Drexl, Andreas; Kimms, Alf

Optimization guided lower and upper bounds for the resource investment problem

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 481

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 481

Optimization Guided Lower and Upper Bounds for the Resource Investment Problem

A. Drexl and A. Kimms

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 481

Optimization Guided Lower and Upper Bounds for the Resource Investment Problem

A. Drexl and A. Kimms

September 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

The resource investment problem deals with the issue of providing resources to a project such that a given deadline can be met. The objective is to make the resources available in the cheapest possible way. For each resource, expenses depend on the maximum amount required during the course of the project. In this paper we develop two lower bounds for this $\mathcal{NP}$–hard problem using Lagrangean relaxation and column generation techniques, respectively. Both procedures are capable of yielding feasible solutions as well. Hence, we also have two optimization guided heuristics. A computational study consisting of a set of 3210 instances compares both approaches and allows insight into the performance. E.g., for the instances from Möhring's test set it turns out that in 56% of the cases the heuristic solution derived on the basis of Lagrangean relaxation is optimal. Using column generation, the gap between the lower bound and the optimum objective function value is below 5% in 50% of the cases, it is below 10% in 71% of the cases, and it is below 20% in all cases.

Keywords: resource investment problem, Lagrangean relaxation, column generation, lower bounds, upper bounds

1 Introduction

Completing a project means to find a schedule for the activities, which constitute the project, subject to some side constraints. Usually, precedence constraints have to be taken into account. In such a setting, a common objective is to minimize the makespan of the project, i.e. to answer the question of what is the shortest possible project duration. If the activities compete for scarce resources, it may happen that activities may not be processed in parallel. This resource constrained project scheduling problem is known to be $\mathcal{NP}$–hard [2].

Being motivated by a practical case of bridge construction where time is scarce (see [13]), Möhring [11] invented the resource investment problem. In contrast to the problem above, we face a tight deadline for the project completion here. To be able to finish the project in time, one must provide a sufficient amount of resources. The objective is to do so in the cheapest possible way. This problem is $\mathcal{NP}$–hard, too [11].

The remaining text is organized as follows: A formal statement of the resource investment problem is given in Section 2. In Section 3 lower bounds are obtained by means of Lagrangean relaxation in combination with subgradient optimization. In Section 4 column generation techniques are developed to compute lower bounds. We will see that both algorithms are capable of generating feasible solutions as well. A computational study in Section 5 examines the performance of both procedures. Conclusions and some hints for future work are given in Section 6.

2 Problem Setting

The resource investment problem is about completing a project consisting of a set of $V = \{0, 1, \ldots, n, n+1\}$ activities such that a given deadline $T \in \mathbb{N}$ is met in time. Precedence relations $E \subset V \times V$ among the activities have to be taken into account where $G = (V, E)$ forms an acyclic activity–on–node network. Without loss of generality

we assume that activity 0 is the unique source and that activity $n+1$ is the unique sink, respectively, in the network. For each activity j, a duration $p_j \in \mathbb{N}_0$ is given where once that activity j is started, it runs p_j time units without preemption. Earliest and latest completion times EC_j and LC_j, respectively, of the activities can be computed easily performing forward and backward recursions with $EC_0 = p_0$ and $LC_{n+1} = T$ as starting points. A set K of resources are necessary to fulfill the project. Each activity j requires $r_{jk} \in \mathbb{N}_0$ units of resource $k \in K$ per period in order to be processed. If activities are processed concurrently, the per period availability $R_k \in \mathbb{N}_0$ — a decision variable — must not fall below the request for any resource k. Since providing a resource k incurs a cost $c_k \in \mathbb{R}^+$ per unit, the aim is to find a schedule, i.e. completion times C_j, and resource availabilities R_k such that the project can be completed in the cheapest possible way. Formally, this problem can be described as follows:

$$\min \sum_{k \in K} c_k R_k \tag{1}$$

s.t.

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

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

$$R_k - \sum_{j \in V} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} r_{jk} x_{j\tau} \geq 0 \qquad k \in K; t \in \{1,\ldots,T\} \tag{4}$$

$$x_{jt} \in \{0,1\} \qquad j \in V; t \in \{EC_j,\ldots,LC_j\} \tag{5}$$

$$R_k \geq 0 \qquad k \in K \tag{6}$$

The variable x_{jt} is one, if activity j is completed in period t (zero, otherwise). It directly corresponds to the completion times C_j and allows to state the problem as a linear mixed–binary programming model. The objective (1) is to minimize the sum of the costs for providing the resources. (2) states that every activity must be completed where (3) makes sure that the activities are scheduled in accordance with the precedence constraints. (4) guarantees that the resource units provided are sufficient to implement the schedule. Note, a non–negativity condition (6) for R_k (instead of $R_k \in \mathbb{N}_0$) is sufficient, because all parameters r_{jk} are integer–valued and, thus, (4) in combination with (1) yields integer values for R_k, too, in every optimal solution.

Related to the resource investment problem is the resource constrained project scheduling problem. A feasible solution of that type of problem can be described with (2) thru (5) where the difference is that R_k are known parameters. The objective is to complete the project as soon as possible. Another class of problems which are somehow related are resource levelling problems. Again, the R_k values are parameters (∞ in the unconstrained case). The objective functions, however, depend on the resource requirements within the periods and, for instance, measure the sum of squares of the usage amounts per resource and per period. For a recent survey on these and other related project planning problems, we refer to [4].

Only little research has been undertaken to tackle the presented problem. Möhring [11] has developed an exact solution method for the resource investment problem and solved instances with $n = 16$ activities and four resources. Demeulemeester [5] has presented an

optimal algorithm which outperforms Möhring's procedure. Though the results on small instances (in the sense of a small number of activities and a small number of resources) are impressive, the shortcoming of his approach is that the first feasible solution found is known to be the optimum solution. Hence, it cannot be truncated to a heuristic to be applied to larger instances. Also, his approach is highly sensitive to variations in the number of resources. The results he presents indicate that instances with up to 20 activities must not have more than six resources. Recently, Nübel [12] considered the extension with maximal time lags between the activities and implemented a branch–and–bound procedure to find optimum solutions. He performed a computational study using instances with 10, 20, and 30 non–dummy activities combined with one, three, and five resources. The performance of his procedure highly depends on these parameters. While all instances with 10 activities can be solved optimally, there are instances in his test–bed with 20 and 30 activities, respectively, for which no optimum solution is known yet. For example, given five resources about 79% out of 270 instances with 20 activities and about 53% out of 270 instances with 30 activities can be solved optimally.

