A time-indexed formulation for single-machine scheduling problems : column generation

A Time-Indexed Formulation for Single-Machine Scheduling Problems: Column Generation

J.M. van den Akker [1]
Center for Operations Research and Econometrics
Catholique University of Louvain
Voie du Roman Pays 34, 1348 Louvain-la-Neuve, Belgium

C.A.J. Hurkens
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

M.W.P. Savelsbergh
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205, U.S.A.

April 3, 1996

Abstract

Time-indexed formulations for single-machine scheduling problems have received a lot of attention, because the linear program relaxations provide strong bounds. 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 the difficulties associated with the size of time-indexed formulations and that the application of these techniques still allows the use of cut generation techniques.

Key words: scheduling, Dantzig-Wolfe decomposition, column generation.

[1]Supported by Human Capital and Mobility (HCM), grant number ERBCHBGTC940513; a significant part of the research was carried out while the author was at Eindhoven University of Technology.

1 Introduction

Integer programming approaches to single-machine scheduling problems have received a considerable amount of attention in the past decade. For a survey of the formulations that have been proposed and investigated, we refer the reader to an excellent survey by Queyranne and Schulz [1994]. One of the more powerful formulations, in the sense that it can model many different types of scheduling problems and that its LP relaxation provides strong bounds, is based on time-discretization [Sousa 1989, Sousa and Wolsey 1992, Van den Akker, Van Hoesel, and Savelsbergh 1993, Van den Akker 1994, Van den Akker, Hurkens, and Savelsbergh 1995, Crama and Spieksma 1995]. Unfortunately, these 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. As a result, solution times and memory requirements may be prohibitive.

Dantzig-Wolfe decomposition is a well-known technique that can be applied to large scale structured linear programs to reduce the memory requirements and solution times. The application of Dantzig-Wolfe decomposition techniques results in a reformulation of the linear program with far fewer constraints but many more variables. However, the variables are handled implicitly rather than explicitly. Variables are left out of the linear program because there are too many to handle efficiently and many of them will be equal to zero in an optimal solution anyway. Then to check the optimality of the solution to the linear program, a subproblem, called the pricing problem, is solved to try to identify variables to enter the basis. If such variables are found, the linear program is reoptimized.

In this paper, we investigate whether Dantzig-Wolfe decomposition techniques, also referred to as column generation techniques, can be used to alleviate the difficulties associated with the size of time-indexed formulations.

Experiments with an LP-based branch-and-bound algorithm based on a time-indexed formulation for the problem of minimizing the total weighted completion time on a single-machine subject to release dates have shown that the bounds provided by the LP relaxation of the time-indexed formulation are strong. However, to obtain a robust algorithm, i.e., an algorithm that consistently solves instances in a reasonable amount of time, it is necessary to enhance the algorithm with cut generation [Van den Akker, Hurkens, and Savelsbergh 1995]. Therefore, a major part of the research discussed in this paper deals with the complexities associated with combining approaches based on column generation with cut generation. 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.

In Section 2, we review the time-indexed formulation for single-machine scheduling problems. In Section 3, we present and analyze the reformulation obtained by applying Dantzig-Wolfe decomposition and discuss its advantages and disadvantages. In Section

4, we develop a cutting plane algorithm for a large class of single machine scheduling problems based on the reformulation discussed in Section 3. We elaborate on the issues related to combining column generation and cut generation and present some general results that are also applicable in other contexts. In Section 5, we discuss some extensions and present some concluding remarks.

2 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

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

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \qquad (t = 1, \ldots, T), \tag{2}$$

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

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.

The major advantage of the time-indexed formulation is that it can be used to model different single-machine scheduling problems. Different objective functions can be modeled by appropriate choices of cost coefficients and many constraints, such as deadlines and release dates, can be handled simply by fixing certain variables to zero. In addition, and equally important, the LP relaxation of the time-indexed formulation provides a strong bound; better than 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 $\sum_{j=1}^{n} p_j$. 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 release dates [Van den Akker, Hurkens, and Savelsbergh 1995]. 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 and that the number of constraints is $n + T$.

3 Reformulation

We have applied 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

$$\min \; cx$$

$$Ax \leq b,$$

$$x \in X,$$

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

$$x = \sum_{1 \leq j \leq k} \lambda_j x^j, \quad \sum_{1 \leq j \leq k} \lambda_j = 1, \;\; \lambda_j \geq 0, \; j = 1, ..., k.$$

This leads to the following reformulation, which is known as the Dantzig-Wolfe master problem

$$\min \sum_{1 \leq j \leq k} (cx^j)\lambda_j$$

$$\sum_{1 \leq j \leq k} (Ax^j)\lambda_j = b$$

$$\sum_{1 \le j \le k} \lambda_j = 1$$

$$\lambda_j \ge 0 \;\; j = 1, ..., k.$$

However, since the reformulation frequently contains a huge number of columns, it may be necessary to work with restricted versions that contain only a subset of its columns, and to generate additional columns only as they are needed. Column generation for the restricted master problem is accomplished by solving the pricing problem

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

where (π, α) is an optimal dual solution to the LP relaxation of the restricted master problem.

