Implementing Mixed Integer Column Generation

François Vanderbeck (fv@math.u-bordeaux1.fr)

Laboratoire de Mathématique Appliquées de Bordeaux (MAB),

Université Bordeaux 1, 33405 Talence Cedex, France

Working paper no U-03.28, December 2003 (revised in September 2004).

Abstract

We review the main issues that arise when implementing a column generation approach to solve a mixed integer program: setting-up the Dantzig-Wolfe reformulation, adapting standard MIP techniques to the context of column generation (branching, preprocessing, primal heuristics), and dealing with issues specific to column generation (initialization, stabilization, column management strategies). The description of the different features is done in generic terms to emphasize their applicability across problems. More hand-on experiences are reported in the literature in application specific context, f.i., see Desaulniers, Desrosiers and Solomon (2001) for vehicle routing and crew scheduling applications. This paper summarizes recent work in the field, in particular that of Vanderbeck (2002) and (2003).

Introduction

The Dantzig-Wolfe reformulation approach is an application of a decomposition principle: one chooses to solve a large number of smaller size and typically well-structured subproblems instead of being confronted to the original problem whose size and complexity are beyond what can be solved in reasonable time. The approach is well suited for problems whose constraint set admits a natural decomposition into sub-systems representing well known combinatorial structure.

Consider a mixed integer problem of the form:

$$Z^{MIP} = \min \qquad c\,(x, y) \tag{1}$$

$$[MIP] \quad \text{s.t.} \quad \begin{aligned} A\,(x,y) &\geq a & (2)\\ B\,(x,y) &\geq b & (3)\\ x &\geq 0 & (4)\\ y &\in \mathbb{N}^p & (5) \end{aligned}$$

where $A \in \mathbb{Q}^{l\times(n+p)}$, $B \in \mathbb{Q}^{m\times(n+p)}$, are rational matrices and $c \in \mathbb{Q}^{(n+p)}$, $a \in \mathbb{Q}^l$, and $b \in \mathbb{Q}^m$ are rational vectors. This structure encompasses cases in which the problem has

1. **Difficult Constraints:** $A\,(x,y) \geq a$ represents *difficult constraints.* $B\,(x,y) \geq b$ represents a *more tractable* combinatorial subproblem that can be solved much more efficiently than the global problem. A classic example is the Traveling Salesperson Problem (TSP) where sub-system B represents 1-tree constraints while sub-system A stands for degree-2 constraints at each node. However, the word "efficiently" needs not necessarily mean polynomially solvable.

2. **Linking Constraints:** $B(x,y) \geq b$ has a block diagonal structure while $A(x,y) \geq a$ represents *linking constraints.* An example is the Cutting Stock Problem (CSP) where B defines a feasible way to cut each wide paper roll into products of smaller width (a knapsack subproblem for each wide roll) and A defines product demand covering constraints.

3. **Multiple Sub-Systems:** $A\,(x,y) \geq a$ and $B\,(x,y) \geq b$ represent each a *more tractable* sub-system (possibly having its own block diagonal structure) and the difficulty arises from having them simultaneously. An example is the Capacitated Multi-Item Lot-Sizing (CMILS) problem, where there is a subsystem associated with each item defining a single-item lot-sizing problem and a subsystem associated with each period defining a knapsack capacity constraint.

They are alternative ways to take advantage of such problem structure to compute efficiently strong dual bounds around which an efficient exact optimization approach can be developed: Lagrangian relaxation (Geoffrion, 1974), cutting plane approach (Crowder, Johnson and Padberg, 1983), and variable redefinition (Martin, 1987) are such techniques.

1 Dantzig-Wolfe reformulation

The Dantzig-Wolfe reformulation approach is a special form of variable redefinition. It can be presented using the concept of generating sets: For each sub-system on which the decomposition is based, one defines a finite set of generators from which each subproblem solution can be generated. The variables of the reformulation shall be the weights of the elements of these generating sets.

Saying that G^B is a *generating set* for subproblem

$$X^B \equiv \{(x,y) : \; B\,(x,y) \geq b, \; x \geq 0, \; y \in I\!N^p\} \; . \tag{6}$$

means that

$$X^B = \{(x,y) = \sum_{g \in G^B} g\,\lambda_g : \lambda \in R^B\} \; .$$

Set G^B is typically large but finite even if X^B is not. R^B represents specific restrictions on the weights λ_g, including cardinality and integrality restrictions.

This framework encompasses various definitions of D-W reformulation. In the classic *convexification* approach, the generating set is defined as the set of extreme points and rays of $conv(X^B)$. An alternative approach is to base the decomposition on a property that integer polyhedra can be generated from a finite set of feasible integer solutions plus a non-negative integer linear combination of integer extreme rays (Nemhauser and Wolsey, 1988). This is the *discretization* approach of Vanderbeck (2000). It generalizes to Mixed Integer Programs, by applying discretization in the integer variables (in the y-space) while convexification is used for the continuous variables. In Table 1, we specify the definition of G^B and R^B for each of the above cases. Vanderbeck (2002) discusses other possible definitions of generating sets. In Table 1, we assume boundedness of the subproblem X^B and a single block in matrix B to simplify the notation (the generalization to the unbounded case is straightforward, the case of a block diagonal subsystem is treated below).

The reformulation based on subproblem B takes the form:

$$M(B) \equiv \min\{ \sum_{g \in G^B} (c\,g)\,\lambda_g : \; \sum_{g \in G^B} (A\,g)\,\lambda_g \geq a, \; \lambda \in R^B\} \; . \tag{7}$$

Due to its large number of variables the linear relaxation of the reformulation is solved using a delayed column generation technique. Then, the reformulation is commonly called the master program while the slave is the pricing subproblem. For $M(B)$, the pricing subproblem takes the form

$$\zeta^B(\pi) \equiv \min\{(c - \pi A)\,g : \; g \in G^B\} \; . \tag{8}$$

where π are dual variables associated with constraints $\sum_{g \in G^B} (A\,g)\,\lambda_g \geq a$. The procedure starts by solving a master LP restricted to a subset of columns. While the subproblem

Table 1: Definitions of G^B and R^B under convexification and discretization approaches

Convexification:

$$\begin{cases} G^B = \{(x,y) \in I\!R^n_+ \times I\!N^p : \ (x,y) = \text{ extreme point of } conv(X^B)\} \\ R^B = \{\lambda \geq 0 : \ \sum_{g \in G^B_c} \lambda_g = 1, \ \sum_{g \in G^B} y^g \, \lambda_g \in I\!N^p\} \end{cases}$$

Discretization for an IP:

$$\begin{cases} G^B = \{y \in I\!N^p : \ B\, y \geq b\} \\ R^B = \{\lambda : \ \sum_{g \in G^B} \lambda_g = 1, \ \lambda_g \in I\!N \ \forall \, g \in G^B\} \end{cases}$$

Discretization for a MIP:

$$\begin{cases} G^B_p = \text{proj}_y X^B = \{y \in I\!N^p : \ B\,(x,y) \geq b, \ x \geq 0\} \\ S^B(y) = \{x \in I\!R^n_+ : x \text{ is an extreme point of } B\,(x,y) \geq b\} \\ G^B = \{(x,y) \in I\!R^n_+ \times I\!N^p : \ y \in G^B_p, \ x \in S^B(y)\} \\ G^B(y) = \{g = (x^g, y^g) : \ y^g = y \in G^B_p, \ x^g \in S^B(y)\} \\ R^B = \{\lambda : \ \sum_{g \in G^B} \lambda_g = 1, \ \sum_{g \in G^B(y)} \lambda_g \in I\!N \ \forall \, y \in G^B_p\} \end{cases}$$

returns negative reduced cost columns, they are added to the linear program that is re-optimized. The intermediate master LP values are not valid dual bounds. However, a Lagrangian dual bound can readily be computed from the subproblem value: applying Lagrangian relaxation to M, dualizing the A constraints with weights $\pi \geq 0$ yields a valid bound of the form

$$L(\pi) = \pi\, a + f(\zeta^B(\pi)) \tag{9}$$

where function $f(.)$ takes a form that varies with the definition of the cardinality restrictions in R^B (Vanderbeck, 2003). In the case of a single subproblem taking the form of a bounded integer polyhedron, $f(\zeta^B(\pi))$ simply equals $\zeta^B(\pi)$. Upon completion of the column generation procedure, this Lagrangian bound equals the master LP value. This column generation algorithm is embedded in a branch-and-bound procedure to solve the integer problem. The overall algorithm is known as *Branch-and-Price*.

The interest of the reformulation is twofold. First, it leads to a solution method implementing the desired decomposition of the problem and exploiting our ability to solve the subproblem. Second, it often leads to strong dual bounds: Lagrangian duality theory (Geoffrion, 1974) tells us that the master LP (under a standard definition of the generating set) is equivalent to the Lagrangian dual defined by dualizing the master constraints, itself equivalent to solving an LP on the intersection of the master constraints and the convex hull of the MIP subproblem polyhedron. For $M(B)$, this writes

$$Z_{LP}^{M(B)} = \min\{c\,(x,y) :\ A\,(x,y) \geq a,\ (x,y) \in conv(X^B)\}\ . \tag{10}$$

Thus, $Z_{LP}^{MIP} \leq Z_{LP}^{M(B)} \leq Z^{MIP}$. The first inequality is typically strict unless $X_{LP}^B = conv(X^B)$ (that is when the subproblem has the "integrality property"). In the latter case, although the Dantzig-Wolfe decomposition approach gives a dual bound equal to the standard LP relaxation of the MIP, it might be worthwhile: beyond the motivation of working with a specialized subproblem solver, it may allow to avoid symmetry (as developed below in the case of multiple identical subsystems); it also provides a good strategy for introducing variables dynamically (if the original MIP formulation is itself large scale, as in Briant and Naddef (2004)).

In a standard D-W reformulation, the definitions of G^B and R^B ensure that

Property 1 *The master linear programming relaxation offers a dual bound which is equal to the Lagrangian dual value (10): i.e., we must have*

$$\{(x,y) = \sum_{g \in G^B} g\,\lambda_g : \lambda \in R_{LP}^B\} = conv(X^B)\ ;$$

Property 2 *The master program (7) is a valid mixed integer reformulation of the original MIP problem (1-5).*

However, in extensions of the D-W reformulation principle, one might consider generating set definitions where these properties are relaxed as discussed in Section 3.

When matrix B is block diagonal

$$B = \begin{pmatrix} B^1 & 0 & \dots & 0 \\ 0 & B^2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & B^K \end{pmatrix} \tag{11}$$

with K non-identical blocks, one defines a generating set G^{B^k} for each $k = 1, \dots, K$. But, when the blocks are identical, i.e. $B^1 = B^2 = \dots = B^K$, one only needs to define one generating set $G^B = G^{B^1}$. The weigths associated with the generators can then be aggregated,

$$\lambda_g = \sum_{k=1}^{K} \lambda_g^k \quad \forall g \in G^B \tag{12}$$

and the cardinality constraint in the associated restriction set R^B be written as

$$\sum_{g \in G^B} \lambda_g = K \; . \tag{13}$$

Then, there is a third benefit in working with the D-W reformulation: it does not suffer from the drawback of symmetry present in the original formulation. Indeed, in the original formulation, variables must be indexed by k and a given solution can be represented in several ways by permuting the k indexing. This is quite harmful when it comes to enforcing integrality through branching constraints.

A distinction is to be made between convexification and discretization approaches that relates to the expression of the integrality restrictions. With the former one must translate the master solution in the space of the original variables to express and enforce integrality. As a consequence, when there are multiple identical sub-systems, one cannot aggregate weights using (12) and the reformulation suffers from the same symmetry drawback as the original formulation. The discretization approach, offers a true integer programming reformulation with which it is easier to implement some branching schemes as we shall see.

In most applications, cardinality constraints (13) can be relaxed to either a $\leq$ or a $\geq$ constraint (whether $K = 1$ or $K > 1$) as the combinatorial relaxation yields the same optimum value. This is in particular the case when $0 \in G^B$. A generalization is to write the cardinality constraint in the form:

$$L^B \leq \sum_{g \in G^B} \lambda_g \leq U^B \tag{14}$$

where L^B and U^B define a lower and an upper bound on the number of generators that can be used from subproblem B. They have an economical interpretation: for instance in a vehicle routing problem they stand for the minimum and maximum number of vehicles that must/can be used.

Formulation (7) assumes a simple *Lagrangian relaxation* of a sub-system A while B are being kept as hard constraints. However, one can treat both sub-systems as hard constraints by duplicating variables and dualizing constraints that enforce equality of the copies: this *variable splitting* approach implements a *Lagrangian decomposition* (Guignard, 2004). Then, the master takes the form

$$M(A,B) \equiv \min\{ \sum_{g \in G^A} (\frac{c}{2}\, g)\, \lambda^A_g + \sum_{g \in G^B} (\frac{c}{2}\, g)\, \lambda^B_g :$$

$$\sum_{g \in G^A} g\,\lambda_g^A = \sum_{g \in G^B} g\,\lambda_g^B, \tag{15}$$
$$\lambda^A \in R^A,\ \lambda^B \in R^B\}$$

and the dual bound is

$$Z_{LP}^{M(A,B)} = \min\{c\,(x,y) :\ (x,y) \in conv(X^A) \cap conv(X^B)\}\ . \tag{16}$$

In the sequel, unless said otherwise, we assume a simple Lagrangian relaxation with B as the sub-system.

2 Choosing a solver for the Master

There are alternatives to solving the master LP with the revised simplex algorithm. Seen in the dual space, the latter amounts to applying a cutting plane algorithm known as Kelley's method (Kelley, 1960; Cheney and Goldstein, 1959); the column generation subproblem is then to be seen as a separation routine generating cuts. Other approaches are known to have better performance: the bundle method (Lemaréchal, 1998) and ACCPM (Goffin and Vial, 2002) are cutting plane algorithms for the dual of the master that differ from Kelley's cutting plane method by the choice of proposal that is made to the subproblem solver.

In Kelley's method, the dual solution that is returned to the subproblem is the point that optimizes the current restricted master linear program. In the bundle method the proposal is the point that optimizes a modified objective over the restricted master polyhedron: a quadratic term that penalizes the norm of the deviation to the current best dual solution, $\hat{\pi}$, is included in the objective to stabilize the search ($\hat{\pi}$ is the dual solution that gave the best Lagrangian bound value). In ACCPM, the proposal is the analytic center of the localization set defined by the restricted master polyhedron intersected with a cut on the objective value (defined by the best Lagrangian bound). The latest version of ACCPM (du Merle and Vial, 2002) includes a proximal term like in the bundle method to enhance stability.

Another alternative to the simplex algorithm is the interior point method that is readily available in most commercial MIP solvers. Applied to the dual it has the benefit of (approximately) computing dual prices centered in the optimal face when the dual admits many alternative optimum solutions (as it often happens)– before translating it into an extreme solution. Such "centered dual solutions" may help in stabilizing the column generation procedure. Finally, a sub-gradient algorithm is an alternative that exhibits slow convergence but requires little computations at each iteration. It is often used when one is happy with an approximate dual solution of the master LP. The sub-gradient algorithm also provides a candidate primal solution in the convex hull of generated subproblem solutions, but it may violate the master constraints (Briant et al., 2004).

3 Options in setting-up the D-W reformulation

Given an application, there are typically many ways of formulating it as a MIP and selecting sub-systems on which the D-W decomposition is based. The quality of the resulting dual bounds can be compared theoretically using Lagrangian duality theory (Geoffrion, 1974). But, for practical purposes, the important consideration is the trade-off between the duality gap observed on practical data sets and the computational time required to solve resulting master and subproblems.

The options are:

- To adopt a tighter or looser definition of a sub-system. Some constraints can go in either sub-system A or B or be duplicated in both. Implicit constraints (cuts) can also be added. In particular, one can enforce bounds on subproblem variables that are induced by constraints that are not in the subproblem. Columns whose coefficients take value within the bounds implied by master constraints are called *proper*. Vanderbeck (2002) also defines *strongly proper columns* whose component values satisfy bounds implied by using probing techniques in the master. Including extra restrictions in a subproblem may make it much harder to solve but may yield better dual bounds. The opposite trade-off consists in relaxing the subproblem to ease its solution while weakening the dual bounds. This can be viewed as a form of *state-space-relaxation* –a classical technique in dynamic programming – see Gamache et al. (1999) for an example of its use in a column generation context. Such relaxation can be exploited just to provide a warm start, or to implement a primal-dual heuristic for the subproblem.

- To base the reformulation on one or several sub-systems: a Lagrangian relaxation approach yields reformulation (7) while a Lagrangian decomposition approach yields reformulation (15). Although the latter yields a theoretically stronger dual bound, the master program is typically much harder to solve: the number of dualized constraints (15) is often large and each restricted master typically admits many alternative dual solutions (a degeneracy that causes a large number of iterations in the column generation procedure). Moreover the theoretically better dual bound may in practice be not significantly better than the bound given by simple Lagrangian relaxation (Monneris, 2002). Guignard (2004) proposes to aggregate variables in order to reduce the number of dualized constraints (15); in some case the same quality dual bound can be obtained with aggregate representation of the link between the subsystems.

- To reformulate a sub-system (say A), improving its formulation, either instead of treating it as a subproblem (while decomposition is performed based on a second sub-system(B) as in Michel (2003)) or before applying D-W decomposition (as in Thizy (1991)). One might have a formulation of the convex-hull of subproblem A solutions: for instance, as a network flow problem equivalent to solving the subproblem by dynamic programming (Martin, 1987). Or one might know cutting planes describing the integer polyhedron A. However, the size of such explicit reformulation is often too large. Then an approximate improved formulation can be used: for instance by aggregating variables (as in van Vyve (2003)) or by adding only a subset of cuts. The formulation can also be improved dynamically, using a cutting plane algorithm, which, combined with the column generation procedure, leads to a branch-and-price-and-cut algorithm.

- To disaggregate a sub-system, say B, that has a block diagonal structure or treat it as a single block. As discussed in Vanderbeck (2003), this does not affect the quality of the final dual bound as $conv(X^B) = \otimes_{k=1}^{K} conv(X^{B^k})$. But intermediate dual bounds can be better with the aggregated approach. (However, one can work with a disaggregate master while adopting the Lagrangian bound computation and the column generation strategy of the aggregate approach.) On the other hand, aggregation influences the efficiency of the master solver: the disaggregate formulation typically requires fewer iterations (because the disaggregate columns can be recombined in different implicitly defined aggregate columns); but solving a disaggregate master can be more costly computationally (in particular, when the master is solved using the bundle method, solving the disaggregate form of the quadratic master is much harder than the aggregate form).

4 Branching Scheme

In a branch-and-bound algorithm, the branching scheme consists in partitioning the solution space into sub-spaces so as to cut off the current fractional solution while ensuring that all feasible integer solutions are included in one of the sub-spaces. In a branch-and-price algorithm one can fall back to this standard scheme by using a convexification approach. Then, one returns to the space of the original variables (x, y) to check integrality and derive branching constraints (Villeneuve et al., 2003).

Alternatively, with the discretization approach, one has a mixed integer programming D-W reformulation and branching can be implemented by partitioning the generating sets, $G = G^1 \cup G^2$, and enforcing separate cardinality constraints (14) on each subset:

$$L^1 \leq \sum_{g \in G^1} \lambda_g \leq U^1 \quad \text{and} \quad L^2 \leq \sum_{g \in G^2} \lambda_g \leq U^2$$

so as to eliminate infeasible fractional solutions for which $\sum_{g \in G(y)} \lambda_g \notin I\!N$ for some $y \in G_p$. The recursive partition of the generating set shall be pursued until the integrality constraints $\sum_{g \in G^B(y)} \lambda_g \in I\!N$ are met for all $y \in G_p^B$. The scheme seems to call for associating a subproblem with each subset of generators G^1 and G^2. This, of course, quickly gets out of hand due to the exponentially increasing number of subsets. Instead, the pricing subproblem associated with G is amended to model correctly the reduced cost of generators from any subsets. The resulting modifications to the pricing subproblem range from amending the objective coefficients to adding new variables and constraints that may destroy the initial combinatorial structure and make the subproblem less tractable.

In practice, the separation of the generating set shall define a true partition $G = \hat{G} \cup (G \setminus \hat{G})$. Branching is implemented by choosing $\hat{G}$ such that $\sum_{g \in \hat{G}} \lambda_g = \alpha \notin I\!N$ and by adding a disjunctive branching constraint

$$\sum_{g \in \hat{G}} \lambda_g \leq \lfloor \alpha \rfloor \quad \text{or} \quad \sum_{g \in \hat{G}} \lambda_g \geq \lceil \alpha \rceil \; . \tag{17}$$

An indicator variable $1_{\hat{G}}$ saying whether or not the generated column belongs to $\hat{G}$ is added to the pricing subproblem. Its cost equals the dual value of the branching constraint. MIP constraints can be necessary to define $1_{\hat{G}}$ in terms of the existing subproblem variables while in the simplest case $1_{\hat{G}}$ is equal to an existing subproblem variable. Thus, sets $\hat{G}$ must be carefully chosen, making a trade-off between the efficiency of the branching scheme and the difficulties resulting from the subproblem modifications (Vanderbeck, 2000).

Branching on a single λ_g variable corresponds to choosing $\hat{G}$ to be a singleton. This is typically a poor choice on both regards: it yields an unbalanced branch-and-bound tree and requires heavy modifications to the subproblem. Instead, the most aggregate definitions of $\hat{G}$ that yield the most balanced tree are also the easiest to accommodate in the subproblem. The simplest definition is $\hat{G} = G$, enforcing integrality of the number of generators used from a sub-system:

$$\textbf{Rule 1} \qquad \sum_{g \in G} \lambda_g \in I\!N \; .$$

For instance, in a column generation approach to the facility location problem, this rule enforces $0 - 1$ decisions on opening facilities. The next easiest rule that does not yield any subproblem modification is to branch on existing binary variables: say $y_i \in \{0, 1\}$ is a subproblem variable then define $\hat{G} = \{g \in G : y_i = 1\}$ and enforce

$$\textbf{Rule 2} \qquad \sum_{g \in G : y_i = 1} \lambda_g \in I\!N \; .$$

Then, $1_{\hat{G}} = y_i$.

Next consider an integer variable of the subproblem, say $y_k \in I\!N$, with bounds $l_k \leq y_k \leq u_k$. It admits a binary decomposition:

$$y_k = \sum_{j=0}^{\lfloor \log u_k \rfloor} 2^j y_{kj} \quad \text{with} \quad y_{kj} \in \{0, 1\} \; \forall j \; .$$

One can branch on these binary components starting with the heaviest:

$$\textbf{Rule 3} \qquad \sum_{g \in G : y_{kj} = 1} \lambda_g \in I\!N \; .$$

The modification to the subproblem consists in introducing the binary components either dynamically as they arise in branching constraints (one replaces y_k using $y_k = y'_k + \sum_{l=j}^{\lfloor \log u_k \rfloor} 2^l y_{kl}$ with $y'_k \in I\!N$, $y'_k \leq 2^j - 1$ and bounds $l_k \leq y_k \leq u_k$ must be explicitly enforced on the new expression of y_k); or statically by reformulating the subproblem as a 0-1 program.

Both Rule 2 and 3 can sometimes be enforced directly in the subproblem rather than being set as a master constraint. This results in a stronger dual bound. When the upper bound $\lfloor \alpha \rfloor$ of branching constraint (17) is equal to zero, no generators can be chosen in $\hat{G}$ and the subproblem solution space can be redefined to yield $G \setminus \hat{G}$. Symmetrically, when the lower bound $\lceil \alpha \rceil$ of (17) equals the upper bound U of the cardinality constraint

(14), all the generators must be chosen in $\hat{G}$ and the subproblem solution space can be redefined accordingly.

When the master solution remains fractional beyond these simplest rules, one can branch on pairs (s, t) of binary variables or binary components introduced for Rule 3:

$$\textbf{Rule 4} \qquad \sum_{g \in G: y_s = y_t = 1} \lambda_g \in I\!N \; .$$

This rule generalizes a branching scheme attributed to Ryan and Foster (1981). It admits special cases where it can be implemented directly in the subproblem while it normally requires to introduce a new binary variable in the subproblem, say $1_{\hat{G}} = y_{st}$. Specifically, there are four distinct cases:

1. When $\lfloor \alpha \rfloor = 0$, the constraint $y_s + y_t \leq 1$ is added to the subproblem.
2. When $\lfloor \alpha \rfloor > 0$, y_{st} is introduced in the subproblem along with the constraint $y_{st} \geq y_s + y_t - 1$.
3. When $\lceil \alpha \rceil = U$, the constraint $y_s = y_t$ is added to the subproblem.
4. When $\lceil \alpha \rceil < U$, y_{st} is introduced in the subproblem along with the constraints $y_{st} \leq y_s$ and $y_{st} \leq y_t$.

Columns present in the master that do not satisfy the new subproblem constraints must be deleted. Vanderbeck (2003) presents this rule in a more general context and defines two more special cases.

The above rules may not suffice to generate a solution that obey the integrality restrictions. Then one shall pursue branching by implementing rules based on subset $\hat{G}$ defined by fixing more than 2 column components. Worst case results and subproblem modifications are given in Vanderbeck (2000). However, for many applications, the above rules suffice to ensure integer solutions: either theoretically – for instance when the master is a set partitionning problem – or in practice – for example, most instances of the cutting stock problems can be solved by combining these basic branching rules with primal heuristics (Perrot, 2004).

Choosing specific components (or pairs) on which to branch is done according to branching priorities associated with variables of the original formulation. Which of the disjunctive branch is explored first depends on the priorities defined for original variables.

A tree search strategy (best bound first, depth first, least discrepeancy, ...) must be selected. The above hierarchy of rules along with a best bound first strategy focuses on improving dual bounds. However, if the goal is to obtain incumbent integer solution quickly, branching rules yielding an unbalanced tree may be appropriate along with a depth first or a least discrepancy search strategy. One might even consider branching by rounding-up λ_g if backtracking is not expected (or in a rounding heuristic).

The distinction between convexification and discretization approaches is mostly a matter of presentation framework. The discretization has an apparent drawback: it requires

$$\sum_{g \in G^B(y)} \lambda_g \in I\!N \;\forall\, y \in G^B_p \tag{18}$$

although an integer solution can be obtained for which this isn't true: The current solution in λ may be fractional, while its translation in the original variables y is not. This is in particular the case in applications where the subproblem is a shortest path problem: as we know from the flow decomposition theorem, the arc flow y can be integer while the path flows λ are fractional. Thus one could say that the condition (18) is too strict and might lead to continue on branching when it is not necessary. This misfortune can easily be avoided by checking integrality using the convexification conditions $\sum_{g \in G^B} y^g \lambda_g \in I\!N^p$ while implementing branching with the goal of enforcing (18).

On the other hand, the convexification approach has its drawbacks too. With the plain implementation defined in Table 1, branching constraints are always implemented in the master and are therefore dualized. While, with the discretization approach, special cases can be detected automatically where branching can be enforced directly in the subproblem. Then, the branching constraints are part of the polyhedron that is convexified yielding a better Lagrangian dual bound. However, this apparent drawback of the convexification approach can be circumscribed in an application specific context by recognizing, in the master, constraints that concern the subproblem variables only and can therefore be placed in the subproblem. It is equivalent to redefining the D-W reformulation from the original formulation at each node of the branch-and-bound tree (Villeneuve et al., 2003). We could call this approach the "dynamic convexification approach", it reproduces the subproblem modifications of the discretization approach. Thus, the discretization approach can be understood as a way to implement implicitly/automatically the redefinition of the D-W reformulation at each branch-and-bound node with no need to do it explicitly/manually. While the convexification approach is normally static but can be reseted at each node.

The second drawback of the convexification approach arises when the sub-system on which the decomposition is based has a block diagonal structure, as it is often the case. Even if these blocks are identical, the convexification approach requires to associate different variables with each block to enforce $\sum_{g \in G^{B^k}} y^g \lambda_g^k \in I\!N^p \forall k$, which results in symmetry. Again, this drawback can be overcome: by introducing auxiliary aggregate variables, $y = \sum^k \sum_{g \in G^{B^k}} y^g \lambda_g^k$ on which to branch. Again, this remark reveals what the discretization approach does implicitly: it permits to branch on auxiliary variables. Even when diagonal blocks are not identical but are similar, it is worthwhile branching on sets $\hat{G} \subset G = \cup_{k=1}^K G^k$ to avoid symmetry.

Note that when these two drawbacks of the convexification approach cumulate, i.e. when you have a branching constraint that could go in the subproblem in the presence of multiple blocks, the manipulations that need to be done under convexification to compensate for them become difficult to implement. For instance, consider implementing Rule 4 in the presence of multiple identical blocks. Thus, our view is that the discretization approach provides a theoretical framework in which it is more natural to formulate branching constraints. In particular, any branching rule implemented in the convexification framework can be implemented in the discretization framework in the same way but the converse is not true. The same conclusion holds for adding cutting planes for the master.

5 Preprocessing and variable fixing

Standard preprocessing techniques (Savelsbergh, 1994) can also be used in a column generation approach. Not directly on the column generation re-formulation that is only known implicitly through the solution of subproblems, but in the solution space of the original formulation (1-5), which is also that of the subproblem variables. Bounds on subproblem variables can be tightened not only based on subproblem constraints but also on the basis of their input in the master constraints. Tighter bounds might permit to eliminate redundant constraints (in the subproblem or in the master) or to prove infeasibility. The reduced cost fixing techniques permit to strengthen subproblem variable bounds further. Preprocessing can also be used to estimate the range of dual price values: this dual information is helpful for a warm start of the column generation procedure.

When the sub-system is made of blocks (11), preprocessing can be used to derive bounds on aggregate variables $(x, y) = \sum_{k=1}^K (x^k, y^k)$, which in turn implies bounds on the disaggregate variables (x^k, y^k). The useful formulation where standard preprocessing can be applied is therefore:

$$
\begin{aligned}
\min c\,(x,y) & & & \qquad (19)\\
A\,(x,y) &\geq& a & \qquad (20)\\
B^k\,(x^k,y^k) &\geq& b \ \ \forall k=1,\ldots,K & \qquad (21)\\
(x,y) &=& \sum_{k=1}^{K}(x^k,y^k) & \qquad (22)\\
0 \leq l^k \leq (x^k,y^k) &\leq& u^k \qquad \forall k = 1,\ldots,K & \\
0 \leq l \leq (x,y) &\leq& u\,. & \qquad (23)
\end{aligned}
$$

Using constraints (20) to strengthen bounds (l,u) and constraints (21) to strengthen bounds (l^k,u^k), one can then combine these strengthening through constraints (22).

Formulation (19-23) does not need to be generated explicitly, but these implicit relations can be exploited for preprocessing. As it is standard, the lower bound on a variable x_j with positive coefficient a_{ij} in $\geq$ constraints A_i can be improved if it violates

$$l_j \geq (a_i - \sum_{v\neq j:a_{iv}>0} a_{iv}\,u_v - \sum_{v\neq j:a_{i,v}<0} a_{iv}\,l_v)/a_{ij}$$

and similarly for other coefficient sign and constraint sense and for (x^k,y^k) in the B^k constraints. Then, combining aggregate and disaggregate bounds can result in further strenghtening if one of the following relations is violated:

$$l_i^k \geq l_i - \sum_{t\neq k} u_i^t\,, \qquad u_i^k \leq u_i - \sum_{t\neq k} l_i^t\,,$$

$$l_i \geq \sum_{k=1}^{K} l_i^k\,, \qquad u_i \leq \sum_{k=1}^{K} u_i^k\,.$$

When the diagonal blocks are identical, $(l^k,u^k)=(\hat{l},\hat{u})\ \forall k=1,\ldots,K$, the above procedure simplifies: one only needs to preprocess one sub-block and the strengthening through combining aggregate and disaggregate bounds takes the form

$$\hat{l}_i \geq l_i - (U-1)\,\hat{u}_i\,, \qquad \hat{u}_i \leq u_i - (L-1)\,\hat{l}_i\,,$$

$$l_i \geq L\,\hat{l}_i\,, \qquad u_i \leq U\,\hat{u}_i\,,$$

where (L, U) are the bounds of the cardinality constraint(14). These relations are tested iteratively until no variable bounds can be improved. For integer variables, bounds are rounded to their integer parts.

If an incumbent integer solution INC is available, it can be shown to dominate solutions where a variable takes value within restricted range: if a dual bound conditional to such restriction is available that takes value greater than INC. Lagrangian bound (9) can be made conditional to restricted range (l', u') on subproblem variables:

$$L(\pi | l', u') = \pi\, a + \min\{(c - \pi A)\,(x, y) :\ B\,(x, y) \geq b,\ l' \leq (x, y) \leq u'\}\ . \qquad (24)$$

Thus, further bound strengthening for integer variables y_i can be achieved as follows:

$$\text{while}\ \ L(\pi | l' = l + e^i\,(u_i - l_i), u) \geq INC\ , \quad \text{reset } u_i := u_i - 1\ ,$$

$$\text{while}\ \ L(\pi | l, u' = u - e^i\,(u_i - l_i)) \geq INC\ , \quad \text{reset } l_i := l_i + 1\ ,$$

where e^i is the i^{th} unit vector. The reduced cost indicates which variable bound is candidate for such strengthening (Vanderbeck, 2003).

This so-called "variable fixing" scheme is computationally costly since it requires solving a subproblem for each test. However, a dual bound on the subproblem value is sufficient to define a valid test. Thus, in practice, one may solve a relaxed subproblem: for instance, a combinatorial relaxation of constraints $B\,(x, y) \geq b$ yields a trivial bound. A linear relaxation of the subproblem can yield a stronger bound (if the subproblem has the integrality property this bound has the same quality as (24)– this is the case when the subproblem is a shortest path problem). Yet another strategy is to dualize constraints $B\,(x, y) \geq b$ and apply a dual heuristic (such as several steps of a sub-gradient method) to estimate the Lagrangian prices.

When variable bounds cannot be further strengthened, constraint redundancy and infeasibility can be tested. The improved bounds on subproblem variables shall be used in the definition of the generating set. They guarantee that only *proper* columns shall be generated. However, this can make the subproblem much harder to solve. Take the two dimensional knapsack problem for example. The unbounded version can be solved in pseudo-polynomial time by dynamic programming while the bounded version is strongly NP-hard. For the single-item lot-sizing problem the uncapacitated version is polynomial while the capacitated case is NP-hard. Nevertheless it might be worth enforcing subproblem variable bounds since the D-W reformulation yields a better dual bound then.

Applying preprocessing to the dual of (19-23) allows to estimate the dual prices. This LP dual is a valid dual problem for the mixed integer program (1-5) or (7). However, the bounds on dual prices are not valid bounds for the Lagrangian dual (10). They merely provide estimates of dual price values. Take the example of the cutting stock problem. The primal LP takes the form:

$$\min\{\sum_k y_k : \sum_k x_{ik} \geq d_i \; \forall i, \sum_i w_i \, x_{ik} \leq W \, y_k \; \forall k, \; x_{ik} \geq 0, \; y_k \in [0,1]\}$$

where x_{ik} is the number of pieces of product i cut in wide-roll k, while y_k is 1 if wide-roll k is used. Its dual writes

$$\max\{\sum_i d_i \, \pi_i - \sum_k \nu_k : \; \sigma_k \, W - \nu_k \leq 1 \; \forall k, \; \pi_i - w_i \, \sigma_k \leq 0 \; \forall ik, \; \pi_i, \nu_k, \sigma_k \geq 0\}.$$

It can be solved trivially through preprocessing: $\sigma_k \leq \frac{1+\nu_k}{W} \Rightarrow \pi_i \leq \frac{w_i}{W} (1 + \nu_k)$; the objective is maximized by setting the π_i's equal to those upper bounds and $\nu_k = 0 \; \forall k$ because $K \geq \frac{\sum_i w_i \, d_i}{W}$. Thus $\pi_i = \frac{w_i}{W}$ solves the dual LP. These values provide good estimates for starting up the column generation procedure but do not define valid upper bounds on the Lagrangian dual solution.

6 Initialization

The generalized simplex algorithm used to solve the column generation formulation must be started with a feasible primal solution. It can be constructed heuristically. Alternatively, one can introduce artificial columns and combine phase 1 and phase 2 of the simplex method: at the end of phase 2, if artificial columns remain in the primal solution, their cost is increased and one returns to phase 2. Artificial columns remain useful to restart column generation after adding branching constraints. They also stabilize the column generation procedure. Dual methods need not be started with a primal feasible solution but they can benefit anyway from a warm start.

There are various ways to set artificial variables in the D-W reformulation from the most aggregate to the least: one can define a single artificial column, or one for each subproblem (representing the sort of solutions expected from this subproblem) or one for each master constraint (which offers an individual control on dual prices). In defining artificial column coefficients, one tries to feed the model with relevant information concerning the order of magnitude of the data: after the pre-processing phase, one has estimates of the maximum and minimum value that can be assumed by master constraint

LHS, either globally or for a single subproblem solution; these values can be used to set artificial column coefficients.

In the primal, artificial columns act as slack variables that transform the hard constraints into soft ones: minimizing their cost amounts to minimizing the constraints' violations. In the dual of the master program, artificial columns define cuts (in the same way as regular columns do). Let us look for instance at the primal dual pair for the LP relaxation of (7) augmented with artificial columns associated with individual master constraints (indexed by i). The artificial columns are defined by their cost, constraint coefficient and upper bound: $(\hat{c}, \hat{a}, \hat{u})_i$.

$$\text{Primal:} \left\{ \begin{array}{rcll} \min \sum_{g\in G} c_g\,\lambda_g + \sum_i \hat{c}_i\,\rho_i & & & \\ \sum_{g\in G} a_{ig}\,\lambda_g + \hat{a}_i\,\rho_i & \geq & a_i & \forall i \\ \sum_{g\in G} \lambda_g & = & 1 & \\ \lambda_g & \geq & 0 & \forall g \in G \\ 0 \leq \rho_i & \leq & \hat{u}_i & \forall i \end{array} \right.$$

$$\text{Dual:} \left\{ \begin{array}{rcll} \max \sum_i a_i\,\pi_i - \sigma + \sum_i \hat{u}_i\,\nu_i & & & \\ \sum_i a_{ig}\,\pi_i - \sigma & \geq & c_g & \forall g \in G \\ \hat{a}_i\,\pi_i & \leq & \hat{c}_i + \nu_i & \forall i \\ \pi_i,\ \nu_i & \geq & 0 & \forall i\ . \end{array} \right.$$

It appears that the artificial column cost over its constraint coefficient, $\frac{\hat{c}_i}{\hat{a}_i}$, defines an upper bound on the dual variable π_i that can be overruled (when $\nu_i > 0$) at a cost defined by the upper bound $\hat{u}_i$. In the same way, more aggregate definitions of artificial columns define generalized upper bounds on dual prices. Modifying their definition dynamically is a way to control dual variables. A general presentation of the primal-dual interpretation of modifications done to the master can be found in Briant et al. (2004) – it relies on Fenchel duality.

At a branch-and-price node other than the root, the pool of already generated columns offers a natural set for initialization, once the columns that do not satisfy branching constraints are eliminated, but it might be too large. Selecting only the columns that were potentially active at the parent node (i.e. with zero reduced cost) offers a good filter.

An intelligent initialization of the restricted master program offers a *warm start*. It might be worthwhile working harder at initialization. One can generate an initial set of columns based on the above dual price estimates. In turn, dual price estimates can be refined further using a truncated version of the master solver: for instance, perform a few pivots of the simplex or several iterations of a sub-gradient algorithm with a subproblem optimized over the existing columns. The spirit is to avoid using the computationally costly procedure (whether it is the exact solution of the subproblem or that of the restricted master) while the problem knowledge is approximative. This strategy has sometimes proved helpful (Caprara, Fischetti and Toth, 2000). However, the exact method (for the master or the subproblem) is often very quick at the outset (because of the simplicity of initial master formulation or subproblem objective) and using approximate procedures does not always translate in big time savings.

7 Stabilization

The standard method for solving the LP relaxation of D-W reformulation, i.e. using the simplex algorithm with dynamic pricing of variables, suffers from several drawbacks as illustrated in Figure 1: (i) a slow convergence (a phenomenon called the *"tailing-off effect"* in reference to the long convergence tail); (ii) the first iterations produce irrelevant columns and dual bounds due to poor dual information at the outset (let us call this the *heading-in effect*), (iii) *degeneracy* in the primal and hence multiple optimal solutions in the dual: one can observe that the restricted master solution value remains constant for several iterations (which can be called the *plateau effect*); (iv) instability in the dual solutions that are jumping from one extreme value to another (the *bang-bang effect*); the distance between the dual solution at an intermediate iteration k, π^k, and an optimum dual solution, π^*, does not necessarily decrease at each iteration, i.e., one can have $||\pi^k - \pi^*|| > ||\pi^{k-1} - \pi^*||$; (v) likewise, the intermediate Lagrangian dual bounds (9) do not converge monotonically (the *yo-yo effect*).

Stabilization techniques have been developed to reduce these drawbacks (a review is given in Lübbecke and Desrosiers (2004)). Vignac (2003) classifies them into 4 categories according to the mechanism that is used: (i) defining bounds on dual prices (the dynamic boxes of Marsten, Hogan and Blankenship (1975)), (ii) smoothing dual prices that are returned to the subproblem (Neame (1999) takes a convex combination of the current master dual solution and that of the previous iterate, Wentges (1997) takes a convex combination of the current dual solution and the dual solution that yielded the best Lagrangian bound), (iii) penalizing deviation from a stability center, usually defined as the dual solution that gave the best Lagrangian bound value (as in the bundle method of Lemaréchal (1998)), (iv) working with interior dual solution rather than extreme dual solutions (as in the solution method ACCPM of Goffin and Vial (2002), or in Rousseau, Gendreau and Feillet (2003), where the center of the dual optimal face is computed ap-

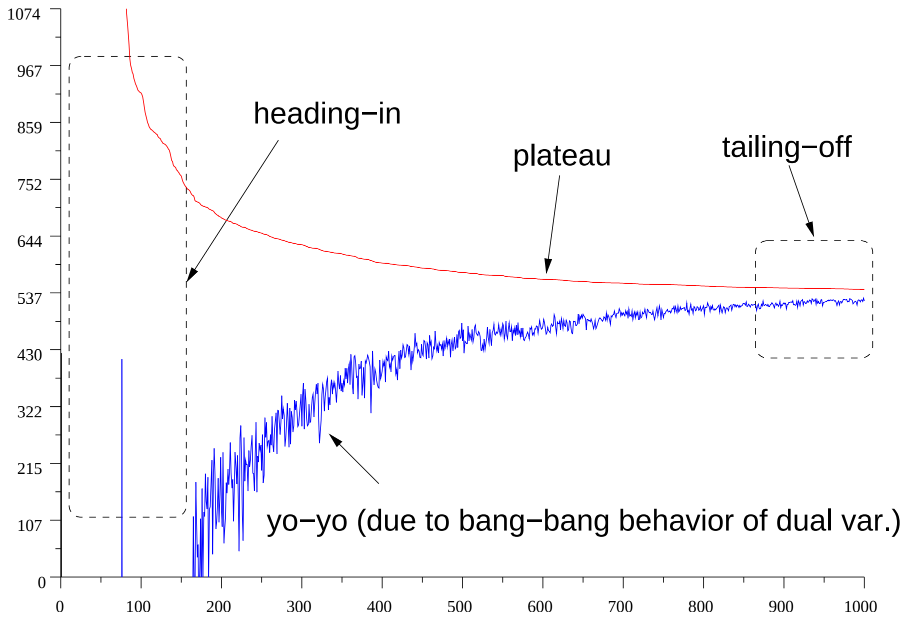

Figure 1: Illustration of the convergence of a simplex-based column generation approach on the TSP instance eil76 of the TSPLIB: the upper (resp. lower) curve gives the primal (resp. dual) bound at each iteration.

proximately). These can be combined into hybrid methods (du Merle et al., 1999; Neame, 1999; du Merle and Vial, 2002).

These techniques often have an intuitive interpretation in the primal space of the λ variables (Briant et al., 2004). The basic techniques (dynamic boxstep and smoothing) can be implemented within the linear programming framework. Observe that using simplex re-optimization after adding a column rather than optimization from scratch can bring some form of stabilization toward the previous primal solution (whether this is desirable or not). Other techniques, such as penalizing deviation from a stability center, can be tuned to remain usable within the linear programming context (by choosing the L_1 norm for instance). Otherwise, an alternative master solver (as presented in Section 2) must be used. It is only by using a different master solver (such as bundle or ACCPM) that one can get theoretical convergence rate better than that of the simplex-method.

There are basic tips to reduce instability. One should avoid including redundant constraints in the master as they create degeneracy. In particular, one should use inequality constraints rather than equalities when this combinatorial relaxation yields the same solution. This also bounds the dual variables to positive values. Including the original variable in the master problem, as proposed in Poggi de Aragão and Uchoa (2003), can yield harmful redundancy. A more aggregate formulation of master constraints can bring stability (Perrot, 2004).

Basic stabilization can be obtained by managing artificial columns dynamically to control dual variables along with an intelligent initialization of the restricted master program that allows to alleviate the heading-in effect (Briant et al., 2004). The next easy step is to use a form of smoothing of dual prices. The more advanced stabilization techniques, in particular the hybrid methods, require intensive parametrization (which may hide the logic). They can be seen as barely attempting to mimic the behavior of alternative dual price updating methods such as an interior point solver, the bundle method or ACCPM. They are attempt at getting the benefits of these alternative approaches while staying in the Simplex world. Therefore, their theoretical convergence rate is that of the simplex method. When instability is problematic beyond the use of basic procedures, one should consider methods based on non-differentiable convex optimization that really bring a different logic.

In a branch-and-price algorithm however the exact solution of the master LP might not always be needed. A node can sometimes be cut by bound before reaching optimality of the master LP. The convergence tail can also be truncated by branching early.

8 Column re-optimization and post-processing

Re-optimizing existing columns before calling on the oracle subproblem to generate new columns, