Column Generation for Extended Formulations

Ruslan Sadykov (1,2) and François Vanderbeck (2,1)

(1) project-team RealOpt, INRIA Bordeaux Sud-Ouest

(2) Institute of Mathematics, University of Bordeaux

July 8, 2011

Abstract

Working in an extended variable space allows one to develop tight reformulations for mixed integer programs. However, the size of the extended formulation grows rapidly too large for a direct treatment by a MIP-solver. Then, one can use projection tools and derive valid inequalities for the original formulation, or consider an approximate extended formulation (f.i., by aggregating variables). Both approaches result in outer approximations of the intended extended formulation. An alternative is to work with inner approximations defined and improved by generating dynamically the variables of the extended formulation. It assumes that the extended formulation stems from a decomposition principle: a subproblem admits an extended formulation from which the original problem extended formulation is derived. Then, one can implement column generation for this extended formulation by transposing the equivalent procedure for the Dantzig-Wolfe reformulation. Pricing subproblem solutions are expressed in the variables of the extended formulation and added to the current restricted version of the extended formulation along with the subproblem constraints that are active for the subproblem solution. Such "column-and-row generation" procedure is reviewed and analysed herein. We compare numerically a direct handling of the extended formulation, a standard column generation approach, and the "column-and-row generation" procedure, highlighting a key benefit of the latter: lifting pricing problem solutions in the space of the extended formulation permits their recombination into new subproblem solutions and results in faster convergence.

Keywords: Extended formulations for MIP, Column-and-Row Generation, Stabilization.

Introduction

Formulating a mixed integer program (MIP) in a higher dimensional space by introducing extra variables is a process to achieve a tight approximation of the integer convex hull. Several classes of reformulation techniques are reviewed in [24]: some are based on variable splitting (including multi-commodity flow, unary or binary expansions), others rely on the existence of a dynamic programming or a linear programming separation procedure, further reformulation techniques rely on exploiting the union of polyhedra or basis

reduction. A unifying framework is presented in [12]. An extended formulation presents the practical advantage to lead to a direct approach: the reformulation can be fed to a MIP-solver. However, such approach remains limited to small size instances because the extended formulation grows rapidly too large for practical purposes. Its size counted as the sum of the number of variables and constraints is often pseudo-polynomial in the input size or polynomial but with a large degree.

To alleviate the curse of dimensionality, one can in some cases project the extended formulation into a lower dimensional space; for example by applying a Benders' decomposition approach [2]. Alternatively, Van Vyve and Wolsey [25] propose to "truncate" the reformulation in order to define a good quality outer approximation of the polyhedron defined by the projection of the full-blown extended formulation. In several application specific contexts, they show that approximating the extended formulation may indeed allow one to achieve significant tightening of the linear programming bound while having manageable size. The techniques range from dropping some of the constraints (the more complex constraints typically bring the most marginal dual bound improvements), or in such case as multi-commodity flow reformulation, aggregating several commodities into one or aggregating nodes, or, more generally applying the extended reformulation paradigm to subsystems only. Such approach preserves the possibility of directly applying a standard MIP approach to the reformulation and allows one to deal with larger size instances. In [25], the level of approximation is controlled by a parameter whose maximum value correspond to the full extended formulation. Their numerical results show that the best trade-off between dual bound quality and size is often achieved for low level approximations.

While Benders' approach results in working with a dynamically improved outer approximation of the intended extended formulation, the 'truncate" extended reformulation of [25] leads to a relaxation that defines a static outer approximation. The approach reviewed herein consists in developing an inner approximation of the intended polyhedron by considering the extended formulation restricted to a subset of variables, delaying the inclusion of some variables and associated constraints. In the spirit of a Dantzig-Wolfe column generation approach, the inner approximation is iteratively improved by adding promising variables along with the constraints that become active once those variables are added. However it relies on specific pricing and separation strategies. Instead of simply doing variable pricing or constraint separation based on enumeration on the columns and rows of a full-blown extended formulation, one prices over the whole set of variables by solving an optimization subproblem and insert the constraints that are binding for that subproblem solution. For the method to apply, the application on hand must have some decomposable structure that makes it amenable to Dantzig-Wolfe decomposition. Then, the pricing subproblem is that of a standard column generation approach applied to Dantzig-Wolfe reformulation. However, subproblem solutions are expressed in the variables of the extended formulation (which can be understood as column "lifting" or "disaggregation") and added to a master program which is a restricted version of the extended formulation.

Therefore, the method is a hybrid between an extended formulation approach and a

standard column generation approach. Compared to a direct use of the extended reformulation, this hybrid approach can be seen as a way to handle dynamically the large size of the reformulation. Compare to applying a standard column generation to the Dantzig-Wolfe reformulation, the hybrid approach has the advantage of an accelerated convergence of the column generation procedure: "lifting/disaggregating" columns acts as a stabilization technique for column generation. Moreover, it offers a richer model in which to define cuts or branching restrictions.

Such *column-and-row generation* procedure is a technique previously described in the literature in application specific context, such as bin packing [19], multi-commodity flow [13], split delivery vehicle routing [6, 7], or network design [7, 8]. The convincing computational results of some of these papers indicate the interest of method. Although the motivations of these studies are mostly application specific, methodological statements made therein are to some extend generic. Moreover, there are recent effort to explore this approach further. In [9], Frangioni and Gendron present a "structured Dantzig-Wolfe decomposition" for which they adapt column generation stabilization techniques (from linear penalties to the bundle method). In [7], Gendreau et al. present a branch-and-price-and-cut algorithm where columns and cuts are generated simultaneously. In [10], Muter et al. consider what they call a "simultaneous column-and-row generation" approach where the subproblem has itself decomposable structure.

Here, we revisit the column-and-row generation approach. Our purpose is to show light on this approach, to emphasize its wide applicability, and to present it with a new angle as a method that is natural when considering a problem reformulation based on an extended reformulation of a subproblem. In the spirit of [23], column-and-row generation is viewed herein as a generalization of standard column generation (the latter is based on a specific subproblem extended formulation). This view leads to a new termination condition and to easy extension of the method's formalism when one only has an approximate extended formulation of subproblems: we establish the validity of the algorithm in a form that encompass all special cases. Our presentation of the methodology completes those of [7, 8, 9, 10, 13, 19]. More importantly perhaps, we analyse the interests of this hybrid approach: the main advantage being the recombination of previously generated columns into new subproblem solutions that results in an acceleration of the convergence.

In the sequel, we start (in Section 1) by presenting applications where a *column-and-row generation* has been used, or could have been used. Next, in Section 2, we formalize the procedure and show the validity of the algorithm. We then show that the method extends to the case where one only has a good reformulation in an extended variable space, but not an exact extended formulation (as in approximate extended formulation arising from the proposal of [25]). We show that in such case, the dual bound that one obtains is at worse that of the linear relaxation of the extended reformulation and at best the Lagrangian dual bound based on that subproblem. We also consider the case of multiple subproblems. In Section 3, we discuss the pros and cons of the method and we highlight the properties that are required for the application in order to find some specific interest in replacing standard column generation by such dynamic column-and-row generation for an extended formulation. Finally, in Section 4, we present some numerical tests on

several applications where we compare a direct solution of the extended formulation linear relaxation, a standard column generation approach for the Dantzig-Wolfe master program, and the column-and-row generation approach applied to the extended formulation LP. The results illustrate the stabilization effect resulting from column disaggregation and recombinations.

1. Specific Examples

1.1 Machine Scheduling

For scheduling problems, time-index formulations are standard extensions resulting from a unary decomposition of the start time variables. Consider a single machine scheduling problem on a planning horizon T as studied by van den Akker et al. [21]. The problem is to schedule the jobs, $j \in J = \{1, \ldots, n\}$, a single one at the time, at minimum cost, which can be modeled as:

$$[F] \equiv \min\{\sum_j c(S_j) : \; S_j + p_j \leq S_i \;\text{ or }\; S_i + p_i \leq S_j \; \forall (i,j) \in J \times J\} \tag{1}$$

where S_j denotes the start time of job j and p_j is its given processing time. Disjunctive program (1) admits an extended formulation written in terms of decision variables $z_{jt} = 1$ if and only if job $j \in J \cup \{0\}$ starts at the outset of period $t \in \{1, \ldots, T\}$, where job 0 with processing time 1 models machine idle time. By convention, period t is associated with time interval $[t-1, t)$ and z_{jt} is only defined for $1 \leq t \leq T - p_j + 1$. The reformulation takes the form:

$$[R] \equiv \min\{\sum_{jt} c_{jt}\, z_{jt} \tag{2}$$

$$\sum_{t=1}^{T-p_j+1} z_{jt} \;=\; 1 \qquad \forall j \in J \tag{3}$$

$$\sum_j z_{j1} \;=\; 1 \tag{4}$$

$$\sum_j z_{jt} \;=\; \sum_{j:\, t-p_j \geq 1} z_{j,t-p_j} \qquad \forall t > 1 \tag{5}$$

$$z_{jt} \;\in\; \{0,1\} \quad \forall j, t\} \tag{6}$$

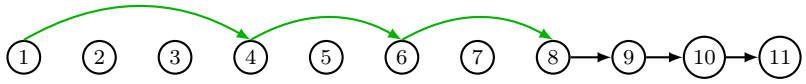
where (3) enforces the assignment of each job, (4) the initialization of the schedule, while (5) forbids the use of the machine for more than one job at the time: a job j can start in t only if one ends in t and therefore releases the machine. (The formulation can be extended to the case in which m identical machines are available; then the rhs of (4) is m and variables z_{0t} represents the number of idle machines at time t.) The objective can model any cost function that depends on job start times (or completion times). Extended reformulation [R] has size $O(|J| \cdot T)$ which is pseudo-polynomial in the input size as $T \geq \sum_j p_j$. The subsystem defined by constraints (4-5) characterize a flow that represents a "pseudo-schedule" satisfying non-overlapping constraints but not the single assignment

constraints. A standard column generation approach based on subsystem (4-5) consists in defining reformulation:

$$[M] \equiv \min\{\sum_{g\in G} c^g\,\lambda_g : \sum_{g\in G}\sum_{t=1}^{T-p_j+1} z^g_{jt}\,\lambda_g = 1\ \forall j,\ \sum_{g\in G}\lambda_g = m,\ \lambda_g \in \{0,1\}\ \forall g \in G\} \quad (7)$$

where G is the set of "pseudo-schedules": vector z^g and scalar c^g define the associated solution and cost for a solution $g \in G$. As done in [20, 21], reformulation [M] can be solved by column generation. The pricing subproblem [SP] is a shortest path problem: find a sequence of jobs and down-times to be scheduled on the single machine with possible repetition of jobs. The underlying graph is defined by nodes that represent periods and arcs $(t, t+p_j)$ associated to the processing of jobs $j \in J \cup \{0\}$ in time interval $[t-1, t+p_j)$. Figure 1 illustrates such path for a numerical instance.

