# Column Generation for Extended Formulations

Ruslan Sadykov, François Vanderbeck

# Column Generation for Extended Formulations

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

(1) INRIA, project-team RealOpt

(2) Univ. of Bordeaux, Institute of Mathematics (CNRS UMR 5251)

INRIA research report hal-00661758, December 2012

**Abstract**

Working in an extended variable space allows one to develop tighter 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 work with inner approximations defined and improved by generating dynamically variables and constraints. When the extended formulation stems from subproblems' reformulations, one can implement column generation for the extended formulation using a Dantzig-Wolfe decomposition paradigm. 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 solutions. This so-called "column-and-row generation" procedure is revisited here in a unifying presentation that generalizes the column generation algorithm and extends to the case of working with an approximate extended formulation. The interest of the approach is evaluated numerically on machine scheduling, bin packing, generalized assignment, and multi-echelon lot-sizing problems. We compare 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 [28]: 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 for a sub-system, 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 or aggregating nodes, or, more generally, applying the extended reformulation paradigm to sub-systems only. Such approach preserves the possibility of directly applying a standard MIP approach to the "truncate" extended reformulation and allows one to deal with larger size instances. In [25], the level of approximation is controlled by a parameter whose maximum value corresponds 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. The method 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, pricing is done by solving an optimization subproblem over the whole set of variables and the inserted constraints are those that are binding for that subproblem solution. The method applies to problems that present some structure that makes them amenable to Dantzig-Wolfe decomposition. Then, the pricing subproblem is that of a standard column generation approach applied to the Dantzig-Wolfe reformulation. However, subproblem solutions must be 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. Compared 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 permits their "recombination" which acts as a stabilization technique for column generation. Moreover, the extended formulation 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 contexts: for bin packing in [22]; for multi-commodity flow in [14] and in [13] where the method is extended to the context of a Lagrangian Decomposition – the resulting methodology is named "Lagrangian pricing"; for the resource-constrained minimum-weight arborescence problem in [8]; for split delivery vehicle routing in [6, 7]; or for network design problems in [7, 9]. The convincing computational results of these papers indicate the interest of the method. Although the motivations of these studies are mostly application specific, methodological statements made therein are to some extend generic. Moreover, there are recent efforts to explore this approach further. In a study developed concomitantly to ours, Frangioni and Gendron [10] present a "structured Dantzig-Wolfe decomposition" for which they adapt column generation stabilization techniques (from linear penalties to the bundle method). Compared to [10], our generic presentation, relying on the definition of an extended formulation, assumes slightly less restrictive assumptions and extends to approximate extended formulation. In [7], Gendreau et al. present a branch-and-price-and-cut algorithm where columns and cuts are generated simultaneously; their presentation considers approximate extended formulation but with a weaker termination condition. In [16], Muter et al. consider what they call a "simultaneous column-and-row generation" approach, but it takes the meaning of a nested decomposition, different from the method reviewed here: the subproblem has itself a decomposable structure.

The goal of revisiting the column-and-row generation approach is to emphasize its wide applicability and to highlight its pros and cons with an analysis supported by numerical experiments. Our paper establishes the validity of the column-and-row generation algorithm in a form that encompasses all special cases of the literature, completing the presentations of [7, 8, 9, 10, 13, 14, 16, 22]. We identify what explains the faster convergence: the recombination of previously generated subproblem solutions. We point out that lifting the master program in the variable space of the extended formulation can be done while carrying pricing in the compact variable space of the original formulation, using any oracle. We show that the method extends to the case where the extended formulation is based on a subproblem reformulation that only defines an approximation of the subproblem integer hull. This extension is essential in practice as it allows one to use approximations for strongly NP-Hard subproblems for which an exact extended formulation is necessarily exponential (unless P=NP). Moreover, we advocate the use of a generalized termination condition in place of the traditional reduced cost criteria, that can lead to stronger dual bounds: solving the integer subproblems yields Lagrangian dual bounds that might be tighter than the extended formulation LP bound. The benefit of the approach depends on the tradeoff between the induced stabilization effect on one hand, and the larger size working formulation and possible weaker dual bound on the other hand. Our analysis should help research fellows to evaluate whether this alternative procedure has potential to outperform classical column generation on a particular problem.

In the sequel, we introduce (in Section 1) the *column-and-row generation* method on specific applications where it has been used, or could have been used. Next, in Section 2, we formalize the procedure and show the validity of the algorithm (including in the case of multiple identical subproblems). We then show how the method extends to the case where one works with an approximate extended formulation (as in the proposal of [25]). We show that in such case, the dual bound that one obtains is between the linear relaxation bound for the extended reformulation and the Lagrangian dual bound based on that subproblem. In Section 3, we discuss the pros and cons of the method, we formalize the lifting procedure, 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. This discussion is corroborated by numerical results presented in Section 4. We carry out experiments on machine scheduling, bin packing, generalized assignment, and multi-echelon lot-sizing problems. 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 that is shown to have a cumulative effect when used in combination with a standard stabilization technique.

# 1. Specific Examples

## 1.1 Machine Scheduling

For scheduling problems, time-indexed 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. [24]. 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:

$$[R] \quad \equiv \quad \min \quad \sum_{j\in J}\sum_{t} c_{jt}\, z_{jt} \tag{2}$$

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

$$\sum_{j\in J} z_{j1} \;=\; 1 \tag{4}$$

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

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

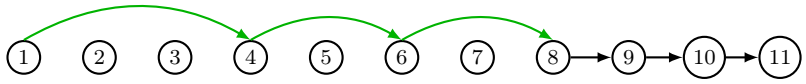
where (3) enforces the assignment of each job, (4) the initialization of the schedule, while (5) forbids the use of the machine for more than one job at the time: a job $j$ can start in $t$ only if one ends in $t$ and therefore releases the machine. The formulation can be extended to the case in which $m$ identical machines are available; then the right-hand-size of (4) is $m$ and variable $z_{0t}$ represents the number of idle machines at time $t$. One can also model in this way a machine with arbitrary capacity where jobs have unit capacity consumption. The objective can model any cost function that depends on job start times (or completion times). Extended reformulation [R] has size $O(|J|\,T)$ which is pseudo-polynomial in the input size, since $T$ is not an input but it is computed as $T = \sum_j p_j$. The subsystem defined by constraints (4-5) characterizes 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:

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

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 [23, 24], 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.

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 [R], for the numerical instance of Figure 1. 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

Figure 1: *Consider the machine scheduling subproblem. A solution is a pseudo-schedule that can be represented by a path as illustrated here 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 periods (represented by straight arcs).*




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 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 $g = 0, \ldots, 3$. Indeed, if $\hat{z} = \sum_{g=0}^{3} z^g \lambda_g$, then $\lambda_0 > 0$ and job 2 must be at least partially scheduled in period 2.

