Combining column generation and Lagrangean relaxation : an application to a single-machine common due date scheduling problem

Eindhoven University
of Technology

Department of Mathematics and Computing Sciences

Memorandum COSOR 98-18

Combining column generation and Lagrangean relaxation
An application to a single-machine common due date scheduling problem

M. van den Akker
J.A. Hoogeveen
S. van de Velde

Eindhoven, August 1998
The Netherlands

COMBINING COLUMN GENERATION AND LAGRANGEAN RELAXATION

AN APPLICATION TO A SINGLE-MACHINE COMMON DUE DATE SCHEDULING PROBLEM

Marjan van den Akker
National Aerospace Laboratory NLR
P.O. Box 90502, 1006 BM Amsterdam, The Netherlands
email address: vdakker@nlr.nl

Han Hoogeveen
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
email address: slam@win.tue.nl

Steef van de Velde
Rotterdam School of Management
Erasmus University
P.O. Box 1738, 3000 DR Rotterdam, The Netherlands
email address: s.velde@fac.fbk.eur.nl

29 June 1998

ABSTRACT

Column generation has proved to be an effective technique for solving the linear programming relaxation of huge set covering or set partitioning problems, and column generation approaches have led to state-of-the-art so-called branch-and-price algorithms for various archetypical combinatorial optimization problems. Usually, if Lagrangean relaxation is embedded at all in a column generation approach, then the Lagrangean bound serves only as a tool to fathom nodes of the branch-and-price tree. We show that the Lagrangean bound can be exploited in more sophisticated and effective ways for two purposes: to speed up convergence of the column generation algorithm and to speed up the pricing algorithm.

Our vehicle to demonstrate the effectiveness of teaming up column generation with Lagrangean relaxation is an archetypical single-machine common due date scheduling problem. Our comprehensive computational study shows that the combined algorithm is by far superior to two existing purely column generation algorithms: it solves instances with up to 125 jobs to optimality, while purely column generation algorithm can solve instances with up to only 60 jobs.

1980 Mathematics Subject Classification (Revision 1991): 90B35.
Keywords and Phrases: column generation, Lagrangean relaxation, set covering, linear programming, earliness-tardiness scheduling, common due date.

1 Introduction

For many an NP-hard combinatorial minimization (maximization) problem, a remarkably strong lower (upper) bound on the optimal solution value can be computed by formulating the problem as an integer linear program (usually a set covering or set partitioning problem) with a huge number of variables and then solving the linear programming relaxation by a *column generation* method. This approach has led to state-of-the-art branch-and-bound (also called branch-and-price) algorithms for such archetypical combinatorial optimization problems as the vehicle routing problem with time windows (Desrochers *et al.*, 1992; Desrosiers *et al.*, 1995), the generalized assignment problem (Savelsbergh, 1997), the parallel-machine scheduling problem (Van den Akker *et al.*, 1995; Chen and Powell, 1995) and the graph coloring problem (Mehrotra and Trick, 1997) as well as for various real-life problems including airline crew scheduling, bus driver scheduling, and production scheduling (see for a selection of references, Soumis (1997)).

The idea behind column generation is to solve the linear programming relaxation on only a small subset of all the variables. A so-called *pricing algorithm* verifies global optimality of the solution for the linear programming relaxation of the minimization (maximization) problem by checking if there exist one or more variables that were left out of the linear program with negative (positive) reduced cost. If there is one or more, then some of them, depending on the design and the implementation of the algorithm, are added to the linear program, since they may possibly improve the solution value, and the procedure is repeated. If there is none, then the current solution is optimal for the linear programming relaxation. For an introduction to the ideas and fundamentals behind column generation and branch-and-bound algorithms based on this concept, we refer to Barnhart *et al.* (1994), Vanderbeck (1998), and Freling *et al.* (1998).

In theory, the linear programming bound obtained by column generation can alternatively be computed by means of *Lagrangean relaxation.* In fact, the basic difference between column generation and Lagrangean relaxation, which are both iterative methods, lies in the way the dual multipliers are adjusted in every iteration. The usual methods for updating the dual (or Lagrangean) multipliers, such as the subgradient method and the bundle method, are relatively easy to implement and require hardly no computer storage. There are serious practical problems with their convergence behavior, however: they are quite slow, with a lot of zig-zagging in the beginning; what is more, they cannot be guaranteed to converge to the optimal solution. In general, column generation has much better convergence properties than Lagrangean relaxation. Now, with the recent availability of efficient simplex codes with easy to implement column generation capabilities and computers with sufficient computer memory, it seems as if these properties can finally be exploited to the max and it looks as if column generation has become the preferred method.

Although column generation is faster and shows betters convergence properties, one drawback still stands: it does not give a lower (upper) bound on the optimal solution value till complete convergence. In contrast, Lagrangean relaxation gives a lower (upper) bound in each iteration.

It is well known that Lagrangean relaxation can complement column generation method very nicely in that it can be used in each iteration of the column generation method to compute a lower (upper) bound in only little additional time; see for instance Vanderbeck and Wolsey (1996). The use of the lower (upper) bound, however, is primarily restricted to comparing it to an upper (lower) bound on the optimal solution value for one of two purposes: (i) to see if a node in the branch-and-price tree can be fathomed; or (ii) to see if the column generation algorithm has converged to the optimal solution value. For the first purpose, the speedup can be considerable; for the second, the speedup is usually only modest.

The message of this paper is there are other, more effective ways to exploit Lagrangean relaxation within the framework of a column generation algorithm. We not only show that there are more ways to alleviate the so-called *tailing-off effect*, that is, to speed up convergence of the column generation method, but also that it can be used to speed up the pricing algorithms.

Our vehicle to demonstrate the effectiveness of this combination of column generation and Lagrangean relaxation is a standard single-machine earliness-tardiness problem with asymmetric job weights and a large common due date. Van den Akker *et al.* (1996) and Chen and Powell (1997) have presented pure column generation algorithms for this problem that can solve instances with up to 60 jobs to optimality. In this paper, we present a combined column generation and Lagrangean relaxation algorithm that solves instances with up to 125 jobs to optimality.