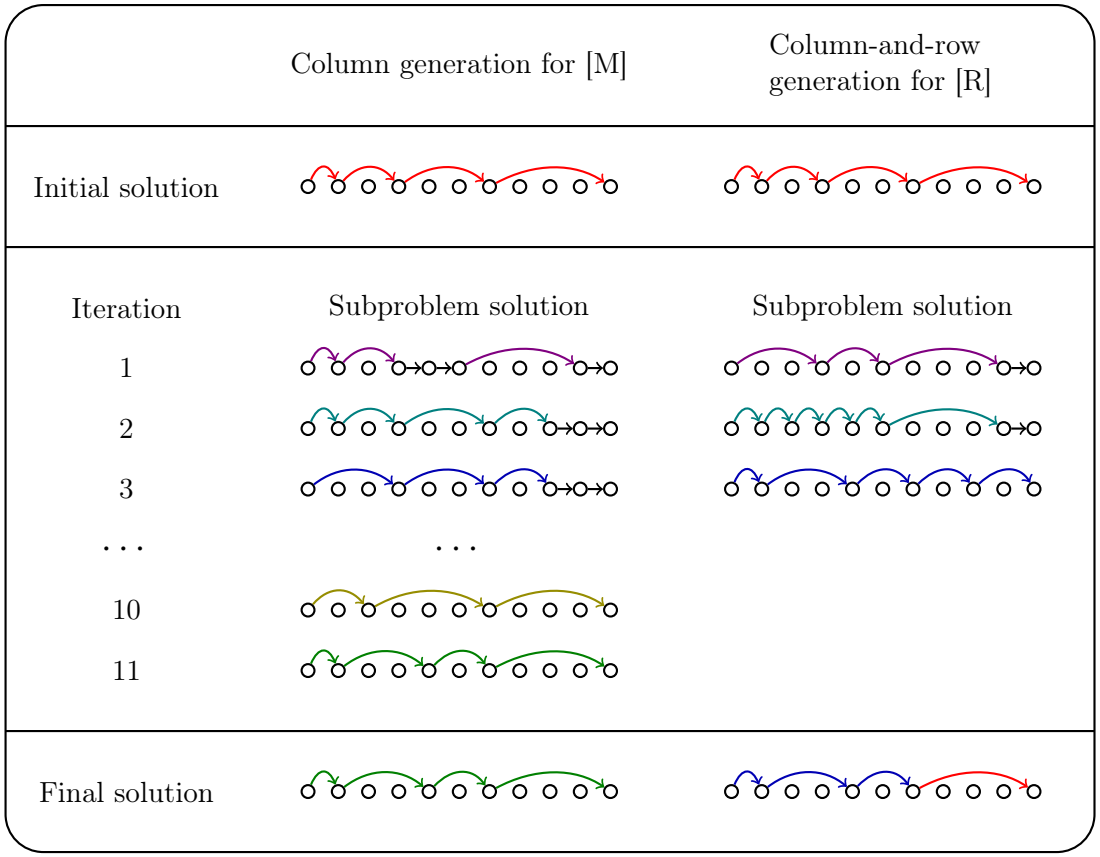
Figure 1: *A path associated to a pseudo-schedule solution to the sub-problem for $T = 10$ and $J = \{1, \ldots, 4\}$ and $p_j = j$ for each $j \in J$: the sequence consists in scheduling job 3, then twice job 2 consecutively, and to complete the schedule with idle times (represented by straight arcs).*



An alternative to the above standard column generation approach for [M] would be to generate dynamically the z variables for [R], not one at the time, but by solving the shortest path pricing problem [SP] and by adding to [R] the components of the subproblem solution z^g in the time index formulation along with the flow conservation constraints that are binding for that solution. To illustrate the difference between the two approaches, Figure 2 shows several iterations of the column generation procedure for [M] and for the numerical instance of Figure 1. In Figure 2, each bended arc represents a job, and straight arcs represent idle times. Formulations [M] and [R] are initialized with the variable(s) associated to the same pseudo-schedule depicted in Figure 2 as the initial subproblem solution. Note that the final solution of [M] is the solution obtained as the subproblem solution generated at iteration 11; while, for formulation [R], the final solution is a recombination of the subproblem solution of iteration 3 and the initial solution.

As illustrated by the numerical example of Figure 2, the interest of implementing column generation for [R] instead of [M] is to allow for the recombination of previously generated solutions. Observe that the final solution of [R] in Figure 2. Let us denote it by $\hat{z}$. It would not have its equivalent in formulation [M] even if the same four subproblem solutions had been generated: z^g, for $g = 1, \ldots, 4$. Indeed, if $\hat{z} = \sum_{g=1}^{4} z^g\,\lambda_g$, then $\lambda_1 > 0$ and job 2 must be at least partially scheduled in period 2.

Figure 2: *Solving the example by column versus column-and-row generation (assuming the same data as in of Figure 1).*



1.2 Bin Packing

A column-and-row generation approach for an extended formulation has been applied to the bin packing problem by Valerio de Carvalho [19]. The bin packing problem consists in assigning n items $i \in I = \{1, \ldots, n\}$ of given size s_i into bins of identical capacity C using a minimum number of bins. A compact formulation is:

$$[F] \equiv \min\{\sum_k \delta_k \quad : \tag{8}$$

$$\sum_k x_{ik} = 1 \quad \forall i \tag{9}$$

$$\sum_i s_i \, x_{ik} \leq C \, \delta_k \quad \forall k \tag{10}$$

$$x_{ik} \in \{0,1\} \quad \forall i,k \tag{11}$$

$$\delta_k \in \{0,1\} \quad \forall k\} \tag{12}$$

where $x_{ik} = 1$ if item $i \in I$ is assigned to bin k for $k = 1, \ldots, n$ and $\delta_k = 1$ if bin k is used. The standard column generation approach consists in reformulating the problem in terms of the solution to the knapsack subproblem (10-12) for a fixed k. The master

program takes the form:

$$[M] \equiv \min\{\min \sum_g \lambda_g : \tag{13}$$

$$\sum_g x_i^g \lambda_g = 1 \ \ \forall i \in I \tag{14}$$

$$\lambda_g \in \{0,1\} \ \ \forall g \in G.\} \tag{15}$$

where G denotes the set of "feasible packings" satisfying (10-12) and vector x^g defines the associated solution $g \in G$. When solving its linear relaxation by column generation, the pricing problem takes the form:

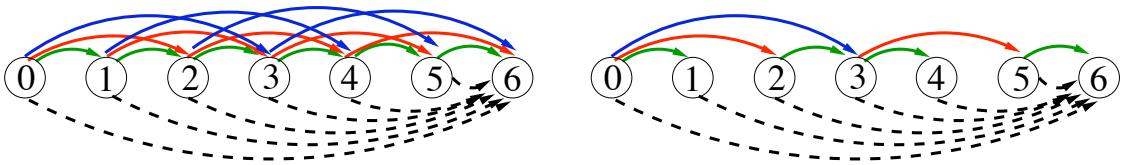
$$[SP] \equiv \min\{\delta - \sum_i \pi_i x_i : (x, \delta) \in X\} \tag{16}$$

where π are dual variables associated to (14) and

$$X = \{(x^g, \delta^g)\}_{g \in G} = \{(x, \delta) \in \{0,1\}^{n+1} : \sum_i s_i x_i \leq C\ \delta\}.$$

The subproblem can be set as the search for a shortest path in an acyclic network corresponding to the decision graph that underlies a dynamic programming solution of the knapsack subproblem: the nodes $v \in \{0, \ldots, C\}$ are associated with capacity consumption levels; each item, $i \in I$, gives rise to arcs (u, v), with $v = u + s_i$; wasted bin capacity is modeled by arcs (v, C) for $v = 0, \ldots, C-1$. A numerical example is given on the left of Figure 3.

Figure 3: *Knapsack network for* $C = 6$, $n = 3$, *and* $s = (1, 2, 3)$



The network flow model for the subproblem yields an extended formulation for the subproblem in terms of variables $f_{uv}^i = 1$ if item i is in the bin in position u to $v = u + s_i$. The subproblem reformulation takes the form:

$$\{(f, \delta) \in \{0,1\}^{n*m+1} : \sum_{i,v} f_{0v}^i + f_{0C} = \delta \tag{17}$$

$$\sum_{i,u} f_{uv}^i = \sum_{i,u} f_{vu}^i + f_{v,C} \quad v = 1, \ldots, C-1 \tag{18}$$

$$\sum_{i,u} f_{uC}^i + \sum_v f_{vC} = \delta \tag{19}$$

$$0 \leq f_{uv}^i \leq 1 \quad \forall i, u, v = u + s_i \ \} \tag{20}$$

(Superscript i is redundant, but it shall help to simplify the notation below.)

Subproblem extended formulation (17-20) leads in turn to an extended formulation for the original problem in terms of aggregate arc flow variables over all subproblems associated with each bin $k = 1, \ldots, n$: $F^i_{uv} = \sum_k f^{ik}_{uv}$, $F_{vC} = \sum_k f^k_{vC}$, and $\Delta = \sum_k \delta^k$. The extended formulation takes the form:

$$[R] \equiv \min\{\Delta : \tag{21}$$

$$\sum_{(u,v)} F^i_{uv} = 1 \quad \forall i \tag{22}$$

$$\sum_{i,v} F^i_{0v} + F_{0C} = \Delta \tag{23}$$

$$\sum_{i,u} F^i_{uv} = \sum_{i,u} F^i_{vu} + F_{vC} \quad v = 1, \ldots, C-1 \tag{24}$$

$$\sum_{i,u} F^i_{uC} + \sum_{v} F_{vC} = \Delta \tag{25}$$

$$F^i_{uv} \in \{0,1\} \quad \forall i, (u,v) : v = u + s_i \ \}. \tag{26}$$

Valerio de Carvalho [19] proposes to solve the linear relaxation of (21-26) by column generation: iteratively solve a partial formulation stemming from a restricted set of variables F^i_{uv}, collect the dual solution π associated to (22), solve pricing problem (16), transform its solution, x^*, into a path flow that can be decomposed into a flow on the arcs, a solution $f(x^*)$ to (17-20), and add in (21-26) the missing arc flow variables $\{F^i_{vu} : f^i_{vu}(x^*) > 0\}$, along with the missing flow balance constraints active for $f(x^*)$.

Observe that (17-20) is only an approximation of the extended network flow formulation associated to the dynamic programming recursion to solve a 0-1 knapsack problem. A dynamic programming recursion for the bounded knapsack problem yields state space $\{(j,b) : j = 0, \ldots, n; b = 0, \ldots, C\}$, where (j,b) represents the state of a knapsack filled up to level b with a combination of items $i \in \{1, \ldots, j\}$. Here, it has been aggregated into state space: $\{(b) : b = 0, \ldots, C\}$. This entails relaxing the subproblem to an unbounded knapsack problem. Hence, feasible solutions to (17-20) can involve multiple copies of the same item. Because (17-20) models only a relaxation of the 0-1 knapsack subproblem, the LP relaxation of (21-26) is weaker than that of the standard master program (13-15). For instance, consider the numerical example with $C = 100$, $n = 5$ and $s = (51, 50, 34, 33, 18)$. Then, the LP value of (13-15) is 2.5, while in (21-26) there is a feasible LP solution of value 2 which is $F^1_{0,51} = 1$, $F^3_{51,85} = F^5_{51,69} = F^5_{69,87} = \frac{1}{2}$, $F^2_{0,50} = F^2_{50,100} = \frac{1}{2}$, $F^3_{0,34} = F^4_{34,67} = F^4_{67,100} = \frac{1}{2}$. In [19], to avoid symmetries and to strengthen the extended formulation, arcs associated to an item are only defined if the tail node can be reached with a filling using items larger than s_i (as in the right part of Figure 3). Note that our numerical example of a weaker LP solution for [R] does not hold after such strengthening.

1.3 Multi-Commodity Capacitated Network Design

Frangioni and Gendron [8] applied a column-and-row generation technique to a multi-commodity capacitated network design problem. Given a directed graph $G = (V, A)$ and commodities: $k = 1, \ldots, K$, with a demand d^k of flow between origin and destination $(o^k, t^k) \in V \times V$, the problem is to assign an integer number of nominal capacity on each arc to allow for a feasible routing of traffic for all commodities, while minimizing routing and capacity installation cost. In [8], split flows are allowed and hence flow variables are continuous. A formulation is:

$$[F] \equiv \min\{\sum_{ijk} c_{ij}^k \, x_{ij}^k + \sum_{ij} f_{ij} \, y_{ij} \tag{27}$$

$$\sum_j x_{ji}^k - \sum_j x_{ij}^k = d_i^k \quad \forall i, k \tag{28}$$

$$\sum_k x_{ij}^k \leq u_{ij} \, y_{ij} \quad \forall i, j \tag{29}$$

$$0 \leq x_{ij}^k \leq d^k \, y_{ij} \quad \forall i, j, k \tag{30}$$

$$x_{ij}^k \geq 0 \quad \forall i, j, k \tag{31}$$

$$y_{ij} \in I\!N \quad \forall i, j\} \tag{32}$$

where $d_i^k = d^k$ if $i = o^k$, $d_i^k = -d^k$ if $i = t^k$, and $d_i^k = 0$ otherwise. Variables x_{ij}^k denote the flow of commodity k in arc $(i, j) \in A$. The design variables, y_{ij}, consists in selecting an integer number of nominal capacity on each arc $(i, j) \in A$. The problem decomposed into a continuous knapsack subproblem with varying capacity for each arc $(i, j) \in A$: $X^{ij} = \{(x, y) \in \mathbb{R}_+^K \times I\!N : \sum_k x^k \leq u_{ij} \, y, \; x^k \leq d^k \, y \; \forall k\}$. An extended formulation for the subproblem arises from unary disaggregation of the design variables: let $y_{ij}^s = 1$ and $x_{ij}^{ks} = x_{ij}^k$ if $y_{ij} = s$ for $s \in \{1, \ldots, s_{ij}^{\max}\}$ with $s_{ij}^{\max} = \lceil \frac{\sum_k d^k}{u_{ij}} \rceil$. Then, the subproblem associated to arc (i, j) can be reformulated as:

$$Z^{ij} = \{(x_{ij}^{ks}, y_{ij}^s) \in \mathbb{R}_+^{K \times s_{ij}^{\max}} \times \{0, 1\}^{s_{ij}^{\max}} :$$

$$\sum_s y_{ij}^s \leq 1, \quad (s-1) \, u_{ij} \, y_{ij}^s \leq \sum_k x_{ij}^{ks} \leq s \, u_{ij} \, y_{ij}^s \; \forall s, \quad x_{ij}^{ks} \leq \min\{d^k, s \, u_{ij}\} \, y_{ij}^s \; \forall k, s\}$$

Its continuous relaxation gives the convex hull of its integer solutions and its projection gives the convex hull of X^{ij} solutions as shown by Croxton, Gendron and Magnanti [5]: reformulation Z^{ij} of subproblem X^{ij} can be obtained as the union of polyhedra associated with each integer value of $y_{ij} = s$ for $s = 0, \ldots, s_{ij}^{\max}$.

