INFORMS Journal on Computing
Vol. 12, No. 2, Spring 2000

0899-1499/100/1202-0111 $05.00

Time-Indexed Formulations for Machine Scheduling Problems: Column Generation

J.M. VAN DEN AKKER / *Information and Communication Technology Division, National Aerospace Laboratory NLR, P.O. Box 90502, 1006 BM Amsterdam, The Netherlands, Email: vdakker@nlr.nl*

C.A.J. HURKENS / *Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, Email: wscor@win.tue.nl*

M.W.P. SAVELSBERGH / *School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0205, Email: martin.savelsbergh@isye.gatech.edu*

(Received March 1996; revised November 1997, December 1998; accepted August 1999)

Time-indexed formulations for machine scheduling problems have received a great deal of attention; not only do the linear programming relaxations provide strong lower bounds, but they are good guides for approximation algorithms as well. Unfortunately, time-indexed formulations have one major disadvantage—their size. Even for relatively small instances the number of constraints and the number of variables can be large. In this paper, we discuss how Dantzig-Wolfe decomposition techniques can be applied to alleviate, at least partly, the difficulties associated with the size of time-indexed formulations. In addition, we show that the application of these techniques still allows the use of cut generation techniques.

In this paper, we study computational issues related to the use of time-indexed formulations for the solution of single-machine scheduling problems. Although we concentrate on single-machine scheduling problems, we hope that our results provide insights that will be applicable when time-indexed formulations are used to solve more complex machine scheduling problems with many machines and many restrictions such as release dates, deadlines, time windows, machine downtime constraints, etc.

Time-indexed formulations are used in optimization as well as in approximation algorithms for machine scheduling problems. Because the bound provided by the solution to the LP-relaxation of a time-indexed formulation is very strong, stronger than the bounds provided by the LP-relaxations of many other integer programming formulations (Dyer and Wolsey,[5] and Queyranne and Schulz[11]), it is a natural candidate to be used in a branch-and-bound or branch-and-cut algorithm (de Sousa[16]; de Sousa and Wolsey[17]; Lee and Sherali[9]; Crama and Spieksma[4]; and van den Akker et al.[1]). Since the value of the LP-relaxation of a time-indexed formulation provides a strong bound, we may also expect that the solution to the LP-relaxation provides useful information to guide list-scheduling algorithms. Computational experience with list-scheduling algorithms guided by the solution to the LP-relaxation of time-indexed formulations has been very positive (van den Akker et al.[1]; and Savelsbergh et al.[13]). Recently, several researchers (Phillips et al.[10]; Goemans[6]; and Hall et al.[7]) have shown that list-scheduling algorithms guided by the solution to the LP-relaxation of time-indexed formulations have constant worst-case performance ratios for certain single-machine and parallel machine scheduling problems; in fact, for some of these scheduling problems, these are the only approximation algorithms known that have a constant performance ratio.

Unfortunately, the promising computational results reported in the papers mentioned above have all been for relatively small instances. This brings us to the main weakness of time-indexed formulations—their size. Even for relatively small instances the number of constraints and the number of variables can be huge. As a result, the memory required to store an instance and the time required to solve just the LP-relaxation may be prohibitively large. Therefore, for time-indexed formulations to have a major impact on optimization as well as on approximation algorithms for machine scheduling problems, we need to find ways to reduce the memory requirements and the solution times of the LP-relaxation. In this paper, we show that Dantzig-Wolfe decomposition in combination with column generation can be used effectively to alleviate, at least partly, the difficulties associated with the size of time-indexed formulations. Furthermore, we show that it is possible, using the Dantzig-Wolfe reformulation, to quickly compute a very good approximate solution to the LP-relaxation.

Dantzig-Wolfe decomposition is a well-known technique that has been applied successfully to solve large-scale structured linear programs. The application of Dantzig-Wolfe decomposition results in a reformulation of the linear program with far fewer constraints but many more variables. The large number of variables does not pose a real problem, however, since we can use a column generation technique.

Lagrangian relaxation provides an alternative to the combination of Dantzig-Wolfe decomposition and column generation. Lee and Sherali[9] and Sherali et al.[15] employ this technique to derive good lower and upper bounds for a scheduling problem with parallel unrelated machines, subject to time windows and machine downtime constraints.

Subject classifications: Integer programming, column generation, machine scheduling.
Other key words: Scheduling, time-indexed formulation, Dantiz-Wolfe decompositon, column generation

It is well-known that valid inequalities and especially facet-inducing inequalities of the polyhedron associated with the set of feasible solutions to an integer program can often be used effectively to improve the bound provided by the linear programming relaxation. The polyhedral structure of the set of feasible solutions to the time-indexed formulation of single-machine scheduling problems has been studied extensively (de Sousa[16]; de Sousa and Wolsey[17]; van den Akker[2]; Crama and Spieksma[4]; and van den Akker et al.[1]), and it has been shown that the bounds provided by the LP-relaxation of the time-indexed formulation indeed can be improved and that the use of these improved bounds leads to more robust branch-and-cut algorithms.

Adding cutting planes strengthens the bound provided by the linear programming relaxation, but to compute this bound we have to solve a huge LP problem again. This LP problem is in fact even larger than the LP-relaxation without cuts. This leads us to the important question of whether column generation and cut generation techniques can be combined. In this paper, we show that indeed this is possible not only for time-indexed formulations of single-machine scheduling problems but also in more general situations. To the best of our knowledge, this is one of the few studies in which this difficult but important issue is covered in some detail.

The paper is organized as follows. In Section 1, we review the time-indexed formulation for single-machine scheduling problems. In Section 2, we present, analyze, and experiment with the reformulation obtained by applying Dantzig-Wolfe decomposition. In Section 3, we show how column generation techniques can be combined with cut generation techniques. In Section 4, we present some conclusions and extensions.

1. A Time-Indexed Formulation for Single-Machine Scheduling Problems

A time-indexed formulation is based on time-discretization, i.e., time is divided into periods, where period t starts at time $t-1$ and ends at time t. The planning horizon is denoted by T, which means that we consider the time-periods $1, 2, \ldots, T$. We consider the following time-indexed formulation for single-machine scheduling problems:

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt},$$

subject to

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \qquad (j = 1, \ldots, n), \tag{1}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leqslant 1 \qquad (t = 1, \ldots, T),$$

$$x_{jt} \in \{0, 1\} \qquad (j = 1, \ldots, n;\ t = 1, \ldots, T - p_j + 1), \tag{2}$$

where the binary variable x_{jt} for each job j ($j = 1, \ldots, n$) and time period t ($t = 1, \ldots, T - p_j + 1$) indicates whether job j starts in period t ($x_{jt} = 1$) or not ($x_{jt} = 0$). The assignment constraints (1) state that each job has to be started exactly once, and the capacity constraints (2) state that the machine can handle at most one job during any time period.

An important advantage of the time-indexed formulation is that it can be used to model many types of single-machine scheduling problems. Different objective functions can be modeled by appropriate choices of cost coefficients, and constraints such as time windows and machine downtime constraints can be handled simply by fixing certain variables to zero. Moreover, the LP-relaxation of the time-indexed formulation provides a strong bound: it dominates the bounds provided by other mixed-integer programming formulations.

The main disadvantage of the time-indexed formulation is its size: there are $n + T$ constraints and approximately nT variables, where T is at least $\Sigma_{j=1}^{n} p_j$. As a result, for instances with many jobs or jobs with large processing times, the memory requirements and the solution times will be large. This was confirmed by computational experiments with a branch-and-cut algorithm for the problem of minimizing the total weighted completion time on a single machine subject to release dates (van den Akker et al.[1]). An analysis of the distribution of the total computation time over the various components of the branch-and-cut algorithm revealed that most of the time was spent on solving linear programs. This is not surprising if we recall that the typical number of simplex iterations is proportional to the number of constraints, which in our formulation amounts to $n + T$.

2. Reformulation

We apply Dantzig-Wolfe decomposition techniques to obtain a reformulation in which the number of constraints is reduced from $n + T$ to $n + 1$ at the expense of many more variables. However, the huge number of variables does not pose a real problem because they can be handled implicitly by means of column generation techniques.