The plan of this paper is as follows. In Section 2, we present a formal description of the problem under study, review the relevant literature, and point out the main characteristics of the two existing column generation algorithms. In Section 3, we formulate the earliness-tardiness problem as a set covering problem with two additional constraints and an exponential number of variables, present the column generation algorithm to solve its linear programming relaxation (Section 3.1), analyze its Lagrangean problem (Section 3.2), and point out how column generation and Lagrangean relaxation can be combined to our computational advantage (Section 3.3). In Section 4, we present our computational results for four classes of randomly generated instances. Surprisingly, the column generation method renders an optimal integral solution for each problem instance. Section 5 concludes the paper with some remarks on the general applicability of a combination of column generation and Lagrangean relaxation. We further sketch a branch-and-price algorithm for the single-machine earliness-tardiness problem under study that can be used if the column generation algorithm does not terminate with an integral solution.

2 Problem description

We consider the following single-machine scheduling problem. A set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n independent jobs has to be scheduled on a single machine that is continuously available from time zero onwards. The machine can handle at most one job at a time. Job J_j $(j = 1, \ldots, n)$ requires a positive integral uninterrupted processing time p_j and should ideally be completed exactly on its due date d, which is common to all jobs. We say that this common due date is *large*, if $d \geq \sum_{j=1}^{n} p_j$; otherwise, we call it *small*. In case of a large common due date, the constraint that the machine is not available before time zero is redundant. A *schedule* specifies for each job J_j a completion time C_j such that the jobs do not overlap in their execution. The order in which the machine processes the jobs is called the *job sequence*. For a given schedule, the earliness of J_j is defined as $E_j = \max\{0, d - C_j\}$ and its tardiness as $T_j = \max\{0, C_j - d\}$. Accordingly, J_j is called *early*, *just-in-time*, or *tardy* if $C_j < d$, $C_j = d$, or $C_j > d$, respectively. The cost of a schedule σ is the sum of weighted job earliness and tardiness, that is,

$$f(\sigma) = \sum_{j=1}^{n} [\alpha_j E_j + \beta_j T_j],$$

where α_j and β_j are given positive weights. The problem is to find a schedule with minimum cost.

We assume that the common due date is large. We assert that the analysis of the problem with a small common due date and the design of the algorithms for its solution are pretty much similar to those for the problem with a large common due date; see also Van den Akker *et al.* (1996). We can take advantage of three well-known properties that characterize a class of optimal solutions (Quaddus, 1987; Baker and Scudder, 1989):

— There is no idle time between the execution of the jobs;

— One of the jobs completes exactly on time d;

— The jobs completed at or before d are in order of nondecreasing α_j/p_j ratio, and the jobs started at or after d are in nonincreasing β_j/p_j ratio.

Schedules that possess these three properties are called *V-shaped.* Note that any V-shaped schedule consists of two parts: *the early schedule*, consisting of the jobs completed before or on time d; and *the tardy schedule*, consisting of the jobs completed after time d. We call these two schedules *complimentary.* The characterization implies that the problem is a partitioning problem: we need to select the jobs that are completed at or before d, and the jobs that are completed after d. Furthermore, due to these three properties, the value of the common due date is irrelevant, provided that it is large.

The problem is arguably the most vexing earliness-tardiness scheduling problem remaining; for an overview of such problems, we refer to Baker and Scudder (1990). The problem is NP-hard in the ordinary sense, but it defies the type of pseudopolynomial algorithm that is so common in earliness-tardiness scheduling — it is therefore still an open question whether the problem is solvable in pseudopolynomial time or NP-hard in the strong sense. The NP-hardness of the problem follows from the NP-hardness of its symmetric counterpart where $\alpha_j = \beta_j = w_j$ for each job J_j $(j = 1, \ldots, n)$ (Hall and Posner, 1991). Hall and Posner also present an $O(n \sum_{j=1}^{n} p_j)$ time and space dynamic programming algorithm, thereby establishing the computational complexity of the symmetric problem. This algorithm proceeds by adding the jobs in order of nondecreasing w_j/p_j ratio; hence, the dynamic programming algorithm can be applied to the asymmetric case only if the ratios α_j/p_j and β_j/p_j induce the same job sequence. Furthermore, till column generation algorithms were developed for its solution, the problem was also very hard in practice as well, since it seemed to be impossible to compute strong lower bounds (Hall and Sriskandarajah, 1990; De, Ghosh, and Wells, 1994). De *et al.* formulate the problem as a quadratic 0-1 integer programming problem, which they solve by the branch-and-bound algorithm proposed by Pardalos and Rodgers (1990). Their algorithm solves randomly generated instances with up to 30 jobs without much effort, but it may take more than 300 seconds to solve an instance with $n = 40$ on a VAX 4000-300 machine. De *et al.* present also a specific randomized local search algorithm, a so-called GRASP algorithm, which has empirically only a small erroneous behavior.

Independently, Van den Akker *et al.* (1996) and Chen and Powell (1997) have presented a column generation algorithm for this problem. Although both algorithms can solve instances with up to 60 jobs to optimality, there are some differences. Chen and Powell present in fact a column generation algorithm for the general m-machine problem ($m \geq 2$) and use an $O(n^2 \sum_{j=1}^{n} p_j)$ time pricing algorithm. Focusing on the single-machine case only, Van den Akker *et al.* are able to present a simpler algorithm, which uses a faster $O(n \sum_{j=1}^{n} p_j)$ time pricing algorithm. The computational results differ as well: while Van de Akker *et al.* report that the solution to the linear programming relaxation was integral for all their randomly generated instances, Chen and Powell report that there is a very small integrality gap and that on average just a few branch-and-bound nodes were necessary to close the gap.

3 Lower bound computing

We formulate the large common due date problem as a set covering problem with an exponential number of binary variables, n covering constraints, and two additional side-constraints. Our formulation reflects that we search for an optimal V-shaped schedule.

Let $\mathcal{E}$ and $\mathcal{T}$ be the set of all early and tardy schedules, respectively, where $\mathcal{T}$ includes the empty tardy schedule. Each feasible schedule $s \in \mathcal{E}$ is characterized by a 0-1 vector $a_s = (a_{1s}, \ldots, a_{n,s})$, where $a_{js} = 1$ if J_j is included in s ($j = 1, \ldots, n$). In a similar fashion, each feasible schedule $s \in \mathcal{T}$ is characterized by a 0-1 vector $a_s = (a_{1s}, \ldots, a_{n,s})$, where $a_{js} = 1$ if J_j is included in s ($j = 1, \ldots, n$). Given any $a_s \in \mathcal{T}$ or $\mathcal{E}$, we can recover the corresponding schedule s and its cost, which we denote by c_s, in a straightforward fashion.

Let now x_s be a 0-1 variable that takes the value 1 if schedule s is selected and the value 0, otherwise. The problem, which we refer to as problem (P), is then to find the value z^*, which is the minimum of

$$\sum_{s\in\mathcal{E}} c_s x_s + \sum_{s\in\mathcal{T}} c_s x_s$$

subject to

$$\sum_{s\in\mathcal{E}} a_{js} x_s + \sum_{s\in\mathcal{T}} a_{js} x_s \geq 1, \text{ for } j = 1,\ldots,n, \tag{1}$$

$$\sum_{s\in\mathcal{E}} x_s \leq 1, \tag{2}$$

$$\sum_{s\in\mathcal{T}} x_s \leq 1, \tag{3}$$

$$x_s \in \{0,1\}, \text{ for each } s \in \mathcal{E} \text{ and } s \in \mathcal{T}. \tag{4}$$

Conditions (1) enforce that each job is executed at least once. Conditions (2) and (3) make sure that no more than one early schedule and one tardy schedule are selected. Conditions (4) are the integrality conditions. Of course, there exists an optimal solution in which conditions (1)-(3) will hold with equality in any optimal solution, since all cost coefficients are positive.

We cannot realistically hope that this problem is solvable in time polynomial in n, since the underlying problem is NP-hard. Furthermore, an *explicit* formulation of even a modest problem instance is impossible because of the huge number of schedules involved. We are therefore interested in computing a strong (mathematical programming) lower bound on the optimal solution value. There are two alternative methods available that can handle an exponential number of variables: *column generation*, which we discuss in Section 3.1, and *Lagrangean relaxation*, which we discuss in Section 3.2. Column generation solves the linear programming relaxation of the above formulation, and as we will see, Lagrangean relaxation gives in theory the same linear programming bound. However, we get the best results if we combine the two — how this can be done is discussed in Section 3.3.

3.1 Column generation

The linear programming relaxation of the integer linear programming problem is obtained by replacing conditions (4) by the conditions

$$x_s \geq 0, \text{ for each } s \in \mathcal{E} \text{ and } s \in \mathcal{T}, \tag{5}$$

as conditions (2) and (3) prohibit values greater than 1.

In each iteration of the column generation procedure, we take only a feasible subset of the schedules, say, $\bar{\mathcal{S}}$, into consideration, solve the linear programming relaxation, and add new schedules if needed. We call a subset of schedules *feasible* if they contain at least one feasible solution to problem (P). A feasible subset can easily be generated by some iterative local improvement heuristic, as we will see in Section 4. From the theory of linear programming, we know that adding a schedule s with corresponding variable x_s can decrease the value of the linear programming solution only if s has negative *reduced cost*. The reduced cost c'_s of any $s \in \mathcal{E}$ with vector a_s is defined as

$$c'_s = c_s - \sum_{j=1}^{n} a_{js}\lambda_j + \lambda_{n+1},$$

where $\lambda_j \geq 0$ $(j = 1, \dots, n)$ is the value of the dual variable corresponding to the jth of the constraints (1) and $\lambda_{n+1} \geq 0$ is the value of the dual variable corresponding to condition (2). For any $s \in \mathcal{T}$, the reduced cost c'_s is defined as

$$c'_s = c_s - \sum_{j=1}^{n} a_{js}\lambda_j + \lambda_{n+2},$$

where $\lambda_{n+2} \geq 0$ is the value of the dual variable corresponding to condition (3). All these dual variables follow from the linear programming solution.

We want to solve the *pricing problem* of finding a schedule s with minimal c'_s value; if this minimum is nonnegative, then we know that the value of the linear programming solution will not decrease by taking the remaining schedules into consideration, which implies that we have found the optimal solution of the linear programming relaxation. We solve the pricing problem by finding the early and tardy schedule with minimum reduced cost among all early and tardy schedules, respectively. To that end, we use two pricing algorithms: one to find an early schedule with minimum reduced cost; and one to find a tardy schedule with minimum reduced cost. The latter is essentially the same as the pricing algorithm that we used for the problem of minimizing total weighted completion time on a set of identical parallel machines; see Van den Akker *et al.* (1995). The pricing algorithm to find an early schedule with minimum reduced cost is very similar; we work out the details of this algorithm below.

In case of an early schedule, the pricing problem reduces to minimizing $c_s - \sum_{j=1}^{n} a_{js}\lambda_j$ over all binary vectors a_s; after all, λ_{n+1} is a constant for given λ. Suppose that the jobs have been reindexed in order of nonincreasing α_j/p_j ratios, that is,

$$\frac{\alpha_1}{p_1} \geq \cdots \geq \frac{\alpha_n}{p_n}.$$

Then for any job J_j in the early schedule s with vector a_s, we have that

$$E_j = d - C_j = \sum_{i=1}^{j-1} a_{is}p_i.$$

As s is an early schedule, we have that $c_s = \sum_{j=1}^{n} \alpha_j a_{js} E_j$, from which we obtain that

$$\begin{aligned} c'_s &= \sum_{j=1}^{n} \alpha_j a_{js} \sum_{i=1}^{j-1} a_{is}p_i - \sum_{j=1}^{n} a_{js}\lambda_j + \lambda_{n+1} \\ &= \sum_{j=1}^{n} \left[\alpha_j \sum_{i=1}^{j-1} a_{is}p_i - \lambda_j \right] a_{js} + \lambda_{n+1}. \end{aligned}$$

Hence, if $\sum_{i=1}^{j-1} a_{is}p_i = t$, then including J_j in the early schedule (or putting $a_{js} = 1$) affects c'_s by $\alpha_j t - \lambda_j$.

We present a pseudo-polynomial dynamic programming algorithm to solve the pricing problem. For any given $\lambda \geq 0$, let $F_\lambda(j,t)$ denote the minimum reduced cost for all early schedules that consist of jobs from the set $\{J_1, \ldots, J_j\}$ in which the first job in the schedule starts at time $d - t$. The initialization is

$$F_\lambda(j,t) = \begin{cases} \lambda_{n+1}, & \text{if } j = 0 \text{ and } t = 0, \\ \infty, & \text{otherwise.} \end{cases}$$

The recursion is then, for $j = 1, \ldots, n$, $t = 0, \ldots, \sum_{i=1}^{j} p_i$

$$F_\lambda(j,t) = \min\{\ F_\lambda(j-1,t),\ F_\lambda(j-1,t-p_j) + \alpha_j(t-p_j) - \lambda_j\}, \qquad (6)$$

where the first and second term reflect the decision of leaving J_j out of s and adding J_j to s, respectively. The early schedule with minimum reduced cost is the one corresponding to

$$\min_{0 \le t \le P} F_\lambda(n,t),$$

where $P = \sum_{i=1}^{n} p_i$. Note that *any* value $F_\lambda(n,t) < 0$ corresponds to an early schedule with negative reduced cost. This raises the issue whether just the early schedule with minimum negative reduced cost, a small number of early schedules with most negative reduced cost, or all early schedules with negative reduced cost should be added to the set $\bar{\mathcal{S}}$. This implementation issue is discussed in Section 4.1.

We construct the pricing algorithm for generating tardy schedules in a similar fashion. Let $G_\lambda(j,t)$ denote the minimum reduced cost for all tardy schedules that consist of jobs from the set $\{J_1, \ldots, J_j\}$ in which the last job completes at time t. As our initialization, we have

$$G_\lambda(j,t) = \begin{cases} \lambda_{n+2}, & \text{if } j = 0 \text{ and } t = 0, \\ \infty, & \text{otherwise.} \end{cases}$$

The values $G_\lambda(j,t)$ $(j = 1, \ldots, n; t = 0, \ldots, \sum_{i=1}^{j} p_i)$, are computed through the recurrence relation

$$G_\lambda(j,t) = \min\{G_\lambda(j-1,t), G_\lambda(j-1,t-p_j) + \beta_j t - \lambda_j\}, \qquad (7)$$

and we determine

$$\min_{0 \le t \le P} G_\lambda(n,t),$$

to find the tardy schedule with minimum reduced cost from among all tardy schedules. Again, each value $G_\lambda(n,t) < 0$ corresponds to a tardy schedule with negative reduced cost. The issue of which tardy schedules with negative reduced cost to add to $\mathcal{S}$ is addressed in Section 4.1.

Note that both pricing algorithms run in $O(n \sum_{j=1}^{n} p_j)$ time and space.

3.2 Lagrangean relaxation

Consider now the Lagrangean problem of problem (P) obtained by dualizing constraints (1) with a given vector of Lagrangean multipliers $\rho = (\rho_1, \dots, \rho_n) \geq 0$; we refer to this problem as problem (L_ρ). For any given ρ, the Lagrangean problem (L_ρ) is to find the value $L(\rho)$, which is the minimum of

$$\sum_{s\in\mathcal{E}}(c_s - \sum_{j=1}^{n} \rho_j a_j s)x_s + \sum_{s\in\mathcal{T}}(c_s - \sum_{j=1}^{n} \rho_j a_j s)x_s + \sum_{j=1}^{n} \rho_j$$

subject to conditions (2), (3), and (4). From standard Lagrangean theory (see for instance Fisher (1981)), we know that $L(\rho)$ is a lower bound on the optimum solution value for any $\rho \geq 0$.

Since $\sum_{j=1}^{n} \rho_j$ is a constant, the Lagrangean problem decomposes into two independent subproblems: one problem of finding an early schedule with minimum $c_s - \sum_{j=1}^{n} \rho_j a_{js}$; and one problem of finding a tardy schedule with minimum $c_s - \sum_{j=1}^{n} \rho_j a_{js}$. The key observation is that both these problems are solved to optimality by using the two pricing algorithms presented in the previous section; we just use ρ_j instead of λ_j for $j = 1, \dots, n$ and put $\lambda_{n+1} = 0$ and $\lambda_{n+2} = 0$. Accordingly, we have the following result.

Theorem 1 *For any $\rho = (\rho_1, \dots, \rho_n) \geq 0$ and corresponding $\rho' = (\rho_1, \dots, \rho_n, 0, 0)$, we have that*

$$L(\rho) = \min_{0\leq t\leq P} F_{\rho'}(n,t) + \min_{0\leq t\leq P} G_{\rho'}(n,t) + \sum_{j=1}^{n} \rho_j.$$

□

In fact, we can strengthen the lower bound $L(\rho)$ in the following way. Note that we can add to the Lagrangean problem the constraint

$$\sum_{s\in\mathcal{E}} \sum_{j=1}^{n} a_{js}x_s + \sum_{s\in\mathcal{T}} \sum_{j=1}^{n} a_{js}x_s = \sum_{j=1}^{n} p_j, \tag{8}$$

which simply stipulates that the length of the early schedule plus the length of the tardy schedule should be exactly equal to the sum of the processing times. Clearly, this constraint is redundant for problem (P). And while remarkably enough it is also redundant for the linear programming relaxation of (P), it is not redundant for the Lagrangean problem. What is more, this additional constraint does not complicate the solution algorithm for the Lagrangean problem. Let $L'(\rho)$ be the minimum solution value of the Lagrangean problem (L_ρ) with the extra constraint (8). Then we have that

$$L'(\rho) = \min_{0 \leq t \leq P} \{F_{\rho'}(n,t) + G_{\rho'}(n, P-t)\} + \sum_{j=1}^{n} \rho_j. \qquad (9)$$

Of course, we can try and solve the *Lagrangean dual problem*

$$\max\{L'(\rho) \mid \rho \geq 0\}$$

to find the best possible Lagrangean lower bound, which is equal the to the linear programming bound in this case. In theory, this problem can be solved to optimality by the subgradient method or the bundle method, which are iterative procedures for updating the Lagrangean multipliers.

Accordingly, the basic difference between column generation and Lagrangean relaxation lies in the way the dual or Lagrangean multipliers are adjusted. The advantages of using Lagrangean relaxation to solve the linear programming relaxation are that (i) it is very easy to implement — there is no need to use linear programming to compute the next set of multipliers; (ii) it gives a lower bound in each iteration, while the column generation method gives a lower bound only upon convergence; (iii) no solutions need be stored. However, column generation has much better convergence properties. The main handicap of the subgradient method is that in practice it works as a non-polynomial approximation algorithm, since the theoretical conditions for convergence are so stringent that they cannot be observed in practice. Furthermore, it is typically impossible to establish whether the subgradient method has converged to an optimal vector of Lagrangean multipliers. This handicap is particularly inconvenient in our application, since in our earlier work on this problem we experienced that the linear programming relaxation was tight for all our randomly generated instances (Van den Akker *et al.*, 1996). In this respect, we insist on solving the linear programming relaxation to optimality.

A much more rewarding venue is to combine column generation and Lagrangean relaxation, as we show in the next subsection. In this way, we can still compute the best lower bound possible, compute lower bounds in each iteration, and use these lower bounds to control the size of the linear programming problem, to alleviate the tailing-off effect, and to speed up the pricing algorithms.

3.3 Combining column generation and Lagrangean relaxation

The main handicap of using column generation is that we have no valid lower bound on the optimal solution value until convergence, that is, until there are provably no columns with negative reduced cost any more. For problems with a severe tailing-off effect and much degeneracy, this is very inconvenient. The advantage of Lagrangean relaxation, however, is that we have a lower bound in each step of the column generation algorithm. In this section, we show how to combine them to both alleviate the tailing-off effect and speed up the pricing algorithms. We do not try to solve the Lagrangean dual problem; we simply compute the strengthened Lagrangean lower bound for the linear programming dual variables λ_j $(j = 1, \dots, n)$ — we denote this bound by $L'(\lambda)$. We use the Lagrangean lower bound in four different ways to speed up the column generation algorithm. We discuss these speedups for the root node of the branch-and-price algorithm; the speedups for the other nodes can easily be derived from this discussion. The Lagrangean lower bound is used to try to

1. *Prove optimality of the current solution*

   Clearly, if we have that $L'(\lambda) > UB - 1$ (recall that the outcome values are integral) for some $\lambda \geq 0$, where UB is the solution value of a feasible solution for the original scheduling problem, then we need no further bother about convergence of the column generation algorithm — we have then found a provably optimal solution for the scheduling problem.

   As mentioned earlier, this application of the Lagrangean lower bound in a colum generation algorithm is known (see for instance Vanderbeck and Wolsey (1996)), and the effect is only modest in the root node, since it applies only in the very tail of the convergence process. In the other nodes, however, this application may be very effective; see for instance Gademann and Van de Velde (1998).

2. *Fix variables*

   From standard linear programming theory, we know that if $c'_s > UB - L'(\lambda) - 1$, then $x_s = 0$ in any solution with solution value less than UB, if such a solution exists. Accordingly, we can remove with impunity all columns s from the column pool $\bar{S}$ for which this condition holds.

   The value of this type of variable fixing for controlling the growth of the column pool $\bar{S}$ is marginal. We could also have used one of the many rule-of-thumb *column management techniques* (see for instance Freling (1997)), such as the one that consistently removes all columns s for which c'_s is larger then some heuristically set positive threshold value δ.

   By the same token, we know that if $c'_s < L'(\lambda) + 1 - UB$ and $x_s = 1$ in the current solution, then $x_s = 1$ in any solution with solution value less than UB, if such a solution exists. Accordingly, if this is the case, then we have identified an optimal solution to the scheduling problem: column s together with its complimentary schedule then constitutes an optimal schedule.

   Anticipating on our computational results, we found that this type of variable fixing seldom appeared in practice.

3. *Restrict the range of the state variable t in the dynamic programming recursion for the pricing algorithms*

   If we have for some t and for some $\lambda \geq 0$ that

   $$F_\lambda(n,t) + G_\lambda(n, P-t) + \sum_{j=1}^{n} \lambda_j > UB - 1,$$

   then we may conclude that there is no solution with the length of the early schedule equal to t (and hence the length of the tardy schedule equal to $P - t$) that is better than our best schedule in hand. If we now can derive that this holds for all $t \geq t_0$, for some t_0 as small as possible, then we can speed up the pricing algorithms by restricting the range of the state variable t, without loosing any optimal solution to the pricing problem.

   This is a most effective trick to speed up the pricing algorithm. Note that the speed up can be no more than a factor two.

4. *Fix jobs to either the early or tardy set.*

   Fixing jobs is done by the following type of sensitivity analysis. Consider any job J_k and let $z_k(\mathcal{T})^*$ be the optimal solution value for our earliness-tardiness problem subject to the condition that job J_k is tardy. For any $\lambda \geq 0$, it is straightforward to compute a lower bound on $z_k(\mathcal{T})^*$ by slight adjustments of the two pricing algorithms. If we require that J_k is tardy, then we compute $F_\lambda(k,t)$ as

$$F_\lambda(k,t) = F_\lambda(k-1,t), \quad \text{for } t = 0, \ldots, \sum_{i=1}^{k} p_i$$

   and $G_\lambda(k,t)$ as

$$G_\lambda(k,t) = G_\lambda(k-1, t-p_k) + \beta_k t - \lambda_k, \quad \text{for } t = 0, \ldots, \sum_{i=1}^{k} p_i.$$

   Hence, if $z_k(\mathcal{T})^* > UB - 1$, then we know that J_k must be early in any optimal schedule, if the best schedule found thus far is not optimal.