## 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 [22]. The bin packing problem consists in assigning $n$ items, $i \in I = \{1, \ldots, n\}$, of given size $s_i$, to bins of identical capacity $C$, using a minimum number of bins. A compact formulation is:

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

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

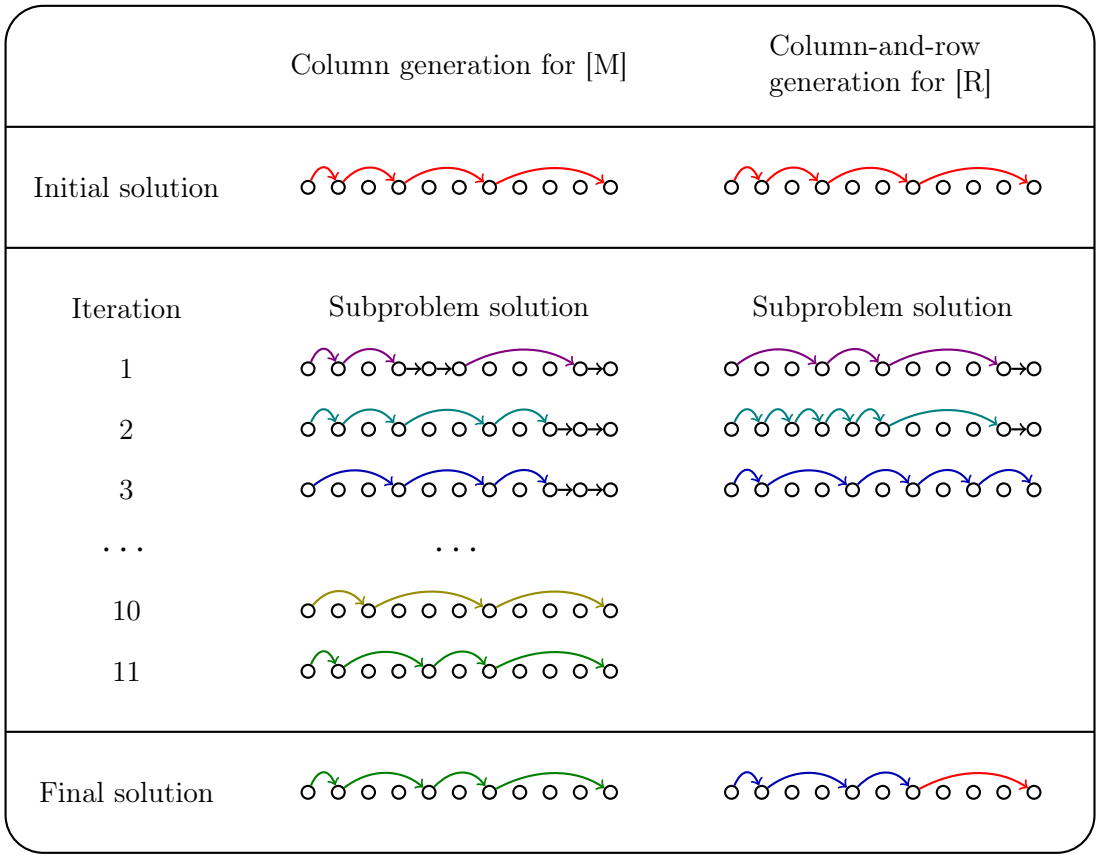
$$\sum_i s_i \, x_{ik} \leq C \, \delta_k \quad \forall k \tag{10}$$

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

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

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. Constraints (9) say that each item must be assigned to one bin, while constraints (10-12) define a knapsack subproblem for each bin $k$. The standard column generation approach consists in reformulating the problem in terms of the knapsack subproblem solutions. The

Figure 2: *Solving the example by column versus column-and-row generation (assuming the same data as in of Figure 1). Each bended arc represents a job, and straight arcs represent idle periods.*




master program takes the form:

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

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

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

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:

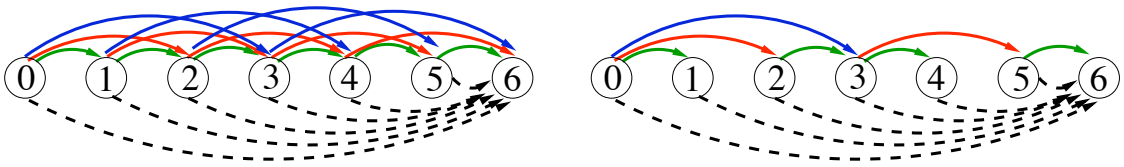
$$[SP] \equiv \min\{\delta - \sum_i \pi_i x_i : (x, \delta) \in X\} \tag{16}$$

where $\pi$ 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 $f^i_{uv} = 1$ if item $i$ is in the bin in position $u$ to $v = u + s_i$. The subproblem reformulation takes the form:

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

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

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

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

(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:

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

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

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

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

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

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

Valerio de Carvalho [22] proposed 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 $\pi$ 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 to (17-20) which we denote by $f(x^*)$, 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 [22], 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 [9] 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 [9], split flows are allowed and hence flow variables are continuous. A formulation is:

$$
\begin{aligned}
[F] \quad \equiv \quad \min \quad & \sum_{i,j,k} c^k_{ij}\, x^k_{ij} + \sum_{ij} f_{ij}\, y_{ij} && && (27)\\
& \sum_j x^k_{ji} - \sum_j x^k_{ij} \;=\; d^k_i \quad \forall i, k && && (28)\\
& \sum_k x^k_{ij} \;\le\; u_{ij}\, y_{ij} \quad \forall i, j && && (29)\\
& 0 \le x^k_{ij} \;\le\; d^k\, y_{ij} \quad \forall i, j, k && && (30)\\
& x^k_{ij} \;\ge\; 0 \quad \forall i, j, k && && (31)\\
& y_{ij} \;\in\; \mathbb{N} \quad \forall i, j && && (32)
\end{aligned}
$$

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 $(i,j) \in A$: $X^{ij} = \{(x,y) \in \mathbb{R}_+^K \times \mathbb{N} : \sum_k x^k \leq u_{ij}\, y,\ x^k \leq d^k\, y\ \forall k\}$. 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 $s_{ij}^{\max} = \left\lceil \frac{\sum_k d^k}{u_{ij}} \right\rceil$. 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:

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

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

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

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

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

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

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 $\sigma$: $y_s^g = 1$ 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 it defines an extreme LP solution for $s = \sigma$. Then, the Dantzig-Wolfe master takes the form

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

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

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

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

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 $(i, j)$.

Frangioni and Gendron [9] proceeded 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 [10]. This comparative advantage of the approach based of reformulation [R] has an intuitive explanation: in [R] a fixed $y_{ij} = s$, can be "recombined" with any alternative associated extreme subproblem solution in the $x$ variables. While, when applying column generation in [M], one might need to generate several columns associated with the same subproblem solution in the $y$ variables but different extreme continuous solution in the $x$ variables.

# 2. The Generic Procedure

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

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

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 notation.

**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:*

$$z \in \mathbb{R}_{+}^{e} \longrightarrow x = (T\, z) \in \mathbb{R}_{+}^{n}\ ;$$

*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)* $Z = Q \cap \mathbb{N}^e$ *defines an extended IP-formulation for* $X$*, i.e.,*

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

Condition $(i)$ is the core of Assumption 1, while condition $(ii)$ is merely a technical restriction that simplifies 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]:

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

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 [27]), 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] denote the dual of $[M_{LP}]$. Let

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

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

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

denote 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 $A\,x \geq a$ in formulation [F], i.e.,*

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

*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 $conv(X) \subset P$).