Lower bounds and/or heuristics have not been developed so far and are the subject of this paper. But, before we turn to study more sophisticated procedures, we give here simple bounds which turn out to be useful later on. First, we consider lower bounds for the R_k values. It is clear that the minimum level cannot fall below of what a single activity requires. Also, it is easy to compute the overall requirement of a resource k which, when distributed equally over the planning horizon, also defines a lower bound for the resource units to be provided. Formally,

$$\underline{R}_k = \max\left\{\max_{j\in V} r_{jk}, \left\lceil \frac{\sum_{j\in V} p_j r_{jk}}{T} \right\rceil\right\} \tag{7}$$

is a lower bound for what must be available of resource k. This defines a lower bound for the optimum objective function value as follows:

$$LB_0 = \sum_{k\in K} c_k \underline{R}_k \tag{8}$$

An upper bound on what should be provided can be derived from the situation in which all activities are executed concurrently, i.e.

$$\overline{R}_k = \sum_{j\in V} r_{jk}. \tag{9}$$

This would allow to compute an upper bound on the optimum objective function value, but we can do better. The earliest start schedule where all activities are completed at their earliest finish times is a feasible solution. The minimum resource units which allow to implement the earliest start schedule define an upper bound on the optimum objective function value:

$$UB_0 = \sum_{k\in K} c_k \max_{t=1}^{T} \sum_{\substack{j\in V \\ t\le EC_j\le t+p_j-1}} r_{jk} \tag{10}$$

3 Lagrangean Relaxation

In this section we consider model (1) thru (6) together with the two additional constraints

$$R_k \ge \underline{R}_k \qquad\qquad k \in K \tag{11}$$

$$R_k \leq \overline{R}_k \qquad k \in K \tag{12}$$

where $\underline{R}_k$ and $\overline{R}_k$, respectively, are lower and upper bounds for the R_k values as defined by (7) and (9). (11) and (12) are redundant in this model, but they are not, if we relax (4). We will consider a Lagrangean relaxation (see, e.g., [7]) with $\lambda_{kt} \in \mathbb{R}_0^+$ being the Lagrangean multipliers associated with (4). Given a set of multipliers, the relaxed problem reads as follows:

$$\min \sum_{k \in K} c_k R_k + \sum_{k \in K} \sum_{t=1}^{T} \lambda_{kt} \left(\sum_{j \in V} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} r_{jk} x_{j\tau} - R_k \right) \tag{13}$$

s.t.

(2), (3), (5), (11), and (12)

Rearranging the terms in the objective (13) yields

$$\min \sum_{k \in K} \left(c_k - \sum_{t=1}^{T} \lambda_{kt} \right) R_k + \sum_{j \in V} \sum_{t=EC_j}^{LC_j} w_{jt} x_{jt} \tag{14}$$

where

$$w_{jt} = \sum_{k \in K} \sum_{\tau=t-p_j+1}^{t} \lambda_{k\tau} r_{jk}. \tag{15}$$

This problem decomposes and can be optimized by solving two subproblems independently. The first subproblem is:

$$\min \sum_{k \in K} \left(c_k - \sum_{t=1}^{T} \lambda_{kt} \right) R_k \tag{16}$$

s.t.

(11) and (12)

The second subproblem is:

$$\min \sum_{j \in V} \sum_{t=EC_j}^{LC_j} w_{jt} x_{jt} \tag{17}$$

s.t.

(2), (3), and (5)

The sum of the optimum objective function values of these two subproblems gives the optimum objective function value (13).

In turn, subproblem one decomposes and can be solved efficiently by inspecting the expression $(c_k - \sum_{t=1}^{T} \lambda_{kt})$ for each $k \in K$. If this expression is negative, the optimum R_k value is $\overline{R}_k$. If the expression is non–negative, the optimum R_k value is $\underline{R}_k$.

Subproblem two is a problem of minimizing the total weighted completion times of the activities subject to precedence constraints. Drexl and Kimms [6] have presented a $\mathcal{O}(n^2 T \max\{n, T\})$ polynomial time dynamic programming algorithm and, therefore, subproblem two can also be solved efficiently. It is remarkable to note that any solution of subproblem two represents a feasible schedule for the problem (1) thru (6). Given such

a schedule with completion times C_j, the smallest possible R_k values which do not violate (4) can be derived in polynomial time:

$$R_k = \max_{t=1}^{T} \sum_{\substack{j \in V \\ t \le C_j \le t+p_j-1}} r_{jk} \tag{18}$$

Hence, for each optimum solution of subproblem two, we can derive a heuristic solution for the resource investment problem in polynomial time.

In summary, for a given set of Lagrangean multipliers λ_{kt}, the relaxed problem can be solved in polynomial time and a suboptimal solution for the resource investment problem can be computed in polynomial time as well.

What remains is to discuss the question of how to find appropriate values for λ_{kt}. We make use of subgradient optimization as presented in [8]. That is to say that starting with $\lambda_{kt} = 0$ we solve the relaxed problem and update the multipliers afterwards before starting a new iteration. Updating is done according to

$$\lambda_{kt} = \max\left\{0, \lambda_{kt} + \delta \frac{(UB^* - LB^*)\Delta_{kt}}{\sum_{\kappa \in K} \sum_{\tau=1}^{T} \Delta_{\kappa\tau}^2}\right\} \tag{19}$$

where UB^* is the best known upper bound and LB^* is the best known lower bound. We update these values after each iteration. Initially, we set $UB^* = \infty$ and $LB^* = -\infty$. The parameter δ is assigned the value two, initially, and it is reduced to its half whenever the lower bound cannot be improved after, say, five iterations. The values Δ_{kt} are computed using the R_k and x_{jt} values just determined for optimizing (16) and (17), respectively:

$$\Delta_{kt} = \sum_{j \in V} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} r_{jk} x_{j\tau} - R_k \tag{20}$$

The Lagrangean relaxation procedure runs until a certain number of iterations are performed. We will refer to the result as LB_1 and UB_1, respectively, denoting the best bounds found during the course of iterating.

4 Column Generation

The second lower bound that we will present is based on column generation techniques (see, e.g., [1, 15]). We do not apply column generation to the model (1) thru (6), but to the following model reformulation:

$$\min \sum_{k \in K} c_k R_k \tag{21}$$

s.t.

$$R_k - \sum_{s=1}^{S} \sum_{j \in V} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} r_{jk} x_{j\tau s} y_s \ge 0 \qquad k \in K; t = 1, \ldots, T \tag{22}$$

$$\sum_{s=1}^{S} y_s = 1 \tag{23}$$

$$y_s \in \{0,1\} \qquad s = 1, \ldots, S \tag{24}$$

$$R_k \ge \underline{R}_k \qquad k \in K \tag{25}$$

In this model, each column s which corresponds to a decision variable y_s represents a schedule. In contrast to what has been defined in former sections, x_{jts} is a 0–1–valued parameter now where the value one indicates that activity j is completed at time t in column s. The binary decision variable y_s models the selection of a schedule. Due to (23) exactly one schedule is selected. (22) makes sure that the resource units provided are sufficient to implement the selected schedule. S is the total number of schedules. The parameters $\underline{R}_k$ used in (25) are defined as given by (7) in our implementation.

