Drexl, Andreas; Kimms, Alf

Minimizing total weighted completion times subject to precedence constraints by dynamic programming

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 475

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 475

Minimizing Total Weighted Completion Times Subject to Precedence Constraints by Dynamic Programming

A. Drexl and A. Kimms

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 475

Minimizing Total Weighted Completion Times Subject to Precedence Constraints by Dynamic Programming

A. Drexl and A. Kimms

July 1998

Andreas Drexl and Alf Kimms
Lehrstuhl für Produktion und Logistik, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24118 Kiel, Germany
email: Drexl@bwl.uni-kiel.de
Kimms@bwl.uni-kiel.de
URL: http://www.wiso.uni-kiel.de/bwlinstitute/Prod
ftp://ftp.wiso.uni-kiel.de/pub/operations-research

Abstract

In this paper we present a polynomial time dynamic programming algorithm for solving a scheduling problem with a (total) weighted completion time objective function where the weights are activity– and time–dependent. We highlight application areas for this type of problem to underscore the relevance of it. A computational study proves that large instances with up to 120 activities can be solved.

Keywords: Total weighted completion time objective, precedence constraints, dynamic programming

1 Problem Definition

We consider a set $V = \{0, 1, \ldots, n, n+1\}$ of activities which are to be scheduled without preemption, i.e., once an activity $j \in V$ is started, it runs $p_j \in \mathbb{N}_0$ time units without interruption. A feasible schedule must take precedence constraints $E \subset V \times V$ among the activities into account. A feasible schedule thus assigns completion times $C_j \in \mathbb{N}_0$ to all activities such that $C_j \geq C_i + p_j$ holds for all $(i,j) \in E$. We assume that the activity–on–node network $G = (V, E)$ contains no cycles. We will subsequently use $\mathcal{P}_j = \{i \in V \mid (i,j) \in E\}$ to denote the set of immediate predecessors of activity j. Consequently, $\bar{\mathcal{P}}_j = \mathcal{P}_j \cup \bigcup_{i \in \mathcal{P}_j} \bar{\mathcal{P}}_i$ is the set of all predecessors of j. Without loss of generality, we assume that the activities 0 and $n+1$, respectively, are unique source and sink nodes in the network, and that the network contains no redundant arcs, i.e. $i \in \mathcal{P}_j$ implies $i \notin \bigcup_{h \in \mathcal{P}_j} \bar{\mathcal{P}}_h$. It is easy to compute earliest and latest finish times EC_j and LC_j, respectively, where $EC_0 = p_0$ and $LC_{n+1} = T$, respectively, are used as starting points for forward and backward recursions, and $T \in \mathbb{N}$ is a deadline for finishing the network. Of course, a feasible solution for the scheduling problem exists, if and only if $EC_{n+1} \leq T$ holds.

The objective is to find a feasible schedule which minimizes the sum of weighted completion times where $g_{jt} \in \mathbb{R}$ for $j \in V$ and $t \in \{EC_j, \ldots, LC_j\}$ are weights for finishing activity j at time t. In summary, we look for an optimum solution of a problem which can be couched formally as a binary program of the form

$$\min \sum_{j=0}^{n+1} \sum_{t=EC_j}^{LC_j} g_{jt} x_{jt} \tag{1}$$

subject to

$$\sum_{t=EC_j}^{LC_j} x_{jt} = 1 \qquad j \in V \tag{2}$$

$$\sum_{t=EC_j}^{LC_j} t \cdot x_{jt} - \sum_{t=EC_i}^{LC_i} t \cdot x_{it} \geq p_j \qquad (i,j) \in E \tag{3}$$

$$x_{jt} \in \{0,1\} \qquad j \in V; t \in \{EC_j, \ldots, LC_j\} \tag{4}$$

where $x_{jt} = 1$ indicates that activity j is finished at time t (otherwise, x_{jt} equals zero).

The remaining text is organized as follows: Section 2 describes some application areas to make clear that the problem just defined is indeed an important one. In Section 3, a dynamic programming algorithm is presented which runs in polynomial time. An illustrative example is given in Section 4. Computational results are provided in Section 5. Some final remarks in Section 6 finish the paper.

2 Application Areas

We first note that the problem stated above is not a generalization of the well–known single machine scheduling problem under a total weighted completion time objective (see, e.g., [1]), because activities may be processed in parallel. Actually, the problem can be seen as an $(n + 2)$ parallel machine scheduling problem, but the number of machines is not restrictive. However, there are important application areas where the above type of problem occurs. We give here two examples from the field of project scheduling (see, e.g., [2]).

2.1 Maximizing the Net Present Value

The first example is project scheduling without resource constraints where the objective is to maximize the so–called net present value. This problem is almost exactly the one above, the only difference is that the weights g_{jt} have a special structure. Note, every maximization problem can be transformed into a minimization problem by multiplying the objective function thru with -1. In this particular problem, a cash flow is associated with each activity. Once an activity j is completed, a real–valued amount of money c_j must be paid (cash outflow, $c_j < 0$) or is received (cash inflow, $c_j \geq 0$). Because having one dollar today is better than receiving one dollar in the future (the reverse is true for cash outflows), future payments are discounted to compute today's worth of future payments. Finance tells us how to do so, i.e.

$$g_{jt} = -c_j \cdot exp(-\alpha \cdot t) \tag{5}$$

is the current value of a payment c_j at time t in the future, where $\alpha \geq 0$ is a discounting factor (actually, $\alpha = \ln(1 + \beta)$ with β being an interest rate). Note, g_{jt} is monotonically decreasing (increasing) in time, if c_j is negative (non–negative). Thus, activities with a positive cash flow should be scheduled as early as feasible, while activities with a negative cash flow should be scheduled as late as possible.

This net present value project scheduling problem is an important and established subject. It has been around for about thirty years now. Russell [11] was, to the best of our knowledge, the first author who dealt with this problem. He formulated a non–linear model and described an approximation method for it. Later on, Grinold [8] transformed this model into a linear one and presented an exact solution procedure to solve the problem. Elmaghraby and Herroelen [7] presented another (optimal) procedure, but Sepil [12] gave a counterexample to show that the algorithm may not find the optimal solution. All these references assumed an activity–on–arc network. However, every activity–on–arc network can be converted into an activity–on–node network and vice versa (see, e.g., [6]), and hence these contributions are relevant in our context also. Recently, Demeulemeester et al. [5] have developed an exact algorithm for activity–on–node networks with a run–time complexity of $\mathcal{O}(n^3)$. They report that their algorithm is on the average about 2.5 times faster than Grinold's procedure.

2.2 Minimizing the Makespan of a Project Under Resource Constraints

Another application area for our problem is in the context of the resource constrained project scheduling problem $PS \mid prec \mid C_{max}$ (see [2]) where the objective is to minimize the makespan of the project network. In this problem, we have a set K of so–called renewable resources where $R_{kt} \in \mathbb{N}_0$ resource units are available for $k \in K$ in time period t. Processing an activity j requires $r_{jk} \in \mathbb{N}_0$ units of each resource k in every period in which it is processed. Using the same symbols as before, the problem can mathematically be modelled as follows:

$$\min \sum_{t=EC_{n+1}}^{T} t \cdot x_{(n+1)t} \tag{6}$$

subject to

$$\sum_{j=0}^{n+1} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} r_{jk} x_{j\tau} \leq R_{kt} \qquad k \in K; t \in \{1,\ldots,T\} \tag{7}$$

and (2), (3), and (4)

Often, it is essential to have good lower bounds which are used either to evaluate solution procedures or to be used as part of solution procedures, like for bounding in a branch–and–bound method, for instance. For the latter case, it is crucial to be able to compute lower bounds fast. Christofides et al. [4] suggest to compute lower bounds by means of a Lagrangean relaxation of the resource constraints combined with subgradient optimization. Let $\lambda_{kt} \in \mathbb{R}_0^+$ be the Lagrangean multipliers associated with restriction (7). The Lagrangean relaxation then looks as follows:

$$\min \sum_{t=EC_{n+1}}^{T} t \cdot x_{(n+1)t} + \sum_{k\in K} \sum_{t=1}^{T} \lambda_{kt} \left(\sum_{j=0}^{n+1} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} r_{jk} x_{j\tau} - R_{kt} \right) \tag{8}$$

subject to

(2), (3), and (4)

Christofides et al. solve this problem with a branch–and–bound procedure. We will now show that the model can be transformed into the form (1) thru (4) and can thus be solved with our polynomial time algorithm which is presented in the next section. With this in mind, we can derive the following:

$$\begin{aligned}
& \min \sum_{t=EC_{n+1}}^{T} t \cdot x_{(n+1)t} + \sum_{k\in K} \sum_{t=1}^{T} \lambda_{kt} \left(\sum_{j=0}^{n+1} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} r_{jk} x_{j\tau} - R_{kt} \right) \\
\Leftrightarrow \quad & \min \sum_{t=EC_{n+1}}^{T} t \cdot x_{(n+1)t} + \sum_{k\in K} \sum_{t=1}^{T} \sum_{j=0}^{n+1} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} \lambda_{kt} r_{jk} x_{j\tau} - \sum_{k\in K} \sum_{t=1}^{T} \lambda_{kt} R_{kt} \\
\Leftrightarrow \quad & \min \sum_{t=EC_{n+1}}^{T} t \cdot x_{(n+1)t} + \sum_{k\in K} \sum_{t=1}^{T} \sum_{j=0}^{n+1} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} \tilde{g}_{jkt} x_{j\tau} - \sum_{k\in K} \sum_{t=1}^{T} \lambda_{kt} R_{kt} \\
& \text{where } \tilde{g}_{jkt} = \lambda_{kt} r_{jk} \\
\Leftrightarrow \quad & \min \sum_{t=EC_{n+1}}^{T} t \cdot x_{(n+1)t} + \sum_{t=1}^{T} \sum_{j=0}^{n+1} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} \bar{g}_{jt} x_{j\tau} - \sum_{k\in K} \sum_{t=1}^{T} \lambda_{kt} R_{kt}
\end{aligned}$$

$$\text{where } \bar{g}_{jt} = \sum_{k \in K} \tilde{g}_{jkt}$$

$$\Leftrightarrow \quad \min \sum_{t=EC_{n+1}}^{T} t \cdot x_{(n+1)t} + \sum_{j=0}^{n+1} \sum_{t=EC_j}^{LC_j} \dot{g}_{jt} x_{jt} - \sum_{k \in K} \sum_{t=1}^{T} \lambda_{kt} R_{kt}$$

$$\text{where } \dot{g}_{jt} = \sum_{\tau=t-p_j+1}^{t} \bar{g}_{j\tau}$$

Since $\sum_{k \in K} \sum_{t=1}^{T} \lambda_{kt} R_{kt}$ is a constant for a given set of multipliers λ_{kt}, we can omit it for the purpose of optimization. Then, we define

$$g_{jt} = \begin{cases} \dot{g}_{jt} + t & \text{if } j = n+1 \\ \dot{g}_{jt} & \text{otherwise} \end{cases} \tag{9}$$

and get the objective function coefficients of the model (1) thru (4).

3 Dynamic Programming

The problem (1) thru (4) can be solved with dynamic programming. To make this clear, let $Z_j^*(t)$ denote the contribution of activity j and all its predecessors to the optimum objective function value when j is completed at time t. It can be computed using the following definition:

$$Z_j^*(t) = g_{jt} + \sum_{i \in \bar{\mathcal{P}}_j} g_{i,C_{ij}^*(t)} \tag{10}$$

In this definition, $C_{ij}^*(t)$ is the time at which activity $i \in \bar{\mathcal{P}}_j$ is finished best when activity j is completed at time t. Formally, $C_{ij}^*(t)$ can recursively be computed as follows:

$$C_{ij}^*(t) = \begin{cases} \arg \min\limits_{\tau=EC_i}^{\min\{t-p_j, LC_i\}} Z_i^*(\tau) & \text{if } i \in \mathcal{P}_j \\ \min\limits_{h \in \mathcal{P}_j} C_{ih}^* \left(\arg \min\limits_{\tau=EC_h}^{\min\{t-p_j, LC_h\}} Z_h^*(\tau) \right) & \text{otherwise} \end{cases} \tag{11}$$

The optimum objective function value is given by

$$\min_{\tau=EC_{n+1}}^{T} Z_{n+1}^*(\tau), \tag{12}$$

and the optimum schedule can be computed by evaluating

$$C_{i(n+1)}^* \left(\arg \min_{\tau=EC_{n+1}}^{T} Z_{n+1}^*(\tau) \right) \tag{13}$$

for each activity $i \in V \backslash \{n+1\}$ where

$$\arg \min_{\tau=EC_{n+1}}^{T} Z_{n+1}^*(\tau) \tag{14}$$

is the completion time of the sink activity $n+1$ in the optimum solution. The run-time complexity is $\mathcal{O}(n^2 T \max\{n, T\})$.

4 An Example

To illustrate the dynamic programming algorithm, we consider a small example with $n = 7$ activities. We make use of the example given in [5], i.e., the objective is to find a schedule which maximizes the net present value of cash flows. Figure 1 provides the relevant data. In addition to that, we use $T = 20$ and $\alpha = 0.01$. From these data, we can derive the earliest and latest finish times as well as the values g_{jt} according to definition (5). Table 1 summarizes this information.

Running the dynamic programming algorithm, the forward recursion (starting with activity 0) yields the values $Z_j^*(t)$ given in Table 2. The optimum schedule can then easily be found in a backward recursion (starting with activity 8). The optimum completion times are shown in Table 3 according to (13) and (14). The optimum objective function value is -39.53 according to (12), the maximum net present value therefore is 39.53.

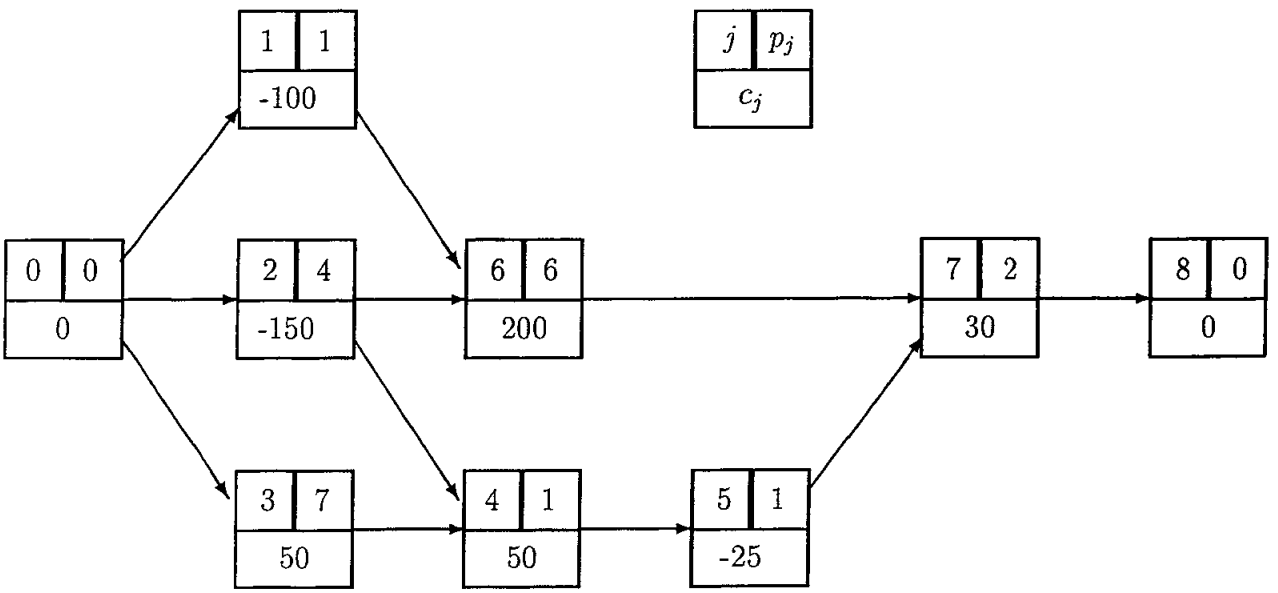

Figure 1: The Network of the Example

5 Computational Results

To test the dynamic programming algorithm, we have implemented it in GNU C. A Pentium II computer with 266 MHz running a LINUX operating system was used as a platform.

To give this computational study a better meaning than just evaluating the run-time of a polynomial time algorithm, we decided to solve instances from a particular application area. As mentioned above, the procedure in [5] is a much more efficient procedure to solve the special case of maximizing the net present value of a project. Hence, we reimplemented the ideas of Christofides et al. [4] heading for lower bounds for the problem $PS \mid prec \mid C_{max}$, using our dynamic programming algorithm instead of branch-and-bound this time. That is to say that given the multipliers λ_{kt} the dynamic programming algorithm solves the instance. Then, the multipliers are updated using subgradient optimization and the whole procedure starts again. Initially, we use $\lambda_{kt} = 0$.

t	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$
0	0.00	—	—	—	—	—	—	—	—
1	0.00	99.00	—	—	—	—	—	—	—
2	0.00	98.02	—	—	—	—	—	—	—
3	0.00	97.04	—	—	—	—	—	—	—
4	0.00	96.08	144.12	—	—	—	—	—	—
5	0.00	95.12	142.68	—	—	—	—	—	—
6	0.00	94.18	141.26	—	—	—	—	—	—
7	0.00	93.24	139.86	-46.62	—	—	—	—	—
8	0.00	92.31	138.47	-46.16	-46.16	—	—	—	—
9	—	91.39	137.09	-45.70	-45.70	22.85	—	—	—
10	—	90.48	135.73	-45.24	-45.24	22.62	-180.97	—	—
11	—	89.58	134.38	-44.79	-44.79	22.40	-179.17	—	—
12	—	88.69	133.04	-44.35	-44.35	22.17	-177.38	-26.61	0.00
13	—	—	—	-43.90	-43.90	21.95	-175.62	-26.34	0.00
14	—	—	—	-43.47	-43.47	21.73	-173.87	-26.08	0.00
15	—	—	—	-43.04	-43.04	21.52	-172.14	-25.82	0.00
16	—	—	—	-42.61	-42.61	21.30	-170.43	-25.56	0.00
17	—	—	—	—	-42.18	21.09	-168.73	-25.31	0.00
18	—	—	—	—	—	20.88	-167.05	-25.06	0.00
19	—	—	—	—	—	—	—	-24.81	0.00
20	—	—	—	—	—	—	—	-24.56	0.00

Table 1: The Weights g_{jt} of the Example

The Lagrangean multipliers λ_{kt} are updated due to

$$\lambda_{kt} = \max\left\{0, \lambda_{kt} + \delta\frac{(UB^* - LB^*)\Delta_{kt}}{\sum\limits_{\kappa\in K}\sum\limits_{\tau=1}^{T}\Delta_{\kappa\tau}^2}\right\} \tag{15}$$

where UB^* is the best known upper bound and LB^* is the best known lower bound of an instance (initially, LB is the critical path based lower bound). Note, LB^* requires updating while iterating. The value two is assigned to δ, initially, and δ is reduced to its half whenever the best known lower bound has not been improved after five iterations. For the schedule just determined, Δ_{kt} denotes the gap between how many units of resource k are requested and how many are available (R_{kt}) in period t where a positive value Δ_{kt} indicates excessive use of the resource. The iterative procedure is terminated after 100 iterations. The resulting LB^* value can be strengthened a bit by rounding it to $\lceil LB^* \rceil$.