The above extends easily to the case where subsystem (46) is block diagonal. Then, there are $K$ independent subproblems, one for each block $k = 1, \ldots, K$, with their own generator set $G^k$ and associated variables in the reformulation $z^k$, and $\lambda^k_g$ respectively. When all subsystem are identical, extended reformulation and master are better defined in terms of aggregate variables to avoid symmetries: $w = \sum_k z^k$ and $\lambda_g = \sum_k \lambda^k_g$. The extended formulation becomes

$$\min c\,T\,w \tag{59}$$

$$\text{[AR]} \qquad A\,T\,w \geq a \tag{60}$$

$$H\,w \geq h \tag{61}$$

$$w \in \mathbb{N}^e \tag{62}$$

where constraints (61) are obtained by summing over $k$ constraints $H^k\,z^k \geq h^k\ \forall k$, as in the Bin Packing example, when aggregating (17-19) into (23-25). The aggregate master takes the form:

$$[AM] \equiv \min\{\sum_{g\in G} c\,x^g\,\lambda_g : \sum_{g\in G} A\,x^g\,\lambda_g \geq a;\ \sum_{g\in G}\lambda_g = K;\ \lambda \in \mathbb{N}^{|G|}\}\ .$$

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

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. The alternative hybrid pricing-and-separation strategy considered here is to generate 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 $x^g = Tz^s$ for some $g \in G$) 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:*

$$
\begin{aligned}
& \min c\,\overline{T}\,\overline{z} && && (63)\\
[\overline{R}] \qquad & A\,\overline{T}\,\overline{z} && \geq\ a && (64)\\
& \overline{H}\,\overline{z} && \geq\ \overline{h} && (65)\\
& \overline{z} && \in\ \mathbb{N}^{|\overline{J}|} && (66)
\end{aligned}
$$

*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 $\overline{S} \subset S$ properly chosen to guarantee the feasibility of $[\overline{R}_{LP}]$ (otherwise, artificial variables can be used to patch non-feasibility until set $\overline{S}$ is expanded). We define the associated set

$$\overline{G} = G(\overline{S}) = \{g \in G :\ x^g = T\,z^s \text{ for some } s \in \overline{S}\} \tag{67}$$

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 values. Although, in the end, $v^R_{LP} = v^M_{LP} = v^*$, 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 and associated $\overline{G}$ as defined in* (67)*. Under Assumption 1, their linear relaxation values are such that:*

$$v^* \leq v^{\overline{R}}_{LP} \leq v^{\overline{M}}_{LP} \tag{68}$$

**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 satisfied by the the current restricted reformulation $[\overline{R}_{LP}]$, which we denote by $\tilde{z}$. More explicitly, 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 $[R_{LP}]$; otherwise, such missing constraint is not satisfied by solutions of $\overline{S}$ in contradiction with the fact that solutions of $\overline{S}$ satisfy all constraints of $Q$. 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 guaranteed 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].

## 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 $[M_{LP}]$ by solving a pricing subproblem over $X$ (or equivalently over $Z$), 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 $v^{\overline{R}}_{LP}$ denote the optimal LP value of $[\overline{R}_{LP}]$, while $(\pi, \sigma)$ denote an optimal dual solution associated to constraints (64) and (65) 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 $[M_{LP}]$, while $(\pi, \zeta)$ defines a feasible solution of [D], the dual of $[M_{LP}]$.*

*(ii) Let $\beta$ denote the best of the above Lagrangian bound encountered in the procedure of Table 1. If $v^{\overline{R}}_{LP} \leq \beta$ (i.e. when the stopping condition in Step 3 is satisfied), then $v^* = \beta$ and $(\pi, \zeta)$ 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 $(\pi, \sigma)$.*

*(iv) Inversely, when $[(c - \pi A)\, T - \sigma\, H] z^* \geq 0$, i.e., if the generated column has non negative aggregate reduced cost for the dual solution of $[\overline{R}_{LP}]$, then $v^{\overline{R}}_{LP} \leq \beta$ (the stopping condition of Step 3 must be satisfied) and $(\pi, \nu)$ 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 (64), $L(\pi)$ defines a valid Lagrangian dual bound on $[F]$. As $\zeta = \min\{(c - \pi\, A)\, T\, z :\ z \in Z\} = \min\{(c - \pi\, A)\, x :\ x \in X\}$, $(\pi, \zeta)$ 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 $(\pi, \sigma)$ 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 $Hz^* \geq h$, $[(c - \pi A)\, T]z^* \geq \sigma\, H\, z^*$ implies $(c - \pi\, A)\, Tz^* \geq \sigma\, h$, i.e., $\zeta \geq \sigma\, h$. In turn, $\zeta \geq \sigma\, h$ implies that $(\pi, \nu)$ with $\nu = \sigma\, h$ is feasible for [D] (all constraints of [D] are satisfied by $(\pi, \nu)$). Note that $\zeta \geq \sigma\, h$ also implies $v_{LP}^{\overline{R}} \leq \beta$, as $v_{LP}^{\overline{R}} = \pi\, a + \sigma\, h \leq \pi\, a + \zeta = L(\pi) \leq \beta$. ∎

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 $v_{LP}^{\overline{R}}$ and the dual solution $\pi$ associated to constraints (64).

**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 $v_{LP}^{\overline{R}} \leq \beta$, 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 1** *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, $\sigma$, associated to subproblem constraints (65).*

**Remark 2** *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 needs 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 $x^* = T\, z^*$.*

**Remark 3** *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, $\lambda_g$, 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:*

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

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

Assumption 2, relaxing Assumption 1-$(i)$, can be more realistic in many applications where the subproblem is strongly 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, $v_{LP} \leq v^R_{LP} \leq v^* = v^M_{LP} = v^D_{LP}$ .*

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

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 value $v^R_{LP}$ of the solution of $[R_{LP}]$, but one has a valid dual bound $\beta$ that is at least as good, since*

$$v^R_{LP} \leq \beta \leq v^* = v^M_{LP}\;.$$

**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 $v^{\overline{R}}_{LP}$. Once the stopping condition, $v^{\overline{R}}_{LP} \leq \beta$, is satisfied however, we have $v^R_{LP} \leq v^{\overline{R}}_{LP} \leq \beta \leq v^*$, proving that the optimal LP value, $v^R_{LP}$, is then guaranteed to lead to a bound weaker than $\beta$. ∎

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

# 3. Interest of the approach

Here, we review the motivations to consider applying column-and-row generation to [R] instead of a 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's 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 the existence of a dynamic programming subproblem solver.

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

Both the hybrid column-and-row generation method for [R] and a standard column generation approach for [M] can be understood as ways to get around the issue of size arising in a direct solution of the extended formulation [R]. The primary interest for implementing a dynamic generation of [R] rather than [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 [22]), 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 does 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 a 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) potentially having to use a subproblem oracle specific to the subproblem extended formulation, if lifting a subproblem solution in the extended space as explained in Remark 2 is not an option; this is an issue when pricing in the $z$-variable space requires higher complexity / computing times than in the $x$-variables.

## 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 from what we call the *"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 guaranteed by the *"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-$(i)$, along with Definition 1, implies Property 2, that is a form of re-wording of Proposition 1. However, the *"convexification property"* does not hold under Assumption 2.

Branching can be performed simply by enforcing integrality restriction on the $z$ variables if the *"integrality property"* holds:

**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 (59-62) [28].

Let us formalize the lifting of Remark 2. 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

$$p^{-1}(x) := \{z \in \mathbb{N}^e : T\ z = x;\ H\ z \geq h\}\ . \tag{69}$$

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* $x = T\ z$.

Then, in Step 2 of the procedure of Table 1, one computes $x^* := argmin\{(c - \pi A)\ x : x \in X\}$, and in Step 4, one defines $z^* = p^{-1}(x^*)$.

**Observation 4** *A generic lifting procedure is to solve the integer feasibility program defined in (69).*

Note that solving (69) is typically much easier than solving the pricing subproblem, as constraint $T\,z = x$ already fixes many $z$ variables. However, in an application specific context, one can typically derive 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 $Z$-subproblem that cannot be modeled in the $X$ space. Then, pricing must be done in the $z$-space.

Finally, let us discuss further the symmetry drawback. It is characterized by the fact that the set $p^{-1}(x^g)$ defined in (69) 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 of the lifted solution, 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 integrality of the $z$ variables would also suffer from such symmetry.) The lifting procedure can break such symmetry by adopting a rule to select a specific representative of the symmetry class $p^{-1}(x^g)$, or by adding constraints in (69).

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.

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 solutions that are not convex combinations of $z^1, \ldots, z^k$: those are 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 $w^A$, $w^B, \ldots$, 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 developed in Section 1.1. 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 reformulation (59-62), 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 [15]. 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 transitions (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.