Dantzig-Wolfe decomposition can be applied to linear programs exhibiting the following structure:

$$\min cx$$

subject to

$$Ax \leqslant b,$$
$$x \in P,$$

where for presentational convenience we assume P is bounded and hence a polytope. The fundamental idea of Dantzig-Wolfe decomposition is that the polytope P is represented by its extreme points $x^1, \ldots, x^K$. Each vector $x \in P$ can be represented as

$$x = \sum_{k=1}^{K} \lambda_k x^k, \text{ with } \sum_{k=1}^{K} \lambda_k = 1,\ \lambda_k \geq 0,\ (k = 1, \ldots, K).$$

Substituting leads to the following problem with variables λ_k ($k = 1, \ldots, K$), which is known as the Dantzig-Wolfe master

problem:

$$\min \sum_{k=1}^{K} (cx^k)\lambda_k,$$

subject to

$$\sum_{k=1}^{K} (Ax^k)\lambda_k \leqslant b,$$

$$\sum_{k=1}^{K} \lambda_k = 1,$$

$$\lambda_k \geqslant 0 \ (k = 1, \ldots, K).$$

The reformulation has far fewer constraints but many more variables. Fortunately, a column generation technique allows us to handle variables implicitly rather than explicitly. We start with only a subset of the variables, i.e., the other variables are implicitly fixed at zero. After solving this restricted version of the master problem, we check whether the solution value may be improved by including some of the variables that are currently implicitly fixed at zero. This is the case if there exist variables with negative reduced costs. If such variables are found, they are added to the linear program, and the resulting linear program is reoptimized. The identification of variables with negative reduced costs is done by solving the so-called pricing problem

$$\min_{x \in P}[(c - \pi A)x - \alpha],$$

where (π, α) is an optimal dual solution to the LP-relaxation of the restricted master problem. The pricing problem will be discussed in more detail in Section 2.2. First, we investigate how Dantzig-Wolfe decomposition can be applied to the time-indexed formulation for single-machine scheduling problems.

2.1 The Dantzig-Wolfe Master Problem

The LP-relaxation of the time-indexed formulation is given by

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt}x_{jt}$$

subject to

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \qquad (j = 1, \ldots, n), \tag{3}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leqslant 1 \qquad (t = 1, \ldots, T),$$

$$x_{jt} \geqslant 0 \qquad (j = 1, \ldots, n;\ t = 1, \ldots\ T - p_j + 1). \tag{4}$$

As the constraints $Ax \leqslant b$, which we keep in the master problem, we take the assignment constraints (3); the capacity constraints (4) and the nonnegativity constraints describe the polytope P, which we express as the convex hull of its extreme points.

We now take a closer look at the extreme points of P. The polytope P is described by the system

$$\begin{pmatrix} D \\ -I \end{pmatrix} x \leqslant \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

where D represents the capacity constraints and I the nonnegativity constraints. Observe that a variable x_{js} occurs in the capacity constraint for time period t if and only if $s \leqslant t \leqslant s + p_j - 1$. This means that the column in D corresponding to x_{js} has a one in the positions $s, \ldots, s + p_j - 1$, i.e., the ones are in consecutive positions. Therefore, D is an *interval matrix*. Interval matrices are known to be totally unimodular (see, for example, Schrijver[14]). This implies that the matrix $\binom{D}{-I}$ describing the polytope P is also totally unimodular. Hence, the extreme points of P are integral.

Because the assignment constraints are not part of the description of P, the extreme points of P represent schedules that satisfy the capacity constraints but not necessarily the assignment constraints. Since the latter constraints state that each job has to be started exactly once, the extreme points of P represent schedules in which jobs can be started more than once, once, or not at all. In the sequel, we will refer to such schedules as *pseudo-schedules*.

Let x^k $(k = 1, \ldots, K)$ be the extreme points of P. Any $x \in P$ can be written as $\Sigma_{k=1}^{K} \lambda_k x^k$ for some nonnegative values λ_k such that $\Sigma_{k=1}^{K} \lambda_k = 1$. The master problem can now be expressed as

$$\min \sum_{k=1}^{K} \left(\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt}x_{jt}^k \right) \lambda_k$$

subject to

$$\sum_{k=1}^{K} \left(\sum_{t=1}^{T-p_j+1} x_{jt}^k \right) \lambda_k = 1 \qquad (j = 1, \ldots, n), \tag{5}$$

$$\sum_{k=1}^{K} \lambda_k = 1,$$

$$\lambda_k \geqslant 0 \qquad (k = 1, \ldots, K). \tag{6}$$

Observe that the coefficient $\Sigma_{t=1}^{T-p_j+1} x_{jt}^k$ of the variable λ_k in the jth row of (5) is precisely the number of times that job j occurs in the pseudo-schedule x^k. This means that the entries of the column corresponding to the pseudo-schedule x^k equal the number of times each job occurs in this schedule. The cost coefficient of the variable λ_k is equal to the cost of the pseudo-schedule x^k.

Example. Consider the following two-job example with $p_1 = 2$ and $p_2 = 3$. If we have a pseudo-schedule x^k given by $x_{11}^k = x_{13}^k = 1$, then the variable λ_k corresponding to this pseudo-schedule has cost coefficient $c_{11} + c_{13}$ and column $(2, 0, 1)^T$, where the one in the last entry stems from the convexity constraint (6).

The reformulation has decreased the number of constraints from $n + T$ to $n + 1$. On the other hand, the number

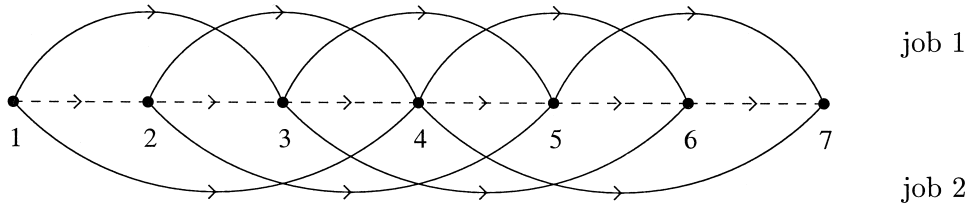

Figure 1. The network N for a 2-job example.

of variables has been increased significantly. Fortunately, the huge number of variables does not pose a real problem, because we can deal with it by using column generation techniques.

2.2 The Pricing Problem

Recall that when column generation techniques are used to solve a linear program only a subset of the variables is taken into account explicitly. In each iteration, this subset of variables is extended by adding one or more variables with negative reduced cost. It is impossible to identify a variable with negative reduced cost simply by enumerating over the variables currently not included in the restricted master problem because each of them corresponds to an extreme point of the polytope P and there are too many extreme points. Therefore, we try to identify a variable with minimum reduced cost by solving an optimization problem over all variables. This problem is called the *pricing problem*.

It follows directly from the theory of linear programming that the reduced cost $\bar{c}_k$ of a variable λ_k is given by

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k - \sum_{j=1}^{n} \pi_j \left(\sum_{t=1}^{T-p_j+1} x_{jt}^k \right) - \alpha,$$

where π_j denotes the dual variable associated with the jth constraint of (5), and α denotes the dual variable of constraint (6). This can be rewritten as

$$\bar{c}_k = \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x_{jt}^k - \alpha. \tag{7}$$

Since α is a constant, we ignore this term when solving the pricing problem. Recall that each extreme point x^k represents a pseudo-schedule, i.e., a schedule in which the capacity constraints are met but in which jobs do not have to start exactly once. If we take a closer look at the structure of a pseudo-schedule, then we see that it can be represented by a path in a network N. This network N has a node for each of the time periods $1, 2, \ldots, T + 1$ and two types of arcs—process arcs and idle time arcs. A *process arc* corresponds to the use of the machine. For each job j and each period t, with $t \leqslant T - p_j + 1$, there is a process arc from t to $t + p_j$ representing that the machine processes job j during the time periods $t, \ldots, t + p_j - 1$. We say that this arc *refers to job-start* (j, t). An *idle time arc* corresponds to the machine being idle. There is an idle time arc from t to $t + 1$ for each time period t, representing that the machine is idle in period t. The path corresponding to pseudo-schedule x^k contains an arc referring to job-start (j, t) for each component x_{jt}^k of x^k with $x_{jt}^k = 1$ complemented by idle time arcs. From now on we refer to this path as path P_k. Note that the correspondence between the extreme points x^k and paths in the network N can also be established directly by observing that the matrix D is a network matrix with associated network N.

Example (continued). Figure 1 depicts the network for our 2-job example with $p_1 = 2$, $p_2 = 3$, and $T = 6$.

If we set the length of the arc referring to job-start (j, t) equal to $c_{jt} - \pi_j$, for all j and t and we set the length of all idle time arcs equal to 0, then the reduced cost of the variable λ_k is precisely the length of path P_k minus the value of dual variable α. Therefore, solving the pricing problem corresponds to finding the shortest path in the network N with arc lengths defined as above. Since the network is directed and acyclic, the shortest path problem (and thus the pricing problem) can be solved in $O(nT)$ time by dynamic programming. When solving the pricing problem, we find one shortest path, which corresponds to a pseudo-schedule and hence to a variable with associated column. We add this one variable to the restricted master problem if its reduced cost is negative.

Observe that the optimal solution of the master problem is given in terms of the variables λ_k and that the columns only indicate how many times each job occurs in the corresponding pseudo-schedule. Therefore, to derive the solution in terms of the original variables, we have to maintain the pseudo-schedules corresponding to the columns. Note that, as described in the solution algorithm of the pricing problem, one column is added in each iteration.

Example. Consider the following three-job example with $p_1 = 2$, $p_2 = 3$, $p_3 = 3$, and $T = 8$. Suppose that the solution in terms of the reformulation has $\lambda_k = \frac{1}{2}$ for the columns $(1, 2, 0, 1)^T$ and $(1, 0, 2, 1)^T$, where the first column corresponds to the pseudo-schedule $x_{11}^1 = x_{23}^1 = x_{26}^1 = 1$ and the second one corresponds to the pseudo-schedule $x_{31}^2 = x_{34}^2 = x_{17}^2 = 1$. In terms of the original formulation this solution is given by $x_{11} = \frac{1}{2}$, $x_{17} = \frac{1}{2}$, $x_{23} = \frac{1}{2}$, $x_{26} = \frac{1}{2}$, $x_{31} = \frac{1}{2}$, and $x_{34} = \frac{1}{2}$.

Remark. The optimal LP solution found by the column generation algorithm may or may not correspond to an extreme point of the original formulation. The LP solution of the above example is in fact an extreme point. However, if in this example the pseudo-schedule corresponding to the second column is replaced by $x_{11}^2 = x_{33}^2 = x_{36}^2 = 1$, then the corresponding LP solution in the original formulation is $x_{11} = 1$, $x_{23} = \frac{1}{2}$, $x_{26} = \frac{1}{2}$, $x_{33} = \frac{1}{2}$, $x_{36} = \frac{1}{2}$, which is a convex combination of the feasible schedules $x_{11} = x_{23} = x_{36} = 1$ and $x_{11} = x_{26} = x_{33} = 1$.

2.3 Computational Validation

We have tested the performance of the column generation algorithm on the LP-relaxation of the time-indexed formulation for the problem of minimizing total weighted completion time on a single machine subject to release dates. We report results for 12 sets of 5 randomly generated instances. Half of the instances have 20 jobs, the others have 30 jobs. The processing times are uniformly distributed in $[1, p_{\max}]$, where $p_{\max}$ equals 5, 10, 20, 30, 50, or 100. The weights and the release dates are uniformly distributed in [1, 10] and in $[0, \frac{1}{2}\Sigma_{j=1}^{n} p_j]$, respectively. We choose $T = \frac{3}{2}\Sigma_{j=1}^{n} p_j$. We have five instances for each combination of n and $p_{\max}$; these instances are denoted by R$n \cdot p_{\max} \cdot i$, where i is the number of the instance. Our computational experiments have been conducted with MINTO 2.0/CPLEX 3.0 and have been run on an IBM RS/6000 model 590. We have used CPLEX's primal simplex algorithm with steepest edge pricing to solve the linear programs when running the column generation algorithm.

The computational results are given in Tables Ia and Ib. These tables show the number of generated columns and the running time of the column generation algorithm (No. of Columns and Time CG, respectively), the time required to solve the LP-relaxation of the original formulation by CPLEX's primal simplex method (Simplex), and the time required to solve the LP-relaxation by CPLEX's barrier method (Barrier). All running times are in seconds.

A "?" in the tables indicates that there was insufficient memory. This only occurred with CPLEX barrier for instances with $(n, p_{\max})$ equal to (30, 100). For these instances the size of the matrix is approximately 2,300 × 56,000 and the number of nonzeros is approximately 2,900,000.

As expected, the computational advantage of the reformulation is apparent for those problems in which $T = \frac{3}{2}\Sigma_{j=1}^{n} p_j$ is large, i.e., for large values of n and $p_{\max}$. For both $n = 20$ and $n = 30$, the column generation algorithm for the reformulation is the fastest for $p_{\max} \geq 20$. The CPU time required by the column generation algorithm for the reformulation appears to grow very slowly with the horizon T.

Observe also that for a given number of jobs n the number of generated columns seems to be almost independent of T. Therefore, the increase in computation time can be fully attributed to the increase in execution time of the shortest path algorithm due to the increase in size of the underlying network.

Table Ia. Performance of the column generation, simplex, and barrier algorithms for the 20-job instances*

Problem	No. of Columns	Time CG	Simplex	Barrier
R20.5.1	349	3.42	0.87	1.14
R20.5.2	380	3.94	0.78	1.12
R20.5.3	360	3.25	0.50	0.88
R20.5.4	446	4.84	0.60	0.98
R20.5.5	342	3.34	0.67	1.04
R20.10.1	326	3.41	2.24	3.51
R20.10.2	291	2.84	2.12	3.37
R20.10.3	272	2.55	2.09	3.17
R20.10.4	251	2.17	1.37	2.04
R20.10.5	312	3.07	1.80	2.59
R20.20.1	358	4.83	7.53	9.87
R20.20.2	296	3.40	7.73	9.46
R20.20.3	292	3.62	10.91	12.17
R20.20.4	304	3.50	5.15	7.78
R20.20.5	300	3.58	8.49	10.74
R20.30.1	324	4.34	12.54	19.89
R20.30.2	381	5.32	10.24	13.50
R20.30.3	368	5.28	11.53	15.69
R20.30.4	464	7.57	9.78	13.77
R20.30.5	269	3.36	13.55	16.14
R20.50.1	337	6.00	46.54	60.28
R20.50.2	284	4.21	24.78	42.88
R20.50.3	283	4.95	46.49	62.27
R20.50.4	264	4.05	41.12	63.75
R20.50.5	365	6.91	45.93	82.44
R20.100.1	314	8.51	349.99	331.78
R20.100.2	412	12.13	158.21	254.62
R20.100.3	401	12.17	265.14	396.38
R20.100.4	418	11.57	165.42	306.14
R20.100.5	314	8.57	348.38	395.75

*No. of Columns, number of generated columns; Time CG, running time of the column generation algorithm; Simplex, time required to solve the LP-relaxation of the original formulation by CPLEX's primal simplex method; Barrier, time required to solve the LP-relaxation by CPLEX's barrier method.

2.4 Approximate Solutions

In the previous sections, we have demonstrated that Dantzig-Wolfe decomposition techniques can be applied to alleviate, at least to some extent, the computational drawbacks associated with the size of time-indexed formulations. However, it still takes a fairly long time to solve the linear programming relaxation of a moderately sized instance, e.g., it takes approximately 40 seconds for a 30-job instance with $p_{\max} = 100$. This is due to the slow convergence of column generation, i.e., it requires a large number of iterations to prove LP optimality. To save computation time, we would like to prematurely end the column generation algorithm as soon as the current value is close to the optimum value. Unfortunately, a column generation algorithm approaches the optimum value from above, i.e., at each iteration it gives an upper bound on the optimum value of the LP-relaxation, whereas we wanted to solve the LP-relaxation to find a lower bound on the integer programming optimum. Hence, ending the column generation prematurely may not give a valid lower bound on the integer programming optimum. However, Lasdon[8] and Vanderbeck and Wolsey[18] describe a simple and relatively easy method for computing a lower bound on the optimum value of the LP-relaxation in each iteration of the column generation algorithm.

Table Ib. Performance of the column generation algorithm, CPLEX simplex, and CPLEX barrier for the 30-job instances*

Problem	No. of Columns	Time CG	Simplex	Barrier
R30.5.1	655	13.58	1.66	2.28
R30.5.2	532	9.06	1.94	2.37
R30.5.3	524	8.67	1.61	2.60
R30.5.4	567	10.93	2.19	2.93
R30.5.5	642	13.68	1.35	2.11
R30.10.1	626	13.92	3.85	4.78
R30.10.2	720	16.46	3.46	5.15
R30.10.3	796	18.97	5.22	6.81
R30.10.4	602	12.78	4.49	5.33
R30.10.5	561	11.95	8.61	8.82
R30.20.1	747	23.46	42.15	27.97
R30.20.2	797	24.93	32.39	25.12
R30.20.3	577	15.58	21.29	25.39
R30.20.4	543	12.71	20.54	14.49
R30.20.5	613	16.19	33.18	30.58
R30.30.1	588	16.78	73.73	41.04
R30.30.2	691	24.94	107.82	72.43
R30.30.3	740	25.56	45.21	55.67
R30.30.4	640	20.50	79.23	46.42
R30.30.5	583	18.57	111.01	66.21
R30.50.1	565	22.67	516.41	195.46
R30.50.2	662	26.31	352.86	130.86
R30.50.3	560	20.47	367.94	130.06
R30.50.4	637	25.10	655.32	160.16
R30.50.5	583	21.77	672.11	152.02
R30.100.1	663	42.32	1373.30	?
R30.100.2	679	40.35	2764.67	?
R30.100.3	571	32.29	1716.26	?
R30.100.4	678	43.90	4565.80	?
R30.100.5	882	57.00	2595.57	?

*No. of Columns, number of generated columns; Time CG, running time of the column generation algorithm; Simplex, time required to solve the LP-relaxation of the original formulation by CPLEX's primal simplex method; Barrier, time required to solve the LP-relaxation by CPLEX's barrier method.

The lower bound is derived in the following way. Consider the linear programming relaxation of the master problem

$$z_{LP} = \min \sum_{k=1}^{K} \left(\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k \right) \lambda_k$$

subject to

$$\sum_{k=1}^{K} \left(\sum_{t=1}^{T-p_j+1} x_{jt}^k \right) \lambda_k = 1 \qquad (j = 1, \ldots, n), \tag{8}$$

$$\sum_{k=1}^{K} \lambda_k = 1,$$

$$\lambda_k \geqslant 0 \qquad (k = 1, \ldots, K). \tag{9}$$

By dualizing constraints (8) with dual variables π_j, we obtain the following lower bound on z_{LP}:

$$\min \sum_{k=1}^{K} \left(\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k \right) \lambda_k + \sum_{j=1}^{n} \pi_j \left(1 - \sum_{k=1}^{K} \left(\sum_{t=1}^{T-p_j+1} x_{jt}^k \right) \lambda_k \right)$$

subject to

$$\sum_{k=1}^{K} \lambda_k = 1,$$

$$\lambda_k \geqslant 0 \qquad (k = 1, \ldots, K).$$

Since $\Sigma_{j=1}^{n} \pi_j$ is a constant, we get that the lower bound on z_{LP} is equal to $\Sigma_{j=1}^{n} \pi_j$ plus the outcome of the following minimization problem:

$$\min \sum_{k=1}^{K} \left[\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x_{jt}^k \right] \lambda_k$$

subject to

$$\sum_{k=1}^{K} \lambda_k = 1,$$

$$\lambda_k \geqslant 0 \qquad (k = 1, \ldots, K).$$

Observe that the cost coefficient of λ_k, i.e., $\Sigma_{j=1}^{n} \Sigma_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x_{jt}^k$, equals $\bar{c}_k + \alpha$ (see (7)), i.e., the reduced cost of variable x^k plus the dual value associated with the convexity constraint. Therefore, for any set of dual values π, a lower bound on the optimal value of the linear program is given by

$$\sum_{j=1}^{n} \pi_j + \min_k \bar{c}_k + \alpha.$$

This allows us to generate at each iteration a lower bound on the value of the optimal solution of the linear programming problem. Furthermore, this lower bound is very cheap to compute since we have to compute the minimum reduced cost anyway. Because the column generation scheme approaches the optimal linear programming value monotonically from above, i.e., it provides an upper bound on the value of the optimal linear programming solution, we can compute at each iteration how close we are to the optimal value. Therefore, we can decide to stop if we are within a prespecified percentage of optimality.

We have conducted the following computational experi-

ment to show the effect of prematurely ending the column generation process on the computation times. We consider 10 sets of 10 randomly generated instances, half of them with 50 jobs and half of them with 100 jobs. In each instance the processing times are drawn from a uniform distribution in $[1, p_{\max}]$, where $p_{\max}$ equals 5, 10, 20, 50, or 100. The weights and release dates are uniformly distributed in [1, 10] and $[0, \frac{1}{2}\Sigma_{j=1}^{n} p_j]$, respectively. We put $T = \frac{3}{2}\Sigma_{j=1}^{n} p_j$. For the set of instances with 50 jobs, we have computed approximate LP solutions to within 0.5, 0.05, and 0.005% of optimality, as well as the optimal LP solution. For the set of instances with 100 jobs, we have computed approximate LP solutions to within 1.05, 0.05, and 0.005% of optimality. The results are presented in Tables II and III and clearly show the tailing-off effect: it takes relatively little time to generate an approximate solution with a solution value that is within a small percentage of optimality (1 or 0.5%), but it takes much longer to generate an optimal solution or an approximate solution that is almost optimal (within 0.005% of optimality).

3. Combining Column and Cut Generation

In the previous section, we have shown that for instances of moderate size the LP-relaxation of the time-indexed formulation can be solved efficiently by a column generation scheme for a reformulation obtained by applying Dantzig-Wolfe decomposition.

It is well-known that valid inequalities (especially facet-inducing inequalities of the polyhedron associated with the set of feasible solutions to an integer program) can often be used effectively to improve the bound provided by the linear programming relaxation. The polyhedral structure of the set of feasible solutions to the time-indexed formulation of single-machine scheduling problems has been studied extensively (de Sousa[16]; de Sousa and Wolsey[17]; van den Akker[2]; Crama and Spieksma[4]; and van den Akker et al.[1]) and it has been shown that the bounds provided by the LP-relaxation of the time-indexed formulation indeed can be improved significantly and that the use of these improved bounds leads to more robust branch-and-cut algorithms.

Therefore, the next natural step is to investigate whether the LP-relaxations that have to be solved after cuts have been added can also be solved efficiently by column generation techniques.

The main difficulty when combining column and cut generation is that the pricing problem may become much more complicated after the addition of extra constraints since each constraint that is added to the master problem introduces an extra term in the reduced cost, which might destroy the nice structure of the pricing problem.

