A branch-and-check algorithm for minimizing the sum of the weights of the late jobs on a single machine with release dates

Ruslan Sadykov*

Abstract

In this paper we consider the scheduling problem of minimizing the sum of the weights of the late jobs on a single machine ($1 \mid r_j \mid \sum w_j U_j$). A branch-and-check algorithm is proposed, where a relaxed integer programming formulation is solved by branch-and-bound and infeasible solutions are cut off using infeasibility cuts. We suggest two ways to generate cuts. First we show how the algorithm by Carlier [7] can be modified to produce tightened "no-good" cuts. We then demonstrate how to create cuts by using constraint propagation. The branch-and-check algorithm proposed is implemented in the *Mosel* modelling and optimization language. Computational experiments show that our algorithm outperforms the exact approach of Péridy at al. [22], which, to our knowledge, is the best reported in the literature.

*CORE, Université Catholique de Louvain, Belgium. E-mail: sadykov@core.ucl.ac.be.

This work is supported in part by the ADONET (Algorithmic Discrete Optimization Network), contract No 504438 of the European Community

1 Introduction

In this paper we study the following scheduling problem. A set of jobs $N = \{1, \ldots, n\}$ has to be processed on a single machine. Only one job can be processed at a time and preemptions are not allowed. Each job $j \in N$ has a release date r_j, a processing time p_j, a due date d_j and a weight w_j. For a given schedule π, let $c_j(\pi)$ denote the completion time of j. If $c_j(\pi) \leq d_j$ then job j is *on time* in schedule π. Otherwise j is *late*. Let $\Pi(N)$ denote the set of all schedules of the jobs from set N. The objective is to find a schedule $\pi \in \Pi(N)$ that minimizes the sum of the weights of the late jobs. The problem is denoted $1 \mid r_j \mid \sum w_j U_j$ in the standard scheduling notation.

The problem is $\mathcal{NP}$-hard in the strong sense even when all weights are the same ($1 \mid r_j \mid \sum U_j$). This was shown by Lenstra et al. [20]. The special case without release dates ($1 \mid\mid \sum w_j U_j$) is $\mathcal{NP}$-hard in the ordinary sense. A pseudopolynomial algorithm for this case has been proposed by Lawler and Moore [18]. Later Lawler et al. [19] developed a pseudopolynomial algorithm for the case of a fixed number of identical machines ($Pm \mid\mid \sum w_j U_j$).

Some special cases can be solved in polynomial time. Moore [21] showed that the problem without release dates and identical weights ($1 \mid\mid \sum U_j$) can be solved in $O(n \log n)$. Kise et al. [15] showed that the problem is polynomial when the weights are identical, and release and due dates are ordered in the same way ($r_i < r_j \Rightarrow d_i \leq d_j$). An $O(n \log n)$ algorithm for this case was given by Lawler in [17]. Another polynomial algorithm by Lawler [16] solves the weighted case without release times when processing times and due dates are ordered in the opposite way.

For the $1 \mid r_j \mid \sum U_j$ problem Baptiste et al. [3] provided an exact algorithm that solved 99% of a set of test instances with 100 jobs and 80% of a set of instances with 160 jobs within one hour.

Recently several approaches for solving the general problem with release dates and unequal weights $1 \mid r_j \mid \sum w_j U_j$ have appeared in the literature. Baptiste et al. [1] proposed a constraint programming approach, which can also be applied when there are several identical parallel machines ($P \mid r_j \mid \sum w_j U_j$). A method based on a mixed-integer linear programming formulation was suggested by Dauzère-Pérès and Sevaux [9]. They used Lagrangean relaxation to solve the problem. Another exact approach based on a time-indexed formulation and Lagrangean relaxation was proposed by Péridy, Pinson and Rivreau [22]. This method is the most efficient algorithm available to the best of our knowledge. Their algorithm was able to solve 84.4% of the instances in a set of 100-job test instances within one hour. To tackle larger instances, some genetic algorithms were proposed by Sevaux and Dauzère-Pérès [26].

The main contribution of this work if another approach for solving the general problem. A branch-and-check algorithm is proposed, where a relaxed integer programming formulation is solved by branch-and-bound. Infeasible solutions are cut off using infeasibility cuts. We emphasis two main components of the algorithm. We first discuss how to make a suitable relaxation of the standard formulation of the problem. We then consider in detail the question

of generating strong infeasibility cuts. Two algorithms for generating two types of cuts are presented. The performance of the branch-and-check algorithm proposed here improves on the performance of the algorithm suggested by Péridy et al. The computational results demonstrate the importance of using strong cuts. An extended abstract of this work appeared in [23].

The paper is organized as follows. In Section 2 we describe the branch-and-check method in general and show how it can be used to solve the problem $1 \mid r_j \mid \sum w_j U_j$. In Sections 3 and 4 we present two approaches for generating strong infeasibility cuts. An algorithm for constructing "no-good" cuts based on "small" subsets of jobs is suggested in Section 3. Section 4 presents an approach for generating another class of cuts. This approach is based on the "edge-finding" technique from constraint programming [2]. We test several variants of the branch-and-check algorithm computationally in Section 5. Section 6 contains our conclusions and a discussion of potential future research.

2 The hybrid Branch-and-Check method

Recently several hybrid methods have been introduced in the literature. These methods solve a problem by using different techniques simultaneously. A standard approach is to decompose the problem into subproblems, where some subproblems are solved using mixed integer programming (MIP) while others are solved using constraint programming (CP). A survey on hybrid methods can be found in [13]. One possible method is the logic-based Benders decomposition. We now give a brief description of this method. For details see [10, 14].

Suppose we are given an optimization problem with two sets of variables x and y, where only the x variables appear in the objective function:

$$\begin{aligned} \min \quad & f(x) && (1)\\ s.t. \quad & x \in X, && (2)\\ & (x, y) \in Y. && (3) \end{aligned}$$

First, the problem obtained by relaxing the constraints (3) is solved. This gives an optimal solution $\bar{x}$ to the master problem (1)-(2). It is then checked if $\bar{x}$ is feasible with respect to the constraints (3), i.e., the following feasibility subproblem is solved:

$$\begin{aligned} \text{find} \quad & y && (4)\\ s.t. \quad & (\bar{x}, y) \in Y && (5) \end{aligned}$$

If the subproblem (4)-(5) has a solution $\bar{y}$, then an optimal solution $(\bar{x}, \bar{y})$ for the problem (1)-(3) is available. Otherwise a cut $b(x) \leq b_0$ is generated that cuts off $\bar{x}$ and is valid for all feasible solutions. The cut is then added to the master problem. This process is repeated until either the subproblem is feasible, or the master problem is infeasible. The latter case implies that the problem (1)-(3) is infeasible.

A disadvantage of this approach is that no general procedure for generating infeasibility cuts is known which guarantees the convergence. It is therefore necessary to develop such a procedure for every problem class. The classical Benders [4] and the generalized Benders decomposition [11] are special cases of this decomposition approach. In both these cases the subproblems are solved by linear programming.

Recently Jain and Grossmann [12] proposed algorithms based on the logic-based Benders decomposition approach to solve a class of mixed 0-1 programming problems. The master problem was solved as a MIP and the subproblems were solved with CP techniques. "No-good" cuts were used to cut off infeasible integer solutions. One of the algorithms was tested on the multi-machine assignment scheduling problem (MMASP), which is a multi-machine variant of the $1 \mid r_j \mid \sum w_j U_j$ problem, and promising results were reported. Later Bockmayr and Pisaruk [6] implemented a more sophisticated algorithm that only solves the IP master problem once with branch-and-bound. Every integer solution produced was checked with CP techniques. In their approach, infeasibility cuts are global, i.e. they are valid throughout the search tree. This approach was named "branch-and-check" by Thorsteinsson [27]. There are now modelling languages available, such as Mosel [8], that allows algorithms based on the branch-and-check approach to be implemented rapidly. A branch-and-check algorithm for the MMASP was tested by Bockmayr and Pisaruk. Their results significantly improved on the results of Jain and Grossmann. Also, it was pointed out in [6, 27] that adding a relaxation of the constraints (3) can play a key role in the efficiency of the algorithm. Such a relaxation was proposed by Sadykov and Wolsey [24] for the disjunctive scheduling constraint. Using this relaxation in a branch-and-check algorithm improved on the results obtained by Bockmayr and Pisaruk on the MMASP.

We now give a compact formulation of the $1 \mid r_j \mid \sum w_j U_j$ problem. Let the binary variable x_j, $j \in N$, take the value 1 if job j is on time, and the value 0 otherwise. Then the problem can be formulated as follows.

$$\min \quad \sum_{j \in N} w_j (1 - x_j) \tag{6}$$

$$s.t \quad \texttt{disjunctive}(x), \tag{7}$$

$$x_j \in \{0,1\}^n \quad \forall j \in N. \tag{8}$$

The vector x satisfies the scheduling constraint (7) if and only if the set of jobs $J = \{j \in N : x_j = 1\}$ can be processed on the machine without violating the release dates and the due dates.