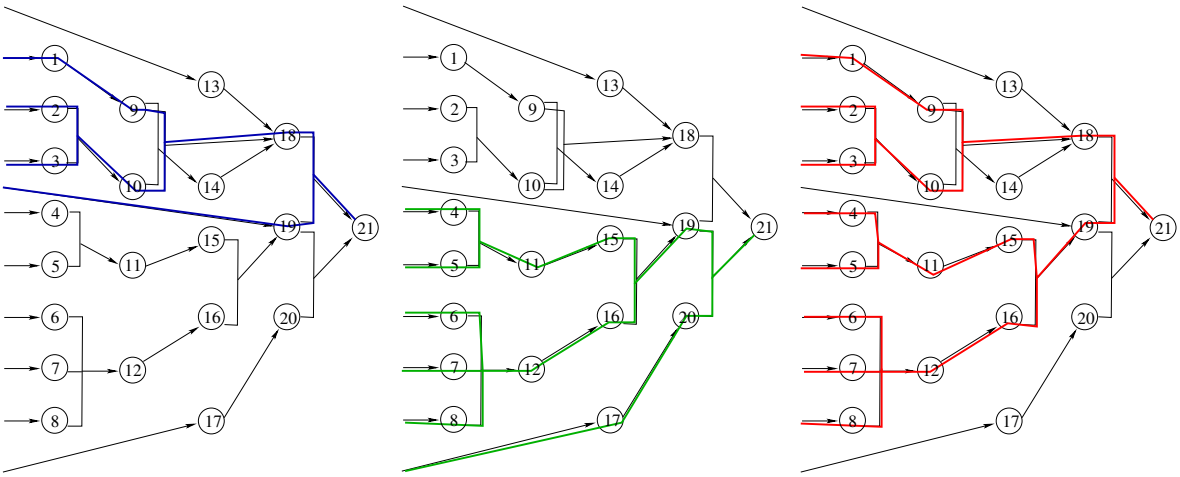
However, more complex dynamic programs may involve the composition of more than one intermediate state (representing partial solutions) into a single state (next stage partial solution). These can be modeled by hyper-arcs with a single head but multiple tails. Then, the extended paradigm developed by [15] 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: *Illustration of the recombination of subproblem solutions that are associated with hyper-paths in the hyper-graph underlying the paradigm of [15]: 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 the final node 21; when a unit flow exits an hyper-arc, a corresponding unit flow must enter in each of the tail nodes; solutions $z^1$ and $z^2$ are depicted on in the hyper-graphs on the left and in the middle; they share a common intermediate node 19; their recombination, $\hat{z}$, is represented in the hyper-graph on the right.*



Following [15], 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:

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

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

$$\max\{u_f :\ u_l - \sum_{j\in J} u_j \leq c(J, l) \quad \forall (J, l) \in \mathcal{A}\} \tag{70}$$

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

$$\min\{ \sum_{(J,l)\in\mathcal{A}} c(J,l)\, z_{(J,l)} \tag{71}$$

$$\sum_{(J,f)\in\mathcal{A}} z_{(J,f)} = 1 \tag{72}$$

$$\sum_{(J,l)\in\mathcal{A}} z_{(J,l)} = \sum_{(J',l')\in\mathcal{A}:l\in J'} z_{(J',l')} \quad \forall l \neq f \tag{73}$$

$$z_{(J,l)} \geq 0 \quad \forall (J,l) \in \mathcal{A}\} \tag{74}$$

that defines the reformulation $Q$ for the subproblem.

In this generalized context, [15] 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 (70); given the specific assumption on the hyper-graph (acyclic consistency and disjointness), this dual solution $z$ can be obtained through a greedy procedure. 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 [15] 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 $z$. 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 $\tilde{z}_{(J^1,l)} = 0$. Such a recombination is illustrated in Figure 4.

# 4. Numerical experimentation

Consider the three formulations introduced in Section 2: the compact formulation [F], the extended formulation [R], and the standard Dantzig-Wolfe reformulation [M], with their respective LP relaxation $[F_{LP}]$, $[R_{LP}]$, and $[M_{LP}]$. Here, we report on comparative numerical experiments with column(-and-row) generation for $[R_{LP}]$ and $[M_{LP}]$, and a direct LP-solver approach applied to $[R_{LP}]$ (or $[F_{LP}]$, when the direct approach to $[R_{LP}]$ is impractical). The column(-and-row) generation algorithm has been implemented generically within the software platform BaPCod [26] (a problem-specific implementation is likely to produce better results). The master program is initialized with either a single artificial variable, or one for each linking constraint, or a heuristic solution. For column-and-row generation, all extended formulation variables $z$ that form the pricing subproblem solution are added to the restricted master, whether or not they have negative reduced

cost. CPLEX 12.1 was used to attempt to solve directly $[R_{LP}]$ or $[F_{LP}]$. CPLEX 12.3 is used to solve the restricted master linear programs, while the MIP subproblems are solved using a specific oracle.

For applications where the subproblem is a knapsack problem, we used the solver of Pisinger [18]. 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 specific lifting procedure selects one representative amongs all equivalent solutions in the arc-space, automatically eliminating some symmetries. It can be extended to integer knapsack problems. 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 $u$.

Results of Tables 2 to 7, are averages over randomly generated instances. The column headed by *"cpu"* reports the average computational time (in seconds on a Dell PowerEdge 1950 workstation with 32 Go of Ram or an Intel Xeon X5460 3.16 GHz processor); *"it"* denotes the average number of iterations in the column(-and-row) generation procedure (it represents the number of calls to the restricted master solver); *"v($\frac{F_1}{F_2}$)"* (resp. *"c($\frac{F_1}{F_2}$)"*) denote the average number of variables (resp. constraints) generated for formulation $F_1$ expressed as a percentage of the number of variables (resp. constraints) generated for $F_2$. Additionally, "*%gap*" (reported in some applications) denotes the average difference between dual bound and best primal bound known (the optimum solution in most cases), as a percentage of the latter. Bounds are rounded to the next integer for integer objectives.

To validate the stabilization effect of the column-and-row generation approach, we show how it compares to applying stabilization in a standard column generation approach. We point out that stabilization by recombination is of a different nature than standard stabilization techniques. To illustrate this, we show that applying both standard stabilization and column recombination together leads to a cumulative effect. For these experiments, the "standard stabilization technique" that we use is a dual price smoothing technique originally proposed by Wentges [29]. It is both simple and among the most effective stabilization techniques. It consists in solving the pricing subproblem for a linear combination, $\bar{\pi}$, of the current restricted master dual solution, $\pi$, and the dual solution which gave the best Lagrangian bound, $\hat{\pi}$: i.e., $\bar{\pi} = \alpha\hat{\pi} + (1-\alpha)\pi$, where $\alpha \in [0, 1)$. Thus, $\alpha = 0$ means that no stabilization by smoothing is used.