3.1 Column and Cut Generation for Single-Machine Scheduling Problems

van den Akker et al.[1] present a complete characterization of all facet-inducing inequalities with integral coefficients and right-hand sides 1 and 2 for the time-indexed formulation. Inequalities with right-hand side 1 are denoted by $x(V) \leqslant 1$, which is a short notation for $\Sigma_{(j,t)\in V} x_{jt} \leqslant 1$. van den Akker et al.[1] show that for any facet-inducing inequality $x(V) \leqslant 1$, V is given by $\{(1, t) \mid t \in [l - p_1, u]\} \cup \{(j, t) \mid j \neq 1, t \in [u -$

Table IIa. Computing approximate linear programming solutions ($n = 50$): CPU time

Name	0.5%	0.05%	0.005%	Optimal
R50.5.1	19.50	52.64	129.81	162.14
R50.5.2	23.57	99.77	250.85	267.33
R50.5.3	19.78	51.35	114.15	141.99
R50.5.4	17.20	55.12	125.67	150.91
R50.5.5	24.20	150.25	489.18	488.58
R50.5.6	23.41	84.30	258.58	271.45
R50.5.7	16.18	37.25	63.99	63.92
R50.5.8	13.98	43.86	95.31	106.33
R50.5.9	19.23	70.73	163.70	183.86
R50.5.10	18.69	48.13	89.26	89.92
R50.10.1	17.67	48.69	183.11	254.04
R50.10.2	29.46	118.42	434.10	549.01
R50.10.3	23.42	58.84	117.68	131.73
R50.10.4	21.20	45.66	169.15	252.24
R50.10.5	25.40	78.41	254.24	289.83
R50.10.6	24.11	85.34	241.69	282.06
R50.10.7	23.77	50.28	89.47	91.25
R50.10.8	20.60	42.73	80.10	88.12
R50.10.9	23.83	52.22	136.09	161.38
R50.10.10	23.73	59.86	116.34	137.00
R50.20.1	28.06	66.12	131.40	227.95
R50.20.2	38.72	106.98	234.80	312.12
R50.20.3	32.58	74.10	141.50	161.61
R50.20.4	29.26	67.63	151.02	237.67
R50.20.5	30.12	77.22	172.05	263.20
R50.20.6	33.57	73.64	132.72	150.36
R50.20.7	29.91	54.90	77.48	87.06
R50.20.8	28.01	55.48	107.12	137.56
R50.20.9	33.62	101.21	240.67	282.34
R50.20.10	33.08	68.95	140.59	156.89
R50.50.1	55.17	105.00	214.51	301.62
R50.50.2	67.05	136.74	205.54	470.33
R50.50.3	59.34	103.91	174.43	203.79
R50.50.4	52.95	109.30	218.00	335.07
R50.50.5	59.94	157.02	281.37	354.44
R50.50.6	57.53	132.09	205.42	252.77
R50.50.7	59.03	105.88	172.14	227.70
R50.50.8	53.45	99.08	188.48	290.73
R50.50.9	55.77	99.39	152.79	170.63
R50.50.10	59.30	104.66	158.27	196.87
R50.100.1	100.99	181.60	287.89	462.73
R50.100.2	104.66	206.09	332.85	436.49
R50.100.3	101.53	192.99	298.00	368.04
R50.100.4	91.72	167.67	290.66	377.46
R50.100.5	89.87	189.34	364.16	449.73
R50.100.6	97.55	179.00	260.07	306.22
R50.100.7	99.38	159.57	235.90	294.57
R50.100.8	90.48	156.50	255.00	446.42
R50.100.9	98.86	176.68	302.15	336.17
R50.100.10	101.27	171.72	247.39	306.85

Table IIb. Computing approximate linear programming solutions ($n = 50$): number of generated columns

Name	0.5%	0.05%	0.005%	Optimal
R50.5.1	539	1023	1690	1861
R50.5.2	625	1547	2518	2606
R50.5.3	531	929	1476	1692
R50.5.4	489	953	1581	1761
R50.5.5	629	1920	3472	3472
R50.5.6	607	1361	2433	2497
R50.5.7	475	778	1079	1079
R50.5.8	435	855	1355	1440
R50.5.9	522	1126	1810	1943
R50.5.10	538	968	1403	1415
R50.10.1	442	819	1856	2167
R50.10.2	631	1656	3087	3392
R50.10.3	534	939	1426	1536
R50.10.4	510	835	1837	2288
R50.10.5	565	1157	2109	2266
R50.10.6	545	1167	2086	2239
R50.10.7	537	863	1243	1264
R50.10.8	480	772	1113	1185
R50.10.9	534	863	1493	1657
R50.10.10	543	970	1426	1590
R50.20.1	489	846	1338	1884
R50.20.2	625	1233	1968	2315
R50.20.3	545	935	1445	1595
R50.20.4	520	918	1602	2101
R50.20.5	540	1009	1669	1933
R50.20.6	557	945	1372	1501
R50.20.7	523	792	994	1084
R50.20.8	486	772	1155	1392
R50.20.9	562	1119	1937	2147
R50.20.10	561	915	1421	1543
R50.50.1	533	872	1478	1890
R50.50.2	650	1096	1465	2165
R50.50.3	564	864	1278	1439
R50.50.4	545	951	1567	2131
R50.50.5	569	1199	1777	2077
R50.50.6	557	1081	1486	1725
R50.50.7	580	899	1301	1623
R50.50.8	511	826	1318	1811
R50.50.9	554	856	1186	1291
R50.50.10	578	891	1198	1415
R50.100.1	562	916	1326	1967
R50.100.2	619	1064	1536	1922
R50.100.3	566	958	1360	1625
R50.100.4	554	922	1425	1761
R50.100.5	561	1032	1697	1958
R50.100.6	554	943	1272	1430
R50.100.7	579	862	1187	1409
R50.100.8	501	795	1181	1860
R50.100.9	578	929	1454	1571
R50.100.10	576	892	1198	1405

Table IIIa. Computing approximate linear programming solutions ($n = 100$): CPU time

Name	1%	0.5%	0.05%	0.005%
R100.5.1	175.46	261.15	1179.85	6570.11
R100.5.2	153.67	231.81	934.81	12615.76
R100.5.3	134.83	194.75	586.55	2522.78
R100.5.4	173.39	239.12	755.42	2283.96
R100.5.5	136.30	186.38	762.05	1986.75
R100.5.6	134.20	191.97	899.87	5856.50
R100.5.7	171.24	251.66	903.40	3580.77
R100.5.8	178.06	274.88	851.87	2570.50
R100.5.9	167.73	247.37	1169.02	6170.37
R100.5.10	170.02	240.73	667.57	1498.16
R100.10.1	217.61	309.52	1068.35	8614.64
R100.10.2	183.83	253.90	882.83	4239.11
R100.10.3	195.51	298.56	939.70	2586.66
R100.10.4	213.40	291.44	836.89	1853.17
R100.10.5	180.39	262.00	897.45	2453.00
R100.10.6	194.14	281.89	956.08	2109.04
R100.10.7	194.10	279.86	849.14	2063.82
R100.10.8	191.17	274.02	904.54	2201.51
R100.10.9	187.50	271.09	1051.64	17793.22
R100.10.10	204.85	290.19	916.33	3400.81
R100.20.1	279.57	401.04	1865.06	8979.14
R100.20.2	260.05	374.26	1117.24	3658.74
R100.20.3	253.71	369.29	1012.66	2143.37
R100.20.4	292.68	418.41	1299.04	2956.84
R100.20.5	246.72	371.13	1321.72	4982.90
R100.20.6	247.11	334.96	980.72	2758.39
R100.20.7	284.63	398.02	1211.66	2705.38
R100.20.8	271.45	344.62	1121.35	2453.73
R100.20.9	247.55	383.95	1419.19	5867.83
R100.20.10	267.93	385.52	1051.47	2331.91
R100.50.1	490.55	658.22	2030.47	4719.59
R100.50.2	479.60	627.24	1801.62	3509.10
R100.50.3	456.41	619.56	1595.22	3389.44
R100.50.4	535.43	685.18	1650.45	3650.04
R100.50.5	428.16	599.95	1885.96	4194.45
R100.50.6	456.54	600.82	1575.94	3500.14
R100.50.7	479.87	602.45	1633.92	3885.24
R100.50.8	460.03	573.29	1699.69	3642.93
R100.50.9	483.81	587.66	1506.94	3891.81
R100.50.10	480.82	594.51	1323.75	2590.05
R100.100.1	829.70	1064.98	2643.34	8293.94
R100.100.2	820.96	1072.32	2531.19	5389.13
R100.100.3	747.22	952.02	2227.23	4248.21
R100.100.4	956.96	1209.97	2408.11	4350.18
R100.100.5	745.41	962.64	2353.56	5681.04
R100.100.6	772.29	949.96	2207.00	4654.92
R100.100.7	882.10	1089.95	2253.28	4475.89
R100.100.8	775.42	990.99	2314.82	4884.82
R100.100.9	803.90	1033.83	2639.90	5857.13
R100.100.10	786.11	967.88	2331.77	4077.75

