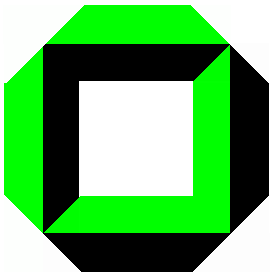
INSTITUT FÜR WIRTSCHAFTSTHEORIE
UND OPERATIONS RESEARCH
UNIVERSITÄT KARLSRUHE

Lower Bounds for RCPSP/max

Roland Heilmann & Christoph Schwindt

Report WIOR–511

November 1997



TECHNICAL REPORT

Universität Karlsruhe

Kaiserstraße 12, D–76128 Karlsruhe

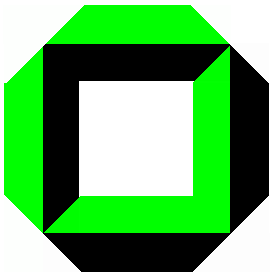
INSTITUT FÜR WIRTSCHAFTSTHEORIE
UND OPERATIONS RESEARCH
UNIVERSITÄT KARLSRUHE

Lower Bounds for RCPSP/max

Roland Heilmann & Christoph Schwindt

Report WIOR–511

November 1997



This research was supported by the Deutsche Forschungsgemeinschaft (Grant Ne 137/4).

Abstract. Lower bounds on the optimal objective function value of minimization problems play a crucial role for the efficiency of enumeration algorithms. Moreover, tight lower bounds allow for an appropriate assessment of heuristic procedures.

Klein & Scholl (1997) distinguish between two strategies for the computation of lower bounds for minimization problems: *Constructive* methods directly calculate a lower bound value by solving a relaxation of a given problem instance. *Destructive improvement* techniques restrict the solution space of a problem instance by setting a maximum objective function value and try to contradict the solvability of this instance by showing that the restricted solution space is empty. These two strategies have been applied to the resource–constrained project duration problem RCPSP by Klein & Scholl (1997).

In this paper, we devise new preprocessing algorithms and new constructive lower bounds for RCPSP/max where in addition to RCPSP arbitrary time lags between the starts of activities may be given. Preprocessing and lower bounds are used for polynomial destructive improvement algorithms. Preliminary computational results show that the relative deviation of lower bounds from the corresponding minimal objective function values can be considerably reduced compared to lower bounds proposed in literature.

Keywords: Lower bound, destructive improvement, resource–constrained project scheduling, maximum time lags

Contents

List of Symbols

0	Fictitious source of project network $\vec{N}$
$\boldsymbol{b}$	Arc weights of project network $\vec{N}$
b_{ij}	Weight of arc $\langle i,j \rangle$ in project network $\vec{N}$
$C^*_{\max}(P^{ijk})$	Optimal objective function value of (P^{ijk})
$\boldsymbol{D}$	Distance matrix
$\overline{d}$	Hypothetical upper bound on minimal S_{n+1}
d_{ij}	Length of longest path from node i to node j in network $\vec{N}$
d_{ij}^{min}	Minimum time lag between activity i and activity j
d_{ij}^{max}	Maximum time lag between activity i and activity j
E	Set of arcs of project network $\vec{N}$
EC_j	Earliest completion time of activity j
ES_j	Earliest start time of activity j
G	Undirected graph based on critical set function ϱ
$[i,j]$	Edge between node i and node j in G
$\langle i,j \rangle$	Arc from node i to node j in project network $\vec{N}$
$\mathcal{K}$	Set of (renewable) resources
LB	Lower bound on minimal S_{n+1}
LC_j	Latest completion time of activity j
LS_j	Latest start time of activity j
$\vec{N}$	Project network
$\mathbb{N}$	Set of positive integers
$\mathbb{N}_0$	Set of nonnegative integers
$n+1$	Fictitious sink of project network $\vec{N}$
$\mathbb{P}(\mathcal{A})$	Power set of set $\mathcal{A}$
p_j	Processing time (duration) of activity j
$p_l^{[\sigma,\gamma]}$	Time during which activity l is at least performed within $[\sigma,\gamma]$
p_l^{ij}	Time during which activity l is at least performed within $W^{ij}(\boldsymbol{S})$
p_l^{ijk}	Processing time of activity l with respect to (P^{ijk})
(P^{ijk})	Instance of scheduling problem $1\|pmtn, r \geq 0, q \geq 0\|C_{\max}$
$\boldsymbol{\Pi}$	Preprocessing matrix $(\pi_{ij})_{i,j \in V}$
$\boldsymbol{\Pi}^{i \to j}$	Preprocessing matrix after addition of arc $\langle i,j \rangle$ with $b_{ij} = p_i$
$\boldsymbol{\Pi}^{\infty}$	Infeasibility matrix
$\mathbb{R}_+$	Set of nonnegative real numbers
$\mathbb{R}_{0,+}$	Set of positive real numbers

ϱ	Critical set function
q_l^{ij}	Tail of activity l with respect to (P^{ijk})
r_{jk}	Usage of resource k by activity j
r_l^{ij}	Head of activity l with respect to (P^{ijk})
R_k	Capacity of resource k
RF	Resource factor of an RCPSP/max instance
RS	Resource strength of an RCPSP/max instance
RT	Restrictiveness of an RCPSP/max instance
$\boldsymbol{S}$	Vector of start times (schedule)
$\mathcal{S}$	Set of feasible schedules (solution space)
$\mathcal{S}(\overline{d})$	Constrained solution space of an RCPSP/max instance
$\mathcal{S}_R$	Set of resource–feasible schedules
$\mathcal{S}_T$	Set of time–feasible schedules
S_j	Start time of activity j
S_{n+1}	Project duration
UB	Upper bound on minimal S_{n+1}
V	Set of activities including 0 and $n+1$
V'	Set of (real) activities
$V(\boldsymbol{S}, t)$	Set of activities being executed at time t
V^{ij}	Set of activities being performed within $W^{ij}(\boldsymbol{S})$
$W^{ij}(\boldsymbol{S})$	Interval $[S_i, S_i + \pi_{ij}]$
$WL_k^{[\sigma,\gamma]}$	Workload being performed on resource k within $[\sigma, \gamma]$
$\mathbb{Z}$	Set of integers

1 Introduction

For the Resource–Constrained Project Scheduling Problem (RCPSP) a large number of bounding algorithms can be found in the open literature (cf. e.g. Christofides et al. 1987, Demeulemeester & Herroelen 1992, Mingozzi et al. 1994, and Brucker et al. 1996). Whereas (classical) *constructive* methods are based on the solution of relaxations of the optimization problem under consideration, the *destructive improvement* approach, introduced by Klein & Scholl (1997), is based on the rejection of hypothetical upper bounds.

The lower bounds which have been used for the Resource–Constrained Project Scheduling Problem with Minimum and Maximum time lags (RCPSP/max) so far rely on the constructive approach. The resource–based bound LBR equals the maximum ratio of workload and resource capacity. The time–lag based lower bound LB_0 corresponds to the objective function value of an optimal solution for the resource relaxation. The tightest lower bound has been proposed by De Reyck & Herroelen (1996). It corresponds to a generalization of the weighted node packing bound devised by Mingozzi et al. (1994).

In this paper, we apply destructive improvement to RCPSP/max. Let $\mathcal{S}$ be the solution space of the RCPSP/max instance in question. By assuming that $\overline{d} \in \mathbb{R}_+$ is a valid upper bound on the minimal project duration we implicitly restrict $\mathcal{S}$. The corresponding constrained solution space is denoted by $\mathcal{S}(\overline{d})$. Applying several tests we try to show that $\mathcal{S}(\overline{d})$ is empty. Clearly, if we are able to establish $\mathcal{S}(\overline{d}) = \emptyset$, i.e., $\overline{d}$ cannot be a valid *upper bound*, $\overline{d}+1$ is proved to be a *lower bound* on the minimal project duration. Moreover, by proving that there is no $\overline{d}$, such that $\mathcal{S}(\overline{d}) \neq \emptyset$, we show that there is no feasible solution, i.e., $\mathcal{S} = \emptyset$.

Let a project consist of a finite set $V' = \{1, 2, \ldots, n\}$ of activities and a finite set of resources $\mathcal{K}$. The capacity of each resource $k \in \mathcal{K}$ is limited by $R_k \in \mathbb{N}$. Each activity $j \in V'$ has a processing time (duration) $p_j \in \mathbb{N}_0$. Once an activity has been started, no preemption is allowed. Moreover, while being performed, each activity $j \in V'$ takes up $r_{jk} \in \mathbb{N}_0$ units of resource $k \in \mathcal{K}$. Furthermore, minimum and maximum time lags $d_{ij}^{min} \in \mathbb{N}_0$ and $d_{ij}^{max} \in \mathbb{N}_0$, respectively, between the starts of two different activities i and j ($i, j \in V'$) have to be observed.