## 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 [20]: 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 a uniform distribution in $[P(1-TF-RDD/2)/m, P(1-TF+RDD/2)/m]$, where $P = \sum_j p_j$, $TF$ is a tardiness factor, and $RDD$ is a relative range of due dates, with $TF, RDD \in \{0.2, 0.4, 0.6, 0.8, 1\}$. For each instance size, 25 instances were generated, one for each pair of parameters $(TF, RDD)$.

| | | | Cplex for $[R_{LP}]$ | Col. gen. for $[M_{LP}]$ | | Col-&-row g. for $[R_{LP}]$ | | Hyb. col-&-row g. for $[R_{LP}]$ | | and relative size | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $m$ | $n$ | $p_{\max}$ | *cpu* | *it* | *cpu* | *it* | *cpu* | *it* | *cpu* | $v(\frac{\overline{R}}{R})$ | $v(\frac{\overline{M}}{R})$ | $c(\frac{\overline{M}}{R})$ |
| 1 | 25 | 50 | 1.6 | 331 | 0.5 | 50 | 0.3 | 106 | 0.3 | 5.3 | 45.6 | 4.1 |
| 1 | 50 | 50 | 18.8 | 1386 | 19.7 | 76 | 2.9 | 207 | 2.8 | 3.2 | 72.7 | 3.7 |
| 1 | 100 | 50 | 304.2 | 8167 | 1449.5 | 104 | 24.8 | 354 | 19.4 | 2.3 | 138.0 | 4.0 |
| 1 | 25 | 100 | 7.1 | 337 | 0.9 | 72 | 0.9 | 124 | 0.8 | 3.8 | 33.8 | 2.0 |
| 1 | 50 | 100 | 132.6 | 1274 | 24.2 | 107 | 8.9 | 246 | 8.6 | 2.7 | 43.1 | 1.8 |
| 1 | 100 | 100 | 2332.0 | 8907 | 1764.4 | 144 | 90.3 | 455 | 61.3 | 1.9 | 103.2 | 1.9 |
| 2 | 25 | 100 | 4.1 | 207 | 0.3 | 63 | 0.2 | 97 | 0.2 | 3.9 | 40.0 | 3.9 |
| 2 | 50 | 100 | 109.2 | 645 | 5.7 | 94 | 1.7 | 173 | 1.9 | 2.8 | 39.9 | 3.5 |
| 2 | 100 | 100 | 3564.4 | 2678 | 115.5 | 117 | 14.3 | 319 | 14.9 | 2.1 | 59.3 | 3.7 |
| 4 | 50 | 100 | 18.7 | 433 | 1.5 | 90 | 0.6 | 167 | 0.7 | 3.0 | 45.0 | 6.6 |
| 4 | 100 | 100 | 485.7 | 1347 | 27.9 | 113 | 4.7 | 295 | 5.2 | 2.2 | 56.2 | 7.2 |
| 4 | 200 | 100 | >2h | 4315 | 409.7 | 148 | 36.1 | 561 | 39.7 | 1.5 | 69.7 | 7.6 |

Table 2: Computational results for Machine Scheduling without using dual price smoothing ($\alpha = 0$).

In Table 2, we compare methods on these instances without using dual price smoothing as a stabilization technique. The table reports on column generation for $[M_{LP}]$, column-and-row generation for $[R_{LP}]$, and solving $[R_{LP}]$ directly using Cplex. In both column generation and column-and-row generation, the master is initialized with a trivial heuristic solution. We note that with column-and-row generation, a lot of recombinations occur during the "heading-in" phase at the outset of the algorithm when the dual information is still very poor. These recombinations slow down the master solution time, while not being very useful as variables generated during this initial phase are not likely to appear in the optimum solution. Hence, we adopt a hybrid approach (also reported in Table 2), starting the algorithm using pure column generation and switching to column-and-row generation only beyond the "heading-in" phase (precisely, when $L(\pi) > 0$). This hybrid technique induces time saving in solving the master at the expense of an increase in the number of iterations (by a factor 2 to 3). The results reveals that solving $[R_{LP}]$ directly using Cplex is not competitive. Thanks to the stabilization effect of column recombinations, column-and-row generation yields a significant reduction in the number of iterations compared to standard column generation. However the restricted master $[\overline{R}_{LP}]$ is typically much larger (and harder to solve) than $[\overline{M}_{LP}]$ (around twice the number of variables and 20 times the number of constraints). Despite this fact, column-and-row generation is much faster. The measure "$v(\frac{\overline{R}}{R})$" shows that only around 2 to 5% of variables are actually generated to solve $[R_{LP}]$ by column-and-row generation ("$c(\frac{\overline{R}}{R})$" is omitted, as it was always close to 100).

| | | | Col. gen. for $[M_{LP}]$, $\alpha = 0.9$ | | Col-&-row g. for $[R_{LP}]$, $\alpha = 0.5$ | | Hyb. col-&-row for $[R_{LP}]$, $\alpha = 0.5$ | | and relative size | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $m$ | $n$ | $p_{\max}$ | *it* | *cpu* | *it* | *cpu* | *it* | *cpu* | $v(\frac{\overline{R}}{R})$ | $v(\frac{\overline{M}}{\overline{R}})$ | $c(\frac{\overline{M}}{\overline{R}})$ |
| 1 | 25 | 50 | 145 | 0.2 | 45 | 0.2 | 84 | 0.2 | 4.1 | 31.5 | 4.3 |
| 1 | 50 | 50 | 341 | 2.5 | 70 | 2.3 | 154 | 1.8 | 2.5 | 36.9 | 3.8 |
| 1 | 100 | 50 | 746 | 26.1 | 91 | 22.0 | 266 | 12.6 | 1.8 | 30.8 | 4.0 |
| 1 | 25 | 100 | 150 | 0.2 | 61 | 0.5 | 96 | 0.4 | 2.6 | 26.6 | 2.2 |
| 1 | 50 | 100 | 354 | 3.8 | 91 | 6.8 | 172 | 4.0 | 1.7 | 25.7 | 1.9 |
| 1 | 100 | 100 | 781 | 39.5 | 115 | 78.6 | 299 | 31.1 | 1.3 | 28.8 | 1.9 |
| 2 | 25 | 100 | 142 | 0.2 | 55 | 0.2 | 87 | 0.2 | 3.3 | 37.5 | 4.1 |
| 2 | 50 | 100 | 323 | 1.7 | 84 | 1.5 | 158 | 1.6 | 2.2 | 32.2 | 3.6 |
| 2 | 100 | 100 | 715 | 17.3 | 102 | 12.4 | 275 | 11.3 | 1.6 | 29.2 | 3.7 |
| 4 | 50 | 100 | 287 | 0.6 | 83 | 0.5 | 154 | 0.6 | 2.6 | 39.5 | 6.7 |
| 4 | 100 | 100 | 638 | 8.7 | 102 | 4.1 | 264 | 4.6 | 1.8 | 38.7 | 7.2 |
| 4 | 200 | 100 | 1553 | 87.7 | 136 | 33.5 | 481 | 33.4 | 1.2 | 36.3 | 7.6 |

Table 3: Computational results for Machine Scheduling using dual price smoothing as a stabilization technique.

In Table 3, we compare column generation for $[M_{LP}]$, pure and hybrid column-and-row generation for $[R_{LP}]$ using dual price smoothing for stabilization purposes. Experimental tuning lead to selecting parameter $\alpha$ to 0.9 and 0.5 respectively, i.e. on average column(-and-row) generation takes the least time to execute when $\alpha$ is fixed to these values. The restricted masters are initialized by a single artificial column, except in pure column-and-row generation, in which the master is initialized with a trivial heuristic solution. Here again, the hybrid approach consists in using standard column generation during the "heading-in" phase and switching to column-and-row generation when $L(\pi) > 0$. Results of Table 3 show that smoothing yields a speed-up in both standard column generation and column-and-row generation, but it is less impressive for the latter as the method was already stabilized by the recombination effect. The difference in cpu time between the two methods increases when either processing times are smaller (allowing for more recombinations) and when number of machines and jobs are larger.

