all predecessors are scheduled, and take its maximum:

$$t := \max(S_i + l^*_{ij} : j \in \mathcal{J}\backslash\mathcal{C} \text{ such that } \forall d_{ij} \geq 0 : i \in \mathcal{C}). \tag{6.2.4}$$

Moreover, the earliest and latest starting times of all not yet scheduled jobs have to be determined, since there are possibly time lags between the start of job j^* and jobs $j \in (\mathcal{J}\backslash\mathcal{C}) \cup \{j^*\}$. The *serial generation scheme* can now be stated as follows:

Algorithm 6.2 Serial Generation Scheme

Input: Instance of $PS|temp|C_{max}$, dynamic priority-list π_d, distance matrix N^{FS}.
Output: Temporal- and resource-feasible schedule $S = (S_0, S_1, \ldots, S_{n+1})$.

```
1: C := {0}, S_0 := 0, t := 0
2: while J\C ≠ ∅ do
3:     Determine E_t by equation (6.1.1)
4:     Compute j* by equation (6.1.2)
5:     Compute t* by equation (6.2.3)
6:     if t* > LS_j* then
7:         Perform procedure 'Unscheduling' (discussed in paragraph 6.4)
8:     else (Schedule j* at time t*)
9:         S_j* := t*, C := C ∪ {j*}
10:        Compute t by equation (6.2.4)
11:        for all j ∈ J\C do
12:            Update ES_j by equation (6.1.4)
13:            Update LS_j by equation (6.1.3)
14:        end for
15:    end if
16:    Update π_d
17: end while
```

Remark 6.1. Input: The distance matrix N^{FS} includes the earliest and latest starting times.

Line 2: As long as not all jobs are scheduled.

Line 10: The updating formula for t. It is the latest temporal-feasible starting time of the not yet scheduled jobs of which all predecessors have been scheduled.

Line 16: Instead of dynamic priority-rule π_d, a static rule π_s could be used. Then, the updating is not needed.

Example 6.2. Consider the $PS|temp|C_{max}$-instance in Figure 6.2 and Table 6.1. There is one resource with capacity $\overline{R} = 10$. It is easily computed that our initial upper bound $T = 23$. Moreover, it turns out that $DLB = 10$, computed with algorithm 4.1. The Forbidden Set Procedure of algorithm 3.1 computes earliest and latest starting times as listed in the table, setting $d_{60} = -23$.

Now this preparation has been done, we can start scheduling. We use the static priority-rule *Shortest Processing Time First*: $\pi_{SPT} = \{0, 5, 7, 2, 1, 6, 0\}$. Ties are broken on increasing activity numbers. The results of the execution of the serial generation scheme are listed in Table 6.2 below.

The Gantt chart for the generated schedule is as in Figure 6.3.

The final schedule is $S = (0, 0, 0, 2, 4, 7, 13)$. It is not a priori seen whether this schedule is optimal, since it does not match with the lower bound of 10.

□

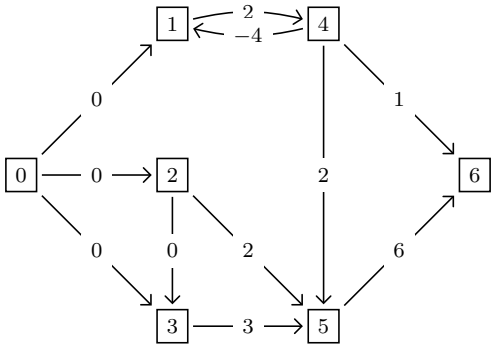

Figure 6.2: Project network N

Job	p_j	r_j	ES_j	LS_j
0	0	0	0	0
1	5	3	0	13
2	7	4	0	14
3	2	6	0	14
4	1	3	2	15
5	6	4	4	17
6	0	0	10	23

Table 6.1

$It.$	$\mathcal{C}$	$\mathcal{E}_t$	j^*	t^*	$Update$
1	$\{0\}$	$\{1,2\}$	2	0	$t := 0$
2	$\{0,2\}$	$\{1,3\}$	3	0	$t := 0$
3	$\{0,2,3\}$	$\{1\}$	1	2	$t := 2,\ ES_4 := 4,\ ES_5 := 6,\ ES_6 := 12,\ LS_4 := 6$
4	$\{0,1,2,3\}$	$\{4\}$	4	4	$t := 6$
5	$\{0,\ldots,4\}$	$\{5\}$	5	7	$t := 13,\ ES_6 := 13$
6	$\{0,\ldots,5\}$	$\{6\}$	6	13	

Table 6.2: Serial generation scheme with SPT-rule

6.3 Parallel Scheduling

Instead of the *activity-incrementation* performed in the serial generation scheme, the parallel generation scheme uses *time-incrementation*. As a consequence, more than one job can be scheduled during each iteration. The same terminology as in the previous paragraph is used. The parallel generation scheme first determines the minimal earliest starting time t^+ of all jobs in $\mathcal{E}_t$, i.e., among the eligible jobs at time t:

$$t^+ := \min_{j \in \mathcal{E}_t}(ES_j). \tag{6.3.1}$$

All jobs with earliest starting time t^+ are put in a new eligible set $\mathcal{E}_{t^+}$:

$$\mathcal{E}_{t^+} := \{j \in \mathcal{E}_t : ES_j = t^+\} \tag{6.3.2}$$

Only jobs belonging to this set can be picked for scheduling. Selection of job j^* and time t^* is done in the same way as in the serial generation scheme, that is, using equations (6.1.2) and (6.2.3). Once a job j^* is picked, it is removed from $\mathcal{E}_{t^+}$. There are three possibilities:

I $t^* > LS_{j^*}$: Then the unscheduling step is performed, which will be the subject of the next paragraph.

II $t^* > t^+$: Job j^* can not be scheduled at time t^+. Hence, its earliest starting time can be increased towards t^*. Moreover, there are possibly not yet scheduled jobs that depend on job j^*, due to minimum time lags. Since ES_{j^*} is increased to t^*, the starting times of these jobs possibly increase as well, with an amount of $l^*_{j^*j}$ time units. So for all $j \in \mathcal{J}\backslash\mathcal{C}$ we do $ES_j := \max(ES_j, t^* + l^*_{j^*j})$. For $j = j^*$ this reads $ES_{j^*} := \max(ES_{j^*}, t^*) = t^*$,

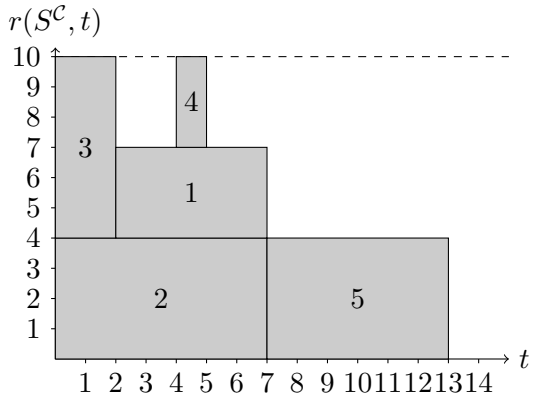

Figure 6.3: Gantt chart for generated schedule

since $t^* > ES_{j^*} = t^+$. Because we have removed j^* from $\mathcal{E}_{t^+}$, it will not be considered anymore during this iteration.

III $t^* = t^+$: Job j^* is scheduled at time t^* and the earliest and latest starting times of the not yet scheduled jobs and t are updated as in the serial generation scheme.

The basic version of the parallel generation scheme is now as follows (cf. [20]).

Algorithm 6.3 Parallel Generation Scheme

Input: Instance of $PS|temp|C_{max}$, dynamic priority-list π_d, distance matrix N^{FS}.
Output: Temporal- and resource-feasible schedule $S = (S_0, S_1, \ldots, S_{n+1})$.

1: $\mathcal{C} := \{0\}$, $S_0 := 0$, $t := 0$
2: **while** $\mathcal{J} \backslash \mathcal{C} \neq \emptyset$ **do**
3: Determine $\mathcal{E}_t$ by equation (6.1.1)
4: Compute t^+ by equation (6.3.1)
5: Determine $\mathcal{E}_{t^+}$ by equation (6.3.2)
6: **while** $\mathcal{E}_{t^+} \neq \emptyset$ **do**
7: Compute j^* by equation (6.1.2)
8: $\mathcal{E}_{t^+} := \mathcal{E}_{t^+} \backslash \{j^*\}$
9: Compute t^* by equation (6.2.3)
10: **if** $t^* > LS_{j^*}$ **then**
11: Perform procedure 'Unscheduling' (discussed in paragraph 6.4)
12: **else**
13: **if** $t^* > t^+$ **then**
14: **for** all $j \in \mathcal{J} \backslash \mathcal{C}$ **do**
15: $ES_j := \max(ES_j, t^* + l^*_{j^* j})$
16: **end for**
17: **else**
18: $S_{j^*} := t^*$, $\mathcal{C} := \mathcal{C} \cup \{j^*\}$
19: $t := \max(S_i + d_{ij} : j \in \mathcal{J} \backslash \mathcal{C}$ such that $\forall l^*_{ij} \geq 0 : i \in \mathcal{C})$
20: **for** all $j \in \mathcal{J} \backslash \mathcal{C}$ **do**
21: Update ES_j by equation (6.1.4)
22: Update LS_j by equation (6.1.3)
23: **end for**
24: **end if**
25: **end if**
26: **end while**
27: Update π_d
28: **end while**

Remark 6.2. Input: The distance matrix N^{FS} includes the earliest and latest starting times.