Now, our aim is to solve the LP–relaxation of the model (21) thru (25), the so–called master problem, optimally which yields a lower bound. In other words, we replace (24) with

$$y_s \geq 0 \qquad\qquad s = 1, \ldots, S \tag{26}$$

and seek for an optimum solution. Note, the constraints $y_s \leq 1$ for $s = 1, \ldots, S$ are redundant and can thus be omitted, because (23) in combination with (26) guarantee that this restriction is fulfilled, too.

The problem with this model is that, if we want to consider all feasible schedules explicitly, S, the number of columns, is extremely large. This is where the idea of column generation comes in. The basic working principle is as follows: Starting with a small set of columns, we try to find out if any other column can exist which, when added, may reduce the objective function value. If such a column exists, we add it and reoptimize with an augmented set of columns. If we figure out that no such column can exist, we stop and the final objective function value is known to be a lower bound. Hopefully, the number of iterations, i.e. the number of columns to be generated, is small compared to the number of feasible schedules.

In our implementation, we start with a single column $s = 1$ which represents the earliest start schedule, i.e. $x_{jt1} = 1$, if $t = EC_j$, and $x_{jt1} = 0$, otherwise.

Suppose now that S schedules are considered explicitly (initially, $S = 1$). The question is, how can we find out that no additional schedule (column) exists which may reduce the objective function value. To give an answer to this question, we consider the dual of the LP–model (21)–(23), (25), and (26) with π_{kt}, μ, and ν_k being the dual variables associated with (22), (23), and (25), respectively:

$$\max \mu + \sum_{k \in K} \underline{R}_k \nu_k \tag{27}$$

s.t.

$$\nu_k + \sum_{t=1}^{T} \pi_{kt} \leq c_k \qquad\qquad k \in K \tag{28}$$

$$\mu - \sum_{j \in V} \sum_{k \in K} \sum_{t=1}^{T} \sum_{\tau = \max\{t, EC_j\}}^{\min\{t+p_j-1, LC_j\}} \pi_{kt} r_{jk} x_{j\tau s} \leq 0 \qquad\qquad s = 1, \ldots, S \tag{29}$$

$$\pi_{kt} \geq 0 \qquad\qquad k \in K; t = 1, \ldots, T \tag{30}$$

$$\mu \in \mathbb{R} \tag{31}$$

$$\nu_k \geq 0 \qquad\qquad k \in K \tag{32}$$

From duality theory we know that the optimum objective function values of the primal (21) and the dual (27) are equal. The question, if there is a column which, when added to the primal, may reduce the optimum objective function value is thus equivalent of

asking whether or not there exists a row which, when added to the dual, may reduce the optimum objective function value. More precisely, we seek for a row of the form (29) which cuts off the current optimum solution, i.e. we look for a schedule $S+1$ for which

$$\mu - \sum_{j \in V} \sum_{k \in K} \sum_{t=1}^{T} \sum_{\tau=\max\{t,EC_j\}}^{\min\{t+p_j-1,LC_j\}} \pi_{kt} r_{jk} x_{j\tau(S+1)} > 0 \tag{33}$$

holds. Note, π_{kt} and μ (and ν_k) are parameters now, and $x_{jt(S+1)}$ are the decision variables. By rearranging the terms, (33) can also be written as

$$\mu > \sum_{j \in V} \sum_{t=EC_j}^{LC_j} v_{jt} x_{jt(S+1)} \tag{34}$$

where

$$v_{jt} = \sum_{k \in K} \sum_{\tau=t-p_j+1}^{t} \pi_{k\tau} r_{jk}. \tag{35}$$

Hence, if, for a given set of weights v_{jt}, the minimal total weighted completion time subject to (2), (3), and (5) (the index $S+1$ in (34) can be ignored) is greater than or equal to μ, pricing out occurs and no further columns need to be added. In other words, our subproblem of finding a new column again reduces to be the problem of minimizing the total weighted completion times subject to precedence constraints which can be solved in polynomial time [6]. This is exactly the same type of problem as subproblem two of the Lagrangean relaxation procedure which was discussed in the previous section. It should be highlighted that generating columns this way guarantees that every schedule, together with (18), represents a feasible solution to the model (1) thru (6). Thus, our column generation procedures produces upper bounds for the resource investment problem as well.

In the sequel we will refer to the lower bound computed with column generation as LB_2. The best upper bound found during the course of generating columns will be denoted as UB_2.

5 Computational Study

To study the performance of the Lagrangean relaxation and the column generation algorithms, we implemented both methods in GNU C. The computer codes were executed on a Pentium II machine with 266 MHz running a LINUX operating system. For solving the master problems of the column generation procedure, we employed the callable library CPLEX.

5.1 ProGen Instances with 30 Non–Dummy Activities

As mentioned before, only optimal solution procedures for the resource investment problem have been presented so far making use of rather small test sets. Hence, we decided to adapt well established instances for the resource constrained project scheduling problem for our purposes. The project scheduling problem library PSPLIB [9] provides a collection of systematically generated instances. We used the 480 instances with $n = 30$

$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
≤ 20	77	58	88	115	139	153
≤ 50	153	113	124	171	203	233
≤ 100	214	128	145	187	228	258
≤ 500	424	311	210	232	266	310
Total	480	480	480	480	480	480

Table 1: Cumulative Frequency Distribution of the Number of Instances

non–dummy activities and four resources. The resource limits of these instances have no meaning in our context, so we simply ignored them. The so–called ProGen instances have different network complexities $NC \in \{1.5, 1.8, 2.1\}$ and different resource factors $RF \in \{0.25, 0.5, 0.75, 1.0\}$ both of which are problem parameters with a big impact on the performance of project scheduling procedures (see [10]). The network complexity reflects the average number of immediate successors of an activity, and the resource factor is a measure of the average number of resources requested per activity. For each parameter level combination of NC and RF, 40 instances are provided which gives the total of $3 \cdot 4 \cdot 40 = 480$ instances. To get a resource investment problem instance, we additionally need a time limit T. In our studies, we computed T on the basis of the length of the critical path in the network. Formally, we defined

$$T = \lfloor \theta \cdot EC_{n+1} \rfloor \tag{36}$$

where $\theta \in \{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$ was used. This gives a test bed consisting of $6 \cdot 480 = 2880$ instances. For each of these instances we generated real–valued objective function coefficients c_k randomly out of the interval $[0, 10]$ with uniform distribution.

In a first experiment, we tested how many columns are generated by the column generation procedure upon termination. Table 1 provides the cumulative frequency distribution of the number of instances with respect to the number of columns upon termination depending on θ. One can see that θ has an impact on the run–time performance of the column generation procedure, because the number of columns upon termination is positively correlated with the run–time upon termination. It seems that a value θ close to 1.2 causes longer run–times than other θ values. We decided to consider only those instances further which can be solved with column generation such that the number of columns does not grow over 500. Thus, 1753 instances were investigated in depth.

Table 2 shows how many of the 1753 instances correspond to different parameter level combinations of NC, RF, and θ. One can observe that especially the problem parameter RF has an impact on the run–time performance of column generation. High RF values make short run–times unlikely. Though not that dramatic, NC also affects the run–time performance. It seems that for low NC values column generation requires to generate more columns than for high NC values. Thus, the choice of adapted ProGen instances as done here for testing the performance of resource investment problem methods appears to be appropriate.

In order to get a bit more insight into the actual run–time performance measured in CPU–seconds, Tables 3 and 4 provide the average figures for the Lagrangean relaxation and the column generation procedure, respectively. Besides for instances with $RF = 0.25$,

	$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$NC = 1.5$	$RF = 0.25$	40	40	40	40	40	40
	$RF = 0.5$	40	32	18	24	27	33
	$RF = 0.75$	29	13	4	5	11	13
	$RF = 1.0$	25	11	2	3	5	7
$NC = 1.8$	$RF = 0.25$	40	40	40	40	40	40
	$RF = 0.5$	40	34	28	27	32	37
	$RF = 0.75$	30	17	6	12	16	23
	$RF = 1.0$	32	14	2	1	1	9
$NC = 2.1$	$RF = 0.25$	40	40	40	40	40	40
	$RF = 0.5$	40	34	22	33	37	40
	$RF = 0.75$	37	24	7	5	15	22
	$RF = 1.0$	31	12	1	2	2	6
	Total	424	311	210	232	266	310

Table 2: Number of Instances per Parameter Level Combination

the run–time for performing 100 iterations of the Lagrangean relaxation method is much shorter than the run–time of the column generation procedure upon termination. While the average run–time of the former method is positively correlated with θ, the average run–time of the latter algorithm has its peak with $\theta = 1.1$ or $\theta = 1.2$ (for $RF \geq 0.75$ this is not always true which is conjectured to be, because the number of evaluated instances is rather small; often it is less than 10). It seems that different levels of NC and RF do not affect the run–time of the Lagrangean relaxation approach, but that column generation is very sensitive towards changes of these values. As mentioned before, the impact of RF on the run–time behavior of column generation is much stronger than the impact of NC. It should be noted that the Lagrangean relaxation method could have solved all 2880 instances in the test–bed within reasonable time, but the average results are given for the aforementioned 1753 instances only to allow a comparison with the column generation results.

To see if Lagrangean relaxation and column generation pay off, we examined whether or not the simple lower bound LB_0 has been improved. Tables 5 and 6, respectively, show the average improvement of LB_0, where the improvement is measured in percentage by

$$\frac{LB_i - LB_0}{LB_0} \cdot 100 \qquad \text{for } i \in \{1, 2\}. \tag{37}$$

It turns out that the column generation procedure yields decidedly better lower bounds than Lagrangean relaxation. For $\theta = 1.0$ and $RF \geq 0.75$ it happens that the lower bounds obtained with Lagrangean relaxation are even worse than LB_0 on average although the information in (11) is used to guide the search for good lower bounds. Due to (25), the column generation lower bound cannot be worse than LB_0. Roughly speaking, the improvement of LB_0 declines if θ increases. For $\theta = 1.5$ neither Lagrangean relaxation nor column generation can improve LB_0; both methods compute LB_0 (neglecting slight improvements). This indicates that studying instances with $\theta \geq 1.5$ is not promising for the approaches presented here. From this result we learn for future work that those who develop exact methods should evaluate, if the optimum solution for instances of the

	$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$NC = 1.5$	$RF = 0.25$	2.37	3.11	3.97	4.87	5.81	6.82
	$RF = 0.5$	2.18	2.72	3.73	4.81	5.88	6.68
	$RF = 0.75$	2.15	2.60	4.86	6.42	7.35	8.74
	$RF = 1.0$	2.00	2.23	2.27	5.21	6.47	7.26
$NC = 1.8$	$RF = 0.25$	2.23	3.05	3.99	4.99	6.04	7.18
	$RF = 0.5$	2.24	3.08	4.11	5.38	6.58	7.58
	$RF = 0.75$	2.28	2.68	4.09	6.48	7.90	9.04
	$RF = 1.0$	2.19	2.55	2.96	3.39	6.99	8.61
$NC = 2.1$	$RF = 0.25$	2.48	3.48	4.66	5.88	7.21	8.64
	$RF = 0.5$	2.45	3.39	4.60	5.93	7.33	8.70
	$RF = 0.75$	2.39	3.14	4.17	5.82	8.71	9.88
	$RF = 1.0$	2.36	2.94	3.28	7.08	9.80	10.48

Table 3: Average Run–Time of the Lagrangean Relaxation Method

	$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$NC = 1.5$	$RF = 0.25$	0.82	1.48	0.98	0.87	0.73	0.81
	$RF = 0.5$	11.32	24.28	31.50	15.68	7.83	11.77
	$RF = 0.75$	20.30	34.15	28.68	57.05	63.91	21.64
	$RF = 1.0$	22.84	36.08	35.37	164.30	201.25	41.62
$NC = 1.8$	$RF = 0.25$	0.74	0.77	1.10	0.65	0.63	0.72
	$RF = 0.5$	6.30	24.51	32.80	17.70	10.41	6.79
	$RF = 0.75$	21.47	30.99	51.93	22.70	23.10	45.02
	$RF = 1.0$	24.59	32.95	42.17	47.93	77.23	195.23
$NC = 2.1$	$RF = 0.25$	0.61	1.03	0.89	0.60	0.67	0.78
	$RF = 0.5$	6.01	25.30	23.94	11.65	4.73	4.41
	$RF = 0.75$	13.37	39.61	42.40	87.89	56.53	30.51
	$RF = 1.0$	22.00	48.36	102.82	42.53	113.57	115.47