By introducing a fictitious source 0 and a fictitious sink $n+1$ (with $p_0 := p_{n+1} := 0$ and $r_{0k} := r_{n+1,k} := 0$ ($k \in \mathcal{K}$)) activities and time lags can be represented by an AoN–network $\vec{N} = \langle V, E; \boldsymbol{b} \rangle$ with node set $V := V' \cup \{0, n+1\}$, arc set E, and arc weights $\boldsymbol{b} : E \to \mathbb{Z}$. A minimum time lag d_{ij}^{min} is represented by an arc $\langle i, j \rangle$ weighted by $b_{ij} := d_{ij}^{min}$, whereas a maximum time lag d_{ij}^{max} corresponds to an arc $\langle j, i \rangle$ weighted by $b_{ji} := -d_{ij}^{max}$.

With $V(\boldsymbol{S},t) := \{j \in V | t - p_j < S_j \leq t\}$ denoting the set of activities being executed at time t the RCPSP/max can be stated as follows:

Min. S_{n+1}

s.t.

$$\begin{aligned} S_j - S_i &\geq b_{ij} && (\langle i,j\rangle \in E) && (1)\\ \sum_{j \in V(\boldsymbol{S},t)} r_{jk} &\leq R_k && (k \in \mathcal{K};\ t \geq 0) && (2)\\ S_j &\geq 0 && (j \in V) && (3)\\ S_0 &= 0 \end{aligned}$$

The objective is to determine a schedule $\boldsymbol{S} = (S_0, S_1, \ldots, S_{n+1})$ of start times $S_j \in \mathbb{R}_{0,+}$ $(j \in V)$ such that the minimum and maximum time lags (1) and resource constraints (2) are met and the start time S_{n+1} of the fictitious sink $n+1$ (project duration) is minimized.

A schedule $\boldsymbol{S}$ obeying (1) and (3) is called *time–feasible*. We call a schedule $\boldsymbol{S}$ *resource–feasible* if it observes (2) and (3). A schedule $\boldsymbol{S}$ which is time– and resource–feasible is called *feasible*. We denote the set of all time–feasible (resource–feasible) schedules by $\mathcal{S}_T$ ($\mathcal{S}_R$). $\mathcal{S} = \mathcal{S}_T \cap \mathcal{S}_R$ stands for the set of all feasible schedules. In the following, we suppose $\mathcal{S}_T$ to be nonempty.

An example for an RCPSP/max instance is given by Figure 1. Activity 5 has to start 1 unit of time after activity 2 at the earliest and 3 units of time after activity 2 at the latest.

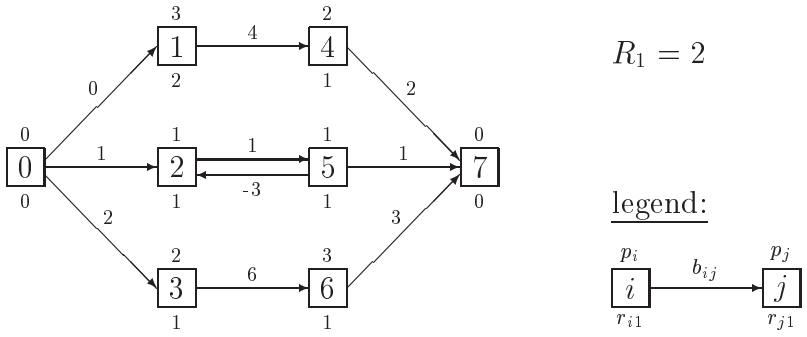


Figure 1: RCPSP/max instance

Clearly, as a generalization of the RCPSP, the RCPSP/max is $\mathcal{NP}$–hard in the strong sense. Even the feasibility problem is $\mathcal{NP}$–hard in the strong sense (cf. Bartusch et al. 1988).

The remainder of this paper is organized as follows. Section 2 is devoted to the basic concepts of our destructive improvement approach. The corresponding algorithms are given in Sections 3 and 4. The results of an experimental performance analysis are reported in Section 5.

2 Basic Concepts

A *resource conflict* occurs within a set $\mathcal{A} \subseteq V$ of activities if the total resource requirement of $\mathcal{A}$ exceeds the capacity of at least one resource $k \in \mathcal{K}$, i.e.:

$$\exists k \in \mathcal{K} : \sum_{j \in \mathcal{A}} r_{jk} > R_k. \tag{4}$$

A set $\mathcal{A}$ fulfilling (4) is called a *forbidden set*. If there is no real subset $\mathcal{A}' \subset \mathcal{A}$ which is forbidden, $\mathcal{A}$ is called *minimal forbidden set* (cf. Bartusch et al. 1988). Obviously, a time–feasible schedule $\boldsymbol{S}$ is feasible iff there is no minimal forbidden set for which all activities are processed simultaneously at one point in time.

Let $\boldsymbol{D} = (d_{ij})_{i,j \in V}$ be the *distance matrix* with d_{ij} being the length of a longest path from node i to node j $(i, j \in V)$ in $\vec{N}$. $\boldsymbol{D}$ can be computed by the Floyd–Warshall algorithm (cf. Lawler 1976) in $\mathcal{O}(|V|^3)$ time. The *time window* of an activity j $(j \in V)$ is an interval $[ES_j, LC_j)$ which is bounded by the earliest start time ES_j and latest completion time LC_j of activity j. The earliest completion time EC_j and latest start time LS_j are defined by $EC_j := ES_j + p_j$ and $LS_j := LF_j - p_j$, respectively. By setting $ES_0 := 0$ and $LF_{n+1} := \overline{d}$, earliest start times and latest completion times of all activities $j \in V$ can be obtained in $\mathcal{O}(|V||E|)$ time by network flow algorithms (cf. Elmaghraby 1977).

Based on the above definitions we state two necessary conditions for the overlapping of the activities in $\mathcal{A}$:

$$d_{ij} < p_i \qquad (i, j \in \mathcal{A}, i \neq j), \tag{5}$$

$$ES_j < LC_i \qquad (i, j \in \mathcal{A}, i \neq j). \tag{6}$$

Inequalities (5) guarantee that there is a time–feasible schedule $\boldsymbol{S}$ such that any two activities $i, j \in \mathcal{A}$ overlap in time. Inequalities (6) ensure that there is a nonempty intersection between the time windows of the activities $j \in \mathcal{A}$.

A minimal forbidden set $\mathcal{A}$ fulfilling (5) and (6) is called a *critical set*. There is a time–feasible schedule $\boldsymbol{S}$ with $S_{n+1} \leq \overline{d}$ such that all activities of a minimal forbidden set $\mathcal{A}$ overlap in time iff $\mathcal{A}$ is critical. Thus, when investigating resource conflicts, we can restrict ourselves to critical sets. Let

$$\varrho : \mathbb{P}(\mathcal{A}) \to \{0, 1\}, \varrho(\mathcal{A}) = \begin{cases} 1, & \text{if } \mathcal{A} \text{ is critical} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

be the *critical set function*.

If we want to show that $\mathcal{S}(\overline{d})$ is empty we can adopt two different approaches. First, we generate additional temporal constraints which have to be fulfilled by any feasible schedule $\mathcal{A}$ with $S_{n+1} \leq \overline{d}$ (*preprocessing check*) (cf. Section 3). These constraints are represented by additional arcs which are added to $\vec{N}$. If owing to these additional arcs a cycle of positive length occurs, i.e., $d_{jj} > 0$ for any $j \in V$, we have shown $\mathcal{S}(\overline{d}) = \emptyset$. Second, we compute constructive lower bounds LB on the minimal project duration of any schedule $\boldsymbol{S} \in \mathcal{S}(\overline{d})$ (*lower bound check*) (cf. Section 4). For $LB > \overline{d}$ we obtain $\mathcal{S}(\overline{d}) = \emptyset$.

In order to classify our preprocessing check algorithms we introduce an (α, β, γ)–notation: $\alpha \in \{con, dis\}$ stands for *conjunctive* and *disjunctive* preprocessing, respectively. Conjunctive preprocessing means that we treat critical sets for which there is at most one possibility to resolve the underlying resource conflict. Disjunctive preprocessing means that we determine alternative ways to resolve resource infeasibility. The parameters β and γ stand for the following: we examine every set $\mathcal{A} \subseteq V'$ with $|\mathcal{A}| = \beta$, such that each subset $\mathcal{A}' \subseteq \mathcal{A}$ with $|\mathcal{A}'| = \gamma$ is critical.

The destructive improvement algorithm for RCPSP/max is given by Algorithm 1. The corresponding lower bound value is denoted by LB. By an interval search within the set $\{LB_0, LB_0 + 1, \ldots, UB\}$ with

$$UB := \sum_{j \in V} \max \left\{ p_j, \max_{\langle j,l \rangle \in E} b_{jl} \right\} \tag{8}$$

we determine the minimal LB–value, for which $\mathcal{S}(LB) \neq \emptyset$. This can be done by performing at most $1 + \lg_2(UB - LB_0 + 1)$ iterations.

$LB := LB_0$
$UB := UB_{start} := \sum_{j \in V} \max \left\{ p_j, \max_{\langle j,l \rangle \in E} b_{jl} \right\}$
$\overline{d} := \left\lceil \frac{LB+UB}{2} \right\rceil$
WHILE $LB \leq UB$ **DO**
    IF $\overline{d}$ can be rejected by one of the algorithms described in Sections 3 and 4 **THEN**
        IF $\overline{d} = UB_{start}$ **THEN**
            $LB := \infty$
        ELSE
            $LB := \overline{d} + 1$
            $\overline{d} := \max \left\{ \left\lceil \frac{LB+UB}{2} \right\rceil, LB \right\}$
        END (*IF*)
    ELSE
        $UB := \overline{d} - 1$
        $\overline{d} := \min \left\{ \left\lceil \frac{LB+UB}{2} \right\rceil, UB \right\}$
    END (*IF*)
END (*WHILE*)
RETURN LB

Algorithm 1: Determine LB

3 Preprocessing Checks

3.1 Conjunctive Preprocessing Check

The $(con, 2, 2)$–preprocessing check investigates every critical set of cardinality two. The algorithm is partly based on the approaches described in Brucker et al. (1996), De Reyck & Herroelen (1996), Klein & Scholl (1997), and Schwindt (1997). For each pair (i, j) $(i, j \in V)$ with $\{i, j\}$ being a critical set we evaluate two necessary conditions in order to find out whether activity j can be completed before the start of activity i.

The first condition, which is called *time lag condition*, is illustrated in Figure 2. Here, the processing time of activity j (4 units of time) is not larger than the maximum time lag between the start of activity j and the start of activity i (5 units of time). Thus, activity j can be completed before the start of activity i.

Figure 2: Time lag condition

The time lag condition can be stated as follows (cf. Brucker et al. 1996 and De Reyck & Herroelen 1996):

$$p_j \leq -d_{ij}. \tag{9}$$

The second condition is called *time window condition*. Figure 3 illustrates the time windows of activity i and activity j. If activity j starts at ES_j and activity i at LS_i, activity j ends after the start of activity i. Therefore, activity j cannot be completed before the start of activity i.

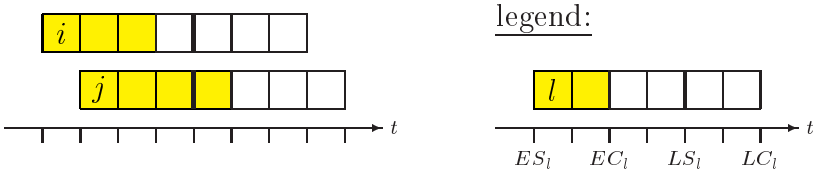


Figure 3: Time window condition

The time window condition can be stated as follows (cf. Brucker et al. 1996):

$$ES_j + p_j \leq LS_i. \tag{10}$$

If at least one of the inequalities (9) and (10) does not hold, activity j cannot be completed before the start of activity i. Since $\{i, j\}$ is critical, activity j has to be started after the completion of activity i, i.e., an arc $\langle i, j\rangle$ with $b_{ij} = p_i$ can be added to the project network $\vec{N}$. In the following, we use a so–called *preprocessing matrix* $\mathbf{\Pi} = (\pi_{ij})_{i,j\in V}$ which reflects the (new) lengths of longest paths from node i to node j $(i, j \in V)$ after arcs have been added to $\vec{N}$. The preprocessing matrix is initialized by $\mathbf{\Pi} := \boldsymbol{D}$.

In analogy to Brucker et al. (1996) we omit the time window condition by adding an arc $\langle n+1, 0\rangle$ with $b_{n+1,0} := -\overline{d}$. For the following, we suppose the current upper bound $\overline{d}$ to be represented in this way.

For each activity $j \in V$ the following equations hold:

$$ES_j = \pi_{0j}, \tag{11}$$
$$EC_j = \pi_{0j} + p_j, \tag{12}$$
$$LS_j = -\pi_{j0}, \tag{13}$$
$$LC_j = -\pi_{j0} + p_j. \tag{14}$$

Adding an arc $\langle i, j\rangle$ with $b_{ij} = p_i$ $(i, j \in V, i \neq j)$ to $\vec{N}$, the resulting preprocessing matrix, denoted by $\mathbf{\Pi}^{i\to j}$, can be obtained by Algorithm 2 in $\mathcal{O}(|V|^2)$ time (cf. Bartusch 1983).

FOR $l_1, l_2 \in V$ **DO**
  $\pi_{l_1 l_2}^{i\to j} := \max\{\pi_{l_1 l_2}, \pi_{l_1 i} + p_i + \pi_{j l_2}\}$
END (*FOR*)
IF $\exists l \in V : \pi_{ll}^{i\to j} > 0$ **THEN**
  $\mathbf{\Pi}^{i\to j} := \mathbf{\Pi}^{\infty}$
END (*IF*)
RETURN $\mathbf{\Pi}^{i\to j}$

Algorithm 2: Determine $\mathbf{\Pi}^{i\to j}$

If the insertion of $\langle i, j\rangle$ leads to a cycle of positive length, i.e., $\pi_{ll}^{i\to j} > 0$ for any $l \in V$, we set $\pi_{l_1 l_2}^{i\to j} := \infty$ for all $l_1, l_2 \in V$. The corresponding *infeasibility matrix* is denoted by $\mathbf{\Pi}^{\infty}$. In that case, the current hypothetical upper bound $\overline{d}$ can be refuted.

In contrast to the authors cited above, we evaluate the time lag condition for any pair (i, j) of activities $i, j \in V$ $(i \neq j)$ until no more arcs can be added. The pseudo–code of the $(con, 2, 2)$–preprocessing check is provided by Algorithm 3.

```
stop := FALSE
WHILE stop = FALSE DO
    stop := TRUE
    FOR i, j ∈ V with ϱ({i, j}) = 1 DO
        IF p_j > −π_ij THEN
            stop := FALSE
            Π := Π^{i→j}
            IF Π = Π^∞ THEN
                RETURN FALSE
            END (*IF*)
        END (*IF*)
    END (*FOR*)
END (*WHILE*)
RETURN TRUE
```

Algorithm 3: $(con, 2, 2)$–preprocessing check

Since at most $\frac{|V|(|V|-1)}{2}$ arcs can be added without obtaining a cycle of positive length, the time complexity is as given by

Proposition 1.

The time complexity of Algorithm 3 is $\mathcal{O}(|V|^4)$.

3.2 Disjunctive Preprocessing Checks

These preprocessing checks are partly based on the approach of Klein & Scholl (1997). Let $\mathcal{A} = \{j_1, j_2, j_3\}$ be a critical set (cf. Figure 4). In order to obtain a feasible schedule $\boldsymbol{S}$, the resource conflict corresponding to $\mathcal{A}$ has to be solved by preventing a simultaneous execution of the activities j_1, j_2, and j_3.

The $(dis, 3, 3)$–preprocessing check examines all possibilities to solve the underlying resource conflict. At least two of activities j_1, j_2, and j_3 have to be performed consecutively. There are two possibilities to enforce the consecutive execution of two activities j_μ and j_ν $(j_\mu, j_\nu \in \mathcal{A}, \mu \neq \nu)$: either we add

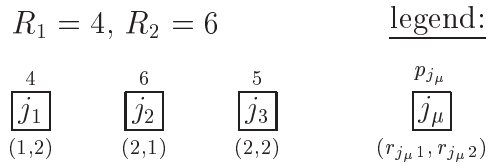


Figure 4: $(dis, 3, 3)$–preprocessing check

an arc $\langle j_\mu, j_\nu \rangle$ weighted by p_{j_μ} or we introduce an arc $\langle j_\nu, j_\mu \rangle$ weighted by p_{j_ν} to project network $\vec{N}$. As $\mathcal{A}$ contains three different subsets $\{j_\mu, j_\nu\}$, there are six alternative possibilities of solving the resource conflict induced by $\mathcal{A}$. Each possibility can be represented by the addition of an arc $\langle j_\mu, j_\nu \rangle$ to $\vec{N}$ corresponding to preprocessing matrices $\mathbf{\Pi}^{j_\mu \to j_\nu}$. The minimal matrix

$$\mathbf{\Pi} := \Big(\min_{\substack{j_\mu, j_\nu \in \mathcal{A} \\ \mu \neq \nu}} \pi_{l_1 l_2}^{j_\mu \to j_\nu} \Big)_{l_1, l_2 \in V}, \tag{15}$$

which can be computed in $\mathcal{O}(|V|^2)$ time, represents the constraints which have to be observed regardless of the way the resource conflict is solved. The current upper bound $\overline{d}$ can be refuted if we establish for one critical set $\mathcal{A}$ that none of the six possible arcs can be added such that

$$\mathbf{\Pi}^{j_\mu \to j_\nu} \neq \mathbf{\Pi}^{\infty}. \tag{16}$$

The pseudo–code of the $(dis, 3, 3)$–preprocessing check is given by Algorithm 4.

FOR $j_1, j_2, j_3 \in V$ with $\varrho(\{j_1, j_2, j_3\}) = 1$ **DO**
  FOR $l_1, l_2 \in V$ **DO**
    $\pi_{l_1 l_2} := \min \left\{ \pi_{l_1 l_2}^{j_1 \to j_2}, \pi_{l_1 l_2}^{j_2 \to j_1}, \pi_{l_1 l_2}^{j_1 \to j_3}, \pi_{l_1 l_2}^{j_3 \to j_1}, \pi_{l_1 l_2}^{j_2 \to j_3}, \pi_{l_1 l_2}^{j_3 \to j_2} \right\}$
  END (∗FOR∗)
  IF $\mathbf{\Pi} = \mathbf{\Pi}^{\infty}$ **THEN**
    RETURN FALSE
  END (∗IF∗)
END (∗FOR∗)
RETURN TRUE

Algorithm 4: $(dis, 3, 3)$–preprocessing check

Proposition 2.

The time complexity of Algorithm 4 is $\mathcal{O}(|V|^5)$.

The $(dis, 3, 2)$–preprocessing check treats every set $\mathcal{A} = \{j_1, j_2, j_3\}$ for which each subset of cardinality two is critical (cf. Figure 5). In other words, we examine all cliques of cardinality three in an undirected graph $G = [V', E']$ with: $[i, j] \in E'$ iff $\varrho(\{i, j\}) = 1$ $(i, j \in V')$ (cf. Brucker et al. 1996).

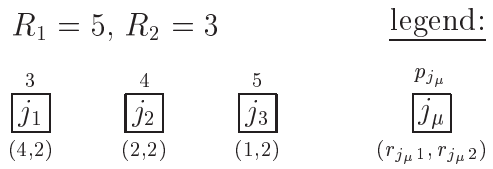


Figure 5: $(dis, 3, 2)$–preprocessing check

In analogy to the $(dis, 3, 3)$–preprocessing check, we prevent simultaneous execution of activities by additional arcs. As each set $\{j_\mu, j_\nu\}$ $(j_\mu, j_\nu \in \mathcal{A}, \mu \neq \nu)$ is critical, all activities in $\mathcal{A}$ have to be executed one after the other. Since there are six permutations of the elements in $\mathcal{A}$, we obtain six preprocessing matrices. As described above, we compute the corresponding minimal matrix. Each of these matrices results from the addition of two different arcs $\langle j_\mu, j_\nu \rangle$. For example, if we look at the permutation (j_2, j_3, j_1), the corresponding additional arcs are $\langle j_2, j_3 \rangle$ weighted by p_{j_2} and $\langle j_3, j_1 \rangle$ weighted by p_{j_3}. The pseudo–code of the $(dis, 3, 2)$–preprocessing check is provided by Algorithm 5.

FOR $j_1, j_2, j_3 \in V$: $\varrho(\{j_1, j_2\}) = \varrho(\{j_1, j_3\}) = \varrho(\{j_2, j_3\}) = 1$ **DO**
  FOR $l_1, l_2 \in V$ **DO**

$$\pi_{l_1 l_2} := \min\left\{ \left(\pi_{l_1 l_2}^{j_1 \to j_2}\right)^{j_2 \to j_3}, \left(\pi_{l_1 l_2}^{j_1 \to j_3}\right)^{j_3 \to j_2}, \left(\pi_{l_1 l_2}^{j_2 \to j_1}\right)^{j_1 \to j_3}, \left(\pi_{l_1 l_2}^{j_2 \to j_3}\right)^{j_3 \to j_1}, \left(\pi_{l_1 l_2}^{j_3 \to j_1}\right)^{j_1 \to j_2}, \left(\pi_{l_1 l_2}^{j_3 \to j_2}\right)^{j_2 \to j_1} \right\}$$

  END (∗FOR∗)
  IF $\mathbf{\Pi} = \mathbf{\Pi}^\infty$ **THEN**
    RETURN FALSE
  END (∗IF∗)
END (∗FOR∗)
RETURN TRUE

Algorithm 5: $(dis, 3, 2)$–preprocessing check

Proposition 3.

The time complexity of Algorithm 5 is $\mathcal{O}(|V|^5)$.

3.3 Machine Scheduling Preprocessing Check

The machine scheduling preprocessing check relies on the one–machine scheduling problem $1|pmtn, r \geq 0, q \geq 0|C_{\max}$ with heads $r \geq 0$ and tails $q \geq 0$ (cf. Schwindt 1997). A similar approach has been used by Brucker et al. (1997) for the solution of complex machine scheduling problems by a branch–and–bound algorithm for a one–machine scheduling problem with minimum and maximum time lags. Optimal solutions to appropriate instances of $1|pmtn, r \geq 0, q \geq 0|C_{\max}$ provide the time which is at least necessary for carrying out activities to be processed within a given time window $W^{ij}(\boldsymbol{S}) := [S_i, S_i + \pi_{ij}]$ $(i, j \in V, i \neq j)$. For any time window $W^{ij}(\boldsymbol{S})$ we define a subset of activities $V^{ij} \subseteq V$ which have to be (at least partially) performed within $W^{ij}(\boldsymbol{S})$. As illustrated in Figure 6, we obtain the minimum time p_l^{ij} during which an activity l $(l \in V)$ has to be executed within $W^{ij}(\boldsymbol{S})$ by

$$p_l^{ij} := \max\Big\{0, \min\big\{p_l, \pi_{ij}, \pi_{il} + p_l, \pi_{lj}\big\}\Big\}. \tag{17}$$

The brightly and darkly shaded boxes correspond to the earliest and latest position of activity j, respectively.

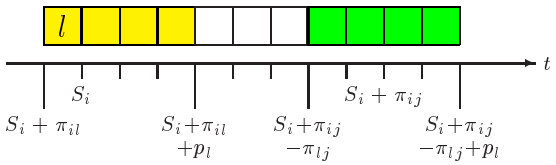


Figure 6: Determine p_l^{ij}

For given pair (i, j) of activities $i, j \in V$ $(i \neq j)$ and resource $k \in \mathcal{K}$ we define an instance (P^{ijk}) of $1|pmtn, r \geq 0, q \geq 0|C_{\max}$ as follows. For each activity $l \in V^{ij}$ we determine an appropriate (resource–dependent) processing time p_l^{ijk}, a head r_l^{ij}, and a tail q_l^{ij} by

$$p_l^{ijk} := \frac{p_l^{ij} r_{lk}}{R_k}, \tag{18}$$

$$r_l^{ij} := \max\{0, \pi_{il}\}, \quad \text{and} \quad q_l^{ij} := \max\{0, \pi_{lj} - p_l\}. \tag{19}$$

The optimal objective function value $C^*_{\max}(P^{ijk})$ can be computed by the algorithm of Carlier (1982) in $\mathcal{O}(|V^{ij}| \log |V^{ij}|)$ time.

Theorem 1.

The minimal objective function value $C^*_{\max}(P^{ijk})$ of the $1|pmtn, r \geq 0, q \geq 0|C_{\max}$ instance (P^{ijk}) represents a lower bound on $S_j - S_i$ for any feasible schedule $\boldsymbol{S} \in \mathcal{S}$.

Proof.

For a subset $\mathcal{I} \subseteq V^{ij}$ we define

$$h(\mathcal{I}) := \min_{l \in \mathcal{I}} r_l^{ij} + \sum_{l \in \mathcal{I}} p_l^{ijk} + \min_{l \in \mathcal{I}} q_l^{ij}. \tag{20}$$

Using (18) we obtain

$$h(\mathcal{I}) = \min_{l \in \mathcal{I}} r_l^{ij} + \frac{\sum\limits_{l \in \mathcal{I}} p_l^{ij} r_{lk}}{R_k} + \min_{l \in \mathcal{I}} q_l^{ij} \quad (\mathcal{I} \subseteq V^{ij}). \tag{21}$$

(19) implies

$$\min_{l \in \mathcal{I}} r_l^{ij} \leq \min_{l \in \mathcal{I}} (S_l - S_i) \tag{22}$$

and

$$\min_{l \in \mathcal{I}} q_l^{ij} \leq \min_{l \in \mathcal{I}} (S_j - (S_l + p_l^{ij})) \quad (\mathcal{I} \subseteq V^{ij}). \tag{23}$$

Furthermore, for each feasible schedule $\boldsymbol{S}$ and any resource $k \in \mathcal{K}$, the workload $\sum_{l \in \mathcal{I}} p_l^{ij} r_{lk}$ requested by set $\mathcal{I}$ must not exceed the workload $(\max_{l \in \mathcal{I}}(S_l + p_l^{ij}) - \min_{l \in \mathcal{I}} S_l) R_k$. Due to $R_k > 0$ we obtain

$$\frac{\sum\limits_{l \in \mathcal{I}} p_l^{ij} r_{lk}}{R_k} \leq \max_{l \in \mathcal{I}} (S_l + p_l^{ij}) - \min_{l \in \mathcal{I}} S_l \quad (\mathcal{I} \subseteq V^{ij}). \tag{24}$$

(22), (23), and (24) imply

$$\begin{aligned} h(\mathcal{I}) &\leq \min_{l \in \mathcal{I}} (S_l - S_i) + \max_{l \in \mathcal{I}} (S_l + p_l^{ij}) - \min_{l \in \mathcal{I}} S_l + \min_{l \in \mathcal{I}} (S_j - S_l - p_l^{ij}) \\ &= \min_{l \in \mathcal{I}} S_l - S_i + \max_{l \in \mathcal{I}} (S_l + p_l^{ij}) - \min_{l \in \mathcal{I}} S_l + S_j - \max_{l \in \mathcal{I}} (S_l + p_l^{ij}) \\ &= S_j - S_i \quad (\mathcal{I} \subseteq V^{ij}). \end{aligned} \tag{25}$$

Hence, for any feasible schedule $\boldsymbol{S}$ it holds that

$$\max_{\mathcal{I} \subseteq V^{ij}} h(\mathcal{I}) \leq S_j - S_i. \tag{26}$$

In Carlier (1982) it has been shown that

$$C^*_{\max}(P^{ijk}) = \max_{\mathcal{I} \subseteq V^{ij}} h(\mathcal{I}). \tag{27}$$

□

If $C^*_{\max}(P^{ijk})$ exceeds the breadth π_{ij} of time window $W^{ij}(\boldsymbol{S})$, we add an arc $\langle i, j \rangle$ with $b_{ij} := C^*_{\max}(P^{ijk})$ to $\vec{N}$. If after the update of preprocessing matrix $\mathbf{\Pi}$ infeasibility occurs, we can reject the current upper bound value $\overline{d}$. The corresponding pseudo–code is given by Algorithm 6.

FOR $i, j \in V$ with $\pi_{ij} > 0$ **DO**
  $V^{ij} := \emptyset$
  FOR $l \in V$ **DO**
    $p_l^{ij} := \max\left\{0, \min\left\{p_l, \pi_{ij}, \pi_{il} + p_l, \pi_{lj}\right\}\right\}$
    IF $p_l^{ij} > 0$ **THEN**
      $V^{ij} := V^{ij} \cup \{l\}$
      $r_l^{ij} := \max\{0, \pi_{il}\}$
      $q_l^{ij} := \max\{0, \pi_{lj} - p_l\}$
    END (*IF*)
  END (*FOR*)
  FOR $k \in \mathcal{K}$ **DO**
    FOR $l \in V^{ij}$ **DO**
      $p_l^{ijk} := \dfrac{p_l^{ij} r_{lk}}{R_k}$
    END (*FOR*)
    $\pi_{ij} := \max\left\{\pi_{ij}, C^*_{\max}(P^{ijk})\right\}$
    Update $\mathbf{\Pi}$
    IF $\mathbf{\Pi} = \mathbf{\Pi}^\infty$ **THEN**
      RETURN FALSE
    END (*IF*)
  END (*FOR*)
END (*FOR*)
RETURN TRUE

Algorithm 6: Machine scheduling preprocessing check

Proposition 4.

The time complexity of Algorithm 6 is $\mathcal{O}(|\mathcal{K}||V|^3 \log |V|)$.

Let us consider the example depicted in Figure 7. The breadth of time window $W^{ij}(\boldsymbol{S})$ equals 4 units of time and set V^{ij} consists of activities i, j', and j''. The processing times, heads, and tails are listed by Table 1. Solving (P^{ijk}) leads to $C^*_{\max}(P^{ijk}) = 5$ (cf. Figure 8). Thus, an arc $\langle i, j \rangle$ with $b_{ij} = 5$ can be added to the project network.

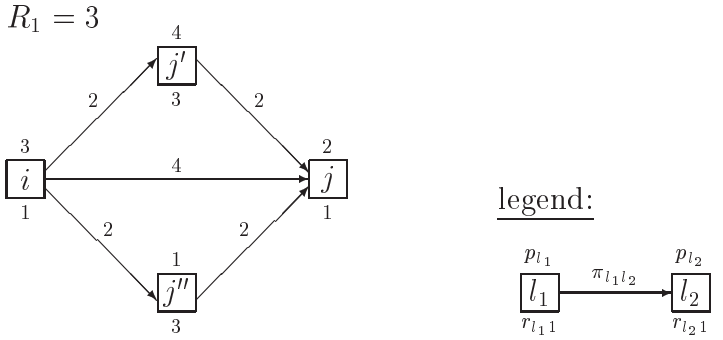


Figure 7: Machine scheduling preprocessing check

Table 1: Computation of $p_l^{ij}, p_l^{ij1}, r_l^{ij}$, and q_l^{ij}

l	p_l^{ij}	p_l^{ij1}	r_l^{ij}	q_l^{ij}
i	$\max\{0, \min\{3,4,3,4\}\} = 3$	$\frac{3*1}{3} = 1$	$\max\{0,0\} = 0$	$\max\{0,1\} = 1$
j'	$\max\{0, \min\{4,4,6,2\}\} = 2$	$\frac{2*3}{3} = 2$	$\max\{0,2\} = 2$	$\max\{0,-2\} = 0$
j''	$\max\{0, \min\{1,4,3,2\}\} = 1$	$\frac{1*3}{3} = 1$	$\max\{0,2\} = 2$	$\max\{0,1\} = 1$

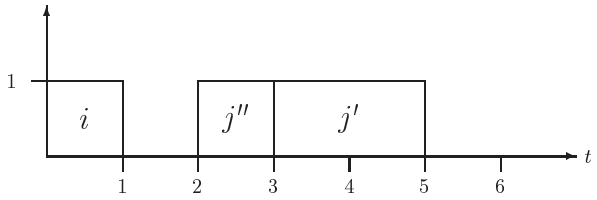


Figure 8: Gantt chart

4 Lower Bound Checks

4.1 Workload Lower Bound Check

Let $p_l^{[\sigma,\gamma]}$ be the minimum time during which an activity $l \in V$ has to be executed within the interval $[\sigma, \gamma]$ (cf. Figure 9). We obtain $p_l^{[\sigma,\gamma]}$ by

$$p_l^{[\sigma,\gamma]} := \max\Big\{0, \min\{p_l, \gamma - \sigma, \pi_{0l} + p_l - \sigma, \gamma + \pi_{l0}\}\Big\}. \tag{28}$$

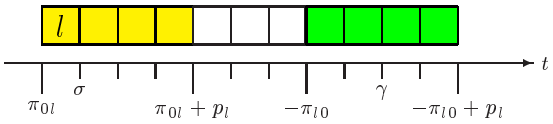


Figure 9: Determine $p_l^{[\sigma,\gamma]}$

$p_l^{[\sigma,\gamma]} r_{lk}$ represents the fraction of workload $WL_k^{[\sigma,\gamma]} := \sum_{l \in V} p_l^{[\sigma,\gamma]} r_{lk}$ which has to be performed on resource k between the points in time σ and $\gamma > \sigma$. Note, that, in contrast to the machine scheduling preprocessing check, σ and γ do not depend on start times of activities.

If the time $\left\lceil \frac{WL_k^{[\sigma,\gamma]}}{R_k} \right\rceil$, which is at least necessary for carrying out $WL_k^{[\sigma,\gamma]}$ on resource k, exceeds the length $\gamma - \sigma$ of interval $[\sigma, \gamma]$, $\overline{d}$ cannot be a valid upper bound on the minimal project duration.

The workload lower bound check relies on the following unconstrained optimization problem:

$$\max_{k \in \mathcal{K}} \max_{\substack{(\sigma,\gamma) \in [0,\overline{d}]^2 \\ \gamma > \sigma}} \left(\frac{WL_k^{[\sigma,\gamma]}}{R_k} - (\gamma - \sigma) \right). \tag{29}$$

Obviously, we can refute $\overline{d}$ as a valid upper bound, if the optimal objective function value of (29) is positive. In Schwindt (1997) it has been shown that (29) can be solved to optimality by the enumeration of the intervals listed in Table 2. Thus, the number of intervals $[\sigma, \gamma]$, which have to be investigated, is polynomially bounded by $\mathcal{O}(|V|^2)$. The pseudo–code of the workload lower bound check is provided by Algorithm 7.