As a test–bed, we used the instances provided in the project scheduling problem library PSPLIB [9]. This library also contains the best known upper bound UB^* for each instance which is not only used in the subgradient optimization, but also as a value for T in our studies. Currently, the library makes four different sizes of instances available. First, there are instances with $n = 30$ activities, but all of these were solved optimally before, so there is no need for computing lower bounds for these. Second, there are instances with $n = 60$ and $n = 90$ activities, respectively. We attacked these with our code, but we

t	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$
0	0.00	—	—	—	—	—	—	—	—
1	0.00	99.00	—	—	—	—	—	—	—
2	0.00	98.02	—	—	—	—	—	—	—
3	0.00	97.04	—	—	—	—	—	—	—
4	0.00	96.08	144.12	—	—	—	—	—	—
5	0.00	95.12	142.68	—	—	—	—	—	—
6	0.00	94.18	141.26	—	—	—	—	—	—
7	0.00	93.24	139.86	-46.62	—	—	—	—	—
8	0.00	92.31	138.47	-46.16	47.08	—	—	—	—
9	—	91.39	137.09	-45.70	46.15	69.93	—	—	—
10	—	90.48	135.73	-45.24	45.23	68.77	59.23	—	—
11	—	89.58	134.38	-44.79	44.31	67.62	58.64	—	—
12	—	88.69	133.04	-44.35	43.41	66.49	58.06	-37.07	-37.07
13	—	—	—	-43.90	42.51	65.36	57.48	-37.17	-37.17
14	—	—	—	-43.47	42.95	64.25	56.91	-37.26	-37.26
15	—	—	—	-43.04	43.38	64.03	56.34	-37.36	-37.36
16	—	—	—	-42.61	43.81	63.82	55.78	-37.45	-37.45
17	—	—	—	—	44.24	63.61	55.23	-37.98	-37.98
18	—	—	—	—	—	63.40	54.68	-38.50	-38.50
19	—	—	—	—	—	—	—	-39.02	-39.02
20	—	—	—	—	—	—	—	-39.53	-39.53

Table 2: The $Z_j^*(t)$ Values of the Example

j	0	1	2	3	4	5	6	7	8
C_j	0	12	12	7	13	18	18	20	20

Table 3: The Optimum Completion Times of the Example

were not able to improve the best known lower bounds due to [3]. However, it should be noted here that the code in [3] is restricted to instances where the resource availability is not time–dependent (i.e., $R_{k1} = \ldots = R_{kT}$ for all every resource k) while our algorithm handles the general case as well. Anyhow, we turned to study the instances with $n = 120$ activities. For these large size instances, no (good) lower bounds are available yet. The code in [3] is not able to solve them as reported to us in a private communication.

The set of instances with $n = 120$ activities consists of 600 cases. They were systematically generated using the instance generator ProGen [10] using three different values of the so–called network complexity $NC \in \{1.5, 1.8, 2.1\}$, four different values of the so–called resource factor $RF \in \{0.25, 0.5, 0.75, 1.0\}$, and five different values of the so–called resource strength $RS \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. The network complexity reflects the average number of immediate successors of an activity. The resource factor is a measure of the average number of resources requested per job. And, the resource strength describes the scarceness of the resource capacities. These parameters are known to have a big impact

on the hardness of a project instance. For each of the $3 \cdot 4 \cdot 5 = 60$ parameter combinations, 10 instances are available which gives the total of 600 instances.

When solving the instances, it turned out that instances with $RS = 0.1$ require excessive run–time and cannot be solved within reasonable time, say less than an hour. For the remaining 480 instances, the ideas of Christofides et al. (who used instances with 25 activities in their original study and restricted the number of iterations to 50) can be applied.

The first observation we made is that some instances in the test–bed have a best known upper bound which equals the critical path based lower bound EC_{n+1}. In other words, these instances have already been solved optimally and need no further consideration. Table 4 shows how many instances for each parameter combination have already been solved optimally.