In this paper we solve the problem (6)-(8) by a branch-and-check algorithm. The constraint `disjunctive` is not included in the master problem, since its formulation in linear form requires a larger number of extra variables and constraints. Instead we use the fact that there exist efficient methods to check if the constraint `disjunctive` is satisfied. These techniques are either from CP or specialized combinatorial algorithms. The constraint (7) corresponds to the constraint (3) in the general framework. As pointed out earlier, it might be

important to include a relaxation of the constraint (3) in the master problem. Otherwise the master problem might have many infeasible integer solutions, i.e., it is likely that many infeasibility cuts would be needed. For the `disjunctive` constraint, Sadykov and Wolsey [24] suggested the following relaxation. For jobs $i, j, l \in N$, let $\alpha_{li} := (r_i - r_l)^+$ and $\beta_{jl} := (d_l - d_j)^+$. Then the constraints

$$\sum_{l \in N} \min \Big[d_j - r_i, (p_l - \max\{\alpha_{li}, \beta_{jl}\})^+ \Big] x_l \leq d_j - r_i \qquad \forall i, j \in N : r_i < d_j \tag{9}$$

are valid for the formulation (7)-(8). A constraint of type (9) is associated with an interval $[r_i, d_j]$. The coefficient on the variable x_l is a part of the processing time p_l that has to be processed in the interval $[r_i, d_j]$ if job l is on-time. The constraints (9) are discussed in more detail in Section 4. In the initial master problem we include the objective function (6), the relaxation (9) and the binary constraints (8).

One of the main components of the branch-and-check approach is a procedure that generates infeasibility cuts. The following "no-good" cuts have been used earlier in the literature [6, 12, 27]. If the solution $\bar{x}$ is not feasible and $\bar{J} = \{j \in N : \bar{x}_j = 1\}$, the inequality

$$\sum_{j \in \bar{J}} x_j \leq \mid \bar{J} \mid -1 \tag{10}$$

cuts off $\bar{x}$ and is valid for (7)-(8).

These cuts can be used to solve small instances with up to 50 jobs. However, our experience shows that a very large number of cuts are needed to solve bigger instances. In the next two sections we therefore suggest methods that attempt to generate stronger cuts. The computational study in Section 5 demonstrates that stronger cuts help in solving bigger instances faster.

3 The modified Carlier algorithm

The standard method for checking feasibility of a solution $\bar{x}$ is to solve a constraint satisfaction problem (CSP) for the set $\bar{J}$ of jobs with one global constraint `disjunctive`. To achieve this, CP algorithms are applied (see, for example, [2] for details). If CSP does not receive a feasible solution, a "no-good" cut (10) is derived. We now present a new algorithm for checking feasibility of a solution $\bar{x}$. The particularity if this algorithm is that it typically outputs more information than just "feasible" or "not feasible". This additional information allows to strengthen "no-good" cuts.

The idea of the strengthening is the following. In many cases when the set $\bar{J}$ of jobs is infeasible, there exists a subset $S \subset \bar{J}$ which is also infeasible. If it is possible to find such a subset S, the "no-good" cut

$$\sum_{j \in S} x_j \leq \mid S \mid -1$$

is stronger than the standard "no-good" cut (10).

However, finding a minimum, or even a minimal infeasible subset of jobs seems to be a very difficult problem. Even checking feasibility is $\mathcal{NP}$-hard in the strong sense. In addition only a limited amount of time can be spent on finding a small set S of jobs, since this computation is a subroutine, which can be called many times. We therefore turn to a heuristic approach.

We first introduce some notation. Let $L_j(\pi)$ denote the lateness of job j in a schedule π. $L_j(\pi)$ is the difference between the completion time $c_j(\pi)$ of job j and its due date d_j:, i.e., $L_j(\pi) = c_j(\pi) - d_j$. For a set S of jobs, let $r_S = \min_{j \in S} r_j$, $p_S = \sum_{j \in S} p_j$ and $d_S = \max_{j \in S} d_j$. Also, let $s_j(\pi)$ denote the starting time of job j in the schedule π. The set $\Pi(S)$ denotes the set of all possible schedules that can be constructed from the set S of jobs. We call a schedule $\pi \in \Pi(S)$ *feasible* with respect to a constant L' if the maximum lateness of π is less than or equal to L', i.e., if $\max_{j \in S} L_j(\pi) \leq L'$. A set S of jobs is *feasible* with respect to L' if and only if there exists a feasible schedule $\pi \in \Pi(S)$. Otherwise S is *infeasible*.

A solution $\bar{x}$ is feasible for the problem (6)-(8) if and only if the set $\bar{J}$ of jobs is feasible with respect to the value 0. Feasibility of the set $\bar{J}$ can be verified by solving the problem of minimizing maximum lateness with release dates on a single machine $1 \mid r_j \mid L_{\max}$, i.e., by finding a schedule $\pi^* \in \Pi(\bar{J})$ that minimizes the maximum lateness L^* and checking if $L^* \leq 0$. To solve such a problem one can use the algorithm by Carlier [7] which performs well in practice.

The Carlier algorithm uses a very efficient branching rule. This rule stays suitable for the following problem, we need to solve. Given a set N of jobs with release dates, processing times, due dates and a constant L', the aim is to find a feasible schedule $\pi^* \in \Pi(N)$, or, if this is not possible, an infeasible subset $S \subseteq N$ of jobs. For solving this problem, here we present an algorithm based on the branching rule suggested by Carler.

The algorithm is of the branch-and-bound type. Triples of job parameters $\{r_j^\nu, p_j^\nu, d_j^\nu\}$, $j \in N$, are associated with each node ν of the search tree. Release dates and due dates of jobs can differ from one node to another. The parameters for the the top node are the initial data.

At each node of the search tree, a Schrage schedule [25] π_σ is constructed using the following procedure. Jobs are scheduled one by one. Suppose the jobs from the subset $N' \subseteq N$ are already assigned to the first $\mid N' \mid$ positions of the schedule. Let $t = \max_{j \in N'} c_j(\pi_\sigma)$ be the maximum completion time of the jobs that are assigned. In the next position in π_σ we place a job that is unscheduled, available (with release date less or equal to t) and with the smallest possible due date. If there are no available jobs, a job with smallest possible release date is chosen. If there are several possibilities, a job with smallest possible due date is chosen among them. If the Schrage schedule is feasible, the algorithm is stopped and the set N is feasible.

Theorem 1 establishes some properties of infeasible Schrage schedules. The branching rule is based on these properties. Theorem 1 uses the following lemma from [7].

Lemma 1 *For any subset* $N' \subseteq N$

$$h(N') = \min_{j \in N'} r_j + \sum_{j \in N'} p_j - \max_{j \in N'} d_j = r_{N'} + p_{N'} - d_{N'}$$

is a lower bound on the maximum lateness of any schedule $\pi \in \Pi(N)$. □

This lower bound is also used in the algorithm itself.

Theorem 1 is a modified version of the main theorem from [7]. The modification is caused by the presence of a constant L' and, as a consequence, the different choice of the job a. In the theorem, jobs are renumbered in the order in which they are scheduled in the Schrage schedule. Two cases considered in the theorem are illustrated in Figures 1 and 2.

Theorem 1 *Suppose the Schrage schedule* $\pi_\sigma \in \Pi(N)$ *is infeasible with respect to constant* L'. *Let* b *be the first job in* π_σ *with the lateness larger than* L', *i.e.,* $L_b(\pi_\sigma) > L'$. *Also, let* a, $a \leq b$, *be the last job in* π_σ *that satisfies*

$$\alpha := s_a(\pi_\sigma) - \min_{a \leq j \leq b} r_j < L_b(\pi_\sigma) - L' := \beta. \tag{11}$$

Let $S = \{a, \ldots, b\}$, *then:*

1. *if there is no job* $c \in S$ *with* $d_c > d_b$ *then the set* S *of jobs is infeasible with respect to constant* L';
2. *otherwise, let* c *be the last job in* S *with* $d_c > d_b$. *If there exists a feasible schedule* $\pi' \in \Pi(N)$, *then job* c *is processed in* π' *either before all the jobs in* J *or after, where* $J = \{c+1, \ldots, b\}$.

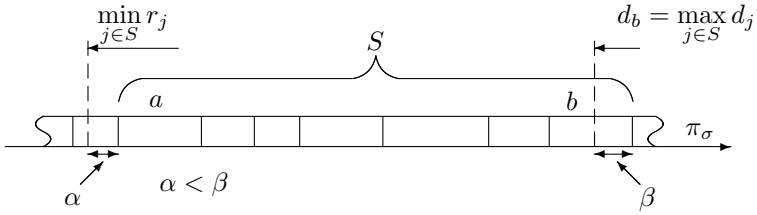


Figure 1: The infeasible Schrage schedule: case 1.

Proof. 1. Observe that there is no idle time between jobs in S in the schedule π_σ. If there was idle time, this would imply the existence of job a', $a' > a$, which is scheduled immediately after the idle time in π_σ and satisfies condition (11), because the following would hold: $s_{a'}(\pi_\sigma) = r_{a'} = \min_{j \in S} r_j$ (by the rules for constructing the Schrage schedule π_σ). This would contradict the condition that a is the last such job in schedule π_σ. We therefore have

$$L_b(\pi_\sigma) = s_a(\pi_\sigma) + \sum_{j \in S} p_j - d_b. \tag{12}$$

Since there is no job $c \in S$ that satisfies $d_c > d_b$, we have $d_b = \max_{j \in S} d_j$. Using this fact with (11) and (12) gives:

$$\min_{j \in S} r_j + \sum_{j \in S} p_j - \max_{j \in S} d_j > L'. \tag{13}$$

By Lemma 1 the left-hand side of (13) is a lower bound on the maximum lateness for any schedule in $\Pi(S)$. It follows that there is no schedule $\pi' \in \Pi(S)$ that satisfies such that $\max_{j \in S} L_j(\pi) \leq L'$, and therefore S is infeasible.

Notice that job a always exists, since the first job in a schedule π_σ satisfies (11).

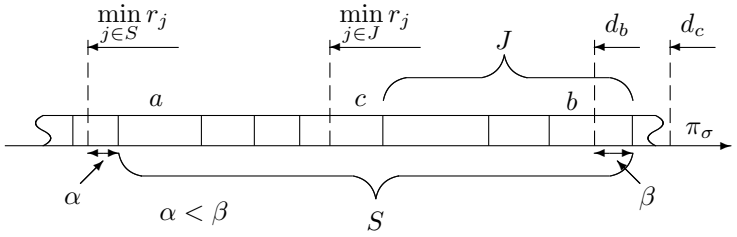


Figure 2: The infeasible Schrage schedule: case 2.