   In a similar fashion, of course, we can test if J_k can be early.

   The purpose of this test is twofold: to restrict the size of the column pool, and more importantly, to speed up the regular pricing algorithm. To see this, suppose for instance that we have established that some J_k must be tardy. First of all, we can then remove from the column pool all early schedules that contain J_k as well as all tardy schedules do not contain J_k. Furthermore, the consequence for the tardy pricing algorithm is that the option of scheduling J_k early need no longer be evaluated. The consequence for the early pricing algorithm is even more drastic and time-saving: J_k need no longer be part of the recursion at all.

   This type of sensitivity analysis is quite time-consuming; it seems therefore prudent to perform it for the first time only when the gap $(UB - L'(\lambda) - 1)$ has become relatively small, and after the first time, to perform it only once every so many iterations.

4 Computational results

In this section, we report on our computational experience with our combined column generation and Lagrangean relaxation algorithm for randomly generated instances; in the remainder, we refer to it as the *combined algorithm.* We first discuss the implementation issues, then give a sketchy description of our implementation of the combined algorithm, and finally report on and analyze our computational results.

4.1 Implementation issues

The algorithms were coded in the computer language C, and the experiments were conducted on an HP9000/710 Unix machine. We used the package CPLEX (CPLEX Optimization, 1990) to solve the linear programs.

The main implementation issues involved are:

— The design of a heuristic to generate the initial set $\bar{S}$ of early and tardy schedules;

— The size of the initial set $\bar{S}$, that is, the number of columns to be generated by the heuristic;

— The columns to add to the linear program per iteration.

The first implementation issue is the design of a heuristic for generating initial columns to compute the initial dual variables with which we start the column generation method. We use a simple iterative improvement procedure for this purpose, which works as follows. First, we generate a feasible solution by deciding randomly whether a job is scheduled early or tardy. Then, we compute the corresponding V-shaped schedule and we search the neighborhood of the current schedule for a better V-shaped schedule. The neighborhood of a V-shaped schedule consists of all V-shaped schedules that can be obtained by three types of *changing operations*: moving a tardy job to the early schedule; moving an early job to the tardy schedule; and swapping an early and a tardy job. As soon as we find a better schedule in the current neighborhood, we adopt it as the new schedule. This process is repeated and terminates when no further improvement can be found.

The second issue is the number of initial solutions to be generated by the heuristic. Note that multiple initial solutions can be obtained by repetitive use of the heuristic described above. In case of a small number, the initial dual variables may be a long shot away from the optimal dual variables; in case of a large number, the size of the linear programs may outweigh the benefit of having better initial dual variables. Our computational experiments indicated that running the heuristic between 20 and 50 times, depending on the number of jobs in the instance, each time with a different starting solution, was a fairly robust choice. We have not tried to finetune this number further with respect to size or any other characteristic of an instance.

The third issue is how many and which schedules (or columns) to add to the set $\bar{S}$ in each iteration. The dilemma we are facing is that the more schedules we add per iteration, the fewer linear programs we (probably) need to solve — which is good; but the more schedules we add per iteration, the bigger the linear programs become — which is bad. In the previous section, we noted that each value $F_n(t) < 0$ and $G_n(t) < 0$ corresponds to a column with negative reduced cost. Hence, using the pricing algorithm, we can determine as many early and tardy schedules with negative reduced cost as there are values t for which $F_n(t) < 0$ or $G_n(t) < 0$, at the expense of a little extra effort. In our computation results, however, this turned out to be not worthwhile. Accordingly, per iteration we add no more than one tardy and one early schedule. There is one exception to this rule, however: If a schedule with minimum reduced cost together with its complementary schedule constitute a better primal solution than the incumbent upper bound, then both schedules are added to $\bar{S}$.

This exception also underlines the great importance of the pricing algorithms for finding better and better feasible solutions to the original scheduling problem. Anticipating on the section with our computational results for randomly generated instances, the iterative local improvement heuristic served its purpose by finding reasonable feasible solutions quickly, but the pricing algorithms always found an optimal solution on the fly.

4.2 The combined algorithm

In this subsection, we give a sketchy description of our implementation of the combined algorithm — we found this implementation the most robust.

COMBINED ALGORITHM

STEP 1. Use the iterative improvement heuristic with between 20 and 50 different starting solutions to generate the initial set $\bar{S}$ of early and tardy schedules.

STEP 2. Solve the linear programming relaxation.

STEP 3. Run both pricing algorithms to determine the early schedule and the tardy schedule with minimum negative reduced cost.

STEP 4. If neither an early, nor a tardy schedule with negative reduced cost exists, then go to STEP 8. If such an early schedule exists, then determine its complementary tardy schedule. If together they constitute a better feasible solution than we have right now, then record the solution and its corresponding solution value, say UB, and add both the early schedule and its complementary tardy schedule to the set $\bar{S}$. If they do not form a better feasible schedule, then just add the early schedule. The same procedure applies to the tardy schedule with minimum negative reduced costs, if it exists.

STEP 5. Compute the Lagrangean lower bound $L'(\lambda)$ and try to restrict the range of the state variables of the pricing algorithms.

STEP 6. Once every n iterations, do the following: first, perform a sensitivity analysis to try and fix jobs, if $L'(\lambda)/UB > 0.99$; and second, remove each tardy (early) column s with $c_s > UB - L'(\lambda) - 1$ or with at least one job that should be early (tardy).

STEP 7. Return to STEP 2.

STEP 8. Stop the column generation procedure. We have solved the linear programming relaxation to optimality. If $L'(\lambda) > UB - 1$, then we have solved the original schedule problem to optimality; if not, then we need to use a branch-and-price algorithm to solve the scheduling problem to optimality.