In Table 4, we compare the column(-and-row) generation approaches on the arc-indexed formulation proposed by Pessoa et al. [17] where a binary variable $z_{ijt}$ is defined for every pair of jobs $(i, j)$ and every period $t$: $z_{ijt} = 1$ if job $i$ finishes and job $j$ starts in period $t$. Then, flow conservation constraints are defined for every couple $(j, t)$. Going to this larger extended space has some advantages: $(i)$ direct repetitions of jobs are forbidden by not defining variable $z_{iit}$; and $(ii)$ a simple dominance rule is incorporated by not defining variable $z_{ijt}$ if permuting jobs $i$ and $j$ decreases the total cost. The resulting strengthening of the LP relaxation bound is, in our experiments, on average 0.38% for single-machine instances, 0.28% for two-machine instances, and 0.13% for four-machine instances. This difference is significant given the fact that the time-indexed formulation

is already very strong. However, the arc-indexed formulation has huge size (solving $[R_{LP}]$ directly is excluded). Moreover, the complexity of the dynamic program for solving the pricing problem increases to $O(n^2T)$ and hence time for pricing dominates the overall cpu time. In this context, reducing the number of iterations is key, which gives the advantage of the column-and-row generation approach. For this reason, we did not use the hybrid column-and-row generation approach here. For the results of Table 4, experimental tuning lead to selecting smoothing parameter $\alpha$ to 0.9 and 0.7 respectively, while the restricted master is initialized by a single artificial column for standard column generation and using a trivial heuristic solution for column-and-row generation. Using a column-and-row generation approach yields a reduction of both iterations and cpu time by factor up to 5.6. The number of variables and constraints in the final restricted master $[\overline{R}_{LP}]$ is only a very small fraction of that of $[R_{LP}]$.

| | | | Col. gen. for $[M^{arc}_{LP}]$, $\alpha = 0.9$ | | Col-&-row g. for $[R^{arc}_{LP}]$, $\alpha = 0.7$ | | and relative size | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $m$ | $n$ | $p_{\max}$ | *it* | *cpu* | *it* | *cpu* | $v(\frac{\overline{R}}{R})$ | $c(\frac{\overline{R}}{R})$ | $v(\frac{\overline{M}}{\overline{R}})$ | $c(\frac{\overline{M}}{\overline{R}})$ |
| 1 | 25 | 50 | 127 | 0.5 | 85 | 0.5 | 0.30 | 1.3 | 23.1 | 11.0 |
| 1 | 50 | 50 | 332 | 11.7 | 117 | 5.6 | 0.09 | 0.8 | 23.7 | 9.5 |
| 1 | 100 | 50 | 734 | 198.1 | 161 | 57.3 | 0.03 | 0.4 | 21.8 | 9.5 |
| 1 | 25 | 100 | 128 | 1.1 | 100 | 1.2 | 0.13 | 0.6 | 24.5 | 12.9 |
| 1 | 50 | 100 | 328 | 25.7 | 160 | 16.0 | 0.05 | 0.4 | 21.5 | 9.7 |
| 1 | 100 | 100 | 758 | 466.3 | 197 | 144.7 | 0.02 | 0.3 | 19.9 | 8.3 |
| 2 | 25 | 100 | 145 | 0.7 | 89 | 0.5 | 0.16 | 1.1 | 42.6 | 12.0 |
| 2 | 50 | 100 | 332 | 13.0 | 140 | 5.9 | 0.07 | 0.7 | 32.6 | 11.3 |
| 2 | 100 | 100 | 730 | 203.7 | 186 | 54.4 | 0.02 | 0.4 | 27.1 | 10.5 |
| 4 | 50 | 100 | 286 | 5.6 | 145 | 3.1 | 0.08 | 1.1 | 42.6 | 13.2 |
| 4 | 100 | 100 | 618 | 83.5 | 197 | 27.5 | 0.03 | 0.6 | 33.6 | 13.3 |
| 4 | 200 | 100 | 1530 | 2233.2 | 275 | 389.9 | 0.01 | 0.2 | 30.4 | 12.0 |

Table 4: Computational results for Machine Scheduling using smoothing and an arc-indexed formulation.

## 4.2 Bin Packing

For the bin packing problem of Section 1.2, we compared solving $[F_{LP}]$ using Cplex (as solving $[R_{LP}]$ directly is impractical), standard column generation for $[M_{LP}]$, and pure column-and-row generation for $[R_{LP}]$. 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. Results in Table 5 are averages over 5 instances. Experimental tuning lead to selecting smoothing parameter $\alpha$ to 0.85 for both approaches, while the restricted master is initialized with a trivial heuristic solution. Here, *"gap"* denotes the absolute value of the difference between dual bound and the optimum solution (which we computed by branch-and-price); it is always zero for formulations $[M_{LP}]$ and $[R_{LP}]$. The percentage gap is also given under "*%gap*". 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. The column-and-row generation for $[R_{LP}]$ outperforms column generation for $[M_{LP}]$ only when the number of items per bin increases (i.e., for class "a4"), as otherwise the potential for column recombination is very limited. The number of iterations is reduced when using column-and-row, but this might not compensate for the extra time required to solve the master, as here pricing required only between 4% and 30% of the overal cpu time. The average relative size ($v(\frac{\overline{M}}{R})$, $c(\frac{\overline{M}}{R})$) in percent is respectively $(82, 36)$ for class "a2", $(39, 37)$ for class "a3", and $(52, 32)$ for class "a4"; while the percentage ($v(\frac{\overline{R}}{R})$, $c(\frac{\overline{R}}{R})$) reported in Table 5 are very small.

| | | Cplex for $[F_{LP}]$ | | | Col. gen. for $[M_{LP}]$, $\alpha = 0.85$ | | Col-&-row g. for $[R_{LP}]$, $\alpha = 0.85$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| class | $n$ | *gap* | *%gap* | *cpu* | *it* | *cpu* | *it* | *cpu* | $v(\frac{\overline{R}}{\overline{R}})$ | $c(\frac{\overline{R}}{\overline{R}})$ |
| "a2" | 200 | 5.6 | 5.2 | 0.1 | 439 | 0.3 | 281 | 0.5 | 0.21 | 14.9 |
| | 400 | 8.6 | 4.0 | 0.8 | 1001 | 1.2 | 599 | 2.0 | 0.15 | 25.7 |
| | 800 | 6.6 | 1.6 | 10.4 | 2725 | 6.8 | 1331 | 12.2 | 0.13 | 40.8 |
| "a3" | 200 | 4.0 | 6.0 | 0.1 | 158 | 0.2 | 124 | 0.2 | 0.16 | 15.1 |
| | 400 | 8.6 | 6.4 | 0.6 | 298 | 0.7 | 192 | 0.8 | 0.10 | 24.7 |
| | 800 | 17.4 | 6.5 | 7.7 | 596 | 5.5 | 297 | 4.8 | 0.08 | 38.7 |
| "a4" | 200 | 0.8 | 1.5 | 0.1 | 400 | 0.8 | 253 | 1.0 | 0.27 | 18.9 |
| | 400 | 1.8 | 1.7 | 0.6 | 841 | 5.4 | 414 | 4.5 | 0.17 | 29.0 |
| | 800 | 2.8 | 1.3 | 5.8 | 1662 | 38.6 | 602 | 16.3 | 0.13 | 41.6 |

Table 5: Computational results for Bin Packing using dual price smoothing as a stabilization technique.