2. Since $c \in S$ is the last job in S satisfying $d_c > d_b$, $d_b = \max_{j \in J} d_j$. Since there is no idle time between the jobs in S in the schedule π_σ, $L_b(\pi_\sigma) = s_c(\pi_\sigma) + p_c + \sum_{j \in J} p_j - d_b$. Notice that $s_c(\pi_\sigma) < \min_{j \in J} r_j$, since otherwise some job in J would be scheduled instead of c in the Schrage schedule π_σ (as $d_c > d_b = \max_{j \in J} d_j$). If schedule $\pi' \in \Pi(N)$ is feasible and we schedule c inside J in π', the lateness of job $k \in J \subset N$, if scheduled last among jobs in $J \cup \{c\}$, would satisfy

$$L_k(\pi') \geq \min_{j \in J} r_j + p_c + \sum_{j \in J} p_j - d_k \geq L_b(\pi_\sigma) > L' \tag{14}$$

since $d_k \leq \max_{j \in J} d_j < d_b$. (14) contradicts the condition that $\max_{j \in N} L_j(\pi') \leq L'$. Therefore, in a feasible schedule π', job c is processed either before all the jobs in J or after. $\square$

Theorem 1 shows that, if at a node of the search tree the Schrage schedule is infeasible, and case 2 of Theorem 1 holds, two child nodes can be created. At the first node, job c is processed before J, and at the second node, job c is processed after all jobs in J. Notice that, because of the way the Schrage schedule is constructed, it suffices to change the due date of job c to $d_c = d_b - p_J$ in the former case, and to change the release date of job c to $r_c := r_J + p_J$ in the latter case.

If case 1 of Theorem 1 holds, we have an infeasible subset S of jobs, and the node is closed. We now consider a node of the search tree which is not a leaf node. From its child nodes it receives two infeasible subsets of jobs. Theorem 2

below shows how an infeasible subset of jobs can be constructed at a non-leaf node. In the theorem we assume the constant L' as given.

Theorem 2 *Consider a node ν in the search tree of the algorithm. Suppose that the case 2 of Theorem 1. If $S_a \subset N$ is an infeasible subset of jobs at the child node ν_a where c is scheduled before all the jobs in J, and $S_b \subset N$ is an infeasible subset of jobs at the child node ν_b where c is scheduled after all the jobs in J, then:*

1. *if $c \notin S_a$ then S_a is infeasible at the node ν;*
2. *if $c \notin S_b$ then S_b is infeasible at the node ν;*
3. *if $c \in S_a$ and $c \in S_b$ then $S = J \cup S_a \cup S_b$ is infeasible at the node ν.*

Proof. 1. If $c \notin S_a$, then the parameters of the jobs in S_a are the same at nodes ν and ν_a. Since at the node ν_a set S_a is infeasible, it is also infeasible at the node ν.

2. The proof is similar to case 1.

3. The proof is by contradiction. Let $\pi \in \Pi(S)$ be a feasible schedule. Since $c \in S$ and $J \subset S$, there are three possible ways of scheduling job c in π. First, c can be scheduled before J in π, but in this case $S_a \subset S$ is infeasible. Second, c can be scheduled after J in π, but in this case subset $S_b \subset S$ is infeasible. Therefore, c should be scheduled inside J in π, but this contradicts with the case 2 of Theorem 1. Thus, there is no feasible schedule $\pi \in \Pi(S)$ and the set S of jobs is infeasible at the node ν. □

Remark. Suppose a node ν of the search tree received subset S_a from its first child node and $c \notin S_a$. Then it is not necessary to explore the other child node. This follows from Theorem 2, since it is already known that S_a, and thus N, is infeasible at the node ν.

From the last two theorems it follows that, at all nodes of the search tree, we are able to determine an infeasible subset of jobs. Therefore, the top node gives an infeasible subset of jobs for the given instance.

The proposed algorithm uses a backtrack search strategy and can be implemented recursively. To find lower bounds, Lemma 1 is used. Notice that, if $h(S) > L'$, then S is infeasible with respect to constant L'. The modified Carlier algorithm is presented in Algorithm 1. It returns either a feasible schedule π^*, or an infeasible subset S' of jobs.

Example. We now apply the modified Carlier algorithm to a 9-job instance. The parameters of the jobs are shown in Table 1. Constant L' is equal to 0. For each node ν we show an illustration of the Schrage schedule π_σ. Job numbers are given inside the rectangles. For each job j the two numbers shown above are the release date r_j^ν and the due date d_j^ν. Numbers shown below are the completion times $c_j(\pi_\sigma)$ of the jobs.

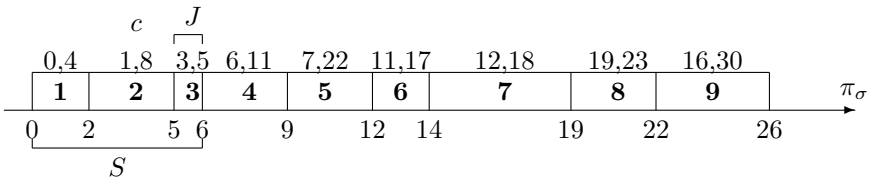
Algorithm 1 The modified Carlier algorithm

1: **MOD_CARLIER**(N, S');
2: **return** S'.

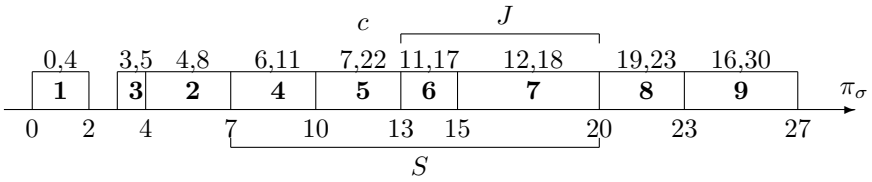
3: **procedure MOD_CARLIER**(N, S')
4: construct the Schrage schedule $\pi_\sigma \in \Pi(N)$;
5: renumber jobs in the order they are scheduled in π_σ;
6: **if** $\max_{j \in N} L_j(\pi_\sigma) \leq L'$ **then**
7: **return** π_σ; **end algorithm**;
8: **end if**
9: find the first job b in π_σ, such that $L_b(\pi_\sigma) > L'$;
10: find the last job a, $a < b$, in schedule π_σ,
such that $s_a(\pi_\sigma) - \min_{j \in S} r_j < L_b(\pi_\sigma) - L'$, where $S = \{a, \ldots, b\}$;
11: **if** there is no job $i \in S$ with $d_i > d_b$ **then**
12: $S' := S$;
13: **else**
14: find the last job $c \in S$ with $d_c > d_b$; $J := \{c+1, \ldots, b\}$;
15: **if** $h(J) = r_J + p_J - d_b > L'$ **then**
16: $S' := J$;
17: **else if** $h(J \cup \{c\}) = r_c + p_c + p_J - d_c > L'$ **then**
18: $S' := J \cup \{c\}$;
19: **else**
20: call **MOD_CARLIER**(N, S_a), where $r_c = r_J + p_J$;
21: **if** $c \in S_a$ **then**
22: call **MOD_CARLIER**(N, S_b), where $d_c = d_b - p_J$;
23: **end if**
24: **if** $c \notin S_a$ **then**
25: $S' := S_a$;
26: **else if** $c \notin S_b$ **then**
27: $S' := S_b$;
28: **else**
29: $S' := J \cup S_a \cup S_b$;
30: **end if**
31: **end if**
32: **end if**
33: **return** S'
34: **end procedure**

j	1	2	3	4	5	6	7	8	9
r_j	0	1	3	6	7	11	12	19	16
p_j	2	3	1	3	3	2	5	3	4
d_j	4	8	5	11	22	17	18	23	30

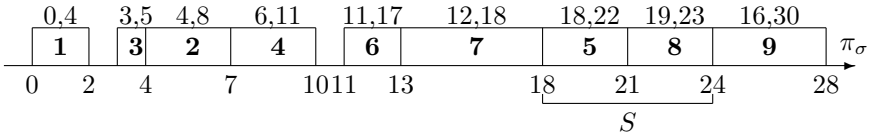
Table 1: The data for the 9-job instance



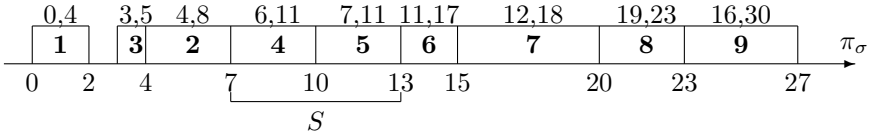
Node 1. Top node. $c = 2$ and $J = \{3\}$. The first child node 2 has $r_2^2 = 4$. Top node receives $S_a = \{4, \ldots, 8\}$. Since $c \notin S_a$, the second child node is not explored. Top node returns $S' := S_a = \{4, \ldots, 8\}$.



Node 2. $c = 5$, $J = \{6, 7\}$. The first child node 3 has $r_5^3 = 18$, node 2 receives $S_a = \{5, 8\}$. The second child node 4 has $d_5^4 := 11$, node 2 receives $S_b = \{4, 5\}$. Since $c = 5 \in S_a$ and $c = 5 \in S_b$, node 2 returns $S' := J \cup S_a \cup S_b = \{4, \ldots, 8\}$.



Node 3. Since the Schrage schedule at node 3 has no c, node 3 returns $S' := S = \{5, 8\}$.



Node 4. Since the Schrage schedule at node 4 has no c, node 4 returns $S' := S = \{4, 5\}$.

Since the top node returned the subset $S' = \{4, \ldots, 8\}$, the set S' is infeasible. In this case set S' is also a *minimal* infeasible subset. In other words, removing any job from S' makes it feasible. The search tree of the algorithm is depicted in Figure 3.

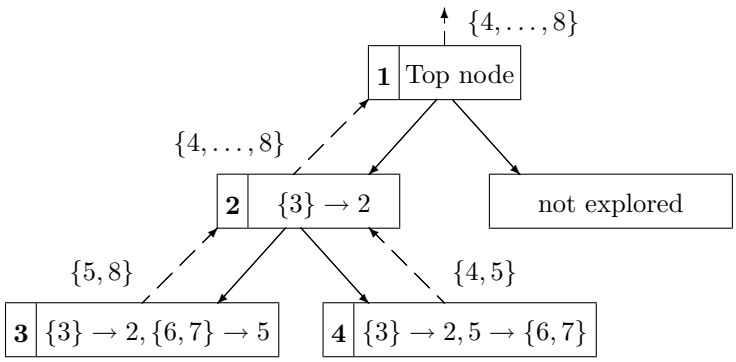


Figure 3: The search tree of the algorithm for the 9-job instance

4 Generalized tightening inequalities

In this section we present an alternative method for generating infeasibility cuts. The inequalities that form the relaxation (9) can be generalized as follows. The constraints (9) are stated for pairs of release and due dates. By means of constraint programming filtering algorithms, we can tighten the time windows of the jobs and obtain new values for release and due dates. The idea is to use the constraints (9) that can be obtained from these new values. However, we can only tighten the time windows of jobs when other jobs are scheduled on-time.

We now consider the case when the release date r_k of a job k can be increased to r_k^Ω if all the jobs in some set Ω are scheduled on-time. The case of decreasing a due date d_k is completely symmetric and is not considered. We do also not consider the case when both release and due dates can be adjusted.

To give some intuition, an example of a generalized tightening inequality is illustrated in Figure 4. The inequality is associated with a set $\Omega \subseteq N$ of jobs and two jobs $k \in N \setminus \Omega$ and $j \in N$. Let $\alpha_{lk}^\Omega = (r_k^\Omega - r_l)^+$ and $\beta_{jl} = (d_l - d_j)^+$. For each job $l \in N$ the time window $[r_l, d_l]$ is shown as a rectangle. The length of the filled area corresponds to the part of the processing time p_l which must be processed within the interval $[r_k^\Omega, d_j]$ when job l is on-time. The length of the hatched area corresponds to the part of processing time which can be executed outside the interval. For each job $l \in N$ the coefficient on the variable x_l in the inequality represents the part of the processing time which must be processed within the interval $[r_k^\Omega, d_j]$. Therefore the sum of the coefficients of all the on-time jobs can not exceed the length of the interval.

Proposition 1 gives the general result for a generalized tightening inequality. Let $a_l(E)$, $l \in N$, and $b(E)$ be the coefficient on variable x_l and the value of the right-hand side in inequality E, respectively. Then inequality E can be written

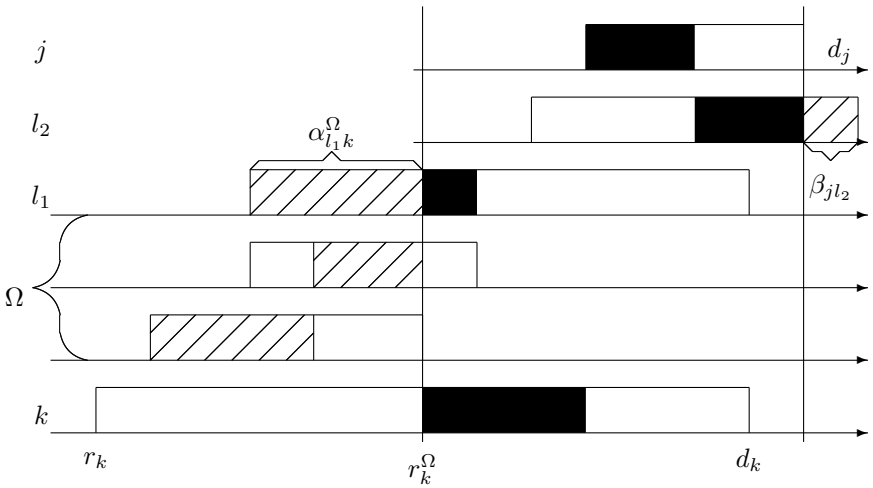


Figure 4: An illustration of a generalized tightening inequality

as

$$\sum_{l \in N} a_l(E) x_l \leq b(E).$$

Let also $A_x(E)$ be the value of the left-hand side in inequality E for solution x, i.e.,

$$A_x(E) = \sum_{l \in N} a_l(E) x_l.$$

Proposition 1 *Consider a subset $\Omega \subset N$ of jobs and a job $k \in N \setminus \Omega$, such that, if all the jobs in Ω are on-time, then k cannot start before r^{Ω}_k, and $r^{\Omega}_k > r_k$. Then the inequality*

$$\sum_{l \in N \setminus \Omega \setminus \{k\}} \min \left[d_j - r^{\Omega}_k, (p_l - \max\{\alpha^{\Omega}_{lk}, \beta_{jl}\})^+ \right] x_l + (p_k - \beta_{jk})^+ x_k \leq d_j - r^{\Omega}_k + (r^{\Omega}_k - r_k)(\mid \Omega \mid - \sum_{o \in \Omega} x_o) \tag{15}$$

is valid for (7)-(8) for each job $j \in N \setminus \Omega$ satisfying $d_j > r^{\Omega}_k$.

Proof. Consider the inequality E of type (15) for set Ω, job k and some job $j \in N \setminus \Omega$ satisfying $d_j > r^{\Omega}_k$.

Consider a feasible schedule with $\sum_{o \in \Omega} x_o \leq \mid \Omega \mid -1$. Let E' be the inequality of type (9), associated with the interval $[r_k, d_j]$. Inequality E' is valid, thus $\sum_{l \in N} a_l(E') x_l \leq b(E')$ for any feasible solution x. We will show that $b(E') \leq b(E)$ and $a_l(E) \leq a_l(E')$, $\forall l \in N$.

We have $b(E) \geq d_j - r_k = b(E')$. From $r^{\Omega}_k \geq r_k$, it follows that $\alpha^{\Omega}_{lk} \geq \alpha_{lk}$ and $a_l(E) \leq a_l(E')$, $\forall l \in N \setminus \Omega \setminus \{k\}$. Then, $a_l(E) = 0 \leq a_l(E')$, $\forall l \in \Omega$. Finally, $a_k(E) = a_k(E')$, and $a_l(E) \leq a_l(E')$, $\forall l \in N$.

Suppose now that $\sum_{o\in\Omega} x_o = \mid \Omega \mid$. Then the release date of job k can be set to r_k^Ω. Let E'' be the inequality of type (9), associated with the interval $[r_k^\Omega, d_j]$. The idea of the proof is the same as in the previous case.

We have $b(E) = b(E'')$ and $a_l(E) = a_l(E'')\ \forall l \in N \setminus \Omega$. Then $a_l(E) = 0 \leq a_l(E'')\ \forall l \in \Omega$. Again, $a_l(E) \leq a_l(E'')\ \forall l \in N$. $\square$

Since the number of inequalities (15) can be exponential, we do not include them in the initial formulation of the master problem. We propose to use these constraints as infeasibility cuts. Given an integer solution $\bar{x}$, the aim is to find an inequality (15), which is violated by $\bar{x}$. In other words, a separation algorithm is needed.

The idea of the separation algorithm for a given integer solution $\bar{x}$ is the following. We use a propagation algorithm, which performs adjustments of release dates of the jobs. Whenever the propagation algorithm adjusts the release date of some job $k \in N$ due to the presence of some set $\Omega \subset N$, we check whether $\bar{x}$ violates an inequality (15), which is associated with Ω, k, and some job $j \in N \setminus \Omega$. This examination takes $O(n^2)$ operations, since all jobs $j \in N \setminus \Omega$ need to be considered.

There are several propagation techniques in constraint programming for adjusting release dates of jobs. We concentrate on the technique called "Edge-Finding", which has the following propagation rule [2]:

$$\forall \Omega \in N, \forall k \in N \setminus \Omega: \text{ if } d_\Omega - r_{\Omega\cup\{k\}} < p_\Omega + p_k \tag{16}$$

$$\text{then } \Omega \text{ precedes } k \text{ and } r_k^\Omega = \max_{\Omega'\subseteq\Omega}\{r_{\Omega'} + p_{\Omega'}\} \tag{17}$$

Again there are several propagation algorithms, which perform all possible adjustments of release dates according to the rule (16)-(17). We use the algorithm presented in [2, p. 24], which can be easily implemented. The complexity of this algorithm is $O(n^2)$. Therefore, the straightforward separation algorithm based on this propagation algorithm has complexity $O(n^4)$. However, if the given integer solution $\bar{x}$ satisfies all the constraints (9), it is possible to decrease the complexity of the separation algorithm to $O(n^3)$. Details of the improved algorithm are presented in the Appendix.

Notice that there exist other methods to adjust release dates of jobs, for example, the "Not-First" propagation technique [2]. Therefore, the separation algorithm presented in the Appendix does not guarantee that, if a violated inequality of type (15) exists, it will be necessarily found.

5 Computational experiments

In this section we report the results of a computational study of the proposed branch-and-check algorithm. We studied five variants of the algorithm. Each variant uses a different strategy to generate infeasibility cuts. Otherwise the algorithms are the same.

Remember that we have three ways to generate infeasibility cuts. The first approach is to use standard "no-good" cuts. To generate these cuts we only

need to know whether solution $\bar{x}$ is feasible or not. In the experiments, to check feasibility, the instance of the problem $1 \mid r_j \mid L_{\max}$ with jobs in $\bar{J}$ was solved by Carlier algorithm [7]. We denote this standard approach "ng" ("no-good" cuts).

The second approach was suggested in Section 3. Here we tighten the "no-good" cuts with the modified Carlier algorithm. Unfortunately, this algorithm is sometimes time consuming. Therefore, in our experiments, the algorithm is interrupted when the number of created nodes reaches 1000. The second approach is denoted as "tng" (tightened "no-good" cuts).

The next approach was discussed in Section 4. Here generalized tightening inequalities are used as infeasibility cuts. To generate them, we apply the separation algorithm. We denote this approach "gti" (generalized tightening inequalities).

Notice that the separation algorithm does not tell us anything about the feasibility of a solution, if a violated inequality (15) is not found. Also, the modified Carlier algorithm cannot determine whether a solution is feasible or not, if it is interrupted after reaching the node limit. Therefore, in all variants of the algorithm, we use the first approach to test infeasibility when the other approaches are unsuccessful.

We consider the following five variants of the algorithm: "ng", "gti+ng", "gti+tng+ng", "tng+ng", "tng+gti+ng". For example, the last variant "tng+ gti+ ng" signifies that first the modified Carlier algorithm is applied. If it is interrupted, we use the separation algorithm. If the latter also does not find a violated inequality (15), the usual Carlier algorithm is used.

For the experiments we used three test sets. The first test set is from [1]. The data for these instances were derived based on four parameters:

- processing times were uniformly generated in the interval $[p_{\min}, p_{\max}]$;
- weights were uniformly generated in the interval $[1, w_{\max}]$;
- release dates were generated from the normal distribution with parameters $(0, \sigma)$, where σ depends on the *load* of the machine, see [1] for more details; the *load* is equal to ratio of the sum of all processing times and the difference between the maximum due date and minimum release date $d_N - r_N$;
- margins ($m_j = d_j - r_j - p_j$) were uniformly generated in the interval $[0, m_{\max}]$.

The values of the parameters were selected from each of the following values:

$(p_{\min}, p_{\max})$	(0,100), (25,75)
$m_{\max}$	50, 200, 350, 500, 650
load	1.0, 1.6, 2.2
$w_{\max}$	1, 10, 100.

For each quadruple of the parameters and for each $n \in \{10, 20, \ldots, 100\}$, there is one instance in the test set. In total there are 90 instances for each dimension.

We denote this test set by "b". Instances from this test set of a particular dimension n are denoted by "bn".

The second test set is from [9]. For these instances processing times were uniformly generated in the interval $[0, 100]$. Given the number of jobs n, the remaining data is derived based on the following three parameters:

- weights were uniformly generated in the interval $[1, w_{\max}]$;
- release dates were uniformly generated in the interval $[0, K_1 n]$;
- due dates were uniformly generated in the interval $[r_j + p_j, r_j + p_j + K_2 n]$.

The parameters used were:

$w_{\max}$	10, 99
K_1	1, 5, 10, 20
K_2	1, 5, 10, 20.

For each triple of parameters and for each $n \in \{20, 40, \ldots, 140\}$, there are ten instances in the test set. In total there are 320 instances for each dimension. We denote this test set by "s". Instances from this test set of a particular dimension n are denoted by "sn".

The third set of instances have been generated in a similar manner as the second test set. The only difference is that the weights are dependent on the processing times. Each w_j, $j \in N$, was generated uniformly in the interval $[\max\{1, p_j - \delta\}, \min\{99, p_j + \delta\}]$, where δ denotes a degree of dependence. We used two values for the parameter δ: 10 and 25. For the parameters K_1 and K_2 the same values as in the previous test set were used: 1, 5, 10, 20. For each triple of parameters and for each $n \in \{80, 100, 120\}$ there are ten instances in the test set. In total there are 160 instances for each dimension and each value of δ. We denote this test set by "d". Instances from this test set of a particular dimension n and δ are denoted by "dδ-n".

In the experiments we were interested in the following statistics.

$P_{1\text{h}}$, $P_{1000\text{s}}$ - the percentage of instances solved to optimality within one hour and within 1000 seconds.

The following statistics were evaluated only for the instances that were solved within the time limit.

T_{av}, $T_{\max}$ - average and maximum time needed to solve an instance to optimality (in seconds).

N_{av}, C_{av} - average number of nodes and average number of infeasibility cuts in the IP search tree.

$PMCE_{\text{av}}$ - average efficiency of the modified Carlier algorithm, i.e. the average difference in percents between the cardinality of the input set $\bar{J}$ of jobs and the cardinality of the found infeasible subset $S \subseteq \bar{J}$:

$$PMCE = \frac{|\,\bar{J}\,| - |\,S\,|}{|\,\bar{J}\,|} \cdot 100\%.$$

We took into account only the cases when the modified Carlier algorithm found an infeasible subset within the node limit.

$PMCL$ - percentage of cases in which the modified Carlier algorithm reached the node limit.

Computational experiments were performed on a computer with a 2 GHz Pentium IV processor and 512 Mb RAM. The algorithm have been implemented in the *Mosel* modelling and optimization language [8], using *XPress-MP* as the IP solver.

Table 2: Results for the five variants of the algorithm

	"ng"		"gti+ng"		"gti+tng+ng"		"tng+ng"		"tng+gti+ng"	
Test	P_{1h}	T_{av}	P_{1h}	T_{av}	P_{1h}	T_{av}	P_{1h}	T_{av}	P_{1h}	T_{av}
s40	99.7%	2.5	100%	0.9	100%	0.6	100%	0.6	100%	0.6
s60	98.8%	16.6	99.7%	6.0	100%	3.3	100%	3.5	100%	3.4
s80	95.3%	50.3	98.8%	15.2	100%	15.3	100%	15.4	100%	11.8
s100	-	-	99.4%	88.5	99.7%	46.0	99.7%	28.6	100%	33.0
s120	-	-	-	-	97.8%	132.4	99.7%	115.7	99.7%	109.0
s140	-	-	-	-	92.5%	127.2	95.9%	189.8	96.3%	183.6
b40	100%	0.8	100%	0.8	100%	0.6	100%	0.5	-	-
b50	98.9%	5.5	100%	3.0	100%	1.7	100%	1.5	-	-
b60	96.7%	38.5	98.9%	5.9	100%	3.3	100%	3.5	-	-
b70	94.4%	79.4	98.9%	27.7	100%	7.0	100%	6.7	-	-
b80	97.8%	91.1	98.9%	25.9	100%	14.5	100%	49.4	-	-
b90	87.8%	137.2	96.7%	41.5	98.9%	36.8	98.9%	26.3	-	-
b100	-	-	-	-	100%	104.2	100%	91.6	-	-

In Table 2 we present results for the five variants of the algorithm. "-" means, that the corresponding experiment has not been performed. We did not test the variant "tng+gti+ng" for the "b" instances because, when solving these instances by the variant "tng+ng", the modified Carlier algorithm very rarely reached the node limit. This fact implies that we would have the same results for the "tng+gti+ng" variant. Based on this, we conclude that the variant "tng+gti+ng" is the best on average for the test instances "s" and "b". The only exception is one of the "b80" instances, which is solved much faster by the "gti+tng+ng" variant. Because of this instance, the average statistics for the "b80" test set and the variants "tng+ng" and "tng+gti+ng" are worse.

If we consider the efficiency of the tightened "no-good" cuts and generalized tightening inequalities separately, the results in Table 2 indicate that the former are more efficient.

In the next experiment, we compared the results for the best variant "tng+ gti+ng" of the algorithm with the best previous results. These were presented in the paper by Péridy et al. [22]. Notice that in that paper the experiments were performed on a 450 MHz computer. We therefore re-set the time limit for our algorithm to 1000 seconds. Exactly the same test instances were used to

run both algorithms. The comparison is shown in Table 3. The results reveal that our algorithm is faster and solves more instances within the time limit, even even taking into account the difference in speed of the computers used.

Table 3: Comparison of the results

	Results in [22]			Results with "tng+gti+ng"		
Test	P_{1h}	T_{av}	T_{max}	P_{1000s}	T_{av}	T_{max}
b50	100%	47.1	409.1	100%	1.5	45.8
b60	100%	157.9	2147.0	100%	3.5	48.5
b70	97.8%	168.6	1745.3	100%	6.7	117.5
b80	97.8%	294.6	3567.6	98.9%	12.7	408.2
b90	88.9%	383.0	3542.7	98.9%	26.3	311.2
b100	83.3%	515.7	3581.6	98.9%	75.6	930.9
s40	100%	26.6	448.6	100%	0.6	6.7
s60	99.4%	178.2	2127.5	100%	3.4	263.8
s80	95.0%	496.5	3552.2	100%	11.8	514.3
s100	84.7%	1049.9	3560.0	99.7%	29.6	953.5

In Table 4 we present additional parameters for the variants "ng" and "tng+gti+ng" of the algorithm. It appears that in comparison with "ng", the average number of nodes in "tng+gti+ng" is significantly lower. Moreover, the number of the stronger cuts generated in "tng+gti+ng" is on average an order of magnitude less, than the number of "no-good" cuts used in "ng". This suggests that the efficiency of the cuts proposed here is quite high. Statistics for the parameter $PMCE_{av}$ partly explains the difference. Tightened "no-good" cuts are based on infeasible subsets of jobs that are of cardinality of roughly 40-50% the cardinality of the initial infeasible sets. Notice that the algorithm used to derive tightened "no-good" cuts is sufficiently fast. Difficulties only occur with the biggest "s" instances. On these instances the algorithm did not terminate before the node limit in roughly 10% of cases. In these cases we can sometimes still add generalized tightening inequalities in the variant "tng+gti+ng" in contrast to the variant "tng+ng". This seems to be the reason why the former variant is superior to the latter.

Results for the test set "d" are shown in Table 5. As expected, instances in which the weights and processing times are dependent are harder to solve. The stronger the dependence, the smaller the percentage of instances solved within the time limit.

Finally, we divided the test instances "s" into subsets that depend on parameter K_2. The larger this parameter, the larger the average margin between the due date d_j and the earliest completion time $r_j + p_j$. Larger margins give larger average time windows of the jobs. Looking at the results in Table 6, the efficiency of the algorithm appears to be highly dependent on the type of instances. The instances with small time windows are harder to solve. Many more nodes and cuts are needed to solve such instances. However, the $PMCE_{av}$ statistics show that for these instances, the modified Carlier algorithm finds in-

Table 4: Impact of stronger cuts

	"ng"		"tng+gti+ng"			
Test	N_{av}	C_{av}	N_{av}	C_{av}	$PMCE_{\text{av}}$	$PMCL$
b50	396.6	85.6	219.9	10.8	42.9%	0.3%
b60	768.0	184.5	345.8	13.8	46.7%	2.7%
b70	1307.1	273.9	527.5	16.2	57.6%	0.1%
b80	1917.9	252.6	1840.3	85.1	42.1%	0.2%
b90	2358.6	274.4	1300.7	20.3	51.6%	0.4%
b100	-	-	3657.9	60.8	51.5%	0.1%
s40	188.0	30.5	85.3	4.1	34.9%	0.1%
s60	400.1	80.7	267.5	10.3	42.0%	1.0%
s80	1019.3	159.5	501.0	16.3	39.1%	2.2%
s100	-	-	849.8	12.0	48.9%	1.6%
s120	-	-	1907.0	17.3	44.3%	11.7%
s140	-	-	2580.0	21.7	42.2%	9.3%

Table 5: Results for the "dependence" instances

Test	P_{1h}	T_{av}	N_{av}	C_{av}
s80	100%	11.8	501.0	16.3
s100	100%	33.0	849.8	12.0
s120	99.7%	109.0	1907.0	17.3
d25-80	99.4%	38.7	2055.6	36.6
d25-100	96.3%	162.8	4728.1	29.3
d25-120	92.5%	163.0	3486.6	44.9
d10-80	96.9%	64.6	3675.5	47.6
d10-100	95.0%	187.2	6045.8	69.8
d10-120	86.9%	186.2	4327.7	59.0

feasible subsets which are two times smaller than for other instances. Moreover, the algorithm never reaches the node limit. So the stronger cuts are especially useful for instances with small time windows.

6 Conclusions

In this paper we presented a branch-and-check algorithm for the $1 \mid r_j \mid \sum w_j U_j$ scheduling problem. Two approaches were suggested to generate stronger infeasibility cuts. First we suggested an approach for tightening the standard "no-good" cuts. Then a constraint propagation based approach was proposed for generating another type of infeasibility cuts. Experiments showed that the tightened "no-good" cuts are typically more efficient than the generalized tightened inequalities used in the algorithm. However the CP based cuts have some complementary strengths: they are useful in cases when the tightened "no-good" cuts cannot always be generated because of insufficient speed of the modified

Table 6: Results for subsets of the test instances with different time windows

Test	P_{1h}	T_{av}	N_{av}	C_{av}	$PMCE_{av}$	$PMCL$
$K_2 = 1$ (the smallest average time window)						
s140	90.0%	654.7	9237.8	61.7	70.7%	0.0%
s120	100%	342.0	6135.1	40.9	71.5%	0.0%
s100	100%	86.0	2372.4	27.4	72.8%	0.0%
s80	100%	12.9	513.6	12.4	72.1%	0.0%
s60	100%	2.3	104.5	4.4	69.3%	0.0%
$K_2 = 5$						
s140	97.5%	77.5	1026.5	21.1	22.8%	23.0%
s120	98.8%	48.7	842.5	20.6	25.1%	21.2%
s100	100%	16.0	540.1	11.4	36.3%	2.1%
s80	100%	17.4	1040.1	37.6	39.3%	0.0%
s60	100%	3.0	322.1	13.8	36.0%	0.1%
$K_2 = 10$						
s140	97.5%	18.3	338.8	3.8	25.9%	6.3%
s120	100%	22.6	414.9	6.2	15.3%	17.6%
s100	100%	10.4	321.5	8.2	18.4%	3.1%
s80	100%	7.0	295.1	11.4	19.6%	1.1%
s60	100%	5.2	467.5	16.9	47.3%	0.6%
$K_2 = 20$ (the largest average time window)						
s140	100%	24.0	288.0	3.9	14.4%	5.3%
s120	100%	21.8	222.4	1.8	15.6%	3.7%
s100	100%	19.6	165.1	1.1	22.9%	7.6%
s80	100%	10.1	155.4	3.8	12.0%	24.3%
s60	100%	3.4	176.1	6.0	27.5%	4.5%

Carlier algorithm.

Comparison with the algorithm by Péridy et al. [22] showed that the variant "tng+gni+ng" of the branch-and-check algorithm works faster and solves more instances from both test sets within the time limit. From our experiments we also learned that the efficiency of the algorithm is highly dependent on the type of instance. Instances with either small time windows, or with weights that dependent on processing times, are harder to solve.

We believe that the branch-and-check method can be successfully applied to solve other scheduling problems with disjunctive constraint(s). The difficult question which usually arises, when trying to apply this method to such a problem, is how to generate strong infeasibility cuts. We hope that the ideas presented here will help overcome this difficulty. One future research direction would be to develop infeasibility cuts based on more complex constraint propagation.

Acknowledgements

The author would like to thank Laurence Wolsey and Kent Andersen. Their advises helped to improve significantly the paper. The author is also grateful to Philippe Baptiste and Marc Sevaux for supplying test instances.

References

[1] Ph. Baptiste, A. Jouglet, C. Le Pape, W. Nuijten. A constraint-based approach to minimize the weighted number of late jobs on parallel machines, Research Report 2000/288, Université Technologie de Compiègne (2000).

[2] Ph. Baptiste, C. Le Pape, W. Nuijten. Constraint-based scheduling: applying constraint programming to scheduling problems, Kluwer Academic Publishers, 2001.

[3] Ph. Baptiste, L. Peridy, E. Pinson. A branch and bound to minimize the number of late jobs on a single machine with release time constraints, European Journal of Operations Research 144(1) (2003) 1-11.

[4] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems, Numerische Mathematik 4 (1962) 238-252.

[5] A. Bockmayr, Th. Kasper. Branch-and-Infer: a unifying framework for integer and finite domain constraint programming, INFORMS Journal on Computing 10 (1998) 287-300.

[6] Bockmayr, A., N. Pisaruk. Detecting infeasibility and generating cuts for MIP using CP, in: Proceedings of the 5th International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'03), Montreal, Canada, 2003.

[7] J. Carlier. The one machine sequencing problem, European Journal of Operatins Research, 11 (1982) 42-47.

[8] Y. Colombani, T. Heipcke. Mosel: an extensible environment for modeling and programming solutions, in: Proceedings of the 4th International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'02), Le Croisic, France, 2002. 277-290.

[9] S. Dauzère-Pérès, M. Sevaux. Using Lagrangean relaxation to minimize the weighted number of late jobs on a single machine, Naval Reseacrh Logistics 50(3) (2003) 273-288.

[10] A. Eremin, M. Wallace. Hybrid Benders decomposition algorithms in constraint logic programming, in: Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP'2001). LNCS 2239, Springer, 2001. 1-15.

[11] A.M. Benders. Generalized Benders decomposition, Journal of Optimization Theory and Applications 10 (1972) 237-260.

[12] V. Jain, I.E. Grossman. Algorithms for hybrid MILP/CLP models for a class of optimization problems, INFORMS Journal on Computing 13(4) (2001) 258-276.

[13] J.N. Hooker. Logic, optimization, and constraint programming, INFORMS Journal on Computing 14(4) (2002) 295-321.

[14] J.N. Hooker, G. Ottosson. Logic-based Benders decomposition, Mathematical Programming 96 (2003) 33-60.

[15] H. Kise, T. Ibaraki, H. Mine. A solvable case of the one machine scheduling problem with ready and due times, Operations Research 26(1) (1978) 121-126.

[16] E.L. Lawler. Sequencing to minimize the weigthed number of tardy jobs, RAIRO Operations Research 10 (1976) 27-33.

[17] E.L. Lawler. Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the "tower od sets" property, Mathematical and Computer Modelling 20(2) (1994) 91-106.

[18] E.L. Lawler, J.M. Moore. A functional equation and its application to resource allocation and sequencing problems, Management Science 16 (1969) 77-84.

[19] E.L. Lawler, J.K. Lenstra, A.G.H. Rinnooy Kan, D.B. Shmoys. Sequencing and scheduling: algorithms and complexity, in: Handbooks in Operations Research and Management Science, vol 4, Logistics of Production and Inventory, North, Amsterdam. 455-522.

[20] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker. Complexity of machine scheduling problems, Annals of Discrete Mathematics 1 (1977) 343-362.

[21] J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs, Management Science 15(1) (1968) 102-109.

[22] L. Péridy, E. Pinson, D. Rivraux. Using short-term memory to minimize the weighted number of late jobs on a single machine, European Journal of Operations Research 148 (2003) 591-603.

[23] R. Sadykov. A hybrid branch-and-cut algorithm for the one-machine scheduling problem, in: Proceedings of the First International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'04). Springer, LNCS, V.3011, 2004. 409-414.