Table IIIb. Computing approximate linear programming solutions ($n = 100$): number of generated columns

Name	1%	0.5%	0.05%	0.005%
R100.5.1	1015	1228	3002	7502
R100.5.2	978	1177	2394	7635
R100.5.3	944	1121	1922	4083
R100.5.4	1027	1215	2087	4450
R100.5.5	914	1069	2130	3705
R100.5.6	918	1105	2658	6872
R100.5.7	1044	1264	2659	5402
R100.5.8	1034	1295	2840	5359
R100.5.9	1030	1245	3222	7117
R100.5.10	1018	1209	2059	3262
R100.10.1	1073	1280	2668	5957
R100.10.2	968	1151	2175	4680
R100.10.3	1032	1296	2387	4262
R100.10.4	1059	1256	2201	3489
R100.10.5	972	1182	2300	3829
R100.10.6	1036	1269	2657	4136
R100.10.7	1028	1240	2196	3441
R100.10.8	990	1205	2398	4125
R100.10.9	1018	1233	2338	7873
R100.10.10	1050	1260	2310	4480
R100.20.1	1051	1308	3281	8512
R100.20.2	1011	1249	2301	4245
R100.20.3	1045	1297	2263	3425
R100.20.4	1107	1354	2486	4008
R100.20.5	993	1271	2669	5153
R100.20.6	1003	1204	2324	4095
R100.20.7	1121	1354	2432	3910
R100.20.8	1028	1194	2395	3765
R100.20.9	1031	1321	2488	4857
R100.20.10	1052	1299	2178	3500
R100.50.1	1072	1324	2535	4741
R100.50.2	1080	1301	2353	3707
R100.50.3	1092	1344	2438	3883
R100.50.4	1194	1404	2468	3913
R100.50.5	1029	1299	2772	4316
R100.50.6	1052	1269	2427	4046
R100.50.7	1132	1319	2427	4036
R100.50.8	1021	1196	2407	4094
R100.50.9	1136	1320	2346	4139
R100.50.10	1093	1269	2110	3157
R100.100.1	1066	1305	2544	5360
R100.100.2	1094	1357	2494	4035
R100.100.3	1062	1300	2373	3709
R100.100.4	1205	1456	2380	3651
R100.100.5	1070	1309	2568	4396
R100.100.6	1039	1236	2335	3900
R100.100.7	1141	1364	2304	3690
R100.100.8	996	1225	2323	4009
R100.100.9	1170	1422	2678	4448
R100.100.10	1075	1265	2198	3432

$p_j, l]\}$, for some l and u with $l < u$ and some special job, which for presentational convenience is assumed to be job 1 and where the intervals $[l - p_1, u]$ and $[u - p_j, l]$ are defined as the sets of time periods $\{l - p_1 + 1, \ldots, u\}$ and $\{u - p_j + 1, \ldots, l\}$, respectively. Such an inequality can be represented by the diagram depicted in Figure 2.

Example. Consider a three-job problem with $p_1 = 4$, $p_2 = 4$, and $p_3 = 3$. The LP solution $x_{15} = x_{19} = x_{27} = x_{2,11} = \frac{1}{2}$, $x_{31} = 1$ violates the inequality with $l = 8$ and $u = 9$ given by the diagram depicted in Figure 3.

Suppose that we add such an inequality $x(V) \leqslant 1$ to the master problem. The reformulated inequality in terms of the variables λ_k is given by

$$\sum_{k=1}^{K} \left(\sum_{(j,t)\in V} x_{jt}^k \right) \lambda_k \leqslant 1.$$

If we add this reformulated inequality to the restricted master problem, then we need to extend the column of λ_k, which corresponds to the pseudo-schedule x^k, with the coefficient of λ_k in this inequality. Observe that this coefficient is equal to the number of arcs in path P_k that refer to job-start (j, t) for each $(j, t) \in V$; this number is readily determined. Therefore, it is easy to compute the coefficient of a reformulated inequality for the columns in the restricted master problem. The same holds for the entries of the columns that will be generated later on.

After a facet-inducing inequality $x(V) \leqslant 1$ has been added, the master problem becomes

$$\min \sum_{k=1}^{K} \left(\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k \right) \lambda_k$$

subject to

$$\sum_{k=1}^{K} \left(\sum_{t=1}^{T-p_j+1} x_{jt}^k \right) \lambda_k = 1 \qquad (j = 1, \ldots, n),$$

$$\sum_{k=1}^{K} \lambda_k = 1,$$

$$\sum_{k=1}^{K} \left(\sum_{(j,t)\in V} x_{jt}^k \right) \lambda_k \leqslant 1,$$

$$\lambda_k \geqslant 0 \qquad (k = 1, \ldots, K).$$

Denote the dual variable of the additional constraint by μ_V. The reduced cost of the variable λ_k is given by

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k - \sum_{j=1}^{n} \pi_j \sum_{t=1}^{T-p_j+1} x_{jt}^k - \alpha - \mu_V \sum_{(j,t)\in V} x_{jt}^k,$$

which can be rewritten as

$$\sum_{(j,t)\in V} (c_{jt} - \pi_j - \mu_V) x_{jt} + \sum_{(j,t)\notin V} (c_{jt} - \pi_j) x_{jt} - \alpha.$$

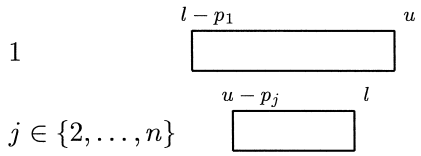

Figure 2. Diagram of a facet-inducing inequality $x(V) \leqslant 1$.

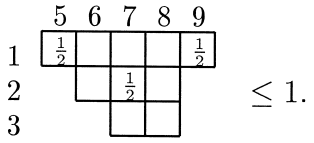

Figure 3. Example.

It is easy to see that the pricing problem now corresponds to determining the shortest path in the network N, where the length of the arc referring to job-start (j, t) equals $c_{jt} - \pi_j - \mu_V$ if $(j, t) \in V$ and $c_{jt} - \pi_j$ if $(j, t) \notin V$. The length of the idle time arcs is again equal to zero. In fact, the only difference with the original pricing problem is that μ_V has been subtracted from the length of the arcs referring to job-starts (j, t) for $(j, t) \in V$. If several constraints have been added, then the length of the arcs is modified in the same way for each constraint. Hence, the structure of the pricing problem does not change: it remains a shortest path problem on a directed acyclic graph. We conclude that we can combine column generation with the addition of facet-inducing inequalities with right-hand side 1.

Summarizing, we have shown that for each reformulated inequality the coefficient of λ_k is equal to the value of the left-hand side of the original inequality when the pseudo-schedule x^k is substituted. Furthermore, the dual variable associated with a reformulated inequality does not change the structure of the pricing problem; it only affects the objective function coefficients.

It is not hard to show that facet-inducing inequalities with right-hand side 2 can be handled similarly.

3.2 Column and Cut Generation for Decomposable Problems

In this subsection, we show that the above ideas and techniques can also be applied in other situations in which column generation is used to solve the LP-relaxation of a reformulation obtained through Dantzig-Wolfe decomposition and where *inequalities given in terms of variables of the original formulation* are added.