These subproblem reformulations yields a reformulation for the original problem:

$$[\mathrm{R}] \equiv \min\{\sum_{ijks} c_{ij}^k \, x_{ij}^{ks} + \sum_{ijs} f_{ij} \, s \, y_{ij}^s \tag{33}$$

$$\sum_{js} x_{ji}^{ks} - \sum_{js} x_{ij}^{ks} = d_i^k \quad \forall i,k \tag{34}$$

$$(s-1)\, u_{ij} \, y_{ij}^s \leq \sum_k x_{ij}^{ks} \leq s \, u_{ij} \, y_{ij}^s \quad \forall i,j,s \tag{35}$$

$$0 \leq x_{ij}^{ks} \leq \min\{d^k, s \, u_{ij}\} \, y_{ij}^s \quad \forall i,j,k,s \tag{36}$$

$$\sum_s y_{ij}^s \leq 1 \quad \forall i,j \tag{37}$$

$$y_{ij}^s \in \{0,1\} \quad \forall i,j,s\} \tag{38}$$

On the other hand, a Dantzig-Wolfe reformulation can be derived based on subsystems X^{ij} or equivalently Z^{ij}. Let $G^{ij} = \{(x^g, y^g)\}_{g \in G^{ij}}$ be the enumerated set of extreme solutions to Z^{ij}. A column g is associated with a given capacity installation level σ: $y_s^g = 1$ for a given $s = \sigma$ and zero for $s \neq \sigma$ while the associated flow vector $x_{ks}^g = 0$ for $s \neq \sigma$ and define an extreme LP solution for $s = \sigma$. Then, Dantzig-Wolfe master takes the form

$$[\mathrm{M}] \equiv \min\{ \sum_{i,j,s,g \in G^{ij}} (c_{ij}^k \, x_{ks}^g + f_{ij} \, s \, y_s^g) \, \lambda_g^{ij} \tag{39}$$

$$\sum_{js} \sum_{g \in G^{ij}} x_{ks}^g \, \lambda_g^{ij} - \sum_{js} \sum_{g \in G^{ij}} x_{ks}^g \, \lambda_g^{ij} = d_i^k \quad \forall i,k \tag{40}$$

$$\sum_{g \in G^{ij}} \lambda_g^{ij} \leq 1 \quad \forall i,j \tag{41}$$

$$\lambda_g^{ij} \in \{0,1\} \quad \forall i,j,g \in G^{ij}\} \tag{42}$$

When solving [M] by column generation, the pricing problems take the form:

$$[SP^{ij}] \equiv \min\{\sum_{ks} c_{ij}^k \, x_{ij}^{ks} + \sum_s f_{ij} \, s \, y_{ij}^s : (\{x_{ij}^{ks}\}_{ks}, \{y_{ij}^s\}_s) \in Z^{ij}\} \tag{43}$$

for each arc (i,j).

Fangioni and Gendron [8] proceed to solve reformulation [R] by adding dynamically the y_{ij}^s variable and associated x_{ij}^{ks} variables for a given s at the time; i.e., for each arc (i,j), they include the solution $y_{ij} = s$ that arises as the solution of a pricing subproblem (43) over Z^{ij}, while a negative reduced cost subproblem solution is found. Constraints (35-36) that are active in the generated pricing problem solutions are added dynamically to [R]. In comparison, a standard column generation approach applied to [M] requires more iterations to converge as shown experimentally in [9].

This comparative advantage of the approach based of reformulation [R] has an intuitive explanation: for a fixed $y_{ij} = s$, one might need to generate several columns in [M] associated to different extreme continuous subproblem solution in the x variables, while

when working with [R], the optimization in the x variables is decoupled from that of the capacity setting, y. Indeed, in a column-and-row generation approach of formulation [R], once variable y^s_{ij} is included, all extreme points of polyhedral descriptions associated to $y_{ij} = s$ are feasible, but in the restricted master [M], the only feasible extreme points in x are those that were generated. Thus, the interest of applying column generation to [R] rather than [M] is to allow for a direct optimization of the trivial continuous part of the subproblem solution instead of proceeding by enumeration of the attractive extreme continuous solutions.

2. The Generic Procedure

Assume a pure integer program that can be stated in the form [F]:

$$\begin{array}{lrcl} & \min c\,x & & (44)\\ [\mathrm{F}] & A\,x & \geq & a \quad (45)\\ & B\,x & \geq & b \quad (46)\\ & x & \in & \mathbb{N}^n \quad (47) \end{array}$$

with an identified subsystem defined by

$$P = \{x \in \mathbb{R}^n_+ : Bx \geq b\} \quad \text{and} \quad X = P \cap \mathbb{Z}^n \tag{48}$$

where $A \in \mathbb{Q}^{m_1 \times n}$ and $B \in \mathbb{Q}^{m_2 \times n}$ are rational matrices, while $a \in \mathbb{Q}^{m_1}$ and $b \in \mathbb{Q}^{m_2}$ are rational vectors. X (resp. [F]) is assumed to be a pure integer program that is feasible and bounded. Extension to the unbounded case or mixed integer case is merely a question of introducing more notations.

Assumption 1 *There exists a polyhedron $Q = \{z \in \mathbb{R}^e_+ : H\,z \geq h, z \in \mathbb{R}^e_+\}$, defined by a rational matrix $H \in \mathbb{Q}^{f \times e}$ and a vector $h \in \mathbb{Q}^f$, and a linear transformation T defining the projection:*

$$z \in \mathbb{R}^p_+ \longrightarrow x = (T\,z) \in \mathbb{R}^n_+ \; ;$$

such that,

(i) Q defines an extended formulation for $conv(X)$, i.e.,

$$conv(X) = proj_x Q = \{x \in \mathbb{R}^n_+ : x = T\,z;\; H\,z \geq h;\; z \in \mathbb{R}^e_+\} \; ;$$

(ii) $Z = Q \cap \mathbb{Z}^e_+$ defines an extended IP-formulation for X, i.e.,

$$X = proj_x Z = \{x \in \mathbb{R}^n_+ : x = T\,z;\; H\,z \geq h;\; z \in \mathbb{Z}^e_+\} \; .$$

Condition (i) is the core of Assumption 1, while condition (ii) is merely a technical restriction that simplify the presentation. It also permits one to define branching restrictions directly in the reformulation. We also assume that Z is bounded to simplify the presentation. The dimension $e + f$ of the reformulation is typically much larger than $n + m_2$: while $n + m_2$ (or n at least) is expected to be polynomial in the input size, $e + f$ can have much higher polynomial degree, or even be pseudo-polynomial/exponential in the input size.

2.1 Reformulations

The subproblem extended formulation immediately gives rise to a reformulation of [F] to which we refer by [R]:

$$\begin{aligned} & \min c\,T\,z && && (49)\\ [\mathrm{R}] \qquad & A\,T\,z && \geq\ a && (50)\\ & H\,z && \geq\ h && (51)\\ & z && \in\ \mathbb{Z}^e_+\ . && (52) \end{aligned}$$

The standard Dantzig-Wolfe reformulation approach is a special case where X is reformulated as:

$$X = \{x = \sum_{g\in G} x^g \lambda_g : \sum_{g\in G} \lambda_g = 1, \lambda_g \in \{0,1\}^{|G|}\}, \tag{53}$$

G defining the set of generators of X (as they are called in [23]), i.e., G is the set of integer solutions of X in the case where X is a bounded pure integer program as assumed here. Then, the reformulation takes a form known as the master program, to which we refer by [M]:

$$\begin{aligned} & \min \sum_{g\in G} c\,x^g\,\lambda_g && && (54)\\ [\mathrm{M}] \qquad & \sum_{g\in G} A\,x^g\,\lambda_g && \geq\ a && (55)\\ & \sum_{g\in G} \lambda_g && =\ 1 && (56)\\ & \lambda && \in\ \{0,1\}^{|G|}\ . && (57) \end{aligned}$$

Let $[R_{LP}]$ and $[M_{LP}]$ denote respectively the linear programming relaxation of [R] and [M], while [D] denotes the dual of $[M_{LP}]$. Let

$$v^R_{LP} = \min\{c\,T\,z :\ A\,T\,z \geq a,\ H\,z \geq h, z \in \mathbb{R}^e_+\}\ ,$$

$$v^M_{LP} = \min\{\sum_{g\in G} c\,x^g\,\lambda_g :\ \sum_{g\in G} A\,x^g\,\lambda_g \geq a,\ \sum_{g\in G} \lambda_g = 1, \lambda \in \mathbb{R}^{|G|}_+\}\ ,\ \text{and}$$

$$v^D_{LP} = \max\{\pi\,a + \nu :\ \pi\,A\,x^g + \nu \leq c\,x^g\ \forall g \in G,\ \pi \in \mathbb{R}^{m_1}_+,\ \nu \in \mathbb{R}^1\} \tag{58}$$

be respectively the linear programming (LP) relaxation value of [R], [M], and [D].

Observation 1 *Under Assumption 1, the linear programming relaxation optimum value of both [R] and [M] are equal to the Lagrangian dual value obtained by dualizing constraints $A\,x \geq a$, i.e.,*

$$v^* = v^R_{LP} = v^M_{LP} = v^D_{LP}\ ,$$

where $v^* := \min\{cx : A\,x \geq a, x \in conv(X)\}$.

This is a direct consequence of Assumption 1. Note that the dual bound v^* obtained via such reformulations is often tighter than the linear relaxation value of the original formulation [F] (as typically $conv(X) \subset P$).

Given the potential large size of [R], both in terms of number of variables and constraints, one can solve its LP relaxation using dynamic column-and-row generation. If one assumes an explicit description of [R], the standard procedure would be to start off with a restricted set of variables and constraints (including possibly artificial variables to ensure feasibility) and to add iteratively negative reduced cost columns and violated rows by inspection. An alternative pricing-and-separation strategy is to implement a hybrid between the standard Dantzig-Wolfe column generation approach for [M] and the above standard dynamic handling of [R]: it consists in generating columns and rows for [R] not one at the time but by lots, each lot corresponding to a solution z^s of Z (which projects onto $x^s = Tz^s$) along with the constraints (51) that need to be enforced for that solution.

2.2 Restricted Extended Formulation

Let $\{z^s\}_{s\in S}$ be the enumerated set of solutions z^s of $Z \subseteq \mathbb{Z}^e_+$. Then, $\overline{S} \subset S$, defines an enumerated subset of solutions: $\{z^s\}_{s\in\overline{S}}$.

Definition 1 *Given a solution z^s of Z, let $J(z^s) = \{j : z^s_j > 0\} \subseteq \{1,\ldots,e\}$ be the support of solution vector z^s and let $I(z^s) = \{i : H_{ij} \neq 0$ for some $j \in J(z^s)\} \subseteq \{1,\ldots,f\}$ be the set of constraints of Q that involve some non zero components of z^s. The "restricted reformulation" $[\overline{R}]$ defined by a subset $\overline{S} \subset S$ of solutions to Z is:*

$$\min c\,\overline{T}\,\overline{z} \tag{59}$$

$$[\overline{R}] \qquad A\,\overline{T}\,\overline{z} \geq a \tag{60}$$

$$\overline{H}\,\overline{z} \geq \overline{h} \tag{61}$$

$$\overline{z} \in \mathbb{Z}^{|\overline{J}|}_+ \tag{62}$$

where $\overline{z}$ (resp. $\overline{h}$) is the restriction of z (resp. h) to the components of $\overline{J} = \cup_{s\in\overline{S}} J(z^s)$, $\overline{H}$ is the restriction of H to the rows of $\overline{I} = \cup_{s\in\overline{S}} I(z^s)$ and the columns of $\overline{J}$, while $\overline{T}$ is the restriction of T to the columns of $\overline{J}$.

Assume that we are given a subset $\overline{S} \subset S$. We define the associated set

$$\overline{G} = G(\overline{S}) = \{g \in G :\ x^g = T\,z^s \text{ for some } s \in \overline{S}\}$$

which in turn defines a restricted formulation $[\overline{M}]$. Let $[\overline{R}_{LP}]$ and $[\overline{M}_{LP}]$ denote the LP relaxation of the restricted formulations $[\overline{R}]$ and $[\overline{M}]$; while $v^{\overline{R}}_{LP}$, $v^{\overline{M}}_{LP}$ denote the corresponding LP value. Although in the end $v^* = v^R_{LP} = v^M_{LP}$, as stated in Observation 1, the value of the restricted formulations may differ.

Proposition 1 *Let [$\overline{R}$] and [$\overline{M}$] be the restricted versions of formulations [R] and [M] both associated to the same subset $\overline{S} \subset S$ of subproblem solutions. Under Assumption 1, their linear relaxation values are such that:*

$$v^* \leq v^{\overline{R}}_{LP} \leq v^{\overline{M}}_{LP} \tag{63}$$

Proof: The first inequality results from the fact that $[\overline{R}_{LP}]$ only includes a subset of variables of [R] whose LP value is $v^R_{LP} = v^*$, while the missing constraints are redundant by definition of the current restricted formulation. Indeed, letting $\tilde{z}$ be a solution to the current restricted reformulation $[\overline{R}_{LP}]$, observe that all the missing constraints must be satisfied by the solution that consists in extending $\tilde{z}$ with zero components, i.e. $(\tilde{z}, 0)$ defines a solution to $[R_{LP}]$; for otherwise such missing constraint would involve some components of subproblem solutions in $S \setminus \overline{S}$ in contradiction to the fact that solutions of $\overline{S}$ satisfy all constraints of Z. The second inequality is derived from the fact that any solution $\tilde{\lambda}$ to the LP relaxation of $\overline{M}$ has its counterpart $\tilde{z} = \sum_{g \in G} z^g \, \tilde{\lambda}_g$ that defines a valid solution for $[\overline{R}_{LP}]$, where $z^g \in Z$ is obtained by lifting solution $x^g \in X$ (the existence of the associated z^g is guarantied by Assumption 1-(ii)). ∎

The second inequality can be strict: solutions in $[\overline{R}_{LP}]$ do not always have their counterpart in $[\overline{M}_{LP}]$ as illustrated in the numerical example of Section 1.1. This is an important observation that justify considering [R] instead of [M].

Remark 1 *As noted in the proof of Proposition 1, (61) defines a feasible system because it is build from feasible solutions to Z: i.e., $\{\overline{z} \in \mathbb{Z}_+^{|\overline{J}|} : \overline{H}\,\overline{z} \geq \overline{h}\} \neq \emptyset$. However, the restricted reformulation [$\overline{R}$] can be infeasible due to constraints (60), hence artificial variables can be used to patch non-feasibility until set $\overline{S}$ is expanded. (Artificial variables are eliminated from the solution latter through raising their costs if need be.) To avoid this technicality, we assume in the sequel that $\overline{S}$ has been properly initialized to guarantee the feasibility of $[\overline{R}_{LP}]$.*

2.3 Column Generation

The procedure given in Table 1 is a dynamic column-and-row generation algorithm for the linear relaxation of [R]. It is a hybrid method that generates columns for $[M_{LP}]$ while getting new dual prices from a restricted version of $[R_{LP}]$. Its validity derives from the observations made in Proposition 2.

Proposition 2 *Let (π, σ) denote an optimal dual solution to $[\overline{R}_{LP}]$, associated to constraints (60) and (61) respectively. Let z^* be the subproblem solution obtained in Step 2 of the procedure of Table 1 and $\zeta = (c - \pi A)\, T\, z^*$ be its value. Then:*

(i) The Lagrangian bound: $L(\pi) = \pi\, a + (c - \pi A)\, T\, z^* = \pi\, a + \zeta$, defines a valid dual bound for [M_{LP}] and (π, ζ) defines a feasible solution of [D], the dual of [M_{LP}] defined in (58). Hence, bound β, that is defined recursively in the procedure of Table 1, is a valid dual bound for [F], and $\beta \leq v^*$.

(ii) If $v^{\overline{R}}_{LP} \leq \beta$ (i.e. when the stopping condition in Step 3 is satisfied), then $v^* = \beta$ and (π, ζ) defines an optimal solution to [D].

(iii) If $v^{\overline{R}}_{LP} > \beta$, then $[(c - \pi A)\, T - \sigma\, H] z^* < 0$. Hence, some of the component of z^* were not present in $[\overline{R}_{LP}]$ and have negative reduced cost for the current dual solution (π, σ).

(iv) Inversely, when $[(c - \pi A)\, T - \sigma\, H] z^* \geq 0$, i.e., if the generated column has non negative reduced cost in $[\overline{R}_{LP}]$, then $v^{\overline{R}}_{LP} \leq \beta$ (the stopping condition of Step 3 must be satisfied) and (π, ν) defines a feasible solution to formulation [D] defined in (58) for $\nu = \sigma\, h$.

Proof:

(i) For any $\pi \geq 0$, and in particular for the current dual solution associated to constraints (60), $L(\pi)$ defines a valid Lagrangian dual bound on $[F]$. Moreover, as $\zeta = \min\{(c - \pi\, A)\, T\, z :\ z \in Z\} = \min\{(c - \pi\, A)\, x :\ x \in X\}$, (π, ζ) defines a feasible solution of $[D]$ and hence its value, $\pi\, a + \zeta$, is a valid dual bound on $[M_{LP}]$.

(ii) From point (i) and Proposition 1, we have $L(\pi) \leq \beta \leq v^* \leq v^{\overline{R}}_{LP}$. When the stopping condition in Step 3 is satisfied, the inequalities turn into equalities.

(iii) When $v^{\overline{R}}_{LP} > \beta \geq L(\pi)$, we note that $\sigma\, h > (c - \pi\, A)\, T z^*$ because $v^{\overline{R}}_{LP} = \pi\, a + \sigma\, h$ and $L(\pi) = \pi\, a + \zeta = \pi\, a + (c - \pi\, A)\, T z^*$. As $H z^* \geq h$ and $\sigma \geq 0$, this implies that $[(c - \pi\, A)\, T - \sigma\, H] z^* < 0$. Assume by contradiction that each component of z^* has non negative reduced cost for the current dual solution of $[\overline{R}_{LP}]$. Then, the aggregate sum, $[(c - \pi\, A) T - \sigma\, H] z^*$ cannot be strictly negative for $z^* \geq 0$. As (π, σ) is an optimal dual solution to $[\overline{R}_{LP}]$, all variables of $[\overline{R}_{LP}]$ have positive reduced cost. Thus, the negative reduced cost components of z^* must have been absent from $[\overline{R}]$.

(iv) Because $H z^* \geq h$, $[(c - \pi A)\, T] z^* \geq \sigma\, H\, z^*$ implies $(c - \pi\, A)\, T z^* \geq \sigma\, h$, i.e., $\zeta \geq \sigma\, h$. In turn, $\zeta \geq \sigma\, h$ implies that (π, ν) with $\nu = \sigma\, h$ is feasible for [D] (all constraints of [D] are satisfied by (π, ν)). Note that $\zeta \geq \sigma\, h$ also implies $v^{\overline{R}}_{LP} \leq \beta$, as $v^{\overline{R}}_{LP} = \pi\, a + \sigma\, h \leq \pi\, a + \zeta = L(\pi) \leq \beta$. ∎

Remark 2 *The column generation pricing problem of Step 2 in Table 1 is designed for formulation $[M_{LP}]$ and not for formulation $[R_{LP}]$: it ignores dual prices, σ, associated to subproblem constraints (61).*

Remark 3 *For the column generation procedure of Table 1, pricing can be operated in the original variables, x, in Step 2. Indeed, $\min\{(c - \pi A)\, T\, z :\ z \in Z\} \equiv \min\{(c - \pi A)\, x :\ x \in X\}$. But, to implement Step 4, one would then need to be able to lift the solution $x^* := argmin\{(c - \pi A)\, x : x \in X\}$ in the z-space in order to add variables to [R], i.e., one must have a procedure to define z^* such as $x^* = T\, z^*$.*

Table 1: *Dynamic column-and-row generation for [R_{LP}].*

Step 0: Initialize the dual bound, $\beta := -\infty$, and the subproblem solution set $\overline{S}$ so that the linear relaxation of $[\overline{R}]$ is feasible.

Step 1: Solve the LP relaxation of $[\overline{R}]$ and record its value $v^{\overline{R}}_{LP}$ and the dual solution π associated to constraints (60).

Step 2: Solve the pricing problem: $z^* := argmin\{(c - \pi A)\, T\, z : z \in Z\}$, and record its value $\zeta := (c - \pi A)\, T\, z^*$.

Step 3: Compute the Lagrangian dual bound: $L(\pi) := \pi\, a + \zeta$, and update the dual bound $\beta := \max\{\beta, L(\pi)\}$. If $v^{\overline{R}}_{LP} \leq \beta$, STOP.

Step 4: Update the current bundle, $\overline{S}$, by adding solution $z^s := z^*$ and update the resulting restricted reformulation $[\overline{R}]$ according to Definition 1. Then, goto Step 1.

Remark 4 *The procedure of Table 1 is a generalization of the standard "text-book" column generation algorithm (see f.i. [4]). Applying this procedure to formulation [M] reproduces exactly the standard column generation approach for solving the LP relaxation of [M]. Indeed, [M] is a special case of reformulation of type [R] where system $H\, z \geq h$ consists of a single constraint, $\sum_{g \in G} \lambda_g = 1$. The latter needs to be incorporated in the restricted reformulation along with the first included column, λ_g, from which point further extensions consist only in including further columns.*

Observation 2 *Note that $\nu = \sigma\, h$ plays the role of the dual solution associated to the convexity constraint (56). It defines a valid cut-off value for the pricing sub-problem, i.e., if $\zeta \geq \sigma\, h$ the stopping condition in Step 3 is satisfied.*

This observation derives from the proof of Proposition 2-(iv).

2.4 Extension to approximate extended formulations

The column-and-row generation procedure for [R] provided in Table 1 remains valid under weaker conditions. Assumption 1 can be relaxed into:

Assumption 2 *Using the notation of Assumption 1, assume:*

(i) reformulation Q defines an improved formulation for X, although not an exact extended formulation: $conv(X) \subset proj_x Q \subset P$ where $proj_x Q\{x = T\, z :\ H\, z \geq h, z \in \mathbb{R}^e_+\}$;

(ii) moreover, assume conditions (ii) of Assumption 1.

Assumption 2, relaxing Assumption 1-(i), is often more realistic in many applications where the subproblem is NP-Hard. It also applies when one develops only an approximation of the extended formulation for X as in the proposal of [25] and in the bin-packing

example of Section 1.2.

Then, Observation 1 and Proposition 1 become respectively:

Observation 3 *Under Assumption 2,* $v_{LP} \leq v_{LP}^R \leq v^* = v_{LP}^M = v_{LP}^D$.

Proposition 3 *Under Assumption 2,* $v_{LP}^{\overline{R}} \leq v_{LP}^{\overline{M}}$ *and* $v^* \leq v_{LP}^{\overline{M}}$*; but one might have* $v_{LP}^{\overline{R}} < v^*$.

Proposition 2 still holds under Assumption 2 except for point (ii). However the stopping condition of Step 3 remains valid.