[24] R. Sadykov, L. Wolsey. Integer programming and constraint programming in solving a multi-machine assignment scheduling problem with deadlines and release dates, to be published in INFORMS Journal on Computing, also available as CORE Discussion Paper 2003/81 (`http://www.core.ucl.ac.be/services/psfiles/dp03/dp2003-81.pdf`).

[25] L. Schrage. Solving resource-constrained network problems by implicit enumeration: non preemptive case, Operations Reseacrh 18 (1970) 263-278.

[26] M. Sevaux, S. Dauzère-Pérès. Genetic algorithms to minimize the weighted number of late jobs on a single machine, European Journal of Operations Research 151 (2003) 296-306.

[27] E.S. Thorsteinsson. Branch-and-Check: a hybrid framework integrating mixed integer programming and constraint logic programming, in: Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP'2001). LNCS 2239, Springer, 2001. 16-30.

Appendix

In the appendix we present in detail the separation algorithm. Given an integer solution $\bar{x}$ of the master problem, an inequality of the type (15) is sought, which violated by $\bar{x}$. The separation algorithm is based on a "Edge-Finding" propagation algorithm [2, p.24], which adjusts release dates according to the rule (16)-(17).

Assume that $\bar{x}$ satisfies constraints (9). The following three propositions describe cases when, if $\bar{x}$ violates any inequality of the type (15), then it should violate at least one inequality (9). Therefore, given the assumption, these three cases can be excluded from consideration.

Remember that $\bar{J} = \{j \in N : \bar{x}_j = 1\}$. Proposition 2 shows that an inequality E of the type (15) can be associated only with a set $\Omega \subset \bar{J}$, if E is violated by $\bar{x}$.

Proposition 2 *If an integer solution $\bar{x}$ violates inequality E of type (15) associated with $\Omega \subset N$, $k, j \in N \setminus \Omega$, and there exists a job $l \in \Omega$ such that $\bar{x}_l = 0$, then $\bar{x}$ also violates at least one inequality (9).*

Proof. Let E' be the inequality of type (9) that is associated with the interval $[r_k, d_j]$. Since E is violated by $\bar{x}$, we have $A_{\bar{x}}(E) > b(E)$. To prove the proposition we show that the right-hand side of inequality E is greater than or equal to the right-hand side of inequality E', and that the left-hand side of E is less then or equal to the left-hand side of inequality E'.

We have $\sum_{o \in \Omega} \bar{x}_o \leq \mid \Omega \mid -1$, thus $b(E) \geq d_j - r_k = b(E')$. From $r_k^\Omega > r_k$, it follows that $\alpha_{lk}^\Omega \geq \alpha_{lk}$ and $a_l(E) \leq a_l(E')\ \forall l \in N \setminus \Omega \setminus \{k\}$. Then $a_l(E) = 0 \leq a_l(E')\ \forall l \in \Omega$. Finally, $a_k(E) = a_k(E')$, and $A_{\bar{x}}(E) \leq A_{\bar{x}}(E')$ $\square$

Proposition 3 shows that an inequality E of the type (15) can be associated only with jobs $k, j \in N \setminus \Omega$, such that $d_j > d_k - p_k$, if E is violated by $\bar{x}$.

Proposition 3 *If an integer solution $\bar{x}$ violates inequality E of type (15), associated with $\Omega \subset N$, $k, j \in N \setminus \Omega$, and $d_j \leq d_k - p_k$, then $\bar{x}$ also violates at least one inequality (9).*

Proof. If $\sum_{o \in \Omega} \bar{x}_o <\mid \Omega \mid$, the proof follows from Proposition 2. Let $\sum_{o \in \Omega} \bar{x}_o =\mid \Omega \mid$. We consider three cases.

1. There exists a job $i \in N \setminus \Omega$, with $r_i < r_k^\Omega$, $r_i + p_i > r_k^\Omega$, $d_i \leq d_j$, and $\bar{x}_i = 1$. Let E' be the inequality of type (9), associated with the interval $[r_i, d_j]$. We have $b(E') = b(E) + (r_k^\Omega - r_i)$. To prove case 1 we show that $A_{\bar{x}}(E')$ is bigger than $A_{\bar{x}}(E)$ by at least $r_k^\Omega - r_i$.

From $r_i < r_k^\Omega$, it follows that $\alpha_{ik} \leq \alpha_{lk}^\Omega$ and $a_l(E) \leq a_l(E')\ \forall l \in N \setminus \Omega \setminus \{k\}$. Then $a_k(E) = 0 = a_k(E')$, since $d_j \leq d_k - p_k$, and $a_l(E) = 0 \leq a_l(E')\ \forall l \in \Omega$. Moreover, from $d_i \leq d_j$, it follows that $a_i(E) = p_i - (r_k^\Omega - r_i)$ and $a_i(E') = p_i$. Therefore, $a_i(E) - a_i(E') = r_k^\Omega - r_i$ and $A_{\bar{x}}(E') - A_{\bar{x}}(E) \geq r_k^\Omega - r_i$.

2. Case 1 does not hold and there exists at most one job $i \in N \setminus \Omega$, with $r_i < r_k^\Omega$, $r_i + p_i > r_k^\Omega$, $d_i - p_i < d_j$, and $\bar{x}_i = 1$. Let $m \in N \setminus \Omega$ be the job with smallest release date, such that $r_m \geq r_k^\Omega$, $r_m < d_j$, $a_m(E) > 0$, and $\bar{x}_m = 1$. Such a job should exist, otherwise E is not be violated by $\bar{x}$, since at most one job i with $\bar{x}_i = 1$ would have nonzero coefficient in E (no coefficient can be bigger than $b(E)$). Let E'' be the inequality of type (9), associated with the interval $[r_m, d_j]$. We have $b(E'') = b(E) - (r_m - r_k^\Omega)$. To prove case 2 we show that $A_{\bar{x}}(E'')$ is smaller than $A_{\bar{x}}(E)$ by at most $r_m - r_k^\Omega$.

We have $r_l + p_l < r_k^\Omega$ or $r_l \geq r_m$ for all jobs in N except at most one job i, thus $a_l(E'') = a_l(E)\ \forall j \in N \setminus \{i\}$. From $\alpha_{im} - \alpha_{ik}^\Omega = r_m - r_k^\Omega$, it follows that $a_i(E'') - a_i(E) \leq r_m - r_k^\Omega$ and $A_{\bar{x}}(E'') - A_{\bar{x}}(E) \leq r_m - r_k^\Omega$.

3. Case 1 does not hold and there exist at least two jobs $i_1, i_2 \in N \setminus \Omega$, with $r_{i_t} < r_k^\Omega$, $r_{i_t} + p_{i_t} > r_k^\Omega$, $d_{i_t} - p_{i_t} < d_j$, and $\bar{x}_{i_t} = 1$ for $t = 1, 2$. As case 1 does not hold, we have $d_{i_1} > d_j$ and $d_{i_2} > d_j$. Without loss of generality let $d' = d_{i_1} \leq d_{i_2}$ and $r' = \max\{r_{i_1}, r_{i_2}\}$. Let E^* be the inequality of type (9), associated with the interval $[r', d']$. We have $b(E^*) = b(E) + (r_k^\Omega - r') + (d' - d_j)$. To prove case 3 we will show that $A_{\bar{x}}(E^*)$ is bigger than $A_{\bar{x}}(E)$ by at least $(r_k^\Omega - r') + (d' - d_j)$.

From $r' < r_k^\Omega$ and $d_j < d'$, it follows that $a_l(E^*) \geq a_l(E)\ \forall l \in N$. First let $r' = r_{i_2}$. Then $\beta_{i_1 i_1} = 0$, $\alpha_{i_1 k}^\Omega - \alpha_{i_1 i_2} = r_k^\Omega - r'$, and $a_{i_1}(E^*) - a_{i_1}(E) \geq r_k^\Omega - r'$. Also, $\beta_{i_2 i_2} = 0$, $\beta_{j i_2} - \beta_{i_1 i_2} = d' - d_j$, and $a_{i_2}(E^*) - a_{i_2}(E) \geq d' - d_j$.

Now let $r' = r_{i_1}$. Then $a_{i_1}(E^*) = p_{i_1}$ and $a_{i_1}(E^*) - a_{i_1}(E) = \max\{r_k^\Omega - r', d' - d_j\}$. Also, $\alpha_{i_2 k}^\Omega - \alpha_{i_2 i_1} = r_k^\Omega - r'$, $\beta_{j i_2} - \beta_{i_1 i_2} = d' - d_j$. Thus $a_{i_2}(E^*) - a_{i_2}(E) \geq \min\{r_k^\Omega - r', d' - d_j\}$. Therefore $A_{\bar{x}}(E^*) - A_{\bar{x}}(E) \geq (r_k^\Omega - r') + (d' - d_j)$.
□

Proposition 4 shows that if an inequality E of the type (15) violates $\bar{x}$, then E can be associated only with jobs $k, j \in N \setminus \Omega$, such that there is no subset $J' \subseteq \bar{J}$ such that $d_{J'} \leq \max\{d_j, d_k\}$, $e_{J'} \geq r_k^\Omega$, $r_{J'} \leq r_k$, and $e_{J''} \leq e_{J'}$, $\forall J'' \subset J'$. Let for any set S of jobs $e_S = r_S + p_S$.

Proposition 4 *If an integer solution $\bar{x}$ violates inequality E of type (15) associated with $\Omega \subset N$, $k, j \in N \setminus \Omega$, and there exists a subset $J' \subseteq \bar{J}$ such that*

$d_{J'} \le \max\{d_j, d_k\}$, $e_{J'} \ge r^\Omega_k$, $r_{J'} \le r_k$, and $e_{J''} \le e_{J'}$, $\forall J'' \subset J'$, then $\bar{x}$ violates at least one inequality (9).