Line 2: As long as not all jobs are scheduled.

Line 10: Case I

Line 13: Case II

Line 17: Case III

Line 19: The updating formula for t. It is the latest temporal-feasible starting time of the not yet scheduled jobs of which all predecessors have been scheduled.

Line 24: Instead of dynamic priority-rule π_d, a static rule π_s could be used. Then, the updating is not needed.

Example 6.3. Consider once again the instance of Figure 6.2 and Table 6.1 in Example 6.2, with $T = 23$ and $DLB = 10$. The results of the execution of the parallel generation scheme, using the dynamic priority-rule LSTd, starting with priority-list $\pi_{LSTd} =$

$\{13, 13, 14, 14, 15, 17, 23\}$ and ties broken on increasing activity numbers, are listed in Table 6.3.

It.	$\mathcal{C}$	$\mathcal{E}_t$	t^+	$\mathcal{E}_{t^+}$	j^*	t^*	*Update*	*Case*
$1a$	$\{0\}$	$\{1,2\}$	0	$\{1,2\}$	1	0	$t := 2, LS_4 := 4$	*III*
$1b$	$\{0,1\}$	$\{2\}$	0	$\{2\}$	2	0	$t := 2, \pi_4 := 4$	*III*
$2a$	$\{0,1,2\}$	$\{3,4\}$	0	$\{3\}$	3	5	$ES_3 := 5, ES_5 := 8$	*II*
$2b$	$\{0,1,2\}$	$\{3,4\}$	2	$\{4\}$	4	2	$t := 3, ES_6 := 14$	*III*
$2c$	$\{0,1,2,4\}$	$\{3\}$	5	$\{3\}$	3	5	$t := 8$	*III*
3	$\{0,\ldots,4\}$	$\{5\}$	8	$\{5\}$	5	8	$t := 14$	*III*
4	$\{0,\ldots,5\}$	$\{6\}$	14	$\{6\}$	6	14		*III*

Table 6.3: Parallel generation scheme with LSTd-rule

Thus, schedule $S = (0, 0, 0, 5, 2, 8, 14)$ is obtained, resulting in the Gantt chart in Figure 6.4.

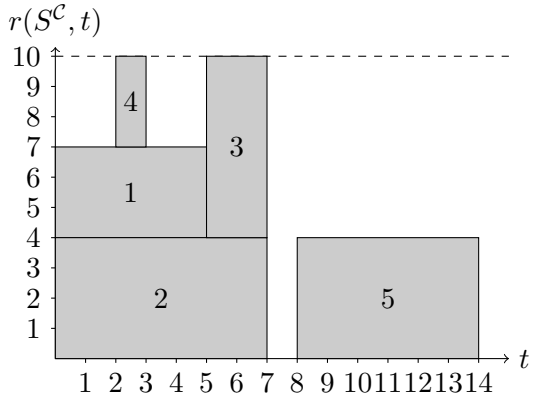

Figure 6.4: Gantt chart for generated schedule

Since in Example 6.2 a lower makespan was found, it is clear that for this instance the serial generation scheme in combination with SPT gives a better schedule than the parallel one with LSTd.

□

6.4 The Unscheduling Step

Maximum time lags, which include the prescribed project deadline $\overline{d}$ or upper bound T, give rise to *latest* starting times LS_j for jobs $j \in \mathcal{J}$. However, once in the serial or parallel generation scheme it occurs that the earliest both resource and temporal feasible starting time t^* of selected job j^* exceeds the latest starting time LS_{j^*}, at least one maximum time lag is harmed. Then, to obtain a feasible schedule, j^* has to be scheduled at an earlier moment. Hence, some jobs $j \in \mathcal{C}$ have to be *unscheduled* and scheduled again at an earlier or later moment, which is done by changing the time windows.

A first step in the Unscheduling Procedure is to determine the set $\mathcal{U}$ of jobs $j \in \mathcal{C}$ which determine the latest starting time of job j^*. That is, job $i \in \mathcal{U}$ if there is a *maximal* (modified)

time lag $l^*_{j^*i} < 0$ such that $LS_{j^*} = S_i - l^*_{j^*i}$. Remember that $l^*_{j^*i} < 0$ indicates that job j^* has to start *at most* $-l^*_{j^*i}$ after the start of job i:

$$\mathcal{U} := \{i \in \mathcal{C} : LS_{j^*} = S_i - l^*_{j^*i}\}. \tag{6.4.1}$$

Since LS_{j^*} should increase, the set $\mathcal{U}$ consist of jobs which have to be scheduled *later* to satisfy the *maximum* time lags between the start of job $i \in \mathcal{U}$ and job j^*. However, if it occurs that the artificial start-job 0 is in this set, no feasible schedule can be found, since the start-job is forced to start at time 0. In that case a different priority-rule and/or generation scheme has to be used.

It suffices forcing jobs $i \in \mathcal{U}$ to start *at least* $t^* - LS_{j^*}$ time units later, which can be done by unscheduling all jobs $i \in \mathcal{U}$ and increasing the *earliest* starting times ES_i by $t^* - LS_{j^*}$ time units:

$$ES_i := S_i + t^* - LS_{j^*}. \tag{6.4.2}$$

As a consequence ('new' corresponds to values used in the next iteration),

$$S_i^{new} \geq ES_i = S_i + t^* - LS_{j^*} = S_i + t^* - (S_i - l^*_{j^*i}) = t^* + l^*_{j^*i}$$

and

$$LS_{j^*}^{new} := S_i^{new} - l^*_{j^*i} \geq t^* + l^*_{j^*i} - l^*_{j^*i} = t^*.$$

Now, $LS_{j^*}^{new} \geq t^*$, so in the next iteration of the serial or parallel generation scheme this case can be fixed, unless $t^*_{new} \geq LS_{j^*}^{new}$. But if that is the case the Unscheduling Procedure can be applied again. There is still one drawback, namely the case that the changed earliest starting time, due to the Unscheduling Procedure, of a job $i \in \mathcal{U}$ exceeds its latest starting time:

$$ES_i > -l^*_{i0} =: LS_i. \tag{6.4.3}$$

Then, no feasible schedule can be found and a different priority-rule and/or generation scheme has to be used.

Since all jobs $i \in \mathcal{U}$ are forced to start later than they did before, resources become available at moments they were scheduled. That is, the integral interval determined by the minimal starting time and maximal finish time of jobs $i \in \mathcal{U}$: $\{\min_{i\in\mathcal{U}} S_i, \ldots, \max_{i\in\mathcal{U}}(S_i + p_i)\}$. Other already scheduled jobs can possibly start in this interval to obtain a better schedule. We define

$$\mathcal{H} := \{i \in \mathcal{C} : S_h > \min_{i\in\mathcal{U}} S_i\}. \tag{6.4.4}$$

Thus, all jobs $h \in \mathcal{H}$ are unscheduled as well.

For the not yet scheduled jobs $j \in \mathcal{J}\backslash\mathcal{C}$ the earliest and latest starting times ES_j and LS_j have to be updated, or computed again if a job was unscheduled. For computing ES_j for $j \in \mathcal{J}\backslash\mathcal{C}$ there are three possible earliest starting times, of which the maximum has to be taken:

I The earliest starting time of job j equals the value that it had in the initial time window, i.e. $ES_j = l^*_{0j}$.

II For a job $i \in \mathcal{U}$ there is a minimal time lag l^*_{ij} between the start of job i and j. Since the earliest starting times of all jobs $i \in \mathcal{U}$ are increased, this have to happen to the ES_j as well: $ES_j = \max_{i\in\mathcal{U}}(ES_i + l^*_{ij})$.

III For $i \in \mathcal{C}$ there is a minimal time lag l^*_{ij} between the start of job i and j. Then ES_j needs to be updated according to equation (6.1.4), in which the maximum value of for all jobs $i \in \mathcal{C}$ have to be taken.

Hence,

$$ES_j := \max(l^*_{0j}, \max_{i\in\mathcal{U}}(ES_i + l^*_{ij}), \max_{i\in\mathcal{C}}(S_i + l^*_{ij})). \tag{6.4.5}$$

A similar analysis holds for the latest starting times. However, the increase of earliest starting times of jobs $i \in \mathcal{U}$ in case II are of no influence for the latest starting times, so only cases similar to I and III make sense. The updating formula for LS_j, $j \in \mathcal{J}\backslash\mathcal{C}$ is thus as follows (cf. equation (6.1.3)):

$$LS_j := \min(-l^*_{j0}, \min_{i\in\mathcal{C}}(S_i - l^*_{ji})) \tag{6.4.6}$$

In theory, the serial or parallel generation scheme can run infinitely long, as unscheduling can cycle among two or more jobs which lead to the Unscheduling Procedure. To fix that, the number u of unscheduling steps is counted. Moreover, a maximum number $\overline{u}$ of unscheduling steps is prescribed, e.g. $\overline{u} = |\mathcal{J}|$ as done in [20]. If $u > \overline{u}$, the algorithm is terminated and no feasible schedule is found. A different priority-rule and/or generation scheme has to be used. Summarizing, the Unscheduling Procedure is described in the following algorithm.

Algorithm 6.4 Unscheduling Procedure

```
u := u + 1
Compute U as in equation (6.4.1)
if 0 ∈ U or u > ū then
    Terminate: no feasible schedule can be found
else
    for i ∈ U do
        Compute ES_i by equation (6.4.2)
        C := C\{i}
        if ES_i > −l*_i0 (equation (6.4.3) holds) then
            Terminate: no feasible schedule can be found
        end if
    end for
end if
Compute H by equation (6.4.4)
C := C\H
for all j ∈ J\C do
    Compute ES_j and LS_j by (6.4.5) and (6.4.6)
end for
```

Example 6.4. In addition to the $PS|temp|C_{max}$-instance of Examples 6.2 and 6.3, an additional maximum time lag $d_{52} = -3$ is introduced, as drawn in Figure 6.5. Again, $\overline{R} = 10$ and the initial upper bound T is set to 23. The Forbidden Set Approach computes earliest and latest starting times as listed in Table 6.4. Moreover, $DLB = 10$.

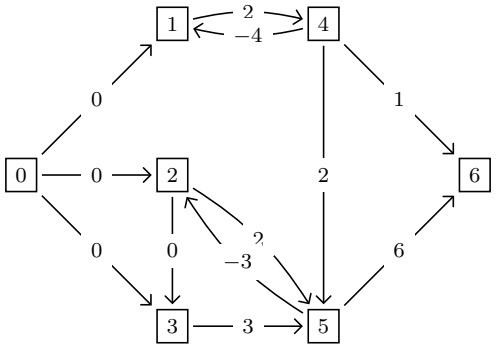

Figure 6.5: Project network N

Job	p_j	r_j	ES_j	LS_j
0	0	0	0	0
1	5	3	0	13
2	7	4	1	14
3	2	6	1	14
4	1	3	2	15
5	6	4	4	17
6	0	0	10	23

Table 6.4

For generating a schedule, we use the priority-rule LSTd, together with the serial generation scheme of algorithm 6.2. Results are listed in Tables 6.5, 6.6, 6.7.

$It.$	$\mathcal{C}$	$\mathcal{E}_t$	j^*	t^*	$Update$
1	$\{0\}$	$\{1\}$	1	0	$t := 2, \pi(4) = LS_4 := 4$
2	$\{0,1\}$	$\{2,4\}$	4	2	$t := 4$
3	$\{0,1,4\}$	$\{2\}$	2	1	$\pi(3) = LS_3 := 1, \pi(5) = LS_5 := 4$

Table 6.5: First three iterations

Now, the Gantt chart for the partial schedule $S^{\{0,1,2,4\}}$ is as in Figure 6.6. Naturally, the eligible set $\mathcal{E}_4 := \{3\}$, so job $j^* := 3$ will be picked for scheduling. However, $t^* = 5$, as the Gantt-chart makes clear, and $LS_3 = 1$. The unscheduling step thus needs to be done. It is easily computed that $\mathcal{U} = \{2\}$. So ES_2 has to be increased by $t^* - LS_3 = 5 - 1 = 4$. Hence, $ES_2 := 1 + 4 = 5$, and $\mathcal{C} := \{0, 1, 4\}$.

Notice that $4 \in \mathcal{H}$, since $S_4 = 2 > S_2 = 1$. Thus, job 4 is unscheduled since it possibly can start earlier than time 2. (In this example, that is not the case, since $ES_4 = 2$ already, but the algorithm do not notice that and in the updating of the earliest starting times, it will remain 2). Hence, $\mathcal{C} = \{0, 1\}$. The earliest and latest starting times of jobs $j \in \mathcal{J}\backslash\mathcal{C}$ are updated as listed in Table 6.6:

j	2	3	4	5	6
ES_j	5	5	2	8	14
LS_j	14	14	4	17	23

Table 6.6: Modified earliest and latest starting times

Moreover, the priority-list π is updated, according to the latest starting times. Now iteration 4 (which was the unscheduling iteration) is done and we can resume scheduling.

The obtained schedule is $S = (0, 0, 5, 5, 2, 8, 14)$, as in Figure 6.7.
The parallel generation scheme with LSTd produces the same outcome.

□

It.	$\mathcal{C}$	$\mathcal{E}_t$	j^*	t^*	*Update*
5	$\{0,1\}$	$\{2,4\}$	4	2	$t := 4$
6	$\{0,1,4\}$	$\{2\}$	2	5	$t := 7, \pi_3 = LS_3 := 5, \pi_5 = LS_5 := 8$
7	$\{0,1,2,4\}$	$\{3\}$	3	5	$t := 8$
8	$\{0,\ldots,4\}$	$\{5\}$	5	8	$t := 14$
9	$\{0,\ldots,5\}$	$\{6\}$	6	14	

Table 6.7: Unscheduling in serial generation scheme with $LSTd$

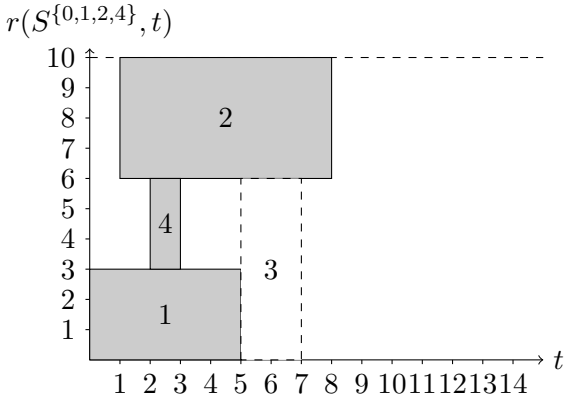

Figure 6.6: Partial schedule $S^{\{0,1,2,4\}}$

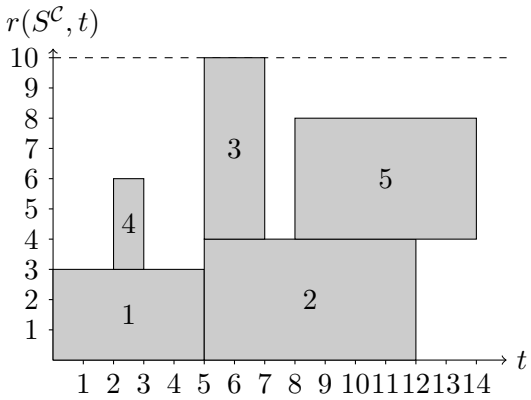

Figure 6.7: Final schedule S

6.5 Direct Method

In this paragraph the *direct method* is discussed. This method is discussed in more detail in [13]. The main idea in this method is to partition the set of jobs in smaller parts. Jobs within such a part strongly depend on each other, as a consequence of minimum and maximum time-lags. Intuitively, these jobs have to be scheduled a short amount of time after each other. The direct method is a modification of the approach of paragraphs 6.2 and 6.3 and it also makes use of the serial or parallel generation scheme. In the next paragraph, modifications of the direct method will be discussed, which are the *decomposition methods*.

A first step is the partitioning of $\mathcal{J}$ into smaller sets, in which the jobs strongly depend on each other. If there is a minimum and maximum time lag between two jobs $i, j \in \mathcal{J}$ in the project network N, say $d_{ij} \geq 0$ and $-\infty < d_{ji} < 0$, then S_i influences ES_j and LS_j very much. If the cycle length is even zero, that is, $d_{ij} = -d_{ji}$, then S_i completely determines $ES_j = LS_j = S_j$. This can be generalized to more than two jobs, by determing the *strongly connected components* of N.

Definition 6.1. A *Strongly Connected Component* (SCC) is a subset $\Sigma = \{i_1, \ldots, i_k\}$ of $\mathcal{J}$ such that there is a path in N from i_h to i_l for each $h, l \in \{1, \ldots, k\}$. Alternatively: $l_{i_h i_l} < \infty$ for each $h, l \in \{1, \ldots, k\}$. The *Set of Strongly Connected Components* $\mathcal{SSCC}$ is a partition of $\mathcal{J}$ such that each $\Sigma \in \mathcal{SSCC}$ is an SCC.

In addition, the $K := |\mathcal{SSCC}|$ SCC's Σ are numbered $\Sigma_1, \ldots, \Sigma_K$. For Σ_κ with $|\Sigma_\kappa| > 1$, $\kappa \in \{1, \ldots, K\}$ scheduling of some job $i \in \Sigma_\kappa$ determines ES_j and LS_j for all $j \in \Sigma_\kappa,\ j \neq i$. Since these possible starting times become restricted, it is intuitively reasonable to schedule

these jobs as soon as possible. Thus, scheduling all $j \in \Sigma_\kappa$ should be done first before selecting activities of different SCC's. This is done in the *direct method*.

At first, $\mathcal{SSCC}$ needs to be computed. This can be done by several different methods, e.g. by Tarjan's algorithm described in [25]. Then, we can use (2.1.1) to obtain an initial upper bound T and apply the Forbidden Set Procedure to determine N^{FS}, thus l_{ij}^* for $i, j \in \mathcal{J}$. For scheduling the activities, the direct method uses a slight modification of the serial or parallel generation scheme: Since all $j \in \Sigma_\kappa$ have to be scheduled directly after each other, we need to know whether all jobs of the current Σ_κ are scheduled, i.e., $\Sigma_\kappa \subseteq \mathcal{C}$. Therefore, a boolean variable q is introduced such that $q = 1$ if for all $j \in \Sigma_\kappa$ it holds that $j \in \mathcal{C}$; $q = 0$ holds if *at least* one but not all $j \in \Sigma_\kappa$ are scheduled. In the latter case, we are not yet finished scheduling the current SCC Σ_κ. Then the eligible set $\mathcal{E}_t$ is updated as follows:

$$\mathcal{E}_t := \{j \in \Sigma_\kappa : \forall i \in \mathcal{J} : t \geq S_i + l_{ij}^*, j \in \mathcal{J} \backslash \mathcal{C}\}. \tag{6.5.1}$$

If $q = 1$, we need to choose a new SCC Σ_ι to schedule next. As a requirement, all predecessors of all jobs $j \in \Sigma_\iota$ have to be scheduled. Remember that t denotes the earliest time on which all eligible jobs can start, since their predecessors have been scheduled. This is covered by the condition $t \geq \max\limits_{i \in \mathcal{C}, j \in \Sigma_\iota} (S_i + d_{ij})$. Thus, in the case $q = 1$,

$$\mathcal{E}_t := \bigcup_{1 \leq \iota \leq K} \{\Sigma_\iota \subseteq \mathcal{J} \backslash \mathcal{C} : t \geq \max_{i \in \mathcal{C}, j \in \Sigma_\iota} (S_i + d_{ij})\} \tag{6.5.2}$$

is used for computing the eligible set. The direct method can now be summarized as follows:

Algorithm 6.5 Direct Method

Input: Instance of $PS|temp|C_{max}$, dynamic priority-list π_d, distance matrix N^{FS}.
Output: Schedule S.