Table 2: Intervals $[\sigma, \gamma]$

Class	$[\sigma, \gamma]$	with
(i)	$[ES_i, LC_j]$	$LC_i \leq LC_j, ES_i \leq ES_j$
(ii)	$[ES_i, EC_j]$	$LC_i \leq EC_j, ES_i \geq LS_j$
(iii)	$[ES_i, ES_j + LC_j - ES_i]$	$ES_j \leq ES_i \leq LC_j, ES_i + LC_i \leq ES_j + LC_j$
(iv)	$[LS_i, LC_j]$	$EC_i \geq LC_j, LS_i \leq ES_j$
(v)	$[LS_i, EC_j]$	$EC_i \geq EC_j, LS_i \geq LS_j$
(vi)	$[LS_i, ES_j + LC_j - LS_i]$	$ES_j \leq LS_i \leq LS_j, ES_i + LC_i \geq ES_j + LC_j$
(vii)	$[ES_i + LC_i - LC_j, LC_j]$	$EC_i \leq LC_j \leq LS_i, ES_i + LC_i \leq ES_j + LC_j$
$(viii)$	$[ES_i + LC_i - EC_j, EC_j]$	$EC_i \leq EC_j \leq LC_i, ES_i + LC_i \geq ES_j + LC_j$

FOR $k \in \mathcal{K}$ **DO**
  FOR $[\sigma, \gamma]$ given by Table 2 with $\gamma > \sigma$ **DO**
    IF $\left\lceil \frac{WL_k^{[\sigma,\gamma]}}{R_\kappa} \right\rceil > \gamma - \sigma$ **THEN**
      RETURN FALSE
    END (∗IF∗)
  END (∗FOR∗)
END (∗FOR∗)
RETURN TRUE

Algorithm 7: Workload lower bound check

Proposition 5.

The time complexity of Algorithm 7 is $\mathcal{O}(|\mathcal{K}||V|^3)$.

The workload lower bound check is illustrated by the example depicted in Figure 10. The minimum time during which the activities j, j', and j'' have to be performed within interval $[\sigma, \gamma]$ (cf. class (v) in Table 2) is 2, 1, and 2, respectively. Thus, the workload $WL_1^{[\sigma,\gamma]}$ with respect to resource 1 is $2*1+1*4+2*2=10$. $\left\lceil \frac{WL_1^{[\sigma,\gamma]}}{R_1} \right\rceil = \left\lceil \frac{10}{4} \right\rceil = 3$ yields a lower bound for carrying out $WL_1^{[\sigma,\gamma]}$. Since the length of interval $[\sigma, \gamma]$ is 2, the hypothetical upper bound $\overline{d}$ has been disproved.

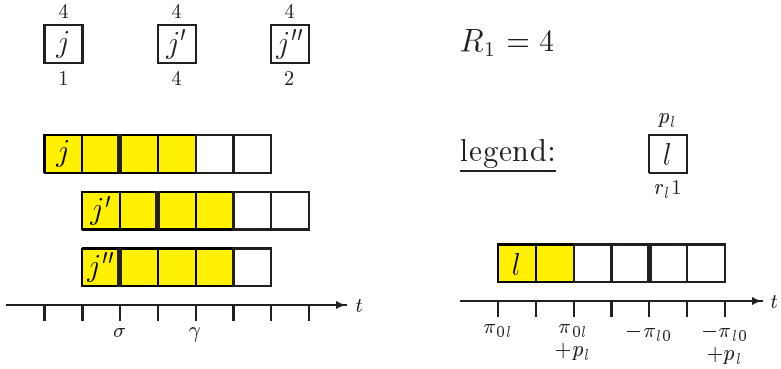


Figure 10: Workload lower bound check

4.2 Active Chain Lower Bound Check

The active chain lower bound check is based on the examination of each clique $\mathcal{A}$ with $|\mathcal{A}| \in \{2,3,4,5\}$ in an undirected graph $G=[V', E']$ with E' as defined in Section 3.2. We check whether the interval $[t_1, t_2]$ with

$$t_1 := \min_{j \in \mathcal{A}} \pi_{0j} \quad \text{and} \quad t_2 := \max_{j \in \mathcal{A}} (-\pi_{j0} + p_j), \tag{30}$$

which is available for the consecutive execution of any sequence of the activities included in $\mathcal{A}$ (*active chain*), is large enough. Since any two activities $j_\mu, j_\nu \in \mathcal{A}$ $(\mu \neq \nu)$ have to be performed one after the other, the time which is at least necessary for the execution of any sequence is given by $\sum_{j \in \mathcal{A}} p_j$. If the latter value exceeds the length $t_2 - t_1$ of $[t_1, t_2]$, we can reject $\overline{d}$ as a valid upper bound on the minimal project duration (cf. Algorithm 8).

FOR $\mathcal{A} \subseteq V:\ |\mathcal{A}| \in \{2,3,4,5\} \wedge \varrho(\{i,j\}) = 1\ (i,j \in \mathcal{A}, i \neq j)$ **DO**
  IF $\sum_{j \in \mathcal{A}} p_j > \max_{j \in \mathcal{A}}(-\pi_{j0} + p_j) - \min_{j \in \mathcal{A}} \pi_{0j}$ **THEN**
    RETURN FALSE
  END (∗IF∗)
END (∗FOR∗)
RETURN TRUE

Algorithm 8: Active chain lower bound check

Proposition 6.

The time complexity of Algorithm 8 is $\mathcal{O}(|V|^5)$.

Let us consider the example depicted in Figure 11. The length of the interval $[t_1, t_2]$ equals 4 units of time whereas the processing times of activities $j_1, j_2, \ldots, j_5$ sum up to 5 units of time. Hence, $\overline{d}$ cannot be a valid upper bound.

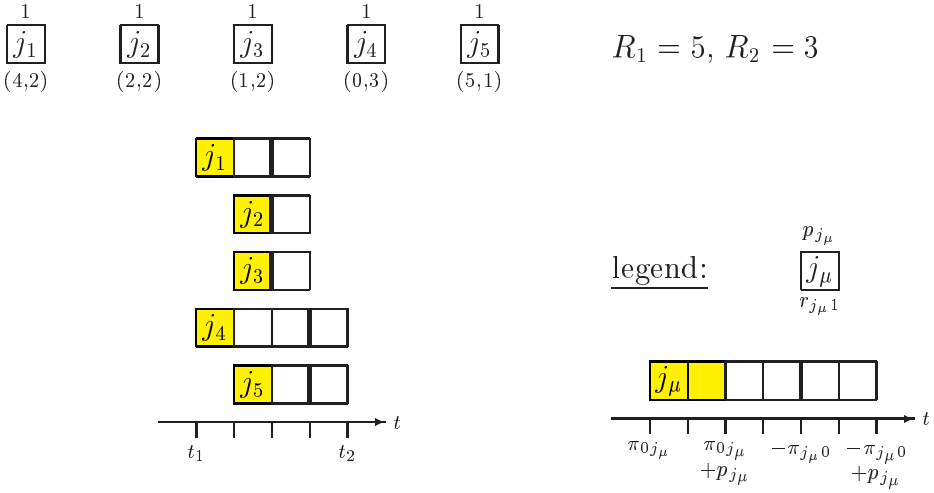


Figure 11: Active chain lower bound check

5 Computational Results

In order to evaluate the performance of the proposed algorithms we have computed lower bound values for the testset SOR96 which has been generated using ProGen/max (cf. Schwindt 1997). This testset consists of 810 RCPSP/max instances. 524 instances have a feasible solution.

The control parameters used for the full factorial design of SOR96 are given by Table 3. For each parameter combination, 10 RCPSP/max instances have been generated.

Table 3: Parameter setting of SOR96

Parameter	Values
$\|V'\|$	10, 15, 20
RT	0.25, 0.50, 0.65
RF	0.50, 0.75, 1.00
RS	0.00, 0.25, 0.50

The restrictiveness RT reflects the degree of parallelity of network $\vec{N}$ (cf. Schwindt 1996). With increasing RT the parallelity of $\vec{N}$ decreases. RT is defined as follows:

$$RT := 1 - \frac{m_d}{m_d^{max}} = 1 - \frac{2m_d}{|V'|(|V'|-1)}. \tag{31}$$

m_d and m_d^{max} denote the number of disjunctive edges in $\vec{N}$ and the maximal number of disjunctive edges in networks with $|\mathcal{V}|$ nodes, respectively. There is a disjunctive edge between node i and node j $(i, j \in V', i \neq j)$ iff $d_{ij} = d_{ji} = -\infty$.