Consider the linear programming problem

$$\min cx$$

subject; to

$$A^{(1)}x \leqslant b^{(1)},$$
$$A^{(2)}x \leqslant b^{(2)},$$

where $c \in R^n$, $A^{(1)} \in R^{m_1} \times n$, and $b^{(1)} \in R^{m_1}$ and where, for presentational convenience, we assume that $A^{(2)}\ x \leqslant b^{(2)}$ describes a bounded set and hence a polytope. Let x^k $(k = 1, \ldots, K)$ be the extreme points of this polytope. The master problem obtained through Dantzig-Wolfe decomposition is as follows:

$$\min \sum_{k=1}^{K} \left(\sum_{j=1}^{n} c_j x_j^k \right) \lambda_k$$

subject to

$$\sum_{k=1}^{K} (A^{(1)}x^k)\lambda_k \leqslant b_i^{(1)} \qquad (i = 1, \ldots, m_1),$$

$$\sum_{k=1}^{K} \lambda_k = 1,$$

$$\lambda_k \geqslant 0 \qquad (k = 1, \ldots, K).$$

The reduced cost of the variable λ_k is equal to

$$\sum_{j=1}^{n} c_j x_j^k - \sum_{i=1}^{m_1} \pi_i \left(\sum_{j=1}^{n} a_{ij}^{(1)} x_j^k \right) - \alpha,$$

where π_i denotes the dual variable of the ith constraint and α denotes the dual variable of the convexity constraint. The pricing problem can hence be written as

$$\min \left\{ \sum_{j=1}^{n} \left(c_j - \sum_{i=1}^{m_1} \pi_i a_{ij}^{(1)} \right) x_j^k - \alpha \mid k = 1, \ldots, K \right\}.$$

Theorem 1. *If the pricing problem can be solved for arbitrary cost coefficients, i.e., if the algorithm for the solution of the pricing problem does not depend on the structure of the cost coefficients, then the addition of a valid inequality $dx \leqslant d_0$ in terms of the original variables does not complicate the pricing problem.*

Proof. In terms of the Dantzig-Wolfe reformulation, the inequality $dx \leqslant d_0$ is given by

$$\sum_{k=1}^{K} (dx^k)\lambda_k \leqslant d_0.$$

The coefficient of λ_k in the reformulated inequality is equal to the value of the left-hand side of the original inequality when the extreme point x^k is substituted. Observe that, after the addition of this inequality, the reduced cost is given by

$$\sum_{j=1}^{n} c_j x_j^k - \sum_{i=1}^{m_1} \pi_i \left(\sum_{j=1}^{n} a_{ij}^{(1)} x_j^k \right) - \alpha - \sigma \left(\sum_{j=1}^{n} d_j x_j^k \right),$$

where σ denotes the dual variable of the additional constraint. The pricing problem is hence given by

$$\min \left\{ \sum_{j=1}^{n} \left(c_j - \sum_{i=1}^{m_1} \pi_i a_{ij}^{(1)} - \sigma d_j \right) x_j^k - \alpha \mid k = 1, \ldots, K \right\}.$$

Table IVa. Combining row and column generation for the 20-job instances

Problem	No. of Columns	CPU	No. of Columns	CPU	No. of Inequalities	No. of Rounds
R20.5.1	349	3.42	349	3.42	0	0
R20.5.2	380	3.94	380	3.94	0	0
R20.5.3	360	3.25	678	14.15	8	1
R20.5.4	446	4.84	446	4.84	0	0
R20.5.5	342	3.34	470	6.85	1	1
R20.10.1	326	3.41	704	23.22	25	3
R20.10.2	291	2.84	680	22.87	25	3
R20.10.3	272	2.55	735	38.57	46	4
R20.10.4	251	2.17	1061	113.05	67	5
R20.10.5	312	3.07	875	46.89	37	2
R20.20.1	358	4.83	358	4.83	0	0
R20.20.2	296	3.40	1059	142.82	73	3
R20.20.3	292	3.62	1049	83.11	40	1
R20.20.4	304	3.50	1322	180.34	64	4
R20.20.5	300	3.58	606	22.77	28	3
R20.30.1	324	4.34	6176	8335.93	112	5
R20.30.2	381	5.32	1150	61.05	15	3
R20.30.3	368	5.28	985	55.31	28	3
R20.30.4	464	7.57	464	7.57	0	0
R20.30.5	269	3.36	3199	1635.43	93	5
R20.50.1	337	6.00	1770	493.60	103	4
R20.50.2	284	4.21	1376	443.87	98	2
R20.50.3	283	4.95	1053	109.24	41	3
R20.50.4	264	4.05	1829	1037.84	170	4
R20.50.5	365	6.91	2610	1421.90	117	3
R20.100.1	314	8.51	2179	1269.67	143	4
R20.100.2	412	12.13	976	60.75	14	3
R20.100.3	401	12.17	844	48.79	11	3
R20.100.4	418	11.57	694	28.25	6	3
R20.100.5	314	8.57	1500	634.04	141	3

Observe that the new pricing problem differs from the original pricing problem only in the objective coefficients: c_j is replaced by $(c_j - \sigma d_j)$. Hence, if we can solve the pricing problem without using some special structure of the objective coefficients, i.e., we can solve this problem for arbitrary c_j, then the addition of constraints does not complicate the pricing problem.

The situation is usually more complicated if a valid inequality $\sum_{k=1}^{K} g_k \lambda_k \leqslant g_0$, in terms of variables of the reformulation, is added. In that case, the pricing problem becomes

$$\min\left\{ \sum_{j=1}^{n} \left(c_j - \sum_{i=1}^{m_1} \pi_i a_{ij}^{(1)} \right) x_j^k - \alpha - \sigma g_k \mid k = 1, \ldots, K \right\}.$$

The addition of the inequality results in an additional term σg_k in the cost of each feasible solution x^k to the pricing problem. As there may be no obvious way to transform the cost g_k into costs on the variables x_j^k, the additional constraint can complicate the structure of the pricing problem. An example of this situation is the addition of clique constraints to the set partitioning formulation of the generalized assignment problem that was discussed by Savelsbergh.[12]

3.3 Computational Validation

We have tested the performance of a cutting plane algorithm with inequalities with right-hand side 1 for the problem of minimizing total weighted completion time on a single machine subject to release dates, where the linear programs are reformulated using Dantzig-Wolfe decomposition and subsequently solved by a column generation scheme.

We present results for the problems $Rn \cdot P_{\max} \cdot i$ with $n = 20, 30$ and $p_{\max} = 5, 10, 20, 30, 50, 100$. The results are found in Tables IVa and IVb. They show the number of columns generated in the solution of the initial LP and the time required to solve this initial LP, the total number of columns generated during the execution of the cutting plane algorithm, the total time required by the execution of the cutting

Table IVb. Combining row and column generation for the 30-job instances

Problem	No. of Columns	CPU	No. of Columns	CPU	No. of Inequalities	No. of Rounds
R30.5.1	655	13.58	1661	143.63	23	3
R30.5.2	532	9.06	1451	182.91	69	2
R30.5.3	524	8.67	1133	61.00	19	3
R30.5.4	567	10.93	885	39.58	20	3
R30.5.5	642	13.68	2065	309.49	37	3
R30.10.1	626	13.92	2044	332.14	41	3
R30.10.2	720	16.46	5068	3473.97	51	3
R30.10.3	796	18.97	2339	485.28	58	3
R30.10.4	602	12.78	2724	769.08	66	3
R30.10.5	561	11.95	2188	493.61	59	3
R30.20.1	747	23.46	1864	351.75	55	3
R30.20.2	797	24.93	2570	696.98	63	3
R30.20.3	577	15.58	2110	780.18	89	3
R30.20.4	543	12.71	2424	777.48	63	3
R30.20.5	613	16.19	1230	181.14	89	3
R30.30.1	588	16.78	2500	862.50	77	3
R30.30.2	691	24.94	1800	379.43	57	3
R30.30.3	740	25.56	1771	322.57	55	3
R30.30.4	640	20.50	2204	455.21	47	3
R30.30.5	583	18.57	1373	233.98	85	3
R30.50.1	565	22.67	1375	248.56	78	3
R30.50.2	662	26.31	1972	387.33	54	3
R30.50.3	560	20.47	1278	221.69	88	3
R30.50.4	637	25.10	1638	315.72	70	3
R30.50.5	583	21.77	2277	670.52	65	3
R30.100.1	663	42.32	1298	256.07	82	3
R30.100.2	679	40.35	1755	467.91	68	3
R30.100.3	571	32.29	1321	264.18	78	3
R30.100.4	678	43.90	1723	445.51	69	3
R30.100.5	882	57.00	2157	574.17	54	3

plane algorithm, the number of inequalities added by the cutting plane algorithm, and the number of cut generation rounds.