1: Determine $\mathcal{SSCC}$ of N
2: Compute T using (2.1.1) (or use prescribed deadline $\overline{d}$)
3: Perform the Forbidden Set Procedure of Algorithm 3.1
4: Choose a proper priority-rule
5: Generate schedule S by using the serial/parallel generation scheme with q, equations (6.5.1) and (6.5.2) and the chosen priority-rule

---

Example 6.5. Consider Example 6.4. Notice that $\mathcal{SSCC} = \{\{0\}, \{1, 4\}, \{2, 3, 5\}, \{6\}\}$. If the direct method in combination with the serial generation scheme and any respecting temporal constraints priority-rule is applied, exactly the same results are obtained, since the order of scheduling the SCC's is: $\{0\}$, $\{1, 4\}$, $\{2, 3, 5\}$, $\{6\}$, as in Example 6.4. Because job 4 is a predecessor of job 5, this is the only possible order of scheduling the SCC's.

□

6.6 Decomposition Methods

As in the direct method, the *decomposition methods*, which we will discuss, use the partition $\mathcal{SSCC}$ proposed in the previous paragraph. In the direct method, we scheduled an SCC within

the main generation scheme. In decomposition methods, this will be not the case: each SCC $\Sigma_\kappa, 1 \leq \kappa \leq K$ is viewed as a separate $PS|temp|C_{max}$-instance. The main idea is to obtain feasible schedules for these sub-instances first, and then 'fitting' them together in a clever way (cf. [13], [20]).

Consider an SCC $\Sigma_\kappa = \{\kappa_1, \ldots, \kappa_\sigma\}$ with $1 < \sigma < n+1$. To formulate this SCC as a separate $PS|temp|C_{max}$-instance, two artificial jobs α and ω need to be introduced, as was done in chapter 2. These are the start- and finish-job respectively. Besides, we set $p_\alpha = p_\omega := 0$ and $r_{\alpha k} = r_{\omega k} := 0$ for each $k \in \mathcal{R}$. Furthermore, time lags (arcs) have to be introduced for these artificial jobs. A time lag $d_{\alpha\kappa_i} := 0$ is introduced if job κ_i has no predecessors in Σ_κ, i.e., there is no minimal time lag $l_{\kappa_j\kappa_i} \geq 0$ for any job $\kappa_j \in \Sigma_\kappa$, $\kappa_j \neq \kappa_i$. That is, if the condition:

$$\max_{\kappa_j \in \Sigma_\kappa \setminus \{\kappa_i\}} l_{\kappa_j\kappa_i} < 0 \tag{6.6.1}$$

is satisfied. For the artificial finish job ω, a time lag $d_{\kappa_i\omega} := p_{\kappa_i}$ is introduced if the termination of activity κ_i possibly can determine the completion time of the subproject. That is, there is no job $\kappa_j \in \Sigma_\kappa$, $\kappa_j \neq \kappa_i$ for which there is a minimal time lag $0 \leq l_{\kappa_i\kappa_j} < p_i$, thus if

$$\max_{\kappa_j \in \Sigma_\kappa \setminus \{\kappa_i\}} l_{\kappa_i\kappa_j} < p_{\kappa_i} \tag{6.6.2}$$

is met. We can construct the subproject network now. The node set is $V_\kappa := \Sigma_k \cup \{\alpha, \omega\}$, with $\sigma + 2$ nodes. The induced arcs from N carry over to this subproject network and arcs from and towards the artificial jobs are introduced as above. What is still left is an upper bound on the duration, that is, an arc (ω, α) with weight $d_{\omega,\alpha} := -\overline{d}^{V_\kappa}$.

For each pair $\kappa_i, \kappa_j \in V_\kappa$ with $1 \leq i, j \leq \sigma$ it is known that $|l_{\kappa_i\kappa_j}| < \infty$, since both are nodes in an SCC and thus reachable by each other in N. If $l_{\kappa_j\kappa_i} < 0$, as in Remark 2.1(1), $-l_{\kappa_j\kappa_i} + p_{\kappa_j}$ is the maximum time lag between the start of job κ_i and the completion of job κ_j. We set:

$$\overline{d}^{V_\kappa} := \max_{\kappa_i \in V_\kappa} \max_{\kappa_j \in V_\kappa} (-l_{ji} + p_j), \tag{6.6.3}$$

which represents the *maximum duration* of the subproject corresponding to V_κ. This duration cannot be exceeded without violation of at least one maximum time lag in V_κ: the maximum time lag between the pair of jobs for which this maximum was attained, and possibly other maximum time lags within V_κ. We add arc (ω, α) with weight $d_{\omega,\alpha} := -\overline{d}^{V_\kappa}$ to the subproject network. The obtained subproject network is denoted by N^{V_κ}, and, with a slight abuse of notation, its adjacency matrix is denoted by N^{V_κ} as well.

In network N^{V_κ}, we can do anything we did before on network N: longest path lengths l can be computed by the Longest Path Approach. Even the Forbidden Set Approach can be applied to obtain stricter time lags l^*. Hence, earliest and latest starting times can be computed, as well as lower bounds. Moreover, feasible subschedules S^{V_κ} can be computed using the serial/parallel generation scheme and a proper priority-rule π^{V_κ}, as illustrated in Example 6.6. If for some SCC no feasible subschedule can be computed, the method terminates.

Once for each Σ_κ, $1 \leq \kappa \leq K$, a feasible subschedule S^{V_κ} has been computed, these have to be 'plugged in' in the final schedule S. At first, the jobs of Σ_κ are ordered according to non-decreasing starting times. That is, $0 = S_{\kappa_{l_1}} \leq S_{\kappa_{l_2}} \leq \ldots \leq S_{\kappa_{l_\sigma}}$. For every two successive

jobs κ_{l_i} and $\kappa_{l_{i+1}}$ a time lag $d^{new}_{\kappa_{l_i}\kappa_{l_{i+1}}} := S_{\kappa_{l_{i+1}}} - S_{\kappa_{l_i}}$ is introduced. By the non-decreasing order, $d^{new}_{\kappa_{l_i}\kappa_{l_{i+1}}} \geq 0$. As a consequence, the minimal amount of time between the start of job κ_{l_i} and $\kappa_{l_{i+1}}$ carries over from the subschedule S^{V_κ}. By adding the arcs according to the introduced minimal time lags and deleting the 'old' arcs between jobs $\kappa_i, \kappa_j \in \Sigma_\kappa$ with weight $d_{\kappa_i,\kappa_j} \geq 0$, a new project network N_1^{new} is constructed. On this network, FSP can be applied to obtain time-windows, as well as a Serial or Parallel Generation Scheme.

This procedure is the first decomposition method. The second one is a slight extension to this method. Not only minimal time lags as above are introduced, the following maximum time lag is added as well: $d^{new}_{\kappa_{l_\sigma}\kappa_{l_1}} := -S_{\kappa_{l_\sigma}}$, while the 'old' arcs within Σ_κ corresponding to maximum time lags are removed. Consequently, a cycle of length 0 arises in N_2^{new}. Hence, Σ_κ will precisely be scheduled in the final schedule S as it was in the subschedule S^{V_κ}, up to translation.

Doing as above for all SCC's, a modified project network N^{new} is obtained. It is immediately clear that $d^{new}_{ij} \geq d_{ij}$ for all jobs $i, j \in \mathcal{J}$, since these new time lags arise from schedules that respect the old time lags. Hence, succesfully applying the Forbidden Set Approach and the serial/parallel generation scheme to N^{new} yields a feasible schedule, as illustrated in Example 6.6. The decomposition methods can be summarized as in Algorithm 6.6.

Algorithm 6.6 Decomposition method 1 and 2

Input: Instance of $PS|temp|C_{max}$.
Output: Schedule S.