The resource factor RF corresponds to the average percentage of resources required per activity (cf. Pascoe 1966):

$$RF := \frac{1}{|V'|}\frac{1}{|\mathcal{K}|}\sum_{j\in V'}\sum_{k\in\mathcal{K}}\begin{cases}1, & \text{if } r_{jk} > 0\\ 0, & \text{otherwise}\end{cases} \tag{32}$$

Roughly speaking, the resource strength RS reflects the ratio of resource capacity and resource requirements (cf. Kolisch 1995). RS is obtained by

$$RS := \frac{1}{|\mathcal{K}|}\sum_{k\in\mathcal{K}} RS_k. \tag{33}$$

For each resource $k \in \mathcal{K}$, RS_k corresponds to the scaling parameter of a convex combination of a minimal demand r_k^{min} and a maximal demand r_k^{max}:

$$\begin{aligned} & R_k = r_k^{min} + RS_k(r_k^{max} - r_k^{min}) \\ \text{with:} \quad & r_k^{min} := \max_{j \in V'} r_{jk}, \\ & r_k^{max} := \max_{t \geq 0} \sum_{j \in V(\boldsymbol{ES},t)} r_{jk}, \\ & \boldsymbol{ES} := (ES_0, ES_1, \ldots, ES_{n+1}). \end{aligned} \tag{34}$$

The relationships between the parameters RT, RF, and RS and the average computation time μ_{CPU} of exact RCPSP/max procedures are listed in Table 4 (cf. De Reyck 1995, Sprecher et al. 1995, and Schwindt 1996).

Table 4: Influence of RT, RF, and RS on μ_{CPU}

Parameter	Influence
RT	$\searrow$
RF	$\nearrow$
RS	$\nearrow$, $\searrow$

As indicated by "$\searrow$", μ_{CPU} decreases with increasing RT, because the number of resource conflicts to be solved decreases, too. With increasing RF the number of possibilities to solve resource conflicts and thus μ_{CPU} increases ("$\nearrow$"). The relationship which has been observed between RS and μ_{CPU} is the following: the maximal value of μ_{CPU} is obtained for an $RS \in (0, 1)$. The reason is, that for $RS = 0$ the number of alternatives to resolve resource infeasibilities is comparatively small and that for $RS = 1$ there is not any resource conflict to be solved.

By the consecutive execution of the algorithms described in Sections 3 and 4 a lower bound value ($LBDI_1$) has been computed for each feasible instance of SOR96. We compare $LBDI_1$ to LBR, LB_0, and the lower bound LB_3^g proposed by De Reyck & Herroelen (1996) with respect to the average percentage deviation (μ_{DEV}) and the maximal percentage deviation ($\max_{DEV}$) from the optimal objective function values. For a given instance, the deviation DEV of a lower bound value LB from the optimal objective function value S_{n+1}^* is given by

$$DEV := \frac{S_{n+1}^* - LB}{S_{n+1}^*} \cdot 100\%. \tag{35}$$

LBR, LB_0, LB_3^g, and $LBDI_1$ have been coded in ANSI C using the MS Visual C++ 5.0 Developer Studio under the operating system Windows NT 4.0. The experimental performance analysis has been conducted on an Intel Pentium–200MHz personal computer with 64 MB RAM. The results of a full factorial analysis are listed in Tables 5 to 7. For each parameter combination, we provide μ_{DEV} and, in parenthesis, $\max_{DEV}$. The first row corresponds to LBR, the second row to LB_0, the third row to LB_3^g, and the fourth row to $LBDI_1$.

Apart from few exceptions, the following (transitive) ranking of the lower bounds with respect to μ_{DEV} and $\max_{DEV}$ can be established:

$$LBDI_1 \succ LB_3^g \succ LB_0 \succ LBR \tag{36}$$

with $a \succ b$ if lower bound a provides smaller μ_{DEV} and $\max_{DEV}$ values than lower bound b.

For the parameter combinations ($RT = 0.25$, $RF = 0.75$, $RS = 0.00$) and ($RT = 0.25$, $RF = 1.00$, $RS = 0.00$) characterizing hard instances we obtain different rankings. For both combinations, LBR turns out to be better than LB_0. For the latter combination, LB_3^g outperforms $LBDI_1$.

The average percentage deviation μ_{DEV} and the average computation time μ_{CPU} in seconds for calculating the lower bound values are given in Table 8. Obviously, the very good performance of $LBDI_1$ is achieved at the expense of a relatively large average computation time μ_{CPU}.

Table 5: $RT = 0.25$

RF \ RS	0.00		0.25		0.50	
0.50	32.4	(63.6)	49.2	(79.6)	60.1	(77.4)
	29.4	(43.6)	9.5	(29.6)	1.7	(17.6)
	7.8	(16.2)	5.2	(18.9)	0.9	(13.7)
	2.3	(14.8)	0.1	(1.4)	0.0	(0.0)
0.75	**35.0**	(**65.2**)	41.4	(60.7)	54.6	(76.5)
	42.6	(**64.4**)	10.3	(29.8)	4.1	(23.2)
	10.5	(24.4)	6.1	(19.7)	3.6	(23.2)
	5.3	(23.6)	0.6	(5.5)	0.1	(1.9)
1.00	**31.6**	(**39.2**)	33.9	(57.9)	41.4	(73.3)
	44.9	(**57.8**)	18.5	(43.4)	4.8	(22.9)
	5.3	(**12.5**)	11.5	(29.8)	4.7	(22.9)
	11.1	(**22.2**)	2.9	(16.0)	0.4	(8.4)

Table 6: $RT = 0.50$

RF \ RS	0.00		0.25		0.50	
0.50	45.8	(60.2)	55.1	(72.6)	63.3	(77.4)
	24.1	(51.3)	10.0	(31.2)	3.3	(21.2)
	6.3	(19.1)	5.6	(22.4)	2.0	(21.2)
	0.1	(0.8)	0.0	(0.0)	0.3	(8.1)
0.75	40.5	(58.9)	47.3	(66.0)	54.4	(73.5)
	33.3	(52.7)	15.3	(40.0)	3.9	(12.5)
	9.4	(18.9)	7.8	(23.6)	3.1	(12.5)
	1.9	(9.9)	0.4	(5.0)	0.2	(4.6)
1.00	34.0	(42.3)	47.6	(64.6)	50.7	(65.0)
	23.8	(40.5)	13.3	(30.8)	5.5	(23.0)
	7.0	(16.7)	5.1	(22.8)	4.3	(21.3)
	3.6	(7.0)	0.3	(5.3)	0.2	(4.2)

Table 7: $RT = 0.65$

RF \ RS	0.00		0.25		0.50	
0.50	51.9	(71.0)	60.9	(75.0)	70.1	(79.9)
	20.3	(46.9)	13.4	(39.4)	3.5	(13.6)
	4.6	(17.5)	5.2	(23.1)	1.9	(12.5)
	0.0	(0.0)	0.0	(0.0)	0.0	(0.0)
0.75	43.5	(61.7)	56.5	(72.3)	64.5	(75.6)
	24.8	(46.2)	14.3	(26.0)	5.7	(18.8)
	6.5	(13.9)	5.7	(19.5)	3.0	(14.9)
	1.0	(6.6)	0.2	(2.9)	0.0	(1.1)
1.00	46.3	(54.4)	54.8	(65.5)	58.4	(72.4)
	21.9	(33.9)	20.3	(44.9)	7.1	(20.0)
	6.5	(16.5)	8.0	(37.0)	3.9	(18.1)
	0.1	(1.0)	0.4	(4.7)	0.1	(1.8)

Table 8: μ_{DEV} and μ_{CPU} for lower bounds

Lower bound	μ_{DEV}	μ_{CPU}
LBR	51.6	<0.01
LB_0	11.7	<0.01
LB_3^g	5.0	<0.01
$LBDI_1$	0.7	0.28

Table 9 illustrates the influence of the parameters RT, RF, and RS on μ_{DEV} for the four lower bounds. The influence of these parameters on LB_0, LB_3^g, and $LBDI_1$ is nearly the same. Whereas the performance of these bounds increases with increasing RT and increasing RS ("$\searrow$"), we obtain larger deviations μ_{DEV} for increasing RF ("$\nearrow$"). Surprisingly, there is no significant influence of RT on the goodness of LB_0 ("—"). The results obtained for LBR are quite contrary. μ_{DEV} increases with increasing RT ("$\nearrow$"), because the decreasing degree of parallelity is not taken into account by LBR. Obviously, LBR performs best for high RF and small RS.