Proof. If $\sum_{o\in\Omega} \bar{x}_o <\mid \Omega \mid$, the proof follows from Proposition 2. Let $\sum_{o\in\Omega} \bar{x}_o =\mid \Omega \mid$. Suppose $d_j > d_k - p_k$, since otherwise the proof follows from Proposition 3. Let $d' = \max\{d_k, d_j\}$.

Let E' be the inequality of type (9), associated with the interval $[r_{J'}, d']$. We have $b(E') = b(E) + (r^\Omega_k - r_{J'}) + (d' - d_j)$. To prove the proposition we show that $A_{\bar{x}}(E')$ is bigger than $A_{\bar{x}}(E)$ by at least $r^\Omega_k - r_{J'} + (d' - d_j)$.

From $d' \ge d_{J'}$ it follows that $a_l(E) \le (p_l - \alpha^\Omega_{lk})^+$, $a_l(E') = p_l \ \forall l \in J'$. Also, $a_k(E') - a_k(E) = d' - d_j$. Then, from $a_l(E) \le a_l(E')\ \forall l \in N$, we have $A_{\bar{x}}(E') - A_{\bar{x}}(E) \ge p_{J'} - \sum_{l\in J'}(p_l - \alpha^\Omega_{lk})^+ + (d' - d_j)$. Therefore, it suffices to show that

$$\sum_{l\in J'}(p_l - \alpha^\Omega_{lk})^+ \le p_{J'} - (r^\Omega_k - r_{J'}). \tag{18}$$

The proof is by contradiction. Let $J'' = \{l \in J' : p_l - \alpha^\Omega_{lk} \ge 0\}$ and $r_{l'} = r_{J''}$. Assume that (18) does not hold. Then

$$e_{J'} = r_{J'} + p_{J'} < \sum_{l\in J''}(p_l - \alpha^\Omega_{lk}) + r^\Omega_k = p_{l'} - (r^\Omega_k - r_{l'})^+ + r^\Omega_k +$$

$$\sum_{l\in J''\setminus\{l'\}}(p_l - (r^\Omega_k - r_{l'})^+) \le r_{l'} + p_{l'} + \sum_{l\in J''\setminus\{l'\}} p_l = r_{J''} + p_{J''} = e_{J''},$$

contradicting the assumption of the proposition. □

The separation algorithm is presented in Algorithm 2. For the algorithm we assume that the input set $\bar{J}$ of jobs is ordered by nondecreasing release dates.

In the l-th outer iteration we consider two subsets $\Omega_\le = \{i \in \bar{J} : d_i \le d_l\}$ and $\Omega_> = \{i \in \bar{J} : d_i > d_l\}$. Set Ω will be sought among the sets $\Omega_{\le,i} = \{j \in \Omega_\le : r_j \ge r_i\}$. For each job $k \in \Omega_>$, we will check whether its release date r_k can be adjusted to C. r_k cannot be adjusted to a value more than C because of the definition of C (line 4) and the rule (17). We will show later why we do not consider values less than C.

In the beginning of the k-th iteration of the inner loop on the line 15, if $d_k > d_l$, we have:

$$P = p_{\Omega_{\le,k}}, \qquad H = \max_{i<k}\{r_i + p_{\Omega_{\le,i}}\} = r_h + p_{\Omega_{\le,h}} = e_{\Omega_{\le,h}}.$$

Now let $H = C$ and suppose that the condition (16) holds for job k and some set $\Omega_{\le,i}$. Notice that $r^\Omega_k \ge r_{\Omega_{\le,h}}$ and $r^\Omega_k \le C$ by the definition of C. Then, $d_{\Omega_{\le,h}} \le d_k$, as otherwise, by (16), we would have $p_{\Omega_\le} + p_k > d_{\Omega_\le\cup\{k\}} - r_{\Omega_\le\cup\{k\}}$ and $\bar{x}$ would violate the inequality of type (9), associated with the interval $[r_{\Omega_\le\cup\{k\}}, d_{\Omega_\le\cup\{k\}}]$. Hence, the set $\Omega_{\le,h} \subset \bar{J}$ of jobs can be taken as the set J' in Proposition 4 and $\bar{x}$ cannot violate any inequality of type (15) associated with set $\Omega_{\le,i}$, job k, and some job j. Therefore, $H < C$.

In line 15, we check the condition (16), which is $r_k + p_k + P > d_l$ for pair $(\Omega_{\le,k}, k)$ and $H + p_k > d_l$ for pair $(\Omega_{\le,h}, k)$. If one of these two condition holds,

Algorithm 2 The separation algorithm

for $l := 1$ to $\mid \bar{J} \mid$ **do**
  $\Omega_{\leq} := \{i \in \bar{J} : d_i \leq d_l\}$;
  $\Omega_{>} := \{i \in \bar{J} : d_i > d_l\}$;
  $P := p_{\Omega_{\leq}}$; $C := \max_{\Omega' \subseteq \Omega_{\leq}} \{r_{\Omega'} + p_{\Omega'}\}$;
  $\alpha'_i := (C - r_i)^+ \ \forall i \in \Omega_{>}$;
  $\bar{d} := \{\}$; $\bar{d} \leftarrow \{d_i; d'_i = d_i - \max\{p_i, \alpha'_i\}; d''_i := d_i - p_i\} \ \forall i \in \Omega_{>}$;
  sort $\bar{d}$; $H := -\infty$;
  for $k := 1$ to $\mid \bar{J} \mid$ **do**
    if $d_k \leq d_l$ **then**
      if $H < r_k + P$ **then**
        $h := k$; $H := r_k + P$;
      end if
      $P := P - p_k$
    else
      if ($r_k + p_k + P > d_l$ **or** $H + p_k > d_l$) **and** $H < C$ **then**
        if $r_k + p_k + P > d_l$ **then**
          $\Omega := \{k+1, \ldots, n\} \cap \Omega_{\leq}$;
        else
          $\Omega := \{h, \ldots, n\} \cap \Omega_{\leq}$;
        end if
        set d'_k to d_k in $\bar{d}$; sort $\bar{d}$;
        CHECK$(\Omega, k, C, \alpha', \bar{d})$
        set d'_k to $d_k - \max\{p_k, \alpha'_k\}$ in $\bar{d}$; sort $\bar{d}$;
      end if
    end if
  end for
end for

procedure CHECK$(\Omega, k, C, \alpha', \bar{d})$
$z := 0$; $s := \sum_{i \in \Omega_{>} \setminus \{k\}} (p_i - \alpha'_i) + p_k$; i is the maximum index in $\bar{d}$;
while $i > 1$ **and** $\bar{d}_i > d_k - p_k$ **do**
  if $s > \bar{d}_i - C$ **then**
    $\bar{x}$ violates the inequality of type (15), associated with set Ω
    and jobs k, j, $d_j = \bar{d}_i$; **end algorithm**;
  end if
  $i := i - 1$; $s := s - z \cdot (\bar{d}_{i+1} - \bar{d}_i)$;
  if $\bar{d}_i$ is some d'_j **then**
    $z := z + 1$;
  else if $\bar{d}_i$ is some d''_j **then**
    $z := z - 1$;
  end if
end while
end procedure

we obtain the set Ω in lines 17 or 19. As $H < C$, the value $C = \max_{\Omega' \subseteq \Omega_\leq}\{e_{\Omega'}\}$ is achieved on some set $\Omega_{\leq,i}$, $i > k$. Thus $\Omega_{\leq,i} \subseteq \Omega_{\leq,k} \subseteq \Omega_{\leq,h}$ and in both cases $r_k^\Omega = C$.

Having the pair (Ω, k) we need to check if there exists a violated inequality of type (15), associated with Ω, k and some job $j \in N$. Now we show how to do this in linear time.

We know before the inner loop in lines 8-26 that, if the "Edge-Finding" condition holds, then $r_k^\Omega = C$, $\forall k \in \Omega_>$. Thus, in any violated inequality E of type (15), $a_i(E) = 0$ $\forall i \in \Omega_\leq$, as $C - r_i > p_i$. Then, before the inner loop we know the values of α_{ik}^Ω for all $i \in \Omega_>$, $i \neq k$. These values are stored in line 5 as α'_i.

Let E' be the inequality of type (15), associated with Ω, some fixed job k and job $j \in \Omega_>$, $d_j = d_{\Omega_>}$. Then $a_i(E') = (p_i - \alpha'_i)^+$, $\forall i \in \Omega_> \setminus \{k\}$. In line 30, we set $s := A_{\bar{x}}(E')$. If the right border d_j of the interval $[C, d_{\Omega_>}]$ is decreased, the coefficient of a variable x_i, $i \neq k$, remains the same as long as $\alpha'_i \geq \beta_{ji}$, i.e. while $d_j \geq d'_i := d_i - \max\{p_i, \alpha'_i\}$. Then the coefficient decreases until it becomes equal to zero when $d_j = d''_i := d_i - p_i$. For the variable x_k, $d'_k = d_k$. In line 6, for each job $i \in \Omega_>$ all three values $\{d_i, d'_i, d''_i\}$ are added to array $\bar{d}$ and $\bar{d}$ is sorted. In line 21, when job k is known, we change the value of d'_k and sort $\bar{d}$. Notice that in this case sorting takes only linear time, as only one value in $\bar{d}$ has been changed. In line 23, $\bar{d}$ is put into the initial state.

Using array $\bar{d}$, the procedure CHECK can check whether there exists a violated inequality in linear time. It just decreases the right border of the interval and adjusts properly the sum s of coefficients of variables from $\Omega_>$. If for some right border d_j, the sum s is greater than the length of the interval $[C, d_j]$ then the desired inequality is found.

As the complexity of the procedure CHECK is $O(n)$, the overall complexity of the separation algorithm is $O(n^3)$.