Proposition 4 *Under Assumption 2, the column-and-row generation procedure of Table 1 remains valid. In particular, the termination of the procedure remains guaranteed. On termination, one may not have the solution* v_{LP}^R *to* $[R_{LP}]$*, but one has a valid dual bound* β *that is at least as good, since*

$$v_{LP}^R \leq \beta \leq v^* = v_{LP}^M \ .$$

Proof: Observe that the proofs of points (i), (iii), and (iv) of Proposition 2 remain valid under Assumption 2. Hence, as long as the stopping condition of Step 3 is not satisfied, negative reduced cost columns are found for $[\overline{R}_{LP}]$ (as stated in Proposition 2-(iii)) that shall in turn lead to further decrease of $v_{LP}^{\overline{R}}$. Once the stopping condition, $v_{LP}^{\overline{R}} \leq \beta$, is satisfied however, we have $v_{LP}^R \leq v_{LP}^{\overline{R}} \leq \beta \leq v^*$, proving that the optimal LP value, v_{LP}^R, is then guaranteed to lead to a bound weaker than β. ∎

Thus, once $v_{LP}^{\overline{R}} \leq \beta$, there is no real incentive to further consider columns z^* with negative reduced cost components in $[\overline{R}_{LP}]$; although this may decrease $v_{LP}^{\overline{R}}$, there is no more guarantee that β shall increase in further iterations. Note that our purpose is to obtain the best possible dual bound for [F], and solving v_{LP}^R is not a goal in itself. Nevertheless, sufficient conditions to prove that $[R_{LP}]$ has been solved to optimality can be found in [7].

2.5 Extension to multiple subsystems

Consider the case where subsystem (46) is block diagonal. Then, original formulation [F] can be written as:

$$\begin{array}{lccccccccl}
\min & c^1x^1 & + & c^2\,x^2 & + & \dots & + & c^K\,x^K & & \\
 & A^1\,x^1 & + & A^2\,x^2 & + & \dots & + & A^K\,x^K & \geq & a \\
 & B^1\,x^1 & & & & & & & \geq & b^1 \\
 & & & B^2\,x^2 & & & & & \geq & b^2 \\
 & & & & & \ddots & & & \geq & \vdots \\
 & & & & & & & B^K\,x^K & \geq & b^K \\
 & x^1 & \in \mathbb{Z}_+^{n_1}, & x^2 & \in \mathbb{Z}_+^{n_2}, & \dots & x^K & \in \mathbb{Z}_+^{n_K} & , &
\end{array}$$

with K independent subproblems: $X^k = P^k \cap \mathbb{Z}^n$ with $P^k = \{x \in \mathbb{R}_+^n : B^k\,x \geq b^k\}$, that are assumed to admit an extended integer reformulation: $Q^k = \{(x,z) \in \mathbb{R}_+^n \times Z_+^p :$

$x = T^k\, z, H^k z \geq h^k\}$ such that $X^k = \text{proj}_x(Q^k)$.

The above analysis carry over to this special case, even when all subsystem are identical. If $A^k = A\ , B^k = B,\ b^k = b,\ T^k = T,\ H^k = H,\ h^k = h\ \forall k$, and hence $X^k = X\ , Q^k = Q\ \forall k$, then the original formulation can be casted in an aggregate form:

$$\begin{aligned} & \min c\, y & & & (64)\\ [\text{AF}] \qquad & A\, y & \geq & a & (65)\\ & y & = & \sum_k x^k & (66)\\ & x^k & \in & X \quad \forall k & (67) \end{aligned}$$

where y variables defined in (66) represent the aggregate value of subproblem solutions. The extended formulation can also be aggregated. It becomes

$$\begin{aligned} & \min c\, T\, w & & & (68)\\ [\text{AR}] \qquad & A\, T\, w & \geq & a & (69)\\ & H\, w & \geq & h & (70)\\ & w & \in & \mathbb{Z}^e_+ & (71) \end{aligned}$$

where

$$w = \sum_k z^k \tag{72}$$

and constraints (70) are obtained by surrogate relaxation: summing over k constraints $H\, z^k \geq h \quad \forall k$. While the aggregate master program takes the form:

$$\begin{aligned} & \min \sum_{g\in G} c\, x^g\, \lambda_g & & & (73)\\ [\text{AM}] \qquad & \sum_{g\in G} A\, x^g\, \lambda_g & \geq & a & (74)\\ & \sum_{g\in G} \lambda_g & = & K & (75)\\ & \lambda & \in & \mathbb{Z}^{|G|}_+\ . & (76) \end{aligned}$$

where $\lambda_g = \sum_k \lambda^k_g$ and constraint (75) is a surrogate relaxation of $\sum_g \lambda^k_g = 1\ \forall k$.

In practice, it is advisable to use the aggregate formulations for their smaller size and more importantly to avoid the symmetry that would arise from carrying different index k. Observe that these surrogate relaxations do not lead to weaker dual bounds.

Proposition 5 *Under Assumption 1, when all subsystem are identical, i.e. $X^k = X\ \forall k$, the linear programming relaxation optimum value of [R], [AR], [M], and [AM] are equal to the Lagrangian dual value obtained by dualizing constraints $A\, x \geq a$, i.e.,*

$$v^* = v^R_{LP} = v^{AR}_{LP} = v^M_{LP} = v^{AM}_{LP}\ ,$$

where $v^* := \min\{cx : \sum_k A^k\, x^k \geq a, x^k \in conv(X^k)\ \forall k\}$.

Proof: Any solution w to $[AR_{LP}]$ can be casted into solution $z^k = \frac{w}{K}$ for $k = 1, \ldots, K$ to $[R_{LP}]$; and any solution λ to $[AM_{LP}]$ can be casted into a solution $\lambda^k = \frac{\lambda}{K}$ for $k = 1, \ldots, K$ to $[M_{LP}]$.

3. Interest of the approach

Here, we review the motivations to consider applying column-and-row generation to [R] instead of standard column generation to [M] or a direct MIP-solver approach to [R]. We summarize the comparative pros and cons of the hybrid approach. We identify properties that are key for the method performance and we discuss two generic cases of reformulations where the desired properties take a special form: reformulations based on network flow models or on dynamic programming subproblem solver.

3.1 Pros and cons of a column-and-row generation approach

When compared to a direct solution of the extended formulation, the hybrid column-and-row approach could be seen as a way to handle dynamically the large size of the reformulation: the formulation is kept under control by way of managing its variables and constraints dynamically at the expense of loosing the comfort of a direct MIP solver handling. However, if the motivation was only to get around the issue of size, one would be better off using a standard column generation approach for [M] that yields a smaller restricted master program. Thus, the hybrid method is rather to be understood as an alternative to applying standard column generation to the Dantzig-Wolfe reformulation.

The primary interest for implementing column generation for [R] rather than for [M] is to exploit the second inequality of Proposition 1: a column generation approach to reformulation $[R_{LP}]$ can converge faster than one for $[M_{LP}]$ when there exist possible re-compositions of solutions in $[\overline{R}_{LP}]$ that would not be feasible in $[\overline{M}_{LP}]$. In the literature (f.i. in [19]), another motivation is put forward for using the column-and-row generation rather than standard column generation: [R] offers a richer model in which to define cuts or branching restrictions. Note however that although [R] provides new entities for branching or cutting decisions, one can implicitly branch or formulate cuts on the variables of [R] while working with [M]: provided one do pricing in the z-space, any additional constraint in the z-variables of the form $\alpha\, z \geq \alpha_0$ for [R] translates into a constraint $\sum_g \alpha\, z^g\, \lambda^g \geq \alpha_0$ for [M] (where z^g's denote generators) that can be accommodated in standard column generation approach for [M].

The drawbacks of a column-and-row approach, compared to applying standard column generation, are:

(i) having to handle a larger restricted linear program ($[\overline{R}_{LP}]$ has more variables and constraints than $[\overline{M}_{LP}]$ for a given $\overline{S}$);

(ii) having to manage dynamic row generation along side column generation;

(iii) having to face potential symmetries in the representation of solutions that might arise in the extended formulation; and

(iv) having to use a subproblem oracle specific to the subproblem extended formulation.

Indeed, as noted in Remark 3, pricing must be done in the z-variable space which might yield greater computing times than pricing in X.

3.2 Key properties characterizing the interest of the approach

From the above discussion, we gather that the applications of interest are those for which the hybrid column-and-row approach can be expected to converge faster than standard column generation, to reach the same quality dual bound, to implicitly provide entities for branching or defining cuts, while allowing the use of a pricing procedure in the original variables if possible and trying to avoid symmetric representations of solutions. These desirable properties are formalized below.

Faster convergence results can be expected only if the following property holds, that we call a *"recombination property"*.

Property 1 *("Recombination")*
Given $\overline{S} \subset S$, $\exists \tilde{z} \in R_{LP}(\overline{S})$, *such that* $\tilde{z} \notin conv(Z(\overline{S}))$.

Property 1 implies that one might not need to generate further columns to achieve some solutions in $Q \setminus conv(Z(\overline{S}))$; hence, the column generation approach to $[R_{LP}]$ might need fewer iterations to converge compared to column generation applied to $[M_{LP}]$.

The dual bound quality is guarantied by what we call a *"convexification property"*.

Property 2 *("Convexification")*
Given $\overline{S} \subset S$, $\forall \tilde{z} \in R_{LP}(\overline{S})$, *one has* $(T\,\tilde{z}) \in conv(X)$;

Assumption 1-(i), with Definition 1, implies Property 2, that can be seen as form of rewording of Proposition 1. Note however that the *"convexification property"* does not hold under Assumption 2.

Branching can be performed simply by enforcing integrality restriction on the z variables if the following property holds, that we call the *"Integrality property"*.

Property 3 *("Integrality")*
Given $\overline{S} \subset S$, $\forall \tilde{z} \in R(\overline{S})$, *one has* $(T\,\tilde{z}) \in X$.

Assumption 1-(ii), together with Definition 1, implies Property 3. But Property 3 does not generalize to the case of multiple identical subsystem giving rise to aggregate formulation [AR] presented in Section 2.5. Indeed, there is no counterpart to Proposition 5 for the relations between integer formulations and reformulations.

To alleviate the drawback of having to do pricing in the z-space, one must be able to lift any subproblem $x \in X$ into an equivalent solution z for the extended formulation of the subproblem. According to Assumptions 1 or 2, any subproblem extended solution $z \in Z$ can be associated with a solution $x \in X$ through the projection operation: $x = p(z) = T\,z$.

Inversely, given a subproblem solution $x \in X$, the system $T\,z = x$ must admit a solution $z \in Z$: one can define

$$p^{-1}(x) := \{z \in \mathbb{Z}^e_+ : T\,z = x;\ H\,z \geq h\}\ . \tag{77}$$

However, in practice, one needs an explicit operator:

$$x \in X \longrightarrow z \in p^{-1}(x)$$

or a procedure that returns $z \in Z$ given $x \in X$. Thus, the desirable property is what we call the *"Lifting property"*:

Property 4 *("Lifting")*
There exists a lifting procedure that transforms any subsystem solution $x \in X$ into a solution to the extended system $z \in Z$ such that $x = T\,z$.

As noted in Remark 3, when this property holds a pricing oracle in the x-space can be used in procedure of Table 1: in Step 2, compute $x^* := argmin\{(c - \pi A)\,x : x \in X\}$; in Step 4, define $z^* = p^{-1}(x^*)$. Otherwise, a pricing oracle on Z is required.

Observation 4 *A generic lifting procedure is to solve the integer feasibility program defined in (77).*

Note that solving (77) is typically much easier than solving the pricing subproblem, as constraint $T\,z = x$ already fix many z variables. However, in application specific context, it might be more efficient to make use of a combinatorial procedure for lifting. When the richer space of z-variables is exploited to derive cutting planes or branching constraints for the master program, it might induce new bounds or a new cost structure in the subproblem in Z that cannot be modeled in the X space. In such case, an optimization in x followed by a lifting procedure is ruled out.

Finally, let us discuss further the symmetry drawback. It is characterized by the fact that the set $p^{-1}(x^g)$ defined in (77) is often not limited to a singleton (as for instance in the bin packing example of Section 1.2, when using the underlying network of the left part of Figure 3). In the lack of uniqueness, the convergence of the solution procedure for $[R_{LP}]$ can be slowed down by iterating between different alternative representations of the same LP solution. Note that a branch-and-bound enumeration based on enforcing integrability of the z variables would also suffer from such symmetry.

In summary, a column generation approach for the extended formulation has any interest only when Property 1 holds; while Assumption 1 guarantees Properties 2 and 3. Property 4 is optional but if it holds any pricing oracle on X will do, until cut or branching constraint expressed in the z variable might require to price in the z-space. The combination of Property 4 and Property 1, leads to the desirable *"disaggregation and recombination property"*. We review below several important special cases where the desired disaggregation and recombination property holds, along side Properties 2 and 3.