Table 4: Average Run–Time of the Column Generation Method

	$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$NC = 1.5$	$RF = 0.25$	9.78	2.38	0.18	0.02	0.00	0.00
	$RF = 0.5$	13.86	7.39	3.54	0.24	0.00	0.00
	$RF = 0.75$	3.23	6.06	2.42	1.51	0.00	0.00
	$RF = 1.0$	-13.80	2.93	0.00	0.00	0.00	0.00
$NC = 1.8$	$RF = 0.25$	13.00	3.77	0.47	0.00	0.00	0.00
	$RF = 0.5$	19.19	8.74	2.88	0.62	0.14	0.02
	$RF = 0.75$	-6.74	1.52	0.00	0.00	0.00	0.00
	$RF = 1.0$	-10.18	3.88	3.05	6.24	0.00	0.00
$NC = 2.1$	$RF = 0.25$	15.65	3.02	0.04	0.00	0.00	0.00
	$RF = 0.5$	20.45	9.54	1.06	0.02	0.00	0.00
	$RF = 0.75$	-1.08	3.04	2.08	1.12	0.00	0.00
	$RF = 1.0$	-18.92	1.97	0.00	0.00	0.00	0.00

Table 5: Average Improvement of LB_0 due to Lagrangean Relaxation

	$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$NC = 1.5$	$RF = 0.25$	13.63	3.27	0.20	0.02	0.00	0.00
	$RF = 0.5$	36.27	17.16	6.65	1.04	0.09	0.05
	$RF = 0.75$	30.62	18.31	6.95	1.95	0.00	0.00
	$RF = 1.0$	20.85	14.03	8.89	2.04	0.00	0.00
$NC = 1.8$	$RF = 0.25$	16.82	6.26	0.90	0.00	0.00	0.00
	$RF = 0.5$	42.11	17.98	5.09	1.20	0.17	0.13
	$RF = 0.75$	28.70	18.95	4.86	0.00	0.00	0.00
	$RF = 1.0$	24.18	17.28	14.01	10.68	0.00	0.00
$NC = 2.1$	$RF = 0.25$	22.96	4.53	0.12	0.00	0.00	0.00
	$RF = 0.5$	43.56	17.93	2.27	0.25	0.03	0.01
	$RF = 0.75$	42.51	21.71	13.71	6.63	0.03	0.00
	$RF = 1.0$	22.53	15.15	16.15	0.00	0.00	0.00

Table 6: Average Improvement of LB_0 due to Column Generation

	$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$NC = 1.5$	$RF = 0.25$	27.08	29.76	28.52	28.39	27.48	26.88
	$RF = 0.5$	22.06	25.84	29.45	33.85	33.89	35.98
	$RF = 0.75$	15.91	26.30	33.54	30.56	35.82	37.34
	$RF = 1.0$	17.85	20.90	29.10	28.10	31.27	35.47
$NC = 1.8$	$RF = 0.25$	22.06	26.24	26.62	24.97	23.95	21.39
	$RF = 0.5$	20.87	25.57	30.02	29.90	30.83	31.84
	$RF = 0.75$	13.47	15.19	22.09	33.46	30.67	32.84
	$RF = 1.0$	8.97	15.96	10.08	23.62	17.95	31.15
$NC = 2.1$	$RF = 0.25$	18.95	24.50	23.07	22.81	22.19	23.22
	$RF = 0.5$	19.03	24.78	25.17	28.92	29.33	28.31
	$RF = 0.75$	10.64	15.83	20.18	24.23	28.98	30.84
	$RF = 1.0$	7.85	17.38	9.56	15.84	31.88	21.41

Table 7: Average Improvement of UB_0 due to Lagrangean Relaxation

	$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$NC = 1.5$	$RF = 0.25$	20.17	23.42	18.91	16.57	16.33	15.96
	$RF = 0.5$	25.72	30.21	31.01	30.36	27.97	28.96
	$RF = 0.75$	26.75	33.94	41.53	30.64	35.64	33.12
	$RF = 1.0$	29.60	30.85	37.49	32.20	29.07	35.42
$NC = 1.8$	$RF = 0.25$	18.21	18.25	17.31	14.16	13.29	12.74
	$RF = 0.5$	22.17	27.85	29.32	25.59	24.87	25.26
	$RF = 0.75$	24.93	27.37	28.21	29.20	28.87	30.39
	$RF = 1.0$	22.30	24.48	17.30	25.66	31.41	39.90
$NC = 2.1$	$RF = 0.25$	15.63	18.07	13.53	12.37	11.72	11.30
	$RF = 0.5$	20.91	25.67	23.82	24.98	24.05	22.76
	$RF = 0.75$	19.45	25.19	25.60	26.54	26.73	25.58
	$RF = 1.0$	24.64	27.11	28.52	29.81	29.52	32.35

Table 8: Average Improvement of UB_0 due to Column Generation

	$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$NC = 1.5$	$RF = 0.25$	16.23	17.76	21.14	21.30	22.72	23.29
	$RF = 0.5$	33.72	35.72	36.57	36.48	39.35	39.03
	$RF = 0.75$	35.43	33.51	38.73	45.91	47.01	47.79
	$RF = 1.0$	40.35	29.08	29.18	38.57	41.88	46.54
$NC = 1.8$	$RF = 0.25$	15.60	17.83	19.81	21.58	22.79	25.03
	$RF = 0.5$	30.58	34.26	35.57	36.49	37.61	38.19
	$RF = 0.75$	42.35	37.46	42.64	43.02	45.45	45.88
	$RF = 1.0$	39.29	31.29	34.54	30.95	49.35	45.71
$NC = 2.1$	$RF = 0.25$	13.94	16.79	20.60	20.99	21.84	20.85
	$RF = 0.5$	30.28	33.57	36.22	37.30	37.65	39.14
	$RF = 0.75$	42.23	38.36	41.91	44.05	44.98	45.80
	$RF = 1.0$	46.01	34.31	38.63	43.72	47.90	50.27

Table 9: Average Gap between LB_1 and the Best Known Upper Bound

	$n = 30$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$NC = 1.5$	$RF = 0.25$	13.42	17.10	21.12	21.30	22.72	23.29
	$RF = 0.5$	21.03	30.00	34.77	35.98	39.30	39.00
	$RF = 0.75$	18.62	25.83	35.65	45.67	47.01	47.79
	$RF = 1.0$	17.84	21.55	22.84	37.21	41.88	46.54
$NC = 1.8$	$RF = 0.25$	12.80	16.05	19.52	21.58	22.79	25.03
	$RF = 0.5$	17.58	28.89	34.25	36.14	37.59	38.12
	$RF = 0.75$	20.80	26.92	39.83	43.02	45.45	45.88
	$RF = 1.0$	16.82	22.41	27.63	28.07	49.35	45.71
$NC = 2.1$	$RF = 0.25$	9.10	15.68	20.55	20.99	21.84	20.85
	$RF = 0.5$	17.07	28.55	35.49	37.15	37.63	39.13
	$RF = 0.75$	17.01	27.36	35.30	41.10	44.97	45.80
	$RF = 1.0$	18.74	25.95	28.72	43.72	47.90	50.27

Table 10: Average Gap between LB_2 and the Best Known Upper Bound

resource investment problem with $\theta \geq 1.5$ is close to LB_0 or if there still remains a gap between LB_0 and the optimum result.