In the next subsections, we investigate how Dantzig-Wolfe can be applied to the time-indexed formulation for single-machine scheduling problems.

3.1 The 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

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

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \le 1 \qquad (t = 1, \ldots, T), \tag{4}$$

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

We place the assignment constraints (3) in the master problem and the capacity constraints (4) plus the nonnegativity constraints in the pricing problem. Therefore, we have to describe the polytope P defined by the capacity constraints plus the nonnegativity constraints as the convex hull of its extreme points.

The polytope P is described by the system

$$\begin{pmatrix} A \\ -I \end{pmatrix} x \le \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \end{pmatrix},$$

where A 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 $s \leq t \leq s + p_j - 1$. This means that the column in A 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, A is an *interval matrix.* Interval matrices are known to be totally unimodular (see for example Schrijver [1986]). This implies that the matrix $\binom{A}{-I}$ 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 $\sum_{k=1}^{K} \lambda_k x^k$ for some nonnegative values λ_k such that $\sum_{k=1}^{K} \lambda_k = 1$. The master problem can now be expressed as:

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

subject to

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

$$\sum_{k=1}^{K} \lambda_k = 1, \tag{6}$$

$$\lambda_k \geq 0 \qquad k = 1, \ldots, K.$$

Observe that the coefficient of λ_k in the jth row of (5), i.e., $\sum_{t=1}^{T-p_j+1} x_{jt}^k$, is precisely the number of times that job j occurs in the pseudo-schedule x^k. This means that the column corresponding to the pseudo-schedule x^k indicates how many 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 $p_2 = 3$. The variable λ_k corresponding to the pseudo-schedule $x_{11} = x_{13} = 1$ 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).

By reformulating the problem in this way, the number of constraints is decreased from $n+T$ to $n+1$. On the other hand, the number of variables is significantly increased. Fortunately, the huge number of variables does not pose a real problem, because they can be handled implicitly by means of column generation techniques. In a column generation scheme, most columns are left out of the LP relaxation because there are too many columns to handle efficiently and most of them will have their associated variable equal to zero in an optimal solution anyway. Then to check the optimality of an LP solution, a subproblem, called the pricing problem, is solved to try to identify columns to enter the basis. If such columns are found, the LP is reoptimized. The pricing problem identifies a column with minimal reduced cost.

3.2 The pricing problem

The reduced cost of a variable λ_k is given by

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

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

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

Recall that each extreme point x^k represents a pseudo-schedule, i.e., a schedule in which the capacity constraints are observed, but in which jobs do not have to start exactly once. Such pseudo-schedules can be represented by paths in a network N as follows. The network 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 $t \leq T-p_j+1$, there is a process arc from t to $t+p_j$ representing that the machine processes job j from time t to time $t+p_j$. 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 A 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 $T = 6$.

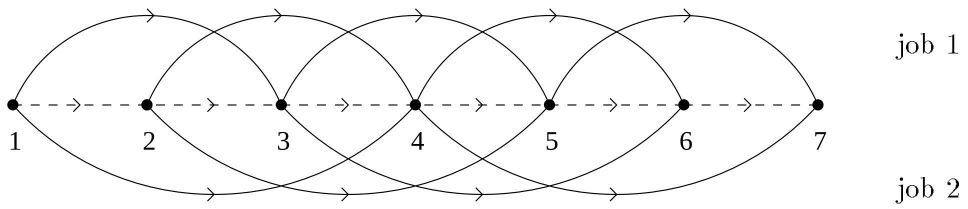


Figure 1: The network N for a 2-job example.

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.

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.

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 $x^1_{11} = x^1_{23} = x^1_{26} = 1$ and the second one to the pseudo-schedule $x^2_{31} = x^2_{34} = x^2_{17} = 1$. 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 $x_{11} = x_{26} = x_{33} = 1$.