3.3 The case of network flow reformulation

Assume that the extended formulation stems from reformulating a subproblem as a network flow problem: a subproblem solution $x \in X$ can be associated with a feasible arc flow in the network, $z \in Z$, that satisfies flow bounds on the arcs and flow balance constraints at the nodes. Note that extreme solutions $z \in Q$ are integer in this case; they map onto integer solutions x by the linear transformation T. In an application specific context, any subproblem solution x can typically easily be interpreted as a feasible flow along paths and/or cycles although the association may not be unique. Then, the flow decomposition theorem [1] yields a unique arc flow z and Property 4 is satisfied: transforming x into path and/or cycle flows and applying flow decomposition define an explicit lifting procedure.

Now, given a set of feasible flows $z^1, \ldots, z^k$, and their combined support graph, let the solution set $\overline{Q} = \overline{Q}(z^1, \ldots, z^k) = \{z \in \mathbb{R}^e_+ : \overline{H}\, z \geq \overline{h}, z \in \mathbb{R}^e_+\}$ be the restriction of the network flow formulation to the support graph of flows $z^1, \ldots, z^k$. Observe that $\overline{Q}$ holds any convex combinations of $z^1, \ldots, z^k$, but also solutions that can be defined from a convex combination plus a flow along a undirected cycle in the support graph. Indeed, for any pair of feasible flows, z^1 and z^2, the difference $w = z^1 - z^2$ is a cycle flow. By the flow decomposition theorem [1], w decomposes into elementary cycle flow w^A, w^B, ..., and $\tilde{z} = z^1 + \alpha\, w^A \in (\overline{Q} \setminus conv(z^1, z^2))$ for any elementary cycle w^A and $\alpha \in (0, 1)$. Hence, Property 1 holds.

This special class also encompasses extended formulations that are "equivalent" to network flow problems; for instance, when H is a consecutive 1 matrix that can be transformed into a node arc incidence matrix [1]. In particular, it encompasses time index formulation for scheduling problems as developped in Section 1.1 (beyond the extension to the case of parallel machines, one can also model in this way a machine with arbitrary capacity where jobs have unit capacity consumption. More generally, flow recombinations can be encountered in any extended formulation that include a network flow model as a subsystem; in particular, in multi commodity flow reformulations of network design problems.

It is interesting to observe that Property 3 remains valid even in the case of multiple identical sub-systems (developed in Section 2.5) when $\{z \in \mathbb{R}^e_+ : H\, z \geq h\}$ models a shortest path in an acyclic network. Then, the aggregate flow w can be decomposed into path flow (by the flow decomposition theorem [1]), each of which corresponds to a solution $x^g \in X$ and therefore an integer aggregate flow w solution to $[\overline{AR}]$ decomposes into an integer solution for [R].

3.4 The case of dynamic programming based reformulations

Another important special case is when the extended formulation is stemming from a dynamic programming solver for the subproblem [14]. Most discrete dynamic program entails finding a shortest (or longest) path in a directed acyclic decision graph, where nodes correspond to states (representing partial solutions) and arcs correspond to tran-

sitions (associated with partial decisions to extend solutions). This directly leads to a reformulation as a unit flow going from origin (empty solution) to destination (complete solution). Then, one is again in the special case of Section 3.3.

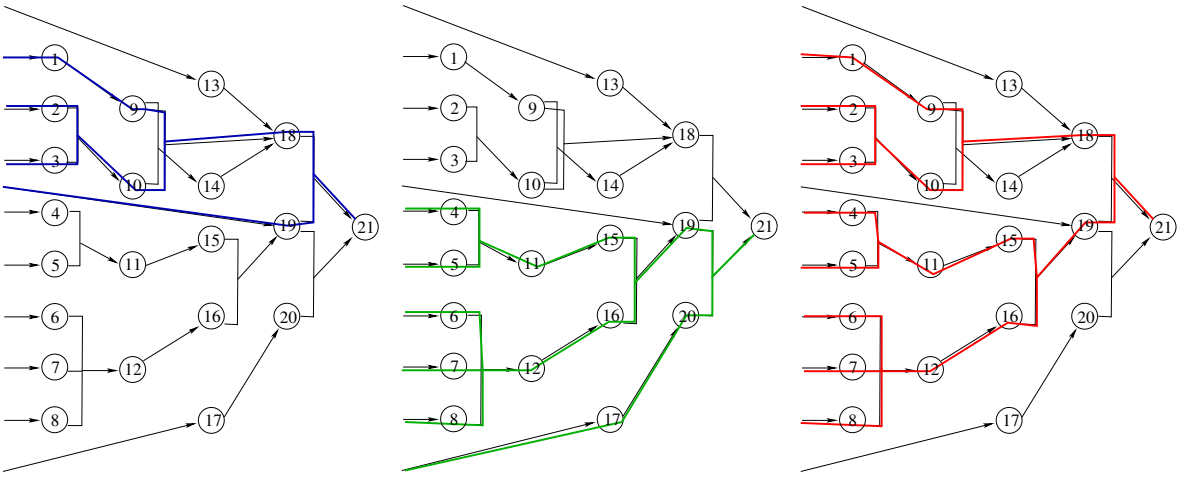
However, more complex dynamic programs may involve the composition of more than one intermediate states (representing partial solutions) into a single state (next stage partial solution). These can be modeled by hyper-arcs with a single head but multiple tails. Then, the extended paradigm developed by [14] consists in seeing a dynamic programming solution as a hyper-path (associated to a unit flow incoming to the final state) in a hyper-graph that satisfy two properties:

(i) *acyclic consistency* – there exists a topological indexing of the nodes such as, for each hyper-arc, the index of the head is larger than the index of the tail nodes;

(ii) *disjointness* – if a hyper-arc has several tails, they must have disjoint predecessor sets.

This characterization avoids introducing an initial state, but instead consider "boundary" arcs that have undefined tails: see Figure 4. The dynamic programs that can be modeled as a shortest path problem are a special case where the hyper-graph only has simple arcs with a single tail and hence the disjointness property does not have to be checked.

Figure 4: *In these hyper-graphs, underlying the paradigm of [14], nodes are indexed in acyclic order; node 21 represents the final state; hyper arcs may have multiple tail but a single head; "boundary" arcs that represent initialization conditions have no tail; a solution is defined by a unit flow reaching final node 21; when a unit flow exit an hyper-arc, a corresponding unit flow must enter in each of the tail nodes. In these graphs are depicted respectively solutions z^1 and z^2 that share a common intermediate node 19, and their recombination, $\hat{z}$.*



Following [14], consider a directed hyper-graph $G = (\mathcal{V}, \mathcal{A})$, with hyper-arc set $\mathcal{A} = \{(J, l) : J \subset \mathcal{V} \setminus \{l\},\ l \in \mathcal{V}\}$ and associated arc costs $c(J, l)$, a node indexing $\sigma : \mathcal{V} \to$

$\{1, \ldots, |\mathcal{V}|\}$ such that $\sigma(j) < \sigma(l)$ for all $j \in J$ and $(J, l) \in \mathcal{A}$ (such topological indexing exists since the hyper-graph is acyclic). The associated dynamic programming recursion takes the form:

$$\gamma(l) = \min_{(J,l)\in\mathcal{A}} \{c(J, l) + \sum_{j\in J} \gamma(j)\}$$

that can be computed recursively following the order imposed by indices σ, i.e., for $l = \sigma^{-1}(1), \ldots, \sigma^{-1}(|\mathcal{V}|)$. Solving this dynamic program is equivalent to solving the linear program:

$$\max\{u_f : \; u_l - \sum_{j\in J} u_j \leq c(J, l) \quad \forall (J, l) \in \mathcal{A}\} \tag{78}$$

where $f = \sigma^{-1}(|\mathcal{V}|)$ is the final state node. Its dual is

$$\min\{ \sum_{(J,l)\in\mathcal{A}} c(J, l)\, z_{(J,l)} \tag{79}$$

$$\sum_{(J,f)\in\mathcal{A}} z_{(J,f)} = 1 \tag{80}$$

$$\sum_{(J,l)\in\mathcal{A}} z_{(J,l)} = \sum_{(J',l')\in\mathcal{A}: l\in J'} z_{(J',l')} \quad \forall l \neq f \tag{81}$$

$$z_{(J,l)} \geq 0 \quad \forall (J, l) \in \mathcal{A}\} \tag{82}$$

that defines the reformulation Q for the subproblem.

In this generalized context, [14] gives an explicit procedure to obtain a solution z defining the hyper-arcs that are in the subproblem solution from the solution of the dynamic programming recursion: the hyper-arc selection is a dual solution to the linear program (78) that characterizes the dynamic program; given the specific assumption on the hyper-graph (acyclic consistency and disjointness), a greedy procedure allows one to obtain the dual complementary solution z. So if one uses the dynamic program as oracle, one can recover a solution x and associated complementary solution z. Alternatively, if subproblem solution x is obtained by another algorithm, one can easily compute distance labels, u_l, associated to nodes of the hyper-graph (u_l = the cost of partial solution associated to node l if this partial solution is part of x and $u_l = \infty$ otherwise) and apply the procedure of [14] to recover the complementary solution z. So, Property 4 is satisfied. The procedure is polynomial in the size of the hyper-graph.

Property 1 also holds. Indeed, given a hyper-path z, let $\chi(z, J, l)$ be the characteristic vector of the set of hyper-arcs (J', l') in the hyper-path defined by z that are such that either $(J', l') = (J, l)$ or $l' \in J$ or l' is a predecessor of a node $j \in J$. Now consider two hyper-paths z^1 and z^2 such that $z^1_{(J^1,l)} = 1$ and $z^2_{(J^2,l)} = 1$ for a given intermediate node l, with $J^1 \neq J^2$. Then, consider $\tilde{z} = z^1 - \chi(z^1, J^1, l) + \chi(z^2, J^2, l)$. Note that we have $\tilde{z} \in Z$ but $\tilde{z} \notin \mathit{conv}(z^1, z^2)$, as $z^1_{(J^1,l)} = 1$ and $\tilde{z}_{(J^1,l)} = 0$. See an illustration in Figure 4.

4. Numerical experimentation

Here we report on our numerical tests highlighting the comparative performance of column-and-row generation for the LP relaxation $[R_{LP}]$ of an extended formulation [R], versus a classic column generation approach applied to the LP relaxation $[M_{LP}]$ of a standard Dantzig-Wolfe reformulation [M], or a direct LP-solver approach applied to $[R_{LP}]$, the LP relaxation extended formulation [R] (when the latter is too large, we compare to solving $[F_{LP}]$, the LP relaxation of a compact formulation [F]). The approaches were implemented generically within the software platform BaPCod [22] (a problem-specific implementation is likely to produce better results). CPLEX 12 is used to solve both the LP relaxation of the extended formulation [R] or the master linear programs, while the MIP subproblems are solved using a specific oracle. Results are averages over randomly generated instances. Field *"cpu"* denotes the computational time (in seconds); *"sp"* denotes the number of calls to the pricing subproblem solver in the column(-and-row) generation procedure; *"%var"* denotes the number of variables generated in the restricted master (this number is expressed as a percentage of the number of variables in the full-blown extended formulation [R]). Additionally, *"%gap"* is reported in some applications to denote the difference between the dual bound obtained and the best known primal bound (the optimum solution in most case), as a percentage of the latter. Bounds are rounded to the next integer for integer objectives.

For applications where the subproblem is a knapsack problem, we used the solver of [15]. Then ,when using column-and-row generation, the solution in the original x variables is "lifted" to recover an associated solution z using a simple combinatorial procedure. For a 0-1 knapsack, we sort the items i for which $x_i^* = 1$ in non-increasing order of their size, s_i, and let $z_{uv}^* = 1$ for the arc (u, v) such that $u = \sum_{j<i} s_j x_j^*$ and $v = \sum_{j \leq i} s_j x_j^*$. Note that this procedure automatically eliminate some symmetries in formulation [R]. It can be extended to integer knapsack. For the other applications considered here, the subproblems are solved by dynamic programming and hence the z^* solution is obtained directly ($z_{uv}^* = 1$ if the optimum label at v is obtained using state transition from u).

4.1 Parallel Machines Scheduling

For the machine scheduling problem of Section 1.1, our objective function is the total weighted tardiness (this problem is denoted as $P \mid\mid \sum w_j T_j$). Instance size is determined by a triple $(n, m, p_{\max})$, where n is the number of jobs, m is the number of machines, and $p_{\max}$ is the maximum processing time of jobs. Instances are generated using the procedure of Potts and van Wassenhove [17]: integer processing times p_j are uniformly distributed in interval $[1, 100]$ and integer weights w_j in $[1, 10]$ for jobs j, $j = 1, \ldots, n$, while integer due dates have been generated from the uniform distribution $[P(1 - TF - RDD/2)/m, P(1 - TF + RDD/2)/m]$, where $P = \sum_j p_j$, TF is the tardiness factor, RDD is the relative range of due dates, $TF, RDD \in \{0.2, 0.4, 0.6, 0.8, 1\}$. For each instance size, 25 instances were generated, one for every couple of parameters (TF, RDD). For the single machine case, trivial instances are discarded: it suffices to schedule jobs in non-decreasing order of their due dates to observe if a solution of cost zero exists.