As explained above, both, the Lagrangean relaxation method and the column generation algorithm, compute feasible solutions as a byproduct. Tables 7 and 8, respectively, reveal, if the simple upper bound UB_0 can be improved on average. Improvements are measured as

$$\frac{UB_0 - UB_i}{UB_0} \cdot 100 \qquad \text{for } i \in \{1,2\}. \tag{38}$$

Both procedures give very promising results and improve UB_0 decidedly. For the Lagrangean relaxation method, there is a tendency that improvements are positively correlated with increases of θ. This is what one would expect, for UB_0 being derived from the earliest start schedule. The earliest start schedule seems to be a poor solution, if θ is large, because periods beyond EC_{n+1} are kept idle which goes along with a high resource request in early periods. Only for $RF = 0.25$ this effect is not clear from the data. For the column generation method, this observation cannot be made. We conjecture that the reason for this is that for large θ values the number of instances for which few columns are generated upon termination is large compared to instances with low θ values (see Table 1). Hence, less feasible solutions are generated and the chance for finding a good feasible solution decreases. Similarly, since the run–time of the column generation procedure is positively correlated with RF (see Table 4), the upper bounds found with column generation are better than the upper bounds found with Lagrangean relaxation, if $RF \geq 0.5$, and are worse, if $RF = 0.25$.

Next, we are interested in the solution quality in terms of the gap between the lower bound and the best known upper bound of an instance. Tables 9 and 10 provide this information in terms of average results. The gap is measured as

$$\frac{\min\{UB_1, UB_2\} - LB_i}{\min\{UB_1, UB_2\}} \cdot 100 \qquad \text{for } i \in \{1,2\}. \tag{39}$$

For both methods there is a clear tendency that the gap increases if θ does (outliers are ignored). This indicates that instances with, say, $\theta = 1.5$ are much harder to solve than instances with $\theta = 1.0$. Note, though not investigated in our studies, we conjecture that if θ is large enough, instances become easier to solve again, because if $T \geq \sum_{j \in V} p_j$ then instances are trivial. RF also seems to have a big impact on the gap, though a tendency is not obvious which might be, because there are only a few instances with a high RF value (often less than 10; see Table 2). NC seems to have a minor effect only, if at all. In summary we have that column generation yields better lower bounds than Lagrangean relaxation combined with subgradient optimization.

5.2 Möhring's Instances

According to Tables 9 and 10 the gap between lower and upper bound does not indicate tight bounds and it is not clear whether this is due to poor lower bounds, due to poor upper bounds, or both. Hence, we need to investigate further what makes the results look that bad. To make a definite statement, one needs instances for which the optimum solutions are known.

For a bridge construction project with $n = 16$ non–dummy activities and four resources, Möhring [11] provides detailed data. He studies 15 instances which differ in

$n = 16$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
1/1/1/1	11.24	16.24	1.03	18.26	11.75	0.00
3/1/1/1	19.22	12.20	2.80	13.00	8.64	0.00
1/3/1/1	13.39	8.09	1.05	13.04	8.74	0.00
1/1/3/1	11.80	8.60	5.88	22.10	23.17	0.00
1/1/1/3	8.55	16.16	6.93	22.87	7.80	0.00
3/3/1/1	14.75	6.77	1.48	13.04	11.11	0.00
3/1/3/1	14.07	15.51	6.26	18.17	1.46	0.00
3/1/1/3	13.80	13.57	6.08	19.14	22.51	0.00
1/3/3/1	11.88	13.00	3.04	19.34	19.33	0.00
1/3/1/3	11.00	9.23	3.58	20.04	6.25	0.00
1/1/3/3	9.57	13.47	5.05	24.14	16.21	0.00
3/3/3/1	12.89	18.82	3.95	15.43	0.92	0.00
3/3/1/3	11.80	6.79	3.59	19.21	20.35	0.00
3/1/3/3	10.65	16.06	7.11	19.98	15.10	0.00
1/3/3/3	10.17	12.96	3.55	20.26	14.08	0.00

Table 11: Gap between LB_1 and the Optimum Solution

$n = 16$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
1/1/1/1	6.67	12.70	0.00	15.00	8.57	0.00
3/1/1/1	14.29	10.39	0.00	11.25	6.12	0.00
1/3/1/1	4.76	5.30	0.00	11.25	6.12	0.00
1/1/3/1	4.17	4.62	3.89	17.73	19.44	0.00
1/1/1/3	4.17	7.62	6.25	19.00	2.38	0.00
3/3/1/1	11.11	3.85	0.00	9.00	4.76	0.00
3/1/3/1	10.00	13.45	3.24	14.74	0.00	0.00
3/1/1/3	10.00	6.72	5.21	15.83	19.70	0.00
1/3/3/1	3.33	10.46	1.04	14.77	15.91	0.00
1/3/1/3	3.33	3.43	0.00	15.83	1.85	0.00
1/1/3/3	3.03	3.16	0.00	19.64	11.69	0.00
3/3/3/1	8.33	16.33	0.89	12.64	0.00	0.00
3/3/1/3	8.33	2.63	0.00	13.57	16.67	0.00
3/1/3/3	7.69	9.94	0.00	16.87	9.89	0.00
1/3/3/3	2.56	7.73	0.00	17.19	9.89	0.00