## 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 on the machines. A compact formulation in terms of binary variables $x_{ij}$ that indicate whether job $j$ is assigned to machine $i$, is:

$$[\text{F}] \equiv \min\{\sum_{i,j} c_{ij}x_{ij} : \sum_{i} x_{ij} = 1 \; \forall j, \; \sum_{j} a_{ij}x_{ij} \leq b_i \; \forall i, \; x_{ij} \in \{0,1\} \; \forall i, j\}, \tag{75}$$

where $c_{ij} \in \mathbb{N}$ is the cost of assigning job $j$ to machine $i$, $a_{ij} \in \mathbb{N}$ is job $j$'s claim on the capacity of machine $i$, and $b_i \in \mathbb{N}$ is the capacity of machine $i$. The binary 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:

$$Z^i = \Big\{z_i \in \{0,1\}^{b_i \times n} : \sum_{j=0}^{n} z_{ij0} = 1, \quad \sum_{j=0}^{n} (z_{ijt} - z_{i,j,t-a_{ij}}) = 0 \; \forall t \in \{1, \ldots, b_i - 1\}\Big\} \tag{76}$$

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

In Table 6, we compare three approaches: solving $[F_{LP}]$ using Cplex (as solving $[R_{LP}]$ directly is impractical); solving $[M_{LP}]$ by standard column generation; and solving $[R_{LP}]$ by pure column-and-row generation. The three approaches were tested on instances from the OR-Library with 100, 200 and 400 jobs, and 5, 10, 20 and 40 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. Results are averages over 3 instances, one for each class. For column(-and-row) generation, experimental tuning lead to selecting smoothing parameter $\alpha$ to 0.85 and 0.5 respectively, while the restricted master is initialized respectively with a single artificial column and a trivial heuristic solution. Column-and-row generation is much faster than standard column generation, but it produces dual bounds that are much worse (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 subproblem to an unbounded knapsack problem (76).

| | | Cplex for $[F_{LP}]$ | | Col. gen. for $[M_{LP}]$, $\alpha = 0.85$ | | | Col-&-row g. for $[R_{LP}]$, $\alpha = 0.5$ | | | and relative size | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $m$ | $n$ | *%gap* | *cpu* | *it* | *%gap* | *cpu* | *it* | *%gap* | *cpu* | $v(\frac{\overline{R}}{R})$ | $v(\frac{\overline{M}}{\overline{R}})$ | $c(\frac{\overline{M}}{\overline{R}})$ |
| 20 | 100 | 1.17 | 0.05 | 201 | 0.09 | 1.4 | 31 | 0.40 | 1.3 | 2.1 | 116.2 | 8.5 |
| 10 | 100 | 0.55 | 0.03 | 229 | 0.10 | 1.2 | 33 | 0.35 | 1.1 | 1.9 | 83.2 | 9.0 |
| 5 | 100 | 0.26 | 0.01 | 295 | 0.05 | 2.2 | 35 | 0.20 | 1.1 | 1.6 | 61.8 | 9.0 |
| 20 | 200 | 0.28 | 0.10 | 358 | 0.02 | 11.9 | 37 | 0.17 | 8.1 | 1.2 | 109.1 | 8.4 |
| 10 | 200 | 0.17 | 0.05 | 448 | 0.04 | 24.6 | 38 | 0.14 | 7.7 | 1.0 | 78.7 | 8.5 |
| 5 | 200 | 0.07 | 0.02 | 637 | 0.02 | 70.5 | 34 | 0.07 | 6.8 | 0.9 | 62.1 | 8.4 |
| 40 | 400 | 0.15 | 0.51 | 591 | 0.03 | 131.1 | 41 | 0.11 | 80.9 | 0.8 | 146.9 | 8.0 |
| 20 | 400 | 0.09 | 0.23 | 696 | 0.03 | 407.1 | 41 | 0.08 | 65.9 | 0.6 | 102.9 | 8.1 |
| 10 | 400 | 0.04 | 0.11 | 909 | 0.01 | 1338.8 | 41 | 0.04 | 58.8 | 0.5 | 75.0 | 8.1 |

Table 6: Computational results for Generalized Assignment using dual price smoothing as a stabilization technique.

We have also experimented with applying the column-and-row generation approach to a larger extended formulation obtained directly from the dynamic programming solver for the pricing sub-problem as explained in Section 3.4. As this larger formulation models a bounded knapsack subproblem, the LP bound then coincides with the one given by the standard column generation. In our experiments, column-and-row generation lead to a reduction of iterations by a factor 3 on average. However, the restricted master of the column-and-row generation approach is much larger than with standard column generation. Hence, this approach is not competitive in terms of cpu time.

## 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 an item 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:

$$[\text{F}] \quad \equiv \quad \min \quad \sum_{ket} (c^k_{et}\, x^k_{et} + f^k_{et}\, y^k_{et}) \tag{77}$$

$$\sum_k y^k_{et} \;\leq\; 1 \quad \forall e, t \tag{78}$$

$$\sum_{\tau=1}^{t} x^k_{e\tau} \;\geq\; \sum_{\tau=1}^{t} x^k_{e+1,\tau} \quad \forall k, e < E, t \tag{79}$$

$$\sum_{\tau=1}^{t} x^k_{E\tau} \;\geq\; D^k_{1t} \quad \forall k, t \tag{80}$$

$$x^k_{et} \;\leq\; D^k_{tT}\, y^k_{et} \quad \forall k, e, t \tag{81}$$

$$x^k_{et} \;\geq\; 0 \quad \forall k, e, t \tag{82}$$

$$y^k_{et} \;\in\; \{0, 1\} \quad \forall k, e, t\ , \tag{83}$$

where variables $x^k_{et}$ are the production of item $k$ at echelon $e$ in period $t$ (at unit cost $c^k_{et}$) and $y^k_{et}$ take value 1 if the production of item $k$ 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 $c^k_{et}$).

For the single item problem, 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 $x^k_{et}\, s^k_{et} = 0\ \forall e, t$. Hence, production can be restricted to lots corresponding to an interval of demands. This dominance property can be exploited to solve single item subproblems by dynamic programming in polynomial time [19]. A backward dynamic program (DP) can be defined where the states are associated with quadruples $(e, t, a, b)$ denoting the fact of having at echelon $e$ in period $t$ accumulated a production that is covering exactly the demand $D^k_{ab}$ for the final product of item $k$. It is defined for $t \leq a \leq b \leq T$ and $e = 1, \ldots, E$. The backward recursion is:

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

for all $e = E, \ldots, 1$, $t = T, \ldots, 1$, $a = T, \ldots, 1$, and $b = T, \ldots, a$. 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 $V^* = V(1,1,1,T)$.

From this 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 $e \in \{1, \ldots, E\}$ in period $t \in \{1, \ldots, T\}$ the production of item $k$ is made to cover demands from period $a \in \{t, \ldots, T\}$ to period $l \in \{a-1, \ldots, T\}$, while the rest of demand interval, i.e. demands from period $l+1$ to period $b \in \{l, \ldots, T\}$, 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, $c^k_{e,t,a,l,b}$, 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:

$$[\mathrm{R}] \quad \equiv \quad \min \sum_{e,t,a,l,b,k} c^k_{e,t,a,l,b}\, z^k_{e,t,a,l,b} \tag{84}$$

$$\sum_{e=1}^{E} \sum_{a,l,b,k:l\geq a, D^k_{al}>0} z^k_{e,t,a,l,b} \leq 1 \quad \forall e,t \tag{85}$$

$$\sum_{l} z^k_{1,1,1,l,T} = 1 \quad \forall k \tag{86}$$

$$\sum_{l} z^k_{e,t,a,l,b} - \sum_{\tau\leq a} z^k_{e,t-1,\tau,a-1,b} - \sum_{\tau\geq b} z^k_{e-1,t,a,b,\tau} = 0 \quad \forall k,e,t,a,b \tag{87}$$

$$z^k_{e,t,a,l,b} \in \{0,1\} \quad \forall k,e,t,a,l,b\,, \tag{88}$$

which results from subproblem reformulation $Z^k$ defined by constraints (86-88) for a fixed $k$. Note that constraints (87) are only defined for $t > 1$ and $b = T$ when $e = 1$; while, when $e > 1$, they are only defined for $a = t$ when $t = 1$.

In Table 7, we compare standard column generation and pure column-and-row generation respectively for $[M_{LP}]$ and $[R_{LP}]$. (Solving formulation $[R_{LP}]$ directly with Cplex is impractical.) Experimental tuning lead to selecting smoothing parameter $\alpha$ to 0.85 and 0.4 respectively, while the restricted master is initialized with a trivial heuristic solution. Results are averages over 5 instances generated randomly, with a number of jobs $K \in \{10, 20, 40\}$, a number of periods $T \in \{50, 100, 200, 400\}$, while setup costs are uniformly distributed in $[20, 100]$, production costs are zero, and storage cost $h^k_e$ are generated as $h^k_{e-1} + \gamma$, where $\gamma$ 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 $[10, 20]$.