Table 9: Influence of RT, RF, and RS on μ_{DEV}

Parameter	LBR	LB_0	LB_3^g	$LBDI_1$
RT	$\nearrow$	—	$\searrow$	$\searrow$
RF	$\searrow$	$\nearrow$	$\nearrow$	$\nearrow$
RS	$\nearrow$	$\searrow$	$\searrow$	$\searrow$

After having compared $LBDI_1$ to other lower bounds, we now investigate the contribution of the components of $LBDI_1$ to the goodness of this bound. For this purpose, we have computed lower bound values for SOR96, which are solely based on one of the algorithms described in Sections 3 and 4. The results are listed in Tables 10 to 12. Again, the first value denotes μ_{DEV} and the value given in parenthesis corresponds to $\max_{DEV}$. It turns out, that the performance of the single algorithms heavily depends on the parameter setting (RT, RF, RS). That is why all algorithms should be combined in order to obtain a lower bound performing well for arbitrary instances. Nevertheless, Table 13 shows, that there are large differences in performance and computational effort between these algorithms.

Table 10: Variation of RT

Lower bound based on:	$RT = 0.25$		$RT = 0.50$		$RT = 0.65$	
$(con, 2, 2)$–p.c.	4.7	(43.9)	2.8	(24.8)	2.2	(23.1)
$(dis, 3, 3)$– and $(dis, 3, 2)$–p.c.	2.2	(31.7)	1.1	(25.0)	0.5	(9.5)
machine scheduling p.c.	9.3	(43.7)	9.0	(34.0)	8.7	(33.7)
workload l.b.c.	8.2	(42.5)	7.2	(36.9)	7.6	(34.6)
active chain l.b.c.	8.2	(42.0)	6.9	(35.9)	7.2	(39.4)
$LBDI_1$	1.4	(23.6)	0.4	(9.9)	0.1	(6.6)

Table 11: Variation of RF

Lower bound based on:	$RF = 0.50$		$RF = 0.75$		$RF = 1.00$	
$(con, 2, 2)$–p.c.	1.5	(23.1)	4.0	(43.9)	4.8	(40.0)
$(dis, 3, 3)$– and $(dis, 3, 2)$–p.c.	0.9	(25.0)	1.3	(31.7)	1.9	(26.7)
machine scheduling p.c.	7.4	(33.7)	9.7	(43.7)	10.2	(39.2)
workload l.b.c.	6.2	(33.8)	8.5	(42.5)	8.5	(38.2)
active chain l.b.c.	5.7	(39.4)	8.3	(42.0)	8.6	(37.0)
$LBDI_1$	0.2	(14.8)	0.7	(23.6)	1.3	(22.2)

Table 12: Variation of RS

Lower bound based on:	$RS = 0.00$		$RS = 0.25$		$RS = 0.50$	
$(con, 2, 2)$–p.c.	5.5	(43.9)	3.2	(28.2)	2.7	(23.2)
$(dis, 3, 3)$– and $(dis, 3, 2)$–p.c.	3.1	(31.7)	1.4	(25.0)	0.6	(17.6)
machine scheduling p.c.	17.1	(43.7)	11.7	(35.8)	4.2	(23.0)
workload l.b.c.	19.0	(42.5)	9.3	(34.6)	2.4	(19.7)
active chain l.b.c.	17.5	(42.0)	8.2	(39.4)	3.4	(23.2)
$LBDI_1$	2.5	(23.6)	0.5	(16.0)	0.2	(8.4)

Table 13: μ_{DEV} and μ_{CPU} for $LBDI_1$–components

Lower bound based on:	μ_{DEV}	μ_{CPU}
$(con, 2, 2)$–p.c.	3.3	<0.01
$(dis, 3, 3)$– and $(dis, 3, 2)$–p.c.	1.3	0.11
machine scheduling p.c.	9.0	0.09
workload l.b.c.	7.7	0.17
active chain l.b.c.	7.4	<0.01
$LBDI_1$	0.7	0.28

Since μ_{CPU} may be prohibitively large for the computation of $LBDI_1$ in every node of a branch–and–bound tree we propose a second variant ($LBDI_2$), which consists of the components $(con, 2, 2)$–preprocessing, machine scheduling preprocessing, and active chain feasibility check. For machine scheduling preprocessing, only the time window $W^{0,n+1}(\boldsymbol{S})$ is considered. As can be seen in Table 14, the average percentage deviation μ_{DEV} increases to 3.1% but the computation time can be drastically reduced.

Table 14: Variants

Lower bound	μ_{DEV}	μ_{CPU}
$LBDI_1$	0.7	0.28
$LBDI_2$	3.1	<0.01

Conclusions

We have presented two approaches for destructive improvement: preprocessing checks and lower bound checks. Two new polynomial lower bounds for the RCPSP/max, which are based on different combinations of these algorithms, have been proposed. The results of a preliminary computational performance analysis show that the new bounds provide encouraging results compared to lower bounds which have been used for the RCPSP/max so far.

We suggest the new lower bounds to be used within branch–and–bound algorithms for RCPSP/max and generalizations like the Multi–Mode RCPSP/max. Moreover, the relationship between lower bound efficiency within enumeration algorithms and control parameters like problem size, restrictiveness, resource factor, and resource strength should be investigated. In particular, during enumeration, expensive bounds might be used on higher levels whereas it seems to be reasonable to use fast algorithms from a certain critical level on. This level should be chosen depending on the aforementioned control parameters.

References

[1] Bartusch, M. (1983): Optimierung von Netzplänen mit Anordnungsbeziehungen bei knappen Betriebsmitteln, *Doctoral Dissertation*, University of Aachen, 56–61

[2] Bartusch, M., Möhring, R.H., and Radermacher, F.J. (1988): Scheduling Project Networks with Resource Constraints and Time Windows, *Annals of Operations Research 16*, 201–240

[3] Brucker, P., Hilbig, T., and Hurink, J. (1997): A Branch & Bound Algorithm for a Single–Machine Scheduling Problem with Positive and Negative Time–Lags, *Osnabrücker Schriften zur Mathematik*, *Reihe P*, *No. 179 (revised version)*, University of Osnabrück, to appear in: *Discrete Applied Mathematics*

[4] Brucker, P., Schoo, A., and Thiele, O. (1996): A Branch & Bound Algorithm for the Resource–Constrained Project Scheduling Problem, *Osnabrücker Schriften zur Mathematik*, *Reihe P*, *No. 178*, University of Osnabrück, to appear in: *European Journal of Operational Research*

[5] Carlier, J. (1982): The One–Machine Sequencing Problem, *European Journal of Operational Research 11*, 42–47

[6] Christofides, N., Alvares–Valdes, R., and Tamarit, J.M. (1987): Project Scheduling with Resource Constraints: A Branch and Bound Approach, *European Journal of Operational Research 29*, 262–273

[7] Demeulemeester, E. and Herroelen, W. (1992): A Branch and Bound Procedure for the Multiple Resource–Constrained Project Scheduling Problem, *Management Science 38*, 1803–1818

[8] De Reyck, B. (1995): On the Use of the Restrictiveness as a Measure of Complexity for Resource–Constrained Project Scheduling, *Onderzoeksrapport 9535*, University of Leuven, Belgium

[9] De Reyck, B. and Herroelen, W. (1996): A Branch–and–Bound Procedure for the Resource–Constrained Project Scheduling Problem with Generalized Precedence Relations, *Onderzoeksrapport 9613*, University of Leuven, Belgium

[10] Elmaghraby, S.E. (1977): *Activity Networks: Project Planning and Control by Network Models*, Wiley, New York

[11] Klein, R. and Scholl, A. (1997): Computing Lower Bounds by Destructive Improvement — An Application to Resource–Constrained Project Scheduling, *Schriften zur quantitativen Betriebswirtschaftslehre 4/97*, University of Darmstadt

[12] Kolisch, R. (1995): *Project Scheduling under Resource Constraints: Efficient Heuristics for Several Problem Classes*, Physica–Verlag, Heidelberg, 54–60

[13] Kolisch, R., Sprecher, A., and Drexl, A. (1995): Characterization and Generation of Resource–Constrained Project Scheduling Problems, *Management Science 41*, 1693–1703

[14] Lawler, E.L. (1976): *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York

[15] Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1994): An Exact Algorithm for Project Scheduling with Resource Constraints Based on a New Mathematical Formulation, *Technical Report 32*, Department of Mathematics, University of Bologna, Italy

[16] Pascoe, T.L. (1966): Allocation of Resources C.P.M., *Revue Francaise Recherche Operationelle 38*, 31–38

[17] Schwindt, C. (1996): Generation of Resource–Constrained Project Scheduling Problems with Minimal and Maximal Time Lags, *Report WIOR 489*, University of Karlsruhe

[18] Schwindt, C. (1997): Verfahren zur Lösung des ressourcenbeschränkten Projektdauerminimierungsproblems mit planungsabhängigen Zeitfenstern, *Doctoral Dissertation*, University of Karlsruhe