The results are presented in Table 2. The column-and-row generation approach significantly outperforms the other two. Moreover, its advantage increases with the increase of the instance size. Compared to standard column generation, cpu times and number of solved subproblems decrease by an order of magnitude for the larger instances. The recombination effect has an important impact: the number of calls to the pricing subproblem is up to two orders of magnitude smaller for column-and-row generation in comparison with standard column generation. However, the difference in solution time is lower due to a larger size of the master problem in the column-and-row generation approach. On average, only 5% of variables are needed to solve $[R_{LP}]$ to optimality by column-and-row generation.

			Cplex 12.1 for $[R_{LP}]$	Col. gen. for $[M_{LP}]$		Col-and-row gen for $[R_{LP}]$		
m	n	$p_{\max}$	*cpu*	*sp*	*cpu*	*sp*	*%var*	*cpu*
1	25	50	2.9	352	1.7	54	8.9	0.8
1	50	50	34.8	1559	41.7	82	6.7	5.9
1	100	50	381.7	9723	2531.0	112	6.1	47.3
1	25	100	11.3	378	2.3	75	5.9	1.6
1	50	100	155.4	1418	44.3	114	4.6	18.4
1	100	100	2039.6	10375	3436.3	155	4.5	182.3
2	25	100	7.2	208	0.7	62	5.4	0.5
2	50	100	198.6	641	10.0	93	4.5	3.1
2	100	100	4038.4	2697	198.2	115	4.5	30.5
4	50	100	35.1	441	3.4	90	4.4	1.5
4	100	100	726.1	1353	47.0	113	4.3	10.7
4	200	100	22441.6	4306	684.7	151	3.1	80.2

Table 2: Computational results for Machine Scheduling

4.2 Bin Packing

For the bin packing problem of Section 1.2, we compared solving $[F_{LP}]$ using Cplex, standard column generation for $[M_{LP}]$, and column-and-row generation for $[R_{LP}]$. We do not report statistics on solving $[R_{LP}]$ directly with Cplex, as solution times are prohibitive due to the huge number of variables. Instance classes "a2", "a3", and "a4" (the number refers to the average number of items per bin) contain instances with bin capacity equal to 4000 where item sizes are generated randomly in intervals $[1000, 3000]$, $[1000, 1500]$, and $[800, 1300]$, respectively. The results in Table 3 are averages over 5 instances. The "*%gap*" is always zero for formulations $[M_{LP}]$ and $[R_{LP}]$. For $[F_{LP}]$, *"gap"* is the absolute value of the difference between dual bound and the optimum solution (which we computed by branch-and-price). The column-and-row generation for $[R_{LP}]$ outperforms column generation for $[M_{LP}]$ for classes "a3", and "a4". Moreover, the advantage of the former increases with the size of test instances. For class "a2", the column-and-row generation does not lead to better cpu time than standard column generation because there are fewer recombinations given the larger average item size. Note that, although the reformulation

is based on Assumption 2, the dual bound obtained solving $[R_{LP}]$ is the same as for $[M_{LP}]$ on the tested instances.

		Cplex 12.1 for $[F_{LP}]$			Col. gen. for $[M_{LP}]$		Col-and-row gen. for $[R_{LP}]$		
class	n	*gap*	*%gap*	*cpu*	*sp*	*cpu*	*sp*	*%var*	*cpu*
"a2"	200	5.6	5.2	0.1	944	0.6	736	0.4	1.1
	400	8.6	4.0	0.8	2863	3.3	1726	0.4	4.8
	800	6.6	1.6	10.4	10358	31.5	4918	0.4	34.6
"a3"	200	4.0	6.0	0.1	148	0.1	116	0.1	0.2
	400	8.6	6.4	0.6	283	1.0	186	0.1	1.0
	800	17.4	6.5	7.7	578	9.3	294	0.1	5.9
"a4"	200	0.8	1.5	0.1	501	1.0	279	0.2	1.0
	400	1.8	1.7	0.6	1045	6.8	431	0.2	3.8
	800	2.8	1.3	5.8	2070	54.3	656	0.1	17.4

Table 3: Computational results for Bin Packing

4.3 Generalized assignment problem

In the Generalized Assignment Problem (GAP), the objective is to find a maximum profit assignment of a set $J = \{1, \ldots, n\}$ of jobs to a set $I = \{1, \ldots, m\}$ of machines such that each job is assigned to precisely one machine subject to capacity restrictions of the machines. A compact formulation in terms of binary variable x_{ij} that indicate whether job j is assigned to machine i, is:

$$[\text{F}] \equiv \min\{\sum_{i,j} c_{ij}x_{ij} : \sum_{i} x_{ij} = 1 \ \forall j, \ \sum_{j} a_{ij}x_{ij} \leq b_i \ \forall i, \ x_{ij} \in \{0,1\} \ \forall i,j\}, \tag{83}$$

where $c_{ij} \in I\!N$ is the cost of assigning job j to machine i, $a_{ij} \in I\!N$ is the claim on the capacity of job j on machine i, and $b_i \in I\!N$ is the capacity of machine i. The $0-1$ knapsack subproblem consists in selecting a job assignment for a single machine i: $X^i = \{x_i \in \{0,1\}^n : \sum_j a_{ij}x_{ij} \leq b_i\Big\}$. It can be reformulated as a shortest path problem:

$$Z^i = \Big\{z_i \in \{0,1\}^{b_i \times n} : \sum_{j=0}^{n} z_{ij0} = 1, \quad \sum_{j=0}^{n}(z_{ijt} - z_{i,j,t-a_{ij}}) = 0 \ \forall t \in \{1, \ldots, b_i - 1\}\Big\}. \tag{84}$$

where binary variable z_{ijt} indicates whether job j use capacity interval $[t, t + a_{ij})$ on machine i.

We compared three approaches: solving $[F_{LP}]$ using Cplex; solving $[M_{LP}]$ by standard column generation; and solving $[R_{LP}]$ by column-and-row generation. The three approaches were tested on instances from the OR-Library with 100 and 200 jobs and 5, 10, and 20 machines. The instances in classes C, D, and E were used, since the instances in classes A and B are easy for modern MIP solvers. The results of Table 4 are averages over 3 instances, one for each class. Missing entries correspond to test for which cpu time

exceeded 1 hour. On this application, we note that although row-and-column generation can be much faster than standard column generation, it leads to dual bounds which are much worse, in fact almost as bad as those obtained solving $[F_{LP}]$. This is explained by the fact that the reformulation is done under Assumption 2, relaxing the 0-1 knapsack subproblem in a unbounded knapsack subproblem.

		Cplex 12.1 for $[F_{LP}]$		Col. gen. for $[M_{LP}]$			Col-and-row gen for $[R_{LP}]$			
m	n	$\%gap$	cpu	sp	$\%gap$	cpu	sp	$\%gap$	$\%z$	cpu
20	100	1.17	0.05	1533	0.09	0.8	640	0.49	2.26	1.4
10	100	0.55	0.03	1927	0.10	1.3	350	0.38	1.95	1.1
5	100	0.26	0.01	3545	0.05	6.2	168	0.23	1.63	1.1
20	200	0.28	0.10	4553	0.02	10.8	793	0.18	1.25	9.3
10	200	0.17	0.05	9453	0.04	68.9	407	0.15	1.06	8.7
5	200	0.07	0.02	31715	0.02	2401.9	190	0.07	0.88	8.7
40	400	0.15	0.51	11837	0.03	120.3	1693	0.12	0.78	88.5
20	400	0.09	0.23	20740	0.03	1077.9	920	0.08	0.63	79.2
10	400	0.04	0.11				457	0.04	0.56	71.4

Table 4: Computational results for Generalized Assignment

4.4 Multi-Item Multi-Echelon Lot-Sizing

The Multi-Item Lot-Sizing problem consists in planning production so as to satisfy demands d_t^k for item $k = 1, \ldots, K$ over a discrete time horizon with period $t = 1, \ldots, T$ either from stock or from production. The production of a product entails production stages (echelons) $e = 1, \ldots, E$, each of which takes place on a different machine that can only process one product in each period (under the so-called small bucket assumption). A compact formulation is:

$$[\mathrm{F}] \equiv \min\{\sum_{ket}(c_{et}^k\, x_{et}^k + f_{et}^k\, y_{et}^k) : \tag{85}$$

$$\sum_k y_{et}^k \leq 1 \quad \forall e, t \tag{86}$$

$$\sum_{\tau=1}^{t} x_{e\tau}^k \geq \sum_{\tau=1}^{t} x_{e+1,\tau}^k \quad \forall k, e < E, t \tag{87}$$

$$\sum_{\tau=1}^{t} x_{E\tau}^k \geq D_{1t}^k \quad \forall k, t \tag{88}$$

$$x_{et}^k \leq D_{tT}^k\, y_{et}^k \quad \forall k, e, t \tag{89}$$

$$x_{et}^k \geq 0 \quad \forall k, e, t \tag{90}$$

$$y_{et}^k \in \{0, 1\} \quad \forall k, e, t\} \tag{91}$$

where variables x_{et}^k are the production of product k at echelon e in period t (at unit cost c_{et}^k) and y_{et}^k take value 1 if the production of product k at echelon e is setup in

period t (at a fixed cost f^k_{et}); $D^k_{1t} = \sum_{\tau=1}^{t} d^k_\tau$. The stock values can be computed as $s^k_{et} = \sum_{\tau=1}^{t} x^k_{e\tau} - \sum_{\tau=1}^{t} x^k_{e+1,\tau}$; their costs have been eliminated (they are included in c^k_{et}).

There exists an optimal solution where at each echelon and period either there is an incoming stock or an incoming production but not both, i.e., such that $x^k_{et}\ s^k_{et} = 0\ \forall e, t$. As a consequence, one can consider only production lot corresponding to an interval of demands. This dominance rule can be exploited to solve single item subproblem can by dynamic programming in polynomial time [16] and an associated network flow reformulation. In the single echelon case, the single item subproblem can reformulated as a shortest path problem:

$$Z^k = \{(z0^k, z^k) \in \{0,1\}^{T+T(T-1)/2} :\ z0^k_t + \sum_{a<t} z^k_{a,t-1} = \sum_{b\geq t} z^k_{tb}\ \forall t = 1, \dots T,\ \sum_{t=1}^{T} z_{t,T} = 1\},$$

where $z^k_{ab} = 1$ if, for item k, the demands for the interval of periods $t = a, \dots, b$ are covered by a lot production in period $t = a$; while $z0^k_t = 1$ if for item k no production take place up to period $t-1$. The projection to the compact formulation is defined by relations: $x^k_t = \sum_{b=t}^{T} \sum_{\tau=t}^{b} d^k_\tau\, z^k_{tb}$ and $y^k_t = \sum_{b=t}^{T} z^k_{tb}$.

For the Multi-Echelon case, a backward dynamic program (DP) can be defined where the states are associated with quadruplets (e, t, a, b) denoting the fact of having at echelon e in period t accumulated a production that is covering exactly the demand D^k_{ab} for final product with $t \leq a \leq b \leq T$ and $e = 1, \dots, E$. The backward recursion is

$$V(e,t,a,b) = \min\{V(e,t+1,a,b), \min_{l=a,\dots,b}\{V(e+1,t,a,l) + c^k_{et}\, D^k_{al} + f^k_{et} + V(e,t+1,l+1,b)\}\}$$

for all $e = E, \dots, 1$, $t = T, \dots, 1$, $a = T, \dots, 1$, and $b = T, \dots, a$. By convention $V(e,t,a,b) = 0$ if $a > b$. The initialization is $V(E+1,t,a,b) = 0$. The optimum is given by $V^* = V(1,1,1,T)$ [16].