These results are obviously disappointing. We have to pay a high price for improving the quality of the lower bounds: the computation times increase significantly (much more so than in standard cutting plane algorithms). Reoptimizing the linear program after a set of cuts has been added seems to be almost as hard as solving the original LP, i.e., roughly the same number of columns needs to be generated to solve the extended LP. This is in stark contrast to standard cutting plane algorithms, i.e., cutting plane algorithms not combined with column generation, where the time to resolve the LP after cuts have been added is typically a fraction of the time it took to solve the first LP! If this observation holds in other contexts as well, this constitutes a major computational drawback of combined column and cut generation approaches.

On a more positive note, we are able to run the cutting plane algorithm on the largest instances, where this was impossible with the simplex and barrier algorithms on the original formulation.

4. Conclusions and Extensions

We have shown that column generation techniques can be applied effectively to solve LP-relaxations of time-indexed formulations for single-machine scheduling problems. As mentioned before, such linear programming solutions can be used to guide list scheduling algorithms to obtain high-quality integral solutions. However, if we want to obtain an optimal integral solution, then we need to embed the column generation scheme in a branch-and-bound algorithm. This requires a branching strategy that does not destroy the structure of the pricing problem. Fortunately, it is not hard to see that any branching strategy that fixes variables in the original formulation can be used. Suppose that at some node the variable x_{js} is fixed at zero. Then we are only allowed to

generate columns that represent a path not containing the arc belonging to the variable x_{js}. This can be achieved by omitting this arc from the network N. Suppose, on the other hand, that this variable is fixed at one. Then all columns that are generated have to correspond to paths containing the arc belonging to x_{js}, i.e., the path determined by the pricing problem has to contain the arc from s to $s + p_j$ corresponding to job j. This means that the pricing problem decomposes into two subproblems. We have to determine the shortest path from 0 to s and the shortest path from $s + p_j$ to $T + 1$.

In our implementation of the column generation scheme, we have taken the straight-forward approach of adding one negative reduced-cost column per iteration. The efficiency of a column generation approach may be improved by implementing a more sophisticated column management scheme. A column management scheme tries to improve the efficiency in two ways: 1) reduce the number of times columns are generated, for example, by generating more than one column per iteration, or 2) keep the size of the active linear program small, for example, by deleting columns with fairly large reduced cost. The efficiency of such column management schemes still needs to be investigated.

Little research has been done on the use of approximate linear programming solutions in LP-based branch-and-bound algorithms and LP-based approximation algorithms. We have shown that good approximate solutions to the linear programming relaxation of a time-indexed formulation can be generated an order of magnitude faster than the optimal solution. This suggests that time-indexed formulations may be well-suited for a study of the use of approximate linear programming solutions in LP-based branch-and-bound algorithms and LP-based approximation algorithms. We intend to do this in the near future.

More research also needs to be done in the area of combining column generation and cut generation techniques. We have shown that for the time-indexed formulation it is possible to do this, but our computational experiences were somewhat disappointing. In any branch-and-cut algorithm there is trade-off between adding many cutting planes and turning to branching as late as possible and adding fewer cutting planes and turning to branching earlier. Adding many cutting planes leads to improved bounds and thus, hopefully, to smaller search trees. However, cutting planes have to be found, which takes time, and when they are added the size of the linear programs increases, which results in larger LP solution times. van den Akker et al.[1] show that for the time-indexed formulation, adding cutting planes leads to more robust algorithms. However, when column generation techniques are used to solve the LP relaxations, it looks to be better to favor branching more. Another possibility for medium-sized instances may be to solve the initial LP-relaxation by column generation and then switch back to the original formulation before any cuts are added.

We have made no computational comparison between our LP-based solution approach and other solution approaches. The results by van den Akker et al.[1] show that optimization algorithms for scheduling problems based on a time-indexed formulation typically have very small search trees because of the quality of the bounds but may be inefficient due to the time it takes to solve the linear programs. Non-LP-based optimization algorithms typically have very large search trees, but may be efficient because the nodes of the search tree can be evaluated extremely quickly. This can be observed, for example, when we compare the results of the branch-and-cut algorithm of van den Akker et al.[1] for the single-machine scheduling problem of minimizing the weighted sum of completion times subject to release dates to the results of the branch-and-bound algorithm of Belouadah et al.[3] Although the search trees produced by the former algorithm are much smaller than the search trees produced by the latter algorithm, the computation times of the former algorithm are larger than those of the latter algorithm. However, we are confident that the computational tools developed and presented in this paper will allow the development of much more efficient LP-based optimization algorithms. Finally, we observe that non-LP-based optimization algorithms are usually specifically designed for one type of scheduling problem, whereas time-indexed formulations have the advantage that they can model many different objective functions and constraints and, therefore, they have a much wider applicability.

Acknowledgments

The research described in this paper was carried out while the first author was a Ph.D. student at Eindhoven University of Technology. The third author was sponsored by National Science Foundation Grant DMI-9410102. The authors want to express their gratitude to Han Hoogeveen for his help in preparing the paper and to the anonymous referees for their comments on an earlier draft of the paper.

References

1. J.M. VAN DEN AKKER, C.P.M. VAN HOESEL, and M.W.P. SAVELSBERGH, 1999. A Polyhedral Approach to Single-Machine Scheduling Problems, *Mathematical Programming 85*, 541–572.
2. J.M. VAN DEN AKKER, 1994. *LP-Based Solution Methods for Single-Machine Scheduling Problems*, Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.
3. H. BELOUADAH, M.E. POSNER, and C.N. POTTS, 1992. Scheduling with Release Dates on a Single Machine to Minimize Total Weighted Completion Time, *Discrete Applied Mathematics 36*, 213–231.
4. Y. CRAMA and F.C.R. SPIEKSMA, 1996. Scheduling Jobs of Equal Length: Complexity and Facets, *Mathematical Programming 72*, 207–227.
5. M.E. DYER and L.A. WOLSEY, 1990. Formulating the Single Machine Sequencing Problem with Release Dates as a Mixed Integer Program, *Discrete Applied Mathematics 26*, 255–270.
6. M.X. GOEMANS, 1997. Improved Approximation Algorithms for Scheduling with Release Dates, *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 591–598.
7. L.A. HALL, A.S. SCHULZ, D.B. SHMOYS, and J. WEIN, 1997. Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms, *Mathematics of Operations Research 22*, 513–544.
8. L.S. LASDON, 1970. *Optimization Theory for Large Systems*, MacMillan, New York.
9. Y. LEE and H.D. SHERALI, 1994. Unrelated Machine Scheduling

with Time-Window and Machine Downtime Constraints: An Application to a Naval Battle-Group Problem, *Annals of Operations Research 50*, 339–365.

10. C. PHILLIPS, C. STEIN, and J. WEIN, 1998. Minimizing Average Completion Time in the Presence of Release Dates, *Mathematical Programming 82*, 199–223.
11. M. QUEYRANNE and A.S. SCHULZ, 1994. *Polyhedral Approaches to Machine Scheduling*, Preprint 408/1994, Revised in October 1996, Department of Mathematics, Technical University of Berlin, Berlin.
12. M.W.P. SAVELSBERGH, 1997. A Branch-and-Price Algorithm for the Generalized Assignment Problem, *Operations Research 45*, 831–841.
13. M.W.P. SAVELSBERGH, R.N. UMA, and J. WEIN, 1998. An Experimental Study of LP-Based Approximation Algorithms for Scheduling Problems, *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 453–462.
14. A. SCHRIJVER, 1986. *Theory of Linear and Integer Programming*, Wiley, New York.
15. H.D. SHERALI, Y. LEE, and D.D. BOYER, 1995. Scheduling Target Illuminators in Naval Battle-Group Anti-Air Warfare, *Naval Research Logistics 42*, 737–755.
16. J.P. DE SOUSA, 1989. *Time-indexed formulations of non-preemptive single-machine scheduling problems*, Ph.D. Thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium.
17. J.P. DE SOUSA and L.A. WOLSEY, 1992. A Time-Indexed Formulation of Non-Preemptive Single-Machine Scheduling Problems, *Mathematical Programming 54*, 353–367.
18. F. VANDERBECK and L.A. WOLSEY, 1996. An Exact Algorithm for IP Column Generation, *Operations Research Letters 19*, 151–160.