$NC = 1.5$	$RS = 0.2$	$RS = 0.3$	$RS = 0.4$	$RS = 0.5$
$RF = 0.25$	0	8	7	10
$RF = 0.5$	0	0	8	10
$RF = 0.75$	0	0	2	10
$RF = 1.0$	0	0	3	7
$NC = 1.8$				
$RF = 0.25$	0	5	7	7
$RF = 0.5$	0	3	5	8
$RF = 0.75$	0	0	1	6
$RF = 1.0$	0	0	2	7
$NC = 2.1$				
$RF = 0.25$	1	2	5	9
$RF = 0.5$	0	0	2	8
$RF = 0.75$	0	0	1	6
$RF = 1.0$	0	0	0	3

Table 4: The Number of Instances with Known Optimal Solution

For the remaining instances, we computed lower bounds as described above. The average run–time (in CPU seconds) for solving one instance is given in Table 5. Note, these figures were yielded on the basis of 100 iterations per instance. That is, our dynamic programming algorithm was called a hundred times per instance. The most important observation is that run–time increases if the resource strength decreases. This is because a low RS value indicates that it is unlikely to find a feasible solution with a makespan close to the length of the critical path due to resource restrictions. Actually, the best known upper bound UB^* of an instance with a low RS value is far away from EC_{n+1}. This in turn means that the intervals $[EC_j, LC_j]$ are large which is the cause for a long run–time.

The average quality of the lower bound is given in Tables 6 and 7, respectively. The former provides the gap between the best known upper bound and the lower bound measured as

$$100 \cdot \frac{UB^* - LB^*}{UB^*} \tag{16}$$

$NC = 1.5$	$RS = 0.2$	$RS = 0.3$	$RS = 0.4$	$RS = 0.5$
$RF = 0.25$	48.66	37.02	40.26	—
$RF = 0.5$	96.32	58.62	42.22	—
$RF = 0.75$	139.44	75.30	52.52	—
$RF = 1.0$	145.64	90.27	59.71	43.38
$NC = 1.8$				
$RF = 0.25$	54.20	45.97	43.40	32.44
$RF = 0.5$	116.64	63.97	34.15	34.70
$RF = 0.75$	140.25	93.81	51.25	37.65
$RF = 1.0$	204.34	108.43	67.40	34.58
$NC = 2.1$				
$RF = 0.25$	67.32	52.50	34.49	39.35
$RF = 0.5$	133.54	73.71	48.21	47.22
$RF = 0.75$	215.71	107.31	67.34	43.67
$RF = 1.0$	257.74	131.15	78.04	47.09

Table 5: The Average Run–Time per Instance (100 Iterations)

while the latter one shows the improvement of the critical path based lower bound defined as

$$100 \cdot \frac{LB^* - EC_{n+1}}{EC_{n+1}}. \tag{17}$$

The results indicate that for instances with a low resource strength Lagrangean relaxation pays off in terms of the improved lower bounds. The results also indicate that a low resource strength makes instances hard to solve especially when combined with a high resource factor.

6 Conclusion

We have introduced a scheduling problem with a total weighted completion time objective where feasible solutions must take precedence constraints among the activities into account.

The problem was motivated by pointed to two applications in the field of project scheduling. The first application was project scheduling for maximizing the net present value of a project in the presence of unlimited resource availability. It was shown that this is a special case of the just introduced problem. The second application occurs in resource–constrained project scheduling where the objective is to minimize the makespan. It was shown that lower bounds, derived from a Lagrangean relaxation of the resource constraints, can be computed by employing the presented approach. This is remarkable, because existing literature proposed a branch–and–bound procedure to solve the problem.

Since it is well–known that many machine scheduling problems, such as job shop, flow shop, and open shop scheduling, are special cases of the resource constrained project scheduling problem, the presented method is of interest in these areas, too.

The solution procedure presented here, is a dynamic programming algorithm which solves the problem in $\mathcal{O}(n^2 T \max\{n, T\})$ time. A computational study revealed that large

$NC = 1.5$	$RS = 0.2$	$RS = 0.3$	$RS = 0.4$	$RS = 0.5$
$RF = 0.25$	10.40	2.41	5.06	—
$RF = 0.5$	20.98	10.30	6.06	—
$RF = 0.75$	27.28	14.18	8.66	—
$RF = 1.0$	26.99	16.58	10.90	4.22
$NC = 1.8$				
$RF = 0.25$	9.07	2.31	3.69	2.39
$RF = 0.5$	20.09	9.88	5.61	3.98
$RF = 0.75$	22.14	15.71	7.75	3.29
$RF = 1.0$	27.23	15.37	11.12	3.85
$NC = 2.1$				
$RF = 0.25$	9.44	5.09	3.39	1.72
$RF = 0.5$	20.36	12.11	6.54	3.94
$RF = 0.75$	26.03	14.67	9.42	3.87
$RF = 1.0$	27.18	17.59	10.53	7.28