For the column generation approach to $[M_{LP}]$, instances get easier as the number of items increases. Indeed, instances with more items have fewer feasible solutions given the single mode constraints. The column-and-row generation clearly outperforms standard column generation on all instances except those with 2 echelons and 50 periods. The number of iterations for column-and-row generation is up to an order of magnitude smaller. This shows 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 $\frac{T}{K}$. Once again, the percentages $(v(\frac{\overline{R}}{R}),\ c(\frac{\overline{R}}{R}))$ are very small. These experiments shows that very large extended formulations can be tractable when solved by column-and-row generation.

| | | Col. gen. for $[M_{LP}]$, $\alpha = 0.85$ | | Col-&-row g. for $[R_{LP}]$, $\alpha = 0.4$ | | | | and relative size | |
|---|---|---|---|---|---|---|---|---|---|
| $K$ | $T$ | *it* | *cpu* | *it* | *cpu* | $v(\frac{\overline{R}}{R})$ | $c(\frac{\overline{R}}{R})$ | $v(\frac{\overline{M}}{\overline{R}})$ | $c(\frac{\overline{M}}{\overline{R}})$ |
| 2 echelons | | | | | | | | | |
| 10 | 50 | 126 | 1.7 | 29 | 1.6 | 0.57 | 1.32 | 26.4 | 3.4 |
| 20 | 50 | 79 | 1.8 | 27 | 3.1 | 0.44 | 1.06 | 17.2 | 2.1 |
| 10 | 100 | 332 | 38.0 | 43 | 8.1 | 0.15 | 0.36 | 34.2 | 3.2 |
| 20 | 100 | 232 | 31.5 | 38 | 20.0 | 0.14 | 0.34 | 21.7 | 1.7 |
| 3 echelons | | | | | | | | | |
| 10 | 50 | 187 | 11.8 | 38 | 5.5 | 0.16 | 1.08 | 24.3 | 3.2 |
| 20 | 50 | 112 | 12.0 | 33 | 9.8 | 0.12 | 0.82 | 16.9 | 2.1 |
| 10 | 100 | 509 | 454.5 | 49 | 36.4 | 0.02 | 0.27 | 33.5 | 3.3 |
| 20 | 100 | 362 | 520.4 | 48 | 103.1 | 0.02 | 0.27 | 21.4 | 1.6 |
| 5 echelons | | | | | | | | | |
| 10 | 50 | 296 | 62.6 | 48 | 16.3 | 0.10 | 0.91 | 24.4 | 3.3 |
| 20 | 50 | 223 | 66.8 | 42 | 34.3 | 0.07 | 0.69 | 18.8 | 2.2 |
| 10 | 100 | 882 | 4855.9 | 61 | 134.0 | 0.01 | 0.24 | 32.9 | 3.2 |
| 20 | 100 | 750 | 4657.8 | 56 | 386.1 | 0.01 | 0.22 | 22.3 | 1.7 |

Table 7: Computational results for multi-echelon multi-item lot-sizing

# Conclusion

The "column-and-row generation" has been presented here as a generalization of standard column generation, in the spirit of [27]. Our aim was 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. Our generic presentation made it straightforward to extend the method to the case where the Dantzig-Wolfe decomposition paradigm is based on a subproblem approximate extended formulation. Then, by pricing on the subproblem integer hull, one can derive Lagrangian dual bounds that serve to define early termination of the algorithm before meeting the reduced cost conditions. We emphasized that the algorithm aims at identifying optimal dual prices (Lagrangian multipliers) for the linking constraints, disregarding the dual value of the other constraints of the extended formulation.

In the literature, a motivation for working with an extended formulation and using the "column-and-row generation" methodology has been the use of the richer variable space of the extended formulation to define cuts or branching constraints. This benefit can be achieved by working with a pricing subproblem in the extended variable space, while working with the traditional master program and a standard column generation approach. Here, we highlighted the benefit of working with a master program expressed in the variable space of the extended formulation, while possibly working with a subproblem compact formulation and implementing a lifting of the subproblem solutions. The interest is to achieve faster convergence thanks to recombinations of previously generated subproblem solutions into new points that are not in the convex hull of currently generated subproblem solutions. By working in the variable space of the extended formu-

lation in both master and subproblem, one can combine the two above benefits to develop branch-and-price-and-cut based on column-and-row generation (such application specific algorithm is beyond the scope of this paper).

We considered two generic situations where the recombination property (Property 1) holds: when the reformulation stems from a network flow model, or a dynamic programming subproblem solver. More generally, this analysis could be extended to generalized flow reformulations, or to reformulations based on a branched polyhedral system [12]. Other examples where the recombination property 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. This is linked to the concept of "base-generators", as developed in [27], that are extracted from regular columns by keeping only the fixed values of the "important" variables in the subproblem solution.

The recombination property leading to a reduction in the number of iterations 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. In the example of Section 1.3, the disaggregation amounts to defining "base-generators" associated to the integer part of the subproblem solution.

The recombination property is closely related to the concept of "exchange vectors" in standard column generation approach [27]; 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 [21].

When the subproblem reformulation is not actually an exact extended formulation for it (i.e., under Assumption 2), there are typically even more recombinations in the relaxed subproblem solution space. The relaxation can imply a weakening of the dual bound, as illustrated on the generalized assignment application, or no difference in dual bound as in the bin packing case. Relaxing Assumption 1 into Assumption 2 is related to the concept of the "state space relaxation" for column generation as presented in [27]. It can also 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].

Our numerical comparative study of column-and-row generation illustrates the experimental trade-off between the comparative acceleration of convergence, the potential losses of quality in dual bounds (under Assumption 2), and the higher computing time required

to solve the restricted master (due to its larger size and potential symmetries). The recommendation that arises from our numerical experiments is to adopt column-and-row generation instead of a standard column generation approach when (*i*) the subproblem solution space offers many possible column recombinations; (*ii*) the extended formulation offers a strong enough relaxation of the pricing problem (when working under Assumption 2); and (*iii*) the pricing procedure dominates the cpu consumption (so that the increase in master solution time is marginalized). We have shown that column-and-row generation makes it tractable to consider much stronger/larger extended formulations, with more combinatorial structure built into the pricing subproblem, while generating only a very small percentage of its variables and constraints.

# Acknowledgments

We thank J. Desrosiers, E. Uchoa, 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] L.-Ph Bigras, M. Gamache, and G. Savard. Time-Indexed Formulations and the Total Weighted Tardiness Problem. *INFORMS Journal on Computing*, 20(1):133–142, 2008.

[4] V. Chvátal. *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] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. Vehicle routing with time windows and split deliveries. Technical Paper 2006-851, Laboratoire d'Informatique d'Avignon, 2006.

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

[8] M. Fischetti and D. Vigo. A branch-and-cut algorithm for the resource-constrained arborescence problem. *Networks*, 29:55–67, 1996.

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

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

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

[12] V. Kaibel and A. 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] A. Löbel. Vehicle scheduling in public transit and lagrangean pricing. *Management Science*, 44(12):1637–1649, 1998.

[14] J. W. Mamer and R. 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.

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

[16] I. Muter, I. Birbil, and K. 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.

[17] A. Pessoa, E. Uchoa, M. Poggi de Aragão, and R. Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2:259–290, 2010.

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

[19] Y. Pochet and L. A. Wolsey. *Production planning by mixed integer programming.* Springer, 2006.

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

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

[22] 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.

[23] 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.

[24] 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.

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

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

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

[28] F. Vanderbeck and L. 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.

[29] P. Wentges. Weighted Dantzig–Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.