1: Determine $\mathcal{SSCC}$ of N
2: **for** each $\Sigma_\kappa \in \mathcal{SSCC}$ **do**
3: Construct subnetwork N^{V_κ} as described
4: Determine the maximal duration $\overline{d}^{V_\kappa}$
5: Add arc (ω, α) with $d_{\omega\alpha} := -\overline{d}^{V_\kappa}$ to N^{V_κ}
6: Perform Forbidden Set Approach on N^{V_κ}
7: Choose proper priority-rule
8: Generate feasible subschedule S^{V_κ} by applying the serial/parallel generation scheme
9: Sort S^{V_κ} according to non-decreasing starting times
10: Add and delete arcs as above to the new project network N^{new}
11: **end for**
12: Compute T using (2.1.1) (or use prescribed deadline $\overline{d}$)
13: Perform the Forbidden Set Procedure on N^{new}
14: Choose a proper priority-rule
15: Generate schedule S by applying the serial/parallel generation scheme

Example 6.6. Consider the project in Figure 6.8. The processing times, resource requirements and earliest/latest starting times are listed in Table 6.8. Furthermore, $\overline{R} = 10$ and $T = 22$. It is easily seen that $\mathcal{SSCC} = \{\{0\}, \{1,4\}, \{2,3,5\}, \{6\}\}$. It is of no use to construct the subproject networks for $\Sigma_1 := \{0\}$ and $\Sigma_4 := \{6\}$, since both only contain one job. Moreover, for $\Sigma_2 := \{1,4\}$ it can be done, but it is immediately seen that the subschedule $S_1^{V_2} = 0, S_4^{V_2} = 2$ is always obtained, since for $j^* = 4$, $t^* = 2$ is feasible in the serial/parallel

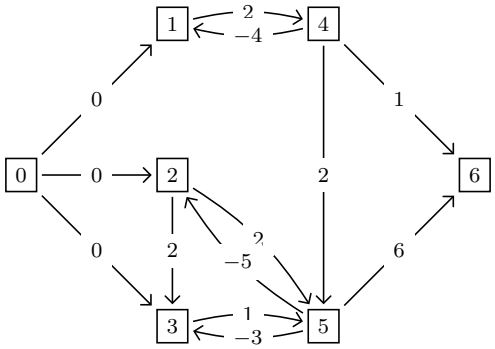

Figure 6.8: Project network N

Job	p_j	r_j	ES_j	LS_j
0	0	0	0	0
1	5	3	0	12
2	7	4	0	13
3	2	6	2	15
4	1	3	2	14
5	6	4	4	16
6	0	0	10	22

Table 6.8

generation scheme. Thus the construction of N^{V_2} is omitted here. Consider $\Sigma_3 := \{2, 3, 5\}$. We can expand this SCC to the subproject network N^{V_3} by the procedure above. That is, we introduce nodes α and ω and arcs with weights $d_{\alpha 2} := 0$, $d_{2\omega} := 7$, $d_{3\omega} := 2$, $d_{5\omega} := 6$. Furthermore, it is computed by equation (6.6.3) that $\overline{d}^{V_3} := 11$. Hence, an arc (ω, α) with $d_{\omega\alpha} = -11$ is introduced. The obtained subnetwork N^{V_3} is drawn in Figure 6.9. The added arcs are dashed. The computed earliest and latest starting times, done by FSP, are listed in Table 6.9.

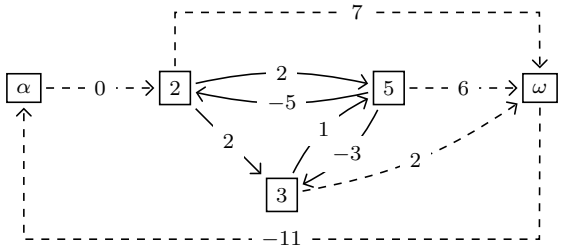

Figure 6.9: Subproject network N^{V_3}

Job	ES^{V_3}	LS^{V_3}
α	0	2
2	0	2
3	2	4
5	3	5
ω	9	11

Table 6.9

A feasible subschedule can be computed by the serial/parallel generation scheme. Notice that each priority-list produces the same subschedule here, as there is no choice in the order of scheduling the jobs, which is always $\alpha, 2, 3, 5, \omega$. Thus, the computed subschedule, which is $S^{V_3} = (0, 0, 2, 4, 10)$, is optimal. However, this is not the case in general.

A new project network N^{new} is constructed as above by adding arcs $d_{23}^{new} := 2$, $d_{35}^{new} := 2$ and $d_{53}^{new} := -4$ only in the second decomposition method. Moreover, $d_{14}^{new} := 2$ and $d_{41}^{new} := -2$, the latter only in the second decomposition method. The resulting networks for decomposition method 1 and 2 are drawn in Figure 6.10 and Figure 6.11 respectively.

If N_1^{new} is used as input for the basic toolkit of generating schedules, using any priority-rule, the schedule $S_1 := (0, 0, 2, 5, 2, 7, 13)$ is obtained. If this is done with N_2^{new}, this results in $S_2 := (0, 0, 3, 5, 2, 7, 13)$. It can easily be verified that both schedules S_i, $i = 1, 2$, are feasible with respect to N_i^{new} and the original project network N. Notice that S_1 is *not* feasible with respect to N_2^{new}. The subschedule S^{V_3} has suffered a translation of $+3$ time units in S_2,

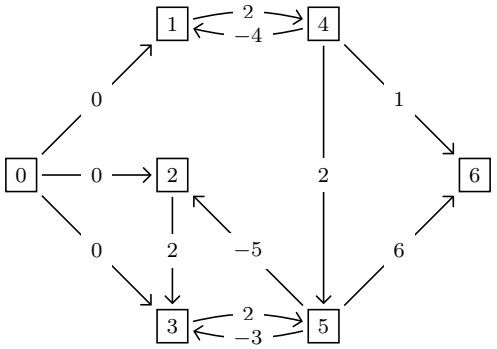

Figure 6.10: N_1^{new} for method 1

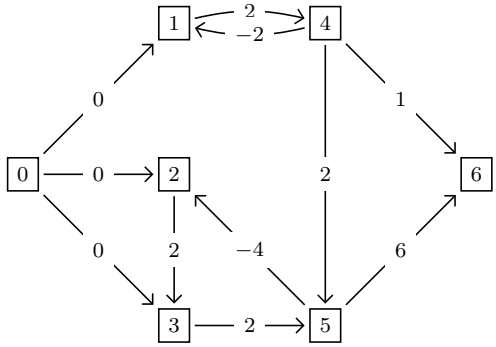

Figure 6.11: N_2^{new} for method 2

while in S_1 only the subschedule is slightly modified. For both modified network, $DLB = 10$, which is not equal to the found upper bound, so nothing can be said about optimality of these schedules.

□

6.7 Regret Biased Random Sampling (RBRS)

Until now, we only considered *deterministic* priority-rules in which we selected the job j^* to be scheduled next deterministically. If there was a tie, we picked a job according to increasing activity number, or via a second priority-rule. Since it is highely unlikely that a chosen priority-rule is the optimal one, it can be useful to search through the 'neighborhood' of a priority-rule, with the purpose of leaving a local optimum. Suppose that in an iteration of the serial generation scheme of paragraph 6.2, the eligible set $\mathcal{E}_t$ defined in equation (6.1.1), consists of more than one candidate. Equation (6.1.2) can be applied to deterministically select the job j^* to schedule next. For the parallel generation scheme, the same holds for $\mathcal{E}_{t^+}$. To avoide making this remark a couple of times, only the serial generation scheme will be considered. One could also choose j^* at random. In the latter, no priority-rule is needed. In this paragraph, these two selection procedures will be mixed, using *Regret Biased Random Sampling*, as introduced in [23]. This method will be expanded to the so-called *Iterative Scheduling Method* proposed in paragraph 6.8.

Let $\mu_j := \mathbb{P}\{j^* = j\}$. That is, the probability of job $j \in \mathcal{E}_t$ to be selected for scheduling next. For each pair of eligible jobs (i, j), $i, j \in \mathcal{E}_t$, if $\pi_i < \pi_j$, then it should hold that $\mu_i > \mu_j$. In the concept of regrets, the regret of a job $j \in \mathcal{J}$, which will be denoted by q_j, is a measure for the consequence of not selecting the job with highest priority.

$$W := \max_{j \in \mathcal{E}_t} \pi_j. \tag{6.7.1}$$

In other words, W is the value of the eligible job with lowest priority: the worst choice the algorithm can make by randomly selecting a job. Remember that we assumed that the job j with the lowest π_j has the highest priority, which may seem quite counter-intuitive. Now the regret q_j of job $j \in \mathcal{E}_t$ can be defined:

$$q_j := W - \pi_j. \tag{6.7.2}$$

As a consequence, $q_j = 0$ for the job with lowest and $q_j = \max_{i \in \mathcal{E}_t} q_i$ for the job with highest priority. Hence for any pair (i, j), $i, j \in \mathcal{E}_t$, it should be ensured that $q_i < q_j \Rightarrow \mu_i < \mu_j$. This is satisfied by the following definition of μ:

$$\mu_j := \frac{(1+q_j)^\beta}{\sum_{i \in \mathcal{E}_t} (1+q_i)^\beta}, \tag{6.7.3}$$

where $\beta \in \mathbb{R}_{\geq 0}$. Notice that the probabilities μ_j are well-defined, since $\mu_j \geq 0$ for each $j \in \mathcal{E}_t$ and $\sum_{j \in \mathcal{E}_t} \mu_j = 1$. It is necessary to add the 1+ term as in equation (6.7.3), because if all jobs would have the same priority, the denominator would become 0. For $\beta := 0$, this method behaves like pure random sampling, as all jobs have the same probability. Moreover, for $\beta \to \infty$, this methods acts as a deterministic priority-rule method. Hence, β is a measure for the variability of Regret Biased Random Sampling.