Table 6: The Average Deviation from the Upper Bound

$NC = 1.5$	$RS = 0.2$	$RS = 0.3$	$RS = 0.4$	$RS = 0.5$
$RF = 0.25$	1.73	0.00	0.00	—
$RF = 0.5$	9.65	0.69	0.00	—
$RF = 0.75$	19.37	3.87	0.39	—
$RF = 1.0$	23.75	10.46	4.98	0.00
$NC = 1.8$				
$RF = 0.25$	1.62	0.00	0.00	0.44
$RF = 0.5$	11.94	1.43	2.25	0.00
$RF = 0.75$	15.71	6.19	1.84	0.00
$RF = 1.0$	27.08	10.48	4.19	0.00
$NC = 2.1$				
$RF = 0.25$	2.37	0.53	0.39	0.00
$RF = 0.5$	11.52	3.67	1.25	1.06
$RF = 0.75$	21.00	7.19	3.22	0.00
$RF = 1.0$	26.66	14.08	4.79	1.99

Table 7: The Average Improvement of the Critical Path Based Lower Bound

instances with $n = 120$ activities can be solved. Thus, if the intervals $[EC_j, LC_j]$ are not too large, the presented algorithm can be used as a submodule in the context of project scheduling in order to compute lower bounds for large instances. As a final remark it should be noted here that it is easy to extend the ideas to cases where minimal finish–start time lags, release times for the activities, and deadlines for the activities are given, too. Future work should be devoted to investigate further extensions.

References

[1] BRUCKER, P., (1998), Scheduling Algorithms, Berlin, Springer, 2nd edition

[2] BRUCKER, P., DREXL, A., MÖHRING, R., NEUMANN, K., PESCH, E., (1998), Resource–Constrained Project Scheduling: Notation, Classification, Models, and Methods, European Journal of Operational Research, to appear

[3] BRUCKER, P., KNUST, S., (1998), A Linear Programming and Constraint Propagation–Based Lower Bound for the RCPSP, Working Paper, University of Osnabrück

[4] CHRISTOFIDES, N., ALVAREZ–VALDES, R., TAMARIT, J.M., (1987), Project Scheduling with Resource Constraints: A Branch and Bound Approach, European Journal of Operational Research, Vol. 29, pp. 262–273

[5] DEMEULEMEESTER, E.L., HERROELEN, W.S., VAN DOMMELEN, P., (1995), An Optimal Recursive Search Procedure for the Deterministic Unconstrained Max–NPV Project Scheduling Problem, Working Paper, University of Leuven

[6] ELMAGHRABY, S.E., (1995), Activity Nets: A Guided Tour Through Some Recent Developments, European Journal of Operational Research, Vol. 82, pp. 383–408

[7] ELMAGHRABY, S.E., HERROELEN, W.S., (1990), The Scheduling of Activities to Maximize the Net Present Value of Projects, European Journal of Operational Research, Vol. 49, pp. 35–49

[8] GRINOLD, R..C., (1972), The Payment Scheduling Problem, Naval Research Logistics, Vol. 19, pp. 123–136

[9] KOLISCH, R., SPRECHER, A., (1997), PSPLIB — A Project Scheduling Problem Library, European Journal of Operational Research, Vol. 96, pp. 205–216

[10] KOLISCH, R., SPRECHER, A., DREXL, A., (1995), Characterization and Generation of a General Class of Resource–Constrained Project Scheduling Problems, Management Science, Vol. 41, pp. 1693–1703

[11] RUSSEL, A.H., (1970), Cash Flows in Networks, Management Science, Vol. 16, pp. 357–373

[12] SEPIL, C., (1994), Comment on Elmaghraby and Herroelen's "The Scheduling of Activities to Maximize the Net Present Value of Projects", European Journal of Operational Research, Vol. 73, pp. 185–187