Furthermore, note that STEP 4 is also important for finding better feasible solutions.

4.3 Performance of the column generation algorithm

We tested our algorithm on four classes of randomly generated instances:

(i) instances with processing times and weights drawn from the uniform distribution [1, 100]. This concurs with the procedure used by De *et al.* (1994) to generate instances, which was also used by Van den Akker *et al.* (1996) and Chen and Powell (1997).

(ii) instances with processing times and weights drawn from the uniform distribution [1, 10]. These instances were also considered by Van den Akker *et al.* (1996) and Chen and Powell (1997).

(iii) instances where the processing times and the job weights are highly correlated. The processing times were drawn from the uniform distribution [10, 100], and each α_j and β_j was drawn from the uniform distribution $[p_j - 5, p_j + 5]$. In this way, the ratios α_j/p_j and β_j/p_j are all close to 1, which means that the jobs have about the same priority.

(iv) instances where the processing times and jobs weights are almost identical. The processing times and weights were drawn from the uniform distribution [90, 100]. This gives instances where the jobs have about the same processing requirement and where furthermore the weight to processing time ratios are all close to 1.

Instance classes (iii) and (iv) contain computationally hard instances, since all the weight to processing time ratios are close to each other. For instance classes (i) and (ii), we tested our algorithm on instances with $n = 10, 25, 50, 75, 100$ and 125 jobs; for instance classes (iii) and (iv), we went no further than instances with 100 jobs, since larger instances took on average too much time. For each combination of n and instance class we generated 100 instances.

For each value of n and for either instance class, we report on the number of times (out of 100) that the optimal linear programming solution was integral, the average computation time, the maximum computation time, the average number of columns generated, the maximum number of columns generated, the average number of linear programming problems solved, and the maximum number of linear programming problems solved.

Tables 1-4 summarize our computational results with the column generation algorithm for the respective instance classes. The headers of the columns are:

n	=	number of jobs;
OPT	=	number of instances out of 100 for which the linear programming solution was integral.
ACT	=	average computation time in seconds;
MCT	=	maximum computation time in seconds;
$ACOL$	=	average number of columns generated;
$MCOL$	=	maximum number of columns generated;
ALP	=	average number of linear programs solved;
MLP	=	maximum number of linear programs solved.

n	OPT	ACT	MCT	$ACOL$	$MCOL$	ALP	MLP
10	100	0.24	0.30	27	53	24	32
25	100	2.57	3.80	146	179	92	124
50	100	21.38	37.94	386	668	207	397
75	100	40.19	62.05	645	899	403	572
100	100	183.01	248.91	1058	1778	669	1591
125	100	467.21	882.87	1575	2168	904	1707

Table 1: Results for the class (i) instances.

n	OPT	ACT	MCT	$ACOL$	$MCOL$	ALP	MLP
10	100	0.10	0.16	44	41	13	21
25	100	1.72	2.40	113	151	56	74
50	100	10.06	12.41	347	397	223	219
75	100	26.88	46.09	541	725	325	481
100	100	61.01	193.21	870	1459	520	792
125	100	387.63	882.87	1344	2161	772	1001

Table 2: Results for the class (ii) instances.

An astonishing but very convenient phenomenon was that the linear programming solution turned out to be integral for each instance: $OPT = 100$

n	OPT	ACT	MCT	$ACOL$	$MCOL$	ALP	MLP
10	100	0.13	0.21	33	52	15	27
25	100	2.16	4.28	122	195	64	102
50	100	38.70	55.24	358	513	193	254
75	100	166.22	1169.10	1100	1409	591	783
100	100	627.16	870.73	1357	1811	763	1048

Table 3: Results for the class (iii) instances.

n	OPT	ACT	MCT	$ACOL$	$MCOL$	ALP	MLP
10	100	0.21	0.31	37	38	17	23
25	100	4.66	5.13	170	203	100	113
50	100	47.53	62.37	429	528	225	304
75	100	226.26	282.00	854	1075	442	562
100	100	823.83	928.71	1538	1669	760	815

Table 4: Results for the class (iv) instances.

for all n and all instance classes. What is more, the solution of each intermediate linear programming problem was always integral as well — and we have solved millions of these. This raises the question whether integrality of the optimal solution of the linear programming relaxation is a structural property. It is not, as was shown by a counterexample with only 5 jobs in Van den Akker *et al.* (1996).

Since the pricing algorithm requires pseudo-polynomial time, we can expect beforehand that the performance of our algorithm deteriorates with the size of the processing times of the jobs. Indeed, the class (ii) instances, with smaller processing times, are easier to solve than the class (i) instances. Furthermore, as expected, our algorithm has much more difficulty in solving the class (iii) and (iv) instances. These are indeed hard instances, for two reasons: the jobs are very similar, since they all have weight to processing time ratios close to one, and the cost difference between an early position and a tardy position is very small for all jobs. One effect is that the sensitivity analysis to try and fix a job to either the early or the tardy set is then less effective.

As a whole, our computational results show that using our algorithm we can solve larger problems to optimality than before: we solve instances with up to 125 jobs, while De *et al.* (1994) went no further than 40 jobs, and the two purely column generation algorithm by Chen and Powell (1997) and Van den Akker *et al.* (1996) can solve class (i) and class (ii) instances with up to 60 jobs only. We note that these algorithms were not tested out on class (iii) and (iv) instances, which are much more difficult to solve.

5 Concluding remarks

We have presented an effective column generation approach combined with Lagrangean relaxation elements for solving the problem of scheduling jobs around a large common due date with asymmetric weights. Using this method, we were able to solve instances with up to 125 jobs to optimality by solving the linear programming relaxation of a set covering formulation of the problem — branch-and-bound was never required for our randomly generated instances. The integrality gap can be positive, unbounded even, however. Note that, if necessary, our lower bounding approach can easily be used in a branch-and-bound algorithm in which we partition by putting some J_j either in the early, or in the tardy schedule; we have discussed the consequences of such a job fixing in Section 3.3.