Table 12: Gap between LB_2 and the Optimum Solution

$n = 16$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
1/1/1/1	0.00	0.00	0.00	16.67	40.00	0.00
3/1/1/1	0.00	18.18	12.50	37.50	28.57	0.00
1/3/1/1	0.00	18.18	0.00	0.00	14.29	0.00
1/1/3/1	0.00	0.00	10.00	0.00	11.11	16.67
1/1/1/3	0.00	0.00	0.00	10.00	14.29	33.33
3/3/1/1	0.00	0.00	0.00	30.00	44.44	37.50
3/1/3/1	0.00	0.00	8.33	0.00	44.44	0.00
3/1/1/3	0.00	11.76	8.33	0.00	45.45	62.50
1/3/3/1	0.00	0.00	25.00	0.00	36.36	0.00
1/3/1/3	0.00	11.76	0.00	8.33	44.44	50.00
1/1/3/3	0.00	0.00	7.14	0.00	18.18	0.00
3/3/3/1	0.00	0.00	0.00	0.00	54.55	0.00
3/3/1/3	0.00	10.53	0.00	21.43	30.77	70.00
3/1/3/3	0.00	0.00	6.25	6.25	7.69	0.00
1/3/3/3	0.00	0.00	0.00	0.00	30.77	0.00

Table 13: Gap between UB_1 and the Optimum Solution

$n = 16$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
1/1/1/1	0.00	0.00	0.00	33.33	20.00	25.00
3/1/1/1	0.00	27.27	37.50	12.50	57.14	50.00
1/3/1/1	0.00	18.18	0.00	25.00	57.14	33.33
1/1/3/1	0.00	0.00	20.00	20.00	33.33	0.00
1/1/1/3	0.00	0.00	20.00	10.00	28.57	16.67
3/3/1/1	0.00	38.46	0.00	60.00	66.67	12.50
3/1/3/1	0.00	17.65	33.33	33.33	44.44	12.50
3/1/1/3	0.00	17.65	16.67	33.33	18.18	25.00
1/3/3/1	0.00	0.00	25.00	33.33	18.18	0.00
1/3/1/3	0.00	11.76	8.33	33.33	44.44	37.50
1/1/3/3	0.00	0.00	7.14	7.14	27.27	37.50
3/3/3/1	0.00	0.00	42.86	42.86	54.55	0.00
3/3/1/3	0.00	26.32	21.43	28.57	30.77	40.00
3/1/3/3	0.00	13.04	18.75	25.00	30.77	60.00
1/3/3/3	0.00	0.00	6.25	12.50	30.77	60.00

Table 14: Gap between UB_2 and the Optimum Solution

the cost coefficients c_k only. Actually, he uses all different cost constellations where $c_k \in \{1,3\}$. We will refer to these instances by means of a tuple $c_1/c_2/c_3/c_4$. Note, because $1/1/1/1$ is essentially the same instance as $3/3/3/3$, the test–bed consists of 15 and not of $2^4 = 16$ instances. Möhring also provides the optimum objective function values for different values of T. We studied instances with T as defined by (36) where, as before, $\theta \in \{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$. In total we thus have $6 \cdot 15 = 90$ instances. These instances are included in Möhring's study, too.

To run the experiment, we used exactly the same method parameters as in the previous subsection. Tables 11 and 12 give the outcome of this study in terms of the deviation of the lower bound from the optimum objective function value OPT measured as

$$\frac{OPT - LB_i}{OPT} \cdot 100 \qquad \text{for } i \in \{1,2\}. \tag{40}$$

One can observe that both procedures are quite sensitive to variations of θ. For almost all cases with $\theta = 1.3$ the lower bounds are loose. It turns out that both algorithms find the optimum objective function value for all instances with $\theta = 1.5$. For $\theta = 1.2$ the lower bounds are tight, too. About all figures are below a 20% deviation from the optimum solution. For column generation, which gives tighter lower bounds than Lagrangean relaxation, the gap is below 10% in about 71% of the cases, and it is below 5% in about 50% of the cases. Variations of the cost coefficients also have an impact on the quality of the bound, and especially for $\theta \leq 1.1$ and $\theta = 1.4$ the sensitivity to these parameters is high for both algorithms.

Tables 13 and 14 provide information on the gap between the upper bound and the optimum objective function value measured as

$$\frac{UB_i - OPT}{OPT} \cdot 100 \qquad \text{for } i \in \{1,2\}. \tag{41}$$

Again, variations of θ and variations of c_k affect the outcome without a clear tendency. For $\theta = 1.0$, both algorithms find feasible solutions which are optimal in all cases. For $\theta \geq 1.1$, the Lagrangean relaxation procedure computes optimum solutions fairly often, but $\theta = 1.4$ is a critical value, because the gaps are large. The upper bounds derived by column generation are poor in almost all cases with $\theta \geq 1.1$. Overall, about 56% of the instances can be solved optimally.

5.3 ProGen Instances with 20 Non–Dummy Activities

As reported above, all (exact) solution procedures proposed so far are highly sensitive to the number of resources. Hence, we want to examine the impact of the number of resources on the run–time performance of our procedures as well. To do so, we used the ProGen software [10] to generate instances with 20 non–dummy activities and $|K| \in \{2,4,6,8\}$ resources. Again, we defined T according to (36) with $\theta \in \{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$. For each parameter level combination of $|K|$ and T we generated 10 instances. In total we thus came up with 240 instances.

The real–valued cost coefficients c_k were randomly drawn out of $[0, 10]$ with uniform distribution. The input for the ProGen software (see [9] for detailed explanations) was chosen to be as follows: Activity durations p_j are integer values out of $[1, 10]$. The project networks have three (non–dummy) start activities and three (non–dummy) finish

$n = 20$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$\|K\| = 2$	0.81	1.10	1.44	1.82	2.17	2.59
$\|K\| = 4$	1.31	1.71	2.14	2.61	3.09	3.51
$\|K\| = 6$	1.65	2.05	2.54	3.03	3.53	4.10
$\|K\| = 8$	1.90	2.37	2.90	3.43	3.97	4.59

Table 15: Average Run–Time of Lagrangean Relaxation

$n = 20$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$\|K\| = 2$	4.01	12.11	10.25	5.09	7.93	3.21
$\|K\| = 4$	12.26	48.93	188.34	338.06	439.95	281.70
$\|K\| = 6$	41.47	134.31	598.46	782.43	1737.41	1651.09
$\|K\| = 8$	29.58	92.15	418.97	704.40	1570.45	3516.03

Table 16: Average Run–Time of Column Generation

activites. The maximal number of predecessors / successors is three and $NC = 1.5$ is the network complexity. The resource demands r_{jk} are integer–valued out of $[1, 10]$. The resource strength is 0.2 and $RF = 1.0$ is the resource factor. All other values are chosen as in [9].

Tables 15 and 16 provide the average run–time (in CPU–seconds) upon termination of the Lagrangean relaxation (performing 100 iterations) and the column generation procedure, respectively. Note, all instances were solved with column generation until pricing out occurs and computation has not been stopped before this criterion has been met. Just one single instance with $|K| = 4$ and $\theta = 1.5$ could not have been solved on our computer and we stopped the computation on this instances after 2500 columns were generated.