3.3 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 the total weighted completion time on a single machine subject to release dates on the jobs. We report results for twelve sets of five randomly generated instances with uniformly distributed weights in $[1, 10]$ and uniformly distributed release dates in $[0, \frac{1}{2}\sum_{j=1}^{n} p_j]$. Half of the instances have twenty jobs, the others have thirty jobs. The processing times are in $[1, p_{\max}]$, where $p_{\max}$ equals $5, 10, 20, 30, 50$ or 100. We have five instances for each combination of n and $p_{\max}$; these instances are denoted by Rn.p$_{\max}$.i, where i is the number of the instance. Our computational experiments have been conducted with MINT0 2.0/CPLEX 3.0 and have been run on a IBM RS/6000 model 590.

The computational results are given in Tables 1a and 1b. These tables show the running time of the column generation algorithm (time cg), the time required to solve the LP-relaxation of the original formulation by CPLEX' primal simplex method (simplex), and the time required to solve the LP-relaxation by CPLEX' 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 2300×56000 and the number of nonzero's is approximately 2,900,000.

As expected, the computational advantage of the reformulation is apparent for those problems in which $T = \sum_{1 \leq j \leq n} p_j$ is large, i.e., large values of n and $p_{\max}$. For both $n = 20$ and $n = 30$, the column generation scheme for the reformulation is the fastest for $p_{\max} \geq 20$. The cpu time required by the column generation scheme for the reformulation appears to grow very slowly with the size of the instance.

Observe also that the number of generated columns seems to be almost independent of the instance size. Therefore the increases in computation time can be fully contributed to the increases in execution times of the shortest path algorithm due to the increase in size of the underlying network.

4 Combining column and cut generation

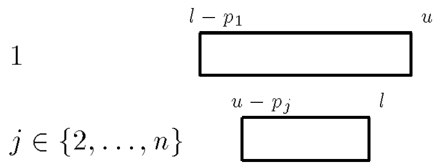
In the previous section, we have shown that for large instances the LP relaxation of the time-indexed formulation can be solved efficiently by a column generation scheme for a reformulation obtained by applying Dantzig-Wolfe decomposition. Van den Akker, Hurkens, and Savelsbergh [1995] have demonstrated that cutting planes strengthen the bounds from the time-indexed formulation considerably and that a branch-and-cut algorithm performs significantly better and is more robust than a plain branch-and-bound algorithm. 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, i.e., whether column generation can be combined effectively with cut generation.

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 a new dual variable that must be handled in the pricing problem.

In the first part of this section, we show that the LP relaxations that have to be solved after one or more cuts have been added to the time-indexed formulation can still be solved by a column generation scheme. We discuss in detail what modifications are necessary in the pricing problem to handle the additional dual variables. In the second part of this section, we show that, under mild assumptions, the same ideas can be applied in other contexts where a column generation scheme based on a reformulation obtained through Dantzig-Wolfe decomposition is combined with the addition of cuts that are given in terms of the original formulation.

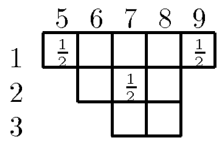
4.1 Column and cut generation for single machine scheduling problems

Van den Akker, Van Hoesel, and Savelsbergh [1993] present a complete characterization of all facet inducing inequalities with right-hand sides 1 and 2 for the time-indexed formulation. Inequalities with right-hand side 1 are denoted by $x(V) \leq 1$, which is a short notation for $\sum_{(j,s)\in V} x_{js} \leq 1$. Van den Akker, Van Hoesel, Savelsbergh [1993] show that for any facet-inducing inequality V is given by $\{(1,s) \mid s \in [l-p_1, u]\} \cup \{(j,s) \mid j \neq 1, s \in [u-p_j, l]\}$, for some l and u with $l < u$ and some special job, which for presentational convenience is assumed to be job 1. Such an inequality can be represented by the following diagram:



$\leq 1.$

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 below



$\leq 1.$

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

$$\sum_{k=1}^{K}(\sum_{(j,s)\in V} x_{js}^k)\lambda_k \leq 1.$$

The coefficient of λ_k is equal to the number of arcs in path P_k, that refer to job-start (j,t) for $(j,t) \in V$.

Now that we have established the structure of the reformulated inequality, two issues have to be addressed:

- How hard is it to update the columns already present in the restricted master problem when a (reformulated) inequality is added;
- How hard is it to generate new columns for the restricted master problem after a (reformulated) inequality is added.

Observe that the number of arcs in path P_k that refer to job-starts in V is readily determined. Therefore, it is easy to compute the coefficient of a reformulated inequality for the columns already present in the restricted master problem.

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

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

subject to

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

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

$$\sum_{k=1}^{K}(\sum_{(j,s)\in V} x_{js}^k)\lambda_k \leq 1,$$

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

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

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

which can be rewritten as

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

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 $c_{js} - \pi_j - \mu_V$ if $(j,s) \in V$ and $c_{js} - \pi_j$ if $(j,s) \notin V$. 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,s) for $(j,s) \in V$. 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 direct 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.