The performance of our algorithm marks quite a computational progress for this problem; after all, two purely column generation algorithms could solve instances with up to only 60 jobs to optimality. The contribution of this paper is the structured way in which Lagrangean relaxation is embedded in the column generation algorithm. Before, the Lagrangean bound was primarily used to try and fathom nodes in a branch-and-price tree. In this paper, we have shown that there are more effective methods to use the Lagrangean bound; indeed, these methods not only speed up convergence but also speed up the pricing algorithms.

We believe that such a combination of column generation and Lagrangean relaxation, where the Lagrangean bound is used for other and more effective purposes than just fathoming nodes in a branch-and-price algorithm, is promising for other applications as well. Indeed, Gademann and Van de Velde (1998) show how Lagrangean relaxation can be embedded in a column generation algorithm for an order batching problem in a parallel-aisle warehouse.

The phenomenon that the randomly generated instances are easy in practice in the sense that the linear programming solution always seems to give an integral solution is intriguing. Hoogeveen *et al.* (1994) showed why randomly generated instances of the symmetric earliness-tardiness problem with unit penalty weights, which is NP-hard in general, can be expected to be computationally easy for large instances. For minimizing total weighted completion time on identical parallel machines, Chan *et al.* (1995) proved that the linear programming solution value of a set covering formulation of the problem is, under mild conditions, asymptotically optimal. But it is still an open question why randomly generated instances of the earliness-tardiness problem under study are computationally relatively easy.

Acknowledgement. The second and third author were supported by a grant from INTAS (Project INTAS-93-257 and INTAS-93-257-Ext).

References

[1] J.M. van den Akker, J.A. Hoogeveen, and S.L. van de Velde (1995). Parallel machine scheduling by column generation. *Operations Research*, to appear.

[2] J.M. van den Akker, J.A. Hoogeveen, and S.L. van de Velde (1996). *A column generation algorithm for common due date scheduling*, Working Paper LPOM-96-13, Faculty of Mechanical Engineering, University of Twente, Enschede, The Netherlands.

[3] K.R. Baker and G.D. Scudder (1989). On the assignment of optimal due dates. *Journal of the Operational Research Society 40*, 93-95.

[4] K.R. Baker and G.D. Scudder (1990). Sequencing with earliness and tardiness penalties: a review. *Operations Research 38*, 22-36.

[5] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance (1994). Branch-and-price: column generation for solving huge integer programs. *Operations Research*, to appear.

[6] L.M.A. Chan, P. Kaminsky, A. Muriel, and D. Simchi-Levi (1995). *Machine scheduling, linear programming and list scheduling*

heuristics, Manuscript, Department of Industrial Engineering, Northwestern University, Evanston.

[7] Z.L. CHEN AND W.P. POWELL (1995). *Solving parallel machine total weighted completion time problems by column generation*, Manuscript, Department of Civil Engineering and Operations Research, Princeton University.

[8] Z.L. CHEN AND W.P. POWELL (1997). *A decomposition approach for a parallel machine just-in-time scheduling problem*, Working Paper, Department of Civil Engineering & Operations Research, Princeton University.

[9] CPLEX OPTIMIZATION, INC. (1990). *Using the* CPLEX^{TM} *Linear Optimizer.*

[10] P. DE, J. GHOSH, AND C.E. WELLS (1994). Solving a generalized model for CON due-date assignment and scheduling. *International Journal of Production Economics*, 179-185.

[11] M. DESROCHERS, J. DESROSIERS, AND M. SOLOMON (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* , 40, 432-354.

[12] J. DESROSIERS, Y. DUMAS, M.M. SALOMON, AND F. SOUMIS (1995). Time constrained routing and scheduling. *Handbooks in Operations Research and Management Science 8*, Volume on Network Routing, Elsevier, 35-139.

[13] R. FRELING (1997). *Models and techniques for integrating vehicle and crew scheduling*, PhD Thesis, Erasmus University, Rotterdam, The Netherlands.

[14] R. FRELING, S.L. VAN DE VELDE, AND A. WAGELMANS (1998). Column generation for dummies: a practical guide. *European Journal of Operational Research*, to appear.

[15] A.J.R.M. GADEMANN AND S.L. VAN DE VELDE (1998). *A column generation algorithm for batch order picking in a parallel-aisle warehouse.* In preparation.

[16] N.G. HALL AND M.E. POSNER (1991). Earliness-tardiness scheduling problems, I: Weighted deviation of completion times about a common due date. *Operations Research 39*, 836-846.

[17] N.G. HALL AND C. SRISKANDARAJAH (1990). The earliness-tardiness problem with asymmetric weights. Presented at the TIMS/ORSA Joint National Meeting, Las Vegas.

[18] J.A. HOOGEVEEN, H. OOSTERHOUT, AND S.L. VAN DE VELDE (1994). New lower and upper bounds for scheduling around a small common due date. *Operations Research 42*, 102-110.

[19] A. MEHROTRA AND M.A. TRICK (1997). A column generation approach for graph coloring. *INFORMS Journal on Computing 8*, 344-354.

[20] P.M. PARDALOS AND G.P. RODGERS (1990). Computational aspects of a branch-and-bound algorithm for zero-one programming. *Computing 45*, 131-144.

[21] M.A. QUADDUS (1987). A generalized model of optimal due date assignment by linear programming. *Journal of the Operational Research Society 38*, 353-359.

[22] M.W.P. SAVELSBERGH (1997). A branch-and-price algorithm for the generalized assignment problem. *Operations Research 45*, 831-841.

[23] F. SOUMIS (1997). Decomposition and column generation, Chapter 8 in *Annotated Bibliography of Combinatorial Optimization* (eds. M. DELL'AMICO, F. MAFFIOLI, AND S. MARTELLO), Wiley, Chichester.

[24] F. VANDERBECK (1998). On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, to appear.

[25] F. VANDERBECK AND L.A. WOLSEY (1996). An exact algorithm for IP Column generation. *Operations Research Letters 19*, 151-159.