We observe that the number of resources affects both, Lagrangean relaxation as well as column generation. For both algorithms, the run–time is correlated with the number of resources. The effect on Lagrangean relaxation, however, is less than linear. This is reasonable, because only the updating of the Lagrangean multipliers and the solution of subproblem one is affected. The computational effort grows linear in both cases, and, therefore, the overall effect is less than linear.

The impact on column generation is stronger. For $|K| \in \{2, 4, 6\}$, run–time grows exponentially as the number of resources inclines. For $|K| = 8$, however, we observe that the computational effort is decidedly less than for $|K| = 6$ in all but one case. This is remarkable, because if the number of resources grows, the number of rows in the master problem grows, too, due to (22) and (25). Hence, one would expect a monotonical growth of the run–time (see, e.g., [3, 14]). The reason for declining run–time is that the number of columns being generated declines (see Table 17) and, thus, the number of master problems to be solved declines, too.

$n = 20$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
$\|K\| = 2$	142.40	268.30	215.90	125.40	138.50	69.90
$\|K\| = 4$	215.70	417.10	717.60	898.10	952.20	708.78
$\|K\| = 6$	289.80	479.60	892.40	1015.60	1312.00	1153.70
$\|K\| = 8$	212.50	401.40	766.50	881.90	1161.10	1496.60

Table 17: Average Number of Columns Upon Termination

6 Conclusions

In this contribution we tackled the resource investment problem. Up to now, only exact algorithms have been proposed and, therefore, there is a need for heuristics and for lower bounds to attack other than very small instances. A first step in this direction has now been made.

We presented two algorithms in order to obtain lower bounds. The first one is based on Lagrangean relaxation combined with subgradient optimization. It turned out that given a set of Lagrangean multipliers, the remaining subproblem decomposes and can be solved in polynomial time. Thus, the Lagrangean relaxation approach works fast. The second algorithm employs column generation techniques. Again, the subproblem of generating a column can be solved in polynomial time, too. However, since the master problem is a linear programming problem, the run–time upon termination highly depends on the number of columns to be generated.

In a computational study with 2880 instances consisting of 30 activities and four resources, the following results were obtained: 1753 of the instances can be solved by column generation after generating 500 columns at most. Many of them need less than 50 columns only. However, column generation is the slower technique compared to Lagrangean relaxation which solves each instance within seconds. Nevertheless, the column generation lower bounds are much tighter than those obtained with Lagrangean relaxation. A simple analytic lower bound can be improved decidedly.

Remarkable to note is that both approaches lead to feasible solutions as a byproduct and thus give upper bounds as well. The gap between lower and upper bound mainly depends on the planning horizon and on the resource factor. The network complexity seems to be of minor importance in this context. The upper bound defined by the earliest start schedule can be improved decidedly.

A further experiment with 90 instances from a bridge construction project with 16 non–dummy activities and four resources reveals that the lower bounds are below a 20% deviation from the optimum result in almost all cases; often it is below a 10% deviation especially when column generation is used (71% of the instances have a gap of less than or equal to 10%, and 50% of the instances have a gap less than 5%). In about 56% of the cases the optimization guided heuristics find an optimum solution. We conjecture that the gaps between lower and upper bounds observed for larger instances are mainly due to poor upper bounds.

In another experiment with 240 instances consisting of 20 non–dummy activities each, the impact of the number of resources on the run–time performance is studied. Instances with two, four, six, and eight resources are solved. Lagrangean relaxation is affected only

slightly. For column generation, the run–time is positively correlated with the number of resources for up to six resources. But, it turned out that for instances with eight resources the run–time is decidedly lower than for instances with six resources. Hence, we can conclude that more resources do not necessarily lead to higher computational effort as it is the case with existing (exact) solution procedures.

This work can serve as a starting point for developing heuristics in the future. A more sophisticated strategy could use the presented Lagrangean relaxation technique for generating several feasible solutions fast. Then, some kind of improvement scheme may take over in order to find improved upper bounds. The column generation method should be refined in order to speed up convergence. Also, the presented column generation procedure could be used as the core of a branch–and–price method seeking for optimal solutions and be compared against existing codes. The advantage of such a strategy would be that it can be truncated to yield a heuristic.

Finally, it should be noted that minimal time lags between the activities, release dates of the activities, and deadlines for the activities are easy to incorporate into the model as well as into the algorithms.

References

[1] BARNHART, C., JOHNSON, E.L., NEMHAUSER, G.L., SAVELSBERGH, M.W.P., VANCE, P.H., (1998), Branch–and–Price: Column Generation for Huge Integer Programs, Operations Research, Vol. 46, pp. 316–329

[2] BŁAŻEWICZ, J., LENSTRA, J.K., RINNOOY KAN, A.H.G., (1983), Scheduling Subject to Resource Constraints: Classification and Complexity, Discrete Applied Mathematics, Vol. 5, pp. 11–24

[3] BORGWARDT, K.H., (1987), The Simplex–Method: A Probabilistic Analysis, Berlin, Springer

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

[5] DEMEULEMEESTER, E., (1995), Minimizing Resource Availability Costs in Time–Limited Project Networks, Management Science, Vol. 41, pp. 1590–1598

[6] DREXL, A., KIMMS, A., (1998), Minimizing Total Weighted Completion Times Subject to Precedence Constraints by Dynamic Programming, Working Paper 475, University of Kiel

[7] FISHER, M.L., (1981), The Lagrangean Relaxation Method for Solving Integer Programming Problems, Management Science, Vol. 27, pp. 1–18

[8] HELD, M., WOLFE, P., CROWDER, H.P., (1974), Validation of Subgradient Optimization, Mathematical Programming, Vol. 6, pp. 62–88

[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] MÖHRING, R.H., (1984), Minimizing Costs of Resource Requirements in Project Networks Subject to a Fixed Completion Time, Operations Research, Vol. 32, pp. 89–120

[12] NÜBEL, H., (1998), A Branch–and–Bound Procedure for the Resource Investment Problem with Generalized Precedence Constraints, Working Paper 516, University of Karlsruhe

[13] RADERMACHER, F.J., (1985/6), Scheduling of Project Networks, Annals of Operations Research, Vol. 4, pp. 227–252

[14] SHAMIR, R., (1987), The Efficiency of the Simplex–Method: A Survey, Management Science, Vol. 33, pp. 301–334

[15] SOUMIS, F., (1997), Decomposition and Column Generation, in: Dell'Amico, M., Maffioli, F., Martello, S., (eds.), Annotated Bibliographies in Combinatorial Optimization, New York, Wiley, pp. 115–126