From the the dynamic program, one can reformulate the single item subproblem as selecting a decision tree in an hyper graph whose nodes are the states of the above DP. The DP transition can be associated to flow on hyper-arcs: $z^k_{e,t,a,l,b} = 1$ if at echelon $e \in \{1, \dots, E\}$ in period $t \in \{1, \dots, T\}$ the production of item k is made to cover demands from period $a \in \{t, \dots, T\}$ to period $l \in \{a-1, \dots, T\}$, while the rest of demand interval, i.e. demands from period $l+1$ to period $b \in \{l, \dots, T\}$, will be covered by production in future periods. If $l = a-1$, there is no production; this can only happen when $a > t$. While if $l = b$, the whole demand interval , D^k_{ab}, is produced in t. The associated cost is $c^k_{e,t,a,l,b}$ is $c^k_{et}\, D^k_{al} + f^k_{et}$ if $l \geq a$ and zero if $l = a-1$. For the initial echelon $e = 1$, variables $z^k_{1,t,a,l,b}$ are only defined for $b = T$. For the first period $t = 1$, they are only defined for

$a = t = 1$. This leads to reformulation:

$$[\mathrm{R}] \equiv \min\{ \sum_{e,t,a,l,b,k} c^k_{e,t,a,l,b}\, z^k_{e,t,a,l,b} \tag{92}$$

$$\sum_{e=1}^{E} \sum_{a,l,b,k:l\geq a, D^k_{al}>0} z^k_{e,t,a,l,b} \leq 1 \quad \forall e,t \tag{93}$$

$$\sum_{l} z^k_{1,1,1,l,T} = 1 \quad \forall k \tag{94}$$

$$\sum_{l} z^k_{e,t,a,l,b} - \sum_{\tau\leq a} z^k_{e,t-1,\tau,a-1,b} - \sum_{\tau\geq b} z^k_{e-1,t,a,b,\tau} = 0 \quad \forall k,e,t,a,b \tag{95}$$

$$z^k_{e,t,a,l,b} \in \{0,1\} \quad \forall k,e,t,a,l,b\}, \tag{96}$$

which results from subproblem reformulation Z^k defined by constraints (94-96) for a fixed k. Note that constraints (95) are only defined for $t > 1$ and $b = T$ when $e = 1$; while when $e > 1$, there are only defined for $a = t$ when $t = 1$.

The three approaches were tested on randomly generated instances with number of jobs $K = 10$, 20 and 40 and number of periods $T = 50$, 100, 200, and 400 periods. Setup costs are uniformly distributed in interval $[20, 100]$, while production costs are zero. Storage cost h^k_e for item k and echelon e is generated as $h^k_{e-1} + \gamma$, where γ is uniformly distributed in interval $[1, 5]$. For each period, there is a positive demand for 3 items on average. Demands are generated using a uniform distribution on interval $[10, 20]$.

The results for the single echelon case are given in Table 5; they are averages over 10 generated instances. There, the column-and-row generation approach performs better than the other two when the ratio between the number of periods and the number of items is big. As a single item is produced in each period, this ratio is an indication of the number of setups per item and hence the number of arcs in a path defining a production planning. Thus, larger ratio means more possible recombinations between these paths. However this advantage decreases with the increase of the dimension of instances. The results for the multi-echelon case given in Table 6 are averages over 5 instances. Solving formulation $[R_{LP}]$ with Cplex took more than one hour for all instances, so these are not reported in the table. For the column generation approach to $[M_{LP}]$, instances get easier with the increase in the number of items because there are fewer feasible production plans due to the simgle mode constraints. The column-and-row generation appears clearly as the only approach of the three that is tractable for the instances size considered therein. This illustrates the benefit of recombinations of decision trees (as illustrated in Figure 4) that take place in this application (this benefit increases with the number of echelons and the ratio $\frac{T}{K}$).

Conclusion

The "column-and-row generation" procedure is reviewed here in an effort to explain exactly when it should be considered, how it works, why it can be comparatively more efficient, and what are its practical performance on a scope of applications. We showed that

		Cplex 12.1 for $[R_{LP}]$	Col. gen. for $[M_{LP}]$		Col-and-row gen for $[R_{LP}]$		
K	T	*cpu*	*sp*	*cpu*	*sp*	*%var*	*cpu*
20	100	6.3	2170	2.5	764	2.9	2.3
40	200	161.9	6216	21.0	2192	1.1	32.0
10	100	2.2	2444	4.5	348	3.6	1.1
20	200	60.5	7024	43.5	1006	1.6	15.4
40	400	1847.9	18668	544.2	2672	0.6	232.5

Table 5: Computational results for single-echelon multi-item lot-sizing

		Col. gen. for $[M_{LP}]$		Col-and-row gen for $[R_{LP}]$		
K	T	*sp*	*cpu*	*sp*	*%var*	*cpu*
2 echelons						
10	50	1612	2.3	286	0.59	1.6
20	50	1380	1.5	492	0.46	2.8
10	100	9468	254.1	368	0.15	7.1
20	100	7268	65.1	684	0.15	19.9
3 echelons						
10	50	3044	20.8	386	0.18	5.2
20	50	2568	12.1	628	0.13	9.1
10	100	16810	3600.3	472	0.03	35.6
20	100	15868	1415.0	936	0.02	98.7
5 echelons						
10	50	9890	774.0	490	0.12	19.1
20	50	11180	281.2	760	0.08	30.2
10	100	>30000	>9h	684	0.02	152.0
20	100	48760	32667.4	1124	0.01	428.9

Table 6: Computational results for multi-echelon multi-item lot-sizing

column-and-row generation is a generalization of the standard column generation procedure. The latter applies to a Dantzig-Wolfe reformulation, while the former can be applied more broadly to any reformulation based on the Dantzig-Wolfe decomposition paradigm: the procedure is suitable for problems that admit an extended reformulation that stems from a reformulation of sub-problems. We formalized the methodology and, in particular, the termination criteria, in a generic procedure that also encompasses the case where one only has a better formulation for subproblem although not an exact extended formulation.

The "column-and-row generation" methodology can be understood as a hybrid between a direct handling of the extended reformulation and using a standard column generation for the associated Dantzig-Wolfe reformulation exploiting the same subproblems. The principal interest of this hybrid technique in comparison to standard column generation is to possibly achieve faster convergence thanks to recombinations of previously generated subproblem solutions into new points that are not in the convex hull of currently gener-

ated subproblem solutions. Examples where such recombination property (Property 1) holds include special cases such as the example of Section 1.3, where subproblem solutions that differ only by the value of the continuous variables are all implicitly defined in the restricted reformulation. We considered two generic situation where the recombination property holds: when the reformulation stems from a network flow model, or a dynamic programming subproblem solver. This analysis could be extended to generalized flow reformulations, or to cases where the subproblem reformulation obeys the rules of a branched polyhedral system [12].

The recombination property leads to a reduction in the number of iterations of the column generation procedure as demonstrated in our numerical results. Therefore, the disaggregation of columns inherent to the column-and-row generation approach can be understood as a stabilization technique for column generation. "Disaggregation" helps convergence as it is numerically demonstrated in many studies related to column generation. For instance, in the presence of block diagonal systems, good practice is to define separate columns for each block, or even to artificially differentiate commodities to create block diagonality as illustrated for origin-destination flow problems in [11]; another example is the disaggregation of the time horizon used by [3] for a scheduling application. When the subproblem reformulation is not actually an exact extended formulation for it (i.e., when Assumption 1 is not satisfied), there are typically even more recombinations in the relaxed subproblem solution space, but the relaxation can imply a weakening of the dual bound, as illustrated on the generalized assignment application. Our numerical comparative study of column-and-row generation illustrates the experimental trade-off between the comparative acceleration of convergence, the potential lost of quality in dual bounds, and the higher computing time required to solve the restricted master (due to its larger size and potential symmetries).

The recombination property is closely related to the concept of "exchange vectors" in standard column generation approach [23]; the latter are columns defining rays in the lattice of subproblem solutions (for instance the elementary cycles of Section 3.3 define rays). Using a convex combination of regular columns and exchange vectors allows one to define new solutions that are outside the convex hull of already generated subproblem solutions. Exchange vectors define so-called dual cuts (valid inequalities for dual prices) in the dual master program [18]. The idea of relaxing the definition of the generator set is related to "base-generators", as developed in [23], that are extracted from regular columns by keeping only the fixed values of the "important" variables in the subproblem solution (in the examples of Section 1.3, the disaggregation amounts to defining "base-generators" associated to the integer part of the subproblem solution). When relaxing Assumption 1 into Assumption 2, the method is then related to the concept of the "state space relaxation" for column generation as presented in [23]; or it can be interpreted as the development of a column-and-row generation approach based on an approximated extended formulation for the subproblem as underlined by the proposal of Van Vyve and Wolsey [25].

Acknowledgments

We thank J. Desrosiers and L. A. Wolsey for constructive exchanges on this paper and their suggestions for improvements.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[3] Louis-Philippe Bigras, Michel Gamache, and Gilles Savard. Time-Indexed Formulations and the Total Weighted Tardiness Problem. *INFORMS Journal on Computing*, 20(1):133–142, 2008.

[4] V. Chvatal. *Linear programming*. Freeman, 1983.

[5] K.L. Croxton, B. Gendron, and T.L. Magnanti. Variable disaggregation in network flow problems with piecewise linear costs. *Operations Research*, 55:146–157, 2007.

[6] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. Vehicle routing with time windows and split deliveries. Technical Paper 2006-851, Laboratoire d'Informatique d'Avignon, 2006.

[7] Dominique Feillet, Michel Gendreau, Andrés L. Medaglia, and Jose L. Walteros. A note on branch-and-cut-and-price. *Operations Research Letters*, 38(5):346 – 353, 2010.

[8] Antonio Frangioni and Bernard Gendron. 0-1 reformulations of the multicommodity capacitated network design problem. *Discrete Applied Mathematics*, 157(6):1229 – 1241, 2009.

[9] Antonio Frangioni and Bernard Gendron. A stabilized structured Dantzig-Wolfe decomposition method. Research report CIRRELT-2010-02, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, 2010.

[10] İbrahim Muter, İlker Birbil, and Kerem Bülbül. Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows. http://www.optimization-online.org/DB_FILE/2010/11/2815.pdf, 2010.

[11] Kim L. Jones, Irvin J. Lustig, Judith M. Farvolden, and Warren B. Powell. Multicommodity network flows: The impact of formulation on decomposition. *Mathematical Programming*, 62:95–117, 1993.

[12] Volker Kaibel and Andreas Loos. Branched polyhedral systems. In Friedrich Eisenbrand and F. Shepherd, editors, *Integer Programming and Combinatorial Optimization*, volume 6080 of *Lecture Notes in Computer Science*, pages 177–190. Springer Berlin / Heidelberg, 2010.

[13] John W. Mamer and Richard D. McBride. A decomposition-based pricing procedure for large-scale linear programs: An application to the linear multicommodity flow problem. *Management Science*, 46(5):693–709, 2000.

[14] R. Kipp Martin, Ronald L. Rardin, and Brian A. Campbell. Polyhedral Characterization of Discrete Dynamic Programming. *Operations Research*, 38(1):127–138, 1990.

[15] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.

[16] Yves Pochet and Laurence Wolsey. *Production planning by mixed integer programming*. Springer, 2006.

[17] Chris N. Potts and Luk N. Van Wassenhove. A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. *Operations Research*, 33(2):363–377, 1985.

[18] José Manuel Valério de Carvalho. Using Extra Dual Cuts to Accelerate Column Generation. *INFORMS Journal on Computing*, 17(2):175–182, 2005.

[19] J.M. Valério de Carvalho. Exact solution of bin packing problems using column generation and branch and bound. *Annals of Operations Research*, 86:629–659, 1999.

[20] J. M. van den Akker, J. A. Hoogeveen, and S. L. van de Velde. Parallel machine scheduling by column generation. *Operations Research*, 47(6):862–872, 1999.

[21] J.M. van den Akker, C.A.J. Hurkens, and M.W.P. Savelsbergh. Time-Indexed Formulations for Machine Scheduling Problems: Column Generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000.

[22] F. Vanderbeck. Bapcod – a branch-and-price generic code. University of Bordeaux, INRIA Research team ReAlOpt.

[23] François Vanderbeck and Martin W. P. Savelsbergh. A generic view of Dantzig-Wolfe decomposition in mixed integer programming. *Operations Research Letters*, 34(3):296–306, 2006.

[24] François Vanderbeck and Laurence A. Wolsey. Reformulation and decomposition of integer programs. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer Berlin Heidelberg, 2010.

[25] Mathieu Van Vyve and Laurence A. Wolsey. Approximate extended formulations. *Mathematical Programming*, 105:501–522, 2006.