Once the probabilities μ_j are computed, one can construct real intervals $[a_j, b_j) \in [0, 1]$ for job j, with $b_j - a_j = \mu_j$. That is, if $\mathcal{E}_t := \{j_1, \ldots, j_{E_t}\}$, where $E_t := |\mathcal{E}_t|$, then define

$$\begin{aligned} &[a_{j_1}, b_{j_1}) = [0, \mu_{j_1}) \\ &[a_{j_i}, b_{j_i}) = [b_{j_i - 1}, \mu_{j_i}) && i \neq 1,\ i \neq E_t \\ &[a_{j_{E_t}}, b_{j_{E_t}}] = [b_{j_{E_t - 1}}, 1] \end{aligned}$$

This yields a partition of the interval $[0, 1]$. A random number $c \in [0, 1]$ can be generated, using a pseudo-random number generator. It is checked in which sub-interval this c is. If $c \in [a_{j_i}, b_{j_i})$, then $j^* := j_i$.

Example 6.7. Assume that in a certain stage of the serial generation scheme, possibly within the direct or a decomposition method, $\mathcal{E}_t := \{3, 5, 8, 14, 21\}$ at a certain time $0 \leq t \leq T$. Besides, $\pi := (\pi_3, \pi_5, \pi_8, \pi_{14}, \pi_{21}) = (2, 0, 0, 3, 1)$. It is easily seen that $W := 3$. The computed probabilities μ are as in Table 6.10, for several β-values.

			$\beta = 0.1$	1	2	5	10	50
Job	π_j	q_j	μ_j					
3	2	1	0.1954	0.1429	0.0870	0.0138	0.0005	0
5	0	3	0.2094	0.2857	0.3478	0.4406	0.4861	0.5
8	0	3	0.2094	0.2857	0.3478	0.4406	0.4861	0.5
14	3	0	0.1823	0.0714	0.0217	0.0004	0	0
21	1	2	0.2035	0.2143	0.1957	0.1046	0	0

Table 6.10: Probabilities for several β-values

Consider the case $\beta = 1$. The constructed partition of $[0, 1]$ is:

$$\begin{aligned} &[a_3, b_3) = [0, 0.1429) \\ &[a_5, b_5) = [0.1429, 0.4286) \\ &[a_8, b_8) = [0.4286, 0.7143) \\ &[a_{14}, b_{14}) = [0.7143, 0.7857) \\ &[a_{21}, b_{21}) = [0.7857, 1] \end{aligned}$$

If for example random number $c := 0.7592$ is generated, then $j^* := 14$, although it has lowest priority.

□

6.8 Iterative Scheduling (IS)

In this paragraph, we propose an alternative method that we have called *Iterative Scheduling (IS)*. Until now only methods that generate just one schedule S were considered, such as the serial or parallel generation scheme, possibly within the direct or a decomposition method, or in combination with Regret Biased Random Sampling. In the remainder, this methods will be called *simple generation methods*. Because of the randomness of Regret Biased Random Sampling, it is recommended to perform a simple generation method a number of times and select the best schedule. However, if always the same priority-rule and the same β are used, the set of possible outcomes is rather restricted, especially for large β-values. To overcome this problem, we can choose a different priority-rule and/or β and apply a simple generation method again. This is the main idea of Iterative Scheduling, discussed in this paragraph. An iteration of Iterative Scheduling is the execution of a simple generation scheme once, resulting in a feasible or infeasible schedule.

Notice that a generated feasible schedule can serve as a priority-list. The priority-rule belonging to this priority-list will be called *Earliest Scheduled Starting Time first* or *ESST*, not to be confused with *EST*: earliest starting time first. The priority-rule *ESST* is not meant to be used in simple generation methods, only in Iterative Scheduling. To distinguish between priority-lists in simple generation schemes and in Iterative Scheduling, π is used for the first, and ν for the latter:

$$\nu_{ESST} := S^{last}, \tag{6.8.1}$$

where the superscript *last* denotes the last obtained feasible schedule. Other priority-rules that arise from actual schedules are e.g. *Earliest Scheduled Completion Time first* or *Earliest α-Completion Time first* (cf. [17]). All these rules are static within the iteration of Iterative Scheduling, but dynamic between iterations. It is clear that ν_{ESST} respects the temporal constraints, as it actually represents a both time- and resource-feasible schedule. Moreover, deterministically applying the serial/parallel generation scheme to priority-list S yields the same schedule.

Little changes to this priority-list can be achieved by applying Regret Biased Random Sampling, which possibly result in minor changes in a schedule. We do this in a simulated annealing like fashion: At first, we begin with small β, such that the set of possible outcomes is very large. As the algorithm evolves in time, β is increased to decrease variability. In contrast to simulated annealing, we do not move to actual solutions (that is, feasible schemes), but to priority-lists, i.e., we do not use $\nu_{ESST} := S^{last}$, but only if a schedule is found that equals or has value less than the best makespan found up to now, this is adapted as priority-list:

$$\nu_{ESST} := S^{best}, \tag{6.8.2}$$

where the superscript *best* denotes the last schedule with minimal makespan found until now. If a priority-list yields a worse schedule, then we 'stay', in terms of simulated annealing. That is, we keep using this priority-list until we found an equally good or better schedule.

To start up the above procedure, an initial priority-list is needed. For this purpose, we can deterministically generate a schedule by a simple generation method and a proper priority-rule, such as *LSTd*, which is known to provide 'good' schedules. This deterministic schedule can be taken as initial priority-list for Iterative Scheduling. Summarizing, the Iterative Scheduling procedure for priority-rule *ESST* is as in Algorithm 6.7. Iterative Scheduling with other priority-rules is similar.

Algorithm 6.7 Iterative Scheduling (IS) for priority-rule *ESST*

Input: $PS|temp|C_{max}$-instance, π, β^{init}, K distinct simple generation schemes, numbered $1, \ldots, K$.
Output: Feasible schedule S

1: Compute initial upper bound T, or use prescribed deadline $\overline{d}$.
2: Perform Forbidden Set Approach to check temporal feasibility and to obtain N^{FS}
3: **for** $k = 1 : K$ **do**
4: S^k is output of deterministic simple generation scheme k with π
5: **if** S^k is infeasible **then**
6: $S^k_{n+1} := \infty$
7: **end if**
8: **end for**
9: $\mathcal{S}^{init} := \{S^k : S^k_{n+1} \leq S^l_{n+1}, 1 \leq k, l \leq K\}$
10: $\nu^{init}_{ESST} := S^{init} \in \mathcal{S}^{init}$

11: $\beta := \beta^{init}$, $S^{best} := S^{init}$, $\nu_{ESST} := \nu^{init}_{ESST}$
12: **repeat**
13: **for** $k = 1 : K$ **do**
14: S^k is output of RBRS(β) simple generation scheme k with ν_{ESST}
15: **if** S^k is infeasible **then**
16: $S^k_{n+1} := \infty$
17: **else**
18: **if** $S^k_{n+1} \leq S^{best}_{n+1}$ **then**
19: $S^{best} := S^k$
20: **end if**
21: **end if**
22: **end for**
23: $\nu_{ESST} := S^{best}$
24: Update β
25: **until** stop-criterion
26: $S := S^{best}$

Remark 6.3. Input: Notice that 8 simple generation methods are considered, so $K \leq 8$. One can choose between serial or parallel generation scheme within direct method or decomposition method 1 or 2, or without a covering method. This yields $2 \times 4 = 8$ different simple generation schemes.

Lines 1-10: An initial schedule and priority-list is computed. The set $\mathcal{S}^{init}$ represents the set of obtained schedules with minimal makespan. An initial priority-list ν is chosen

according to a schedule $S^{init} \in \mathcal{S}^{init}$.

Lines 13-23: The selected simple generation schemes are used, in combination with Regret Biased Random Sampling, for the generation of new schedules. When an equally good or better schedule is found, that is, a schedule with the same or lower makespan than found before, this schedule is stored as S^{best}. This S^{best} is adapted as priority-list ν for the next iteration.

Line 24: The update of β can happen in different ways. For instance, one can increase β by a certain constant in each iteration, or increase β only when an improvement is found. Moreover, the number by which β is increased, can depend on the measure of improvement as well. One could also prescribe a list of β-values and for each β-value a number of iterations that have to be performed, using that β.

Line 25: Several stop-criterions can be used. For instance, one could stop if there was no improvement during the x last iterations, or when a certain β-value or prescribed number of iterations is reached.

6.9 Asymptotic Optimality of Iterative Scheduling

It is well known that in Simulated Annealing a worse solution can be accepted that moves away from a local optimum. Moreover, if SA runs an infinite amount of time, the global optimum will be found. The question arises whether this also is the case for the Iterative Scheduling algorithm of the last paragraph. This is indeed the case, as argued in [18]. In this article, a formal definition of different classes of schedules and types of shifts is given. Here, only the classes and shifts that are important are explained:

Definition 6.2. A *left-shift* is a transformation of a schedule $S := (S_0, \ldots, S_{n+1})$ into a schedule $S' := (S'_0, \ldots, S'_{n+1})$ such that $S'_i \leq S_i$ for all $0 \leq i \leq n+1$ and $S'_i < S_i$ for at least one i.

Definition 6.3. A *global left-shift* is a left-shift that transforms a *feasible* schedule S to a *feasible* schedule S'. A feasible schedule S is called *active* if there is *no* global left-shift from S. The set of all active schedules is denoted by $\mathcal{AS}$.

Definition 6.4. An *order-preserving left-shift* is a global left-shift with the requirement that if $S_i < S_j$, then it should also be ensured that $S'_i < S'_j$, for $0 \leq i, j \leq n+1$. A feasible schedule S is called *quasiactive* if there is *no* order-preserving left-shift from S. The set of all quasiactive schedules is denoted by $\mathcal{QAS}$.

Example 6.8. Consider the project in Figure 6.12 and Table 6.11. The initial deadline $\overline{d} := 13$, the resource availability $\overline{R} := 2$ and the earliest and latest starting times are listed in Table 6.11 as well.

One should notice that the starting times of jobs 2, 3 and 5 are already fixed, since there is a cycle of length 0. In Figure 6.13 a quasiactive schedule S^1 is drawn. This schedule can not be left-shifted into a schedule S' with the same order of activities, since the starting times of jobs 2, 3 and 5 are tied, and both $S_2 < S_1$ and $S_2 < S_4$ hold. So this should be respected in S' as well. Hence, neither job 1 nor job 4 can be left-shifted, and thus $S^1 \in \mathcal{QAS}$.
Schedule S^1 is transformed in schedule S^2 in Figure 6.14 by a global left-shift. It is clear that for S^2 there is no global left-shift possible, and thus neither an order-preserving one. Hence, both $S^2 \in \mathcal{AS}$ and $S^2 \in \mathcal{QAS}$ hold. Moreover, it is clear that this schedule is optimal, since

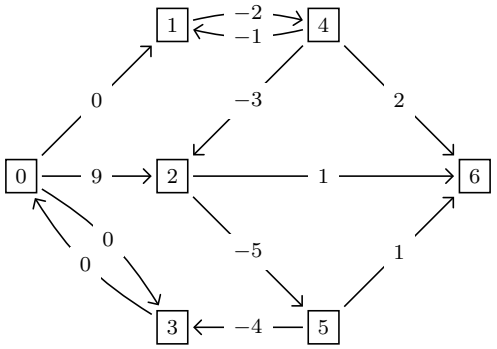

Figure 6.12: Project Network N

Job	p_j	r_j	ES_j	LS_j
0	0	0	0	0
1	2	1	0	11
2	1	1	9	9
3	1	1	0	0
4	2	1	0	11
5	1	1	4	4
6	0	0	10	13

Table 6.11

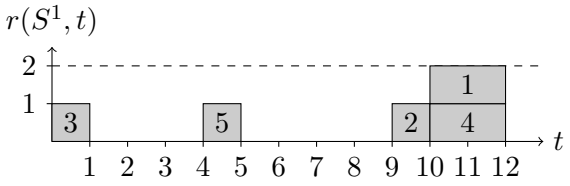

Figure 6.13: Quasiactive schedule

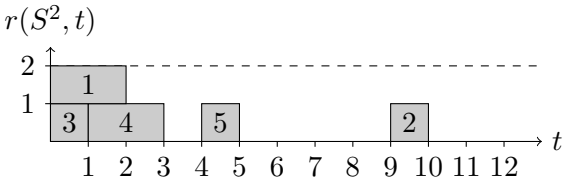

Figure 6.14: Active schedule

its makespan equals the lower bound obtained by the Longest Path approach.

□

Since an order-preserving left-shift is a global left-shift, it is immediately clear that $S \in \mathcal{AS} \Rightarrow S \in \mathcal{QAS}$ and thus $\mathcal{AS} \subseteq \mathcal{QAS}$. In [18] it is argued that constructive schemes provide quasiactive schedules for regular objective functions, of which minimizing the makespan is a special case. All methods considered in this chapter are constructive methods. Moreover, in [18] it is proven that regular objective functions are minimized by active schedules. If Iterative Scheduling runs an infinite amount of time, each quasiactive schedule will be found at least once with probability 1. Since $\mathcal{AS} \subseteq \mathcal{QAS}$, this holds for all active schedules as well. The optimal schedule is in $\mathcal{AS}$, so with probability 1 the optimal schedule will be obtained if Iterative Scheduling runs infinitely long.

7 Computational Study

The heuristics developed in the previous chapters were tested on a large amount of benchmark instances, all from [24] and [26]. For these instances, the destructive, workload based and Lagrangian lower bounds were computed. The best lower bound LB is then determined by $LB := \max(DLB, WLB, LLB)$. The performance of these algorithms is determined by three criteria, as in [20]. These criteria are:

- The *accuracy* of the method. This is measured by the average relative difference with the best found lower bound. Let $\mathcal{I}$ be the test set, and $\mathcal{I}_H \subseteq \mathcal{I}$ the set of instances for which heuristic H found a feasible schedule. If $I \in \mathcal{I}_H$, then the makespan computed by heuristic H is denoted by $C^H_{max}(I)$. Moreover, let $LB(I)$ the best found lower bound for instance I. Then, the average relative error, that is the average relative deviation

of the makespan computed by heuristic H from the best found lower bound is

$$\Delta_H := \frac{1}{|\mathcal{I}_H|} \sum_{I \in \mathcal{I}_H} \frac{C^H_{max}(I) - LB(I)}{LB(I)} \tag{7.0.1}$$

If for two heuristics H_1, H_2 it holds that $\Delta_{H_1} < \Delta_{H_2}$, then H_1 provides better schedules on average than H_2. The accuracy will be measured in percentages, which are denoted by dev_{LB}.

- Let $\mathcal{I}_{LB}$ the set of instances for which a lower bound could be computed. That is, instances that have a time-feasible solution, have a resource-feasible solution as well, and thus a feasible schedule. This is because of the Destructive Lower bound of Algorithm 4.1 verified that there was no cycle of positive length. The question is whether a feasible solution could be computed by heuristic H. This percentage will be denoted by p_{feas}. It is clear that the higher this percentage, the 'better' the method.
- Moreover, the time needed to perform the execution of heuristic H is important as well. Instances were tested on a 1.9GHz AMD Athlon, using an implementation of the heuristics in MATLAB2010a. The average computation time will be denoted by t_{cpu}, and measured in seconds.

7.1 Instances with 10 jobs

The performance of the methods are listed in the following tables, for different sizes of instances. First, five LSTd-methods are tested on the 270 instances with 10 jobs of [26]. These methods are: the Serial and Parallel Generation scheme, Direct Method, and the two Decomposition Methods. The Direct Method and Decomposition Methods are all in combination with the Serial Generation scheme. Results are listed in Table 7.1.

	SerLSTd	ParLSTd	DirectLSTd	Decomp1LSTd	Decomp2LSTd
dev_{LB}	26.62%	25.41%	27.64%	28.10%	27.07%
p_{feas}	93.19%	79.06%	93.72%	93.19%	93.19%
t_{cpu}	0.0460	0.0296	0.0623	0.0723	0.0697

Table 7.1: Comparison of LSTd-methods for 10 jobs

It can be seen that the Parallel Generation scheme provides the best schedules in the fastest way, but it is less likely that this method finds a feasible schedule. Moreover, for this set of very small instances, it can be seen that the Direct Method and Decomposition Methods are outperformed by the Serial Generation scheme, especially in computation time.

In addition, the priority-list arising from the Lagrangian relaxation of chapter 5 can be used in combination with the above methods. It was computationally hard to compute the Lagrangian lower bound by subgradient optimization. For the 270 instances with 10 jobs of [26], it took 8.3883 seconds on average, where $I_1 = 3$ and $I_2 = 10$ where chosen in Algorithm 5.1. However, this could be sped up by chosing lower I_1 and I_2. This priority-list was used in combination with the Serial and Parallel Generation scheme, and the Direct Method. The Decomposition Methods did not give any improvement in comparison with these three, and are therefore not listed in table 7.2. The computation time in this table includes the 8.3883 seconds.

	SerLLB	ParLLB	DirectLLB
dev_{LB}	1.65%	0.72%	1.98%
p_{feas}	73.33%	58.89%	66.67%
t_{cpu}	8.4421	8.4107	8.4467

Table 7.2: Comparison of generation schemes with the Lagrangian lower bound for 10 jobs

The same trend can be seen as in the methods with LSTd: the Parallel generation scheme provides the best schedules and is faster compared to the others, but the feasibility is a drawback. More interesting is the fact that the priority-list arising from the Lagrange relaxation indeed provides far better schedules than LSTd. However, with this priority-list it is less likely to find a feasible schedule. But when a feasible schedule can be found, its makespan almost every time equals the lower bound.

Our Iterative Scheduling method of Algorithm 6.7 has been tested as well. We choose $K := 1$, and the only chosen simply generation scheme is the Serial one. Moreover, as π the priority-list arising from LSTd is selected, and β^{init} is set equal to 1. The update of β is done as follows: if for a particular value of β the algorithm finds a better or equally good schedule, than β is increased by a prescribed $\Delta\beta$: $\beta := \beta + \Delta\beta$. Besides, if in 10 executions of the Serial Generation scheme with the same β no improvement was found, β is also increased by $\Delta\beta$. As stop criterion, the algorithm terminates if in the last five iterations (that is, in the last five values for β) no improvement or equally good schedules were found. In addition, if $\beta = 120$, that is, the method is very deterministic, the algorithm terminates. This method is tested for different values of $\Delta\beta$. Results for the 270 instances with 10 jobs of [26] are listed in Table 7.3.

	$\Delta\beta = 0.01$	$\Delta\beta = 0.1$	$\Delta\beta = 0.5$	$\Delta\beta = 1$	$\Delta\beta = 2$
dev_{LB}	4.88%	5.26%	6.18%	6.47%	6.95%
p_{feas}	95.29%	95.29%	95.29%	95.29%	95.29%
t_{cpu}	12.1455	3.1684	1.3586	1.3527	0.5833

Table 7.3: Comparison of different $\Delta\beta$-values for Iterative Scheduling with 10 jobs

As expected, $\Delta\beta = 0.01$ provides the best schedules, but has also the largest computation time. For every value of $\Delta\beta$, this method finds the same p_{feas}. One would expect that this should be 100%, since that many schedules are generated, such that at least one feasible schedule should have been found. However, it is not trivial whether a resource-feasible schedule exists if a time-feasible exists or not. Hence, for the few instances for which no feasible schedule could be found in the $\frac{1190}{\Delta\beta} + 1$ tries, it is believed that no feasible schedule exists.

7.2 Instances with 20 jobs

The methods above are tested in exactly the same way on the 270 instances of [26] with 20 jobs. The results are listed in Tables 7.4, 7.5 and 7.6.
Here the same trend as for the instances with 10 jobs. In addition, the first decomposition method performs worse with respect to all three criteria in comparison with the second, which performs slightly worse than the Direct Method. Moreover, for the dev_{LB}-criterion, the Serial

	SerLSTd	ParLSTd	DirectLSTd	Decomp1LSTd	Decomp2LSTd
dev_{LB}	23.02%	23.37%	25.17%	27.57%	25.82%
p_{feas}	90.27%	68.65%	92.97%	88.65%	92.43%
t_{cpu}	0.3770	0.2378	0.4314	0.5026	0.4469

Table 7.4: Comparison of LSTd-methods for 20 jobs

Generation scheme is now slightly better than the Parallel one, which still has a bad feasibility percentage. Overall, compared with 10 jobs, less instances could be solved to feasibility, but the difference with the lower bound is smaller.

	SerLLB	ParLLB	DirectLLB
dev_{LB}	1.74%	1.09%	2.90%
p_{feas}	44.07%	28.25%	43.50%
t_{cpu}	95.7715	95.3195	95.5884

Table 7.5: Comparison of generation schemes with the Lagrangian lower bound for 20 jobs

Again, the deviation from the lower bound is very small, although larger than for 10 jobs. But the feasibility is even worse compared to the instances with 10 jobs. Moreover, it took 95.2039 seconds to compute the Lagrangian lower bound, but this could be quickened by modifying I_1 and I_2.

	$\Delta\beta = 0.01$	$\Delta\beta = 0.1$	$\Delta\beta = 0.5$	$\Delta\beta = 1$	$\Delta\beta = 2$
dev_{LB}	7.96%	8.03%	9.03%	9.90%	10.25%
p_{feas}	97.85%	97.85%	97.85%	97.85%	97.85%
t_{cpu}	646.0719	147.5234	49.0817	25.7933	20.5853

Table 7.6: Comparison of different $\Delta\beta$-values for Iterative Scheduling with 20 jobs

Results are as expected: for $\Delta\beta = 0.01$, the computation time is long, while the devation from the lower bound is small compared to the others. However, the difference in dev_{LB} between $\Delta\beta = 0.01$ and $\Delta\beta = 0.1$ is only 0.07%, while the computation time is much longer for $\Delta\beta = 0.01$. In all cases, the algorithm found a feasible solutions for 97.85% of all time-feasible instances. As in the case of 10 jobs, it is believed that no feasible schedule exists for the remaining 2.15% of all time-feasible instances.

7.3 More than 20 jobs

The non-iterative scheduling methods are tested on the 90 instances with 50 and 100 jobs of [24] as well. The Direct Method and Decomposition Methods are still in combination with the Serial Generation scheme only, due to the small feasibilty percentage of the Parallel Generation scheme. Results are listed in tables 7.7 and 7.8 respectively.
The Parallel Generation scheme for the computation time criterion is very beneficial for larger instances. However, its feasibilty percentage is worse compared to the other methods, as in the case with 10 or 20 jobs. One should notice that the more jobs, the smaller the feasibilty percentage for all methods. For one instance with 50 jobs of [24] (instance 4) an improvement

	SerLSTd	ParLSTd	DirectLSTd	Decomp1LSTd	Decomp2LSTd
dev_{LB}	17.76%	24.89%	20.67%	24.10%	21.60%
p_{feas}	79.45%	58.90%	80.82%	73.97%	83.56%
t_{cpu}	13.2950	4.7400	14.8832	19.8389	13.0551

Table 7.7: Comparison of LSTd-methods for 50 jobs

	SerLSTd	ParLSTd	DirectLSTd	Decomp1LSTd	Decomp2LSTd
dev_{LB}	15.45%	14.46%	15.88%	21.50%	20.48%
p_{feas}	61.54%	41.03%	61.54%	53.85%	62.82%
t_{cpu}	293.0817	56.6238	312.7150	325.5442	282.3569

Table 7.8: Comparison of LSTd-methods for 100 jobs

was found using the Parallel Generation scheme in combination with LSTd. The best known schedule up to now had a makespan of 253, while we found one with makespan 228.

It took too much computation time and memory to compute the Lagrangian Lower Bound for instances with 50 jobs, let alone for instances with 100 jobs. Thus unfortunately, there are no results available. Moreover, this holds for the Iterative Scheduling method in combination with instances with 100 jobs. For instances with 50 jobs, the Iterative Scheduling method could be tested, but with different parameter-values compared to instances with 10 or 20 jobs to speed up convergence. However, due to the randomization, this gave very much variation in different runs. Results are therefore not listed.

7.4 Concluding Remarks

The Iterative Scheduling method proposed in this thesis works well for instances with a small number of jobs. However, for large instances, it takes very long to compute a schedule that is near optimal, although optimality is ensured if this method runs infinitely long with suitable parameter values. Another conclusion is that the Parallel generation scheme is the quickest one for instances of different sizes. However, its percentage of feasibility is a drawback. Schedules obtained from the Lagrangian priority-list have the least deviation from their lower bounds, but it takes long, even for small instances, to compute these Lagrangian Lower Bounds. Finally, Decomposition Method 2 outperforms Decomposition Method 1 in all three criteria for different sizes of instances.

Appendices

A Abbreviations

Algorithms

FSP:	Forbidden Set Procedure (paragraph 3.1)
IS:	Iterative Scheduling (paragraph 6.8)
RBRS:	Regret Biased Random Sampling (paragraph 6.7)

Bounds

DLB:	Destructive Lower Bound (paragraph 4.1)
LLB:	Lagrangian Lower Bound (chapter 5)
RLB:	Resource Lower Bound (chapter 4)
TLB:	Temporal Lower Bound (chapter 4)
WLB:	Workload based Lower Bound (paragraph 4.2)

Miscellaneous

LD:	Lagrangian Dual (paragraph 5.3)
LP:	Longest Path (paragraph 3.1)
LS:	Lagrangian Subproblem (paragraph 5.1)
SCC:	Strongly Connected Component (paragraph 6.5)

Priority-rules

ESST:	Earliest Scheduled Starting Time first (paragraph 6.8)
EST:	Earliest Starting Time first (paragraph 6.1)
LST:	Latest Starting Time first (static; paragraph 6.1)
LSTd:	Latest Starting Time first (dynamic; paragraph 6.1)

B List of Algorithms

List of Algorithms

C References

[1] R. Alvarez-Valdés and J. Tamarit. Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis. *Advances in Project Scheduling*, pages 113–134, 1998.

[2] M. Bartusch, R. Möhring, and F. Rademacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16:201–240, 1988.

[3] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[4] D. Bertsekas. *Nonlinear Programming*, chapter 6.3: Nondifferentiable Optimization Methods. Athena Scientific, 2 edition, 1999.

[5] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112:3–41, 1999.

[6] P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107:272–288, 1998.

[7] N. Christofides, R. Alvarez-Valdés, and J. Tamarit. Project scheduling with resource constraints: a branch and bound approach. *European Journal of Operations Research*, 29:262–273, 1987.

[8] B. De Reyck and W. Herroelen. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 111:152–174, 1998.

[9] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[10] R. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):245, 1962.

[11] L. Ford and D. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1954.

[12] B. Franck and K. Neumann. Resource-constrained project scheduling with time windows: Structural questions and priority-rule methods. *Report WIOR*, 492, 1998.

[13] B. Franck, K. Neumann, and C. Schwindt. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR Spectrum*, 23:297–324, 2001.

[14] R. Kolisch. Project scheduling under resource constraints. *Physica*, 1995.

[15] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. In *Project Scheduling: Recent models, algorithms and applications*, pages 147–178. Kluwer Boston, 1999.

[16] A. Land and A. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[17] R. Möhring, A. S. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 3:330–350, 2003.

[18] K. Neumann, H. Nübel, and C. Schwindt. Active and stable project scheduling. *Mathematical Methods of Operations Research*, 52:441–465, 2000.

[19] K. Neumann and C. Schwindt. Project with minimal and maximal time lags: Construction of activity-on-node networks and applications. *Report WIOR*, 447, 1995.

[20] K. Neumann, C. Schwindt, and J. Zimmerman. *Project Scheduling with Time Windows and Scarce Resources*. Springer-Verlag, Berlin, second edition, 2003.

[21] K. Neumann and J. Zhan. Heuristics for the minimum project-duration problem with minimal and maximal time-lags under fixed resource constraints. *Journal of Intelligent Manufacturing*, 6:145–154, 1995.

[22] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, chapter 6.1: The Max-Flow, Min-Cut Theorem, pages 120–128. Dover, 1998.

[23] A. Schirmer. Advanced biased random sampling in serial and parallel scheduling. 1997.

[24] C. Schwindt. Project duration problem rcpsp/max. `http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/rcpspmax.html`.

[25] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[26] J. Weglarz, editor. *Project Scheduling - Recent Models, Algorithms and Applications*, pages 197–212. Kluwer, Boston, 1999.