4.2 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 the original formulation are added.

Consider the linear programming problem

$$\min \ cx$$
$$A^{(1)}x \leq b^{(1)},$$
$$A^{(2)}x \leq b^{(2)},$$

where $c \in \mathcal{R}^n$, $A^{(1)} \in \mathcal{R}^{m_1 \times n}$, and $b^{(1)} \in \mathcal{R}^{m_1}$, and where for presentational convenience, we assume that $A^{(2)}x \leq b^{(2)}$ 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:

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

subject to

$$\sum_{k=1}^{K} (\sum_{j=1}^{n} a_{ij}^{(1)} x_j^k) \lambda_k \leq b_i^{(1)} \qquad (i = 1, \ldots, m_1),$$

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

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

The reduced cost of the variable λ_k is equal to

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

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

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

Theorem 1 *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 $dx \leq d_0$ in terms of the original variables does not complicate the pricing problem.*

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

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

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

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

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

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

Observe that the new pricing problem differs from the original pricing problem only in the objective coefficients. 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. $\square$

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

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

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 [1996].

4.3 Computational validation

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

We present results for the problems Rn.$p_{\max}$.i with $n = 20, 30$ and $p_{\max} = 5, 10, 20, 30, 50, 100$. The results are found in Tables 2a and 2b. 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 that has been generated during the execution of the cutting plane algorithm, the total time required by the execution of the cutting plane algorithm, the number of inequalities that has been added the cutting plane algorithm, and the number of cut generation rounds.

The most important observation that can be made when analyzing the computational results, is that after adding a set of cuts the resulting LP seems to be at least as hard to solve as the original LP, i.e., about the same number of columns needs to be generated to solve the extended LP. This is in stark contrast to standard cutting plane algorithms, 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. This is a major computational drawback of combined column and cut generation approaches, since it means that each round of cut generation results in an LP that has to be solved from scratch.

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.

5 Extensions and conclusions

We have shown that it is possible to implement a cutting plane algorithm, in which the linear programs are reformulated using Dantzig-Wolfe decomposition and subsequently solved by a column generation scheme. To extend such a cutting plane algorithm to a branch-and-cut algorithm, i.e, an LP based branch-and-bound algorithm with cut generation in each node of the search tree, a branching strategy needs to be developed that does not destroy the structure of the pricing problem. 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 $T + 1$.

The conclusions that can be drawn from the research described in this paper are (1) that Dantzig-Wolfe decomposition techniques can be applied effectively to time-indexed formulations for machine scheduling problems, and (2) that column generation techniques can be combined with cut generation techniques.

References

Y. Crama and F.C.R. Spieksma (1995). Scheduling jobs of equal length: complexity, facets and computational results. E. Balas and J. Clausen (eds.). *Proceedings of the 4th International IPCO Conference, Denmark*, Lecture Notes in Computer Science 920, Springer, Berlin, 277-291.

M. Queyranne and A.S. Schulz (1994). *Polyhedral Approaches to Machine Scheduling* Preprint 408/1994, Department of Mathematics, Technical University of Berlin, Berlin.

M.W.P. Savelsbergh (1996). A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, to appear.

A. Schrijver (1986). *Theory of Linear and Integer Programming.* Wiley, New York.

J.P. de Sousa (1989). *Time-indexed formulations of non-preemptive single-machine scheduling problems.* PhD-thesis, Catholic University of Louvain, Louvain-la-Neuve.

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.

J.M. Van den Akker, C.P.M. Van Hoesel, and M.W.P. Savelsbergh (1993). *Facet inducing inequalities for single-machine scheduling problems*, Memorandum COSOR 93-27, Eindhoven University of Technology, Eindhoven.

J.M. Van den Akker (1994). *LP-based solution methods for single-machine scheduling problems* PhD-thesis, Eindhoven University of Technology.

J.M. Van den Akker, C.A.J. Hurkens, and M.W.P. Savelsbergh (1995). *A time-indexed formulation for single-machine scheduling problems: Branch-and-Cut.* Memorandum COSOR 95-24, Eindhoven University of Technology, Eindhoven.

problem	#cols	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

Table 1a: Performance of the column generation, simplex, and barrier algorithms for the 20-job instances.

problem	#cols	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	?

Table 1b: Performance of the column generation algorithm, CPLEX simplex, and CPLEX barrier for the 30-job instances.

problem	# cols	cpu	# cols	cpu	# ineq	# 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

Table 2a: Combining row and column generation for the 20-job instances.

problem	# cols	cpu	# cols	cpu	# ineq	# 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

Table 2b: Combining row and column generation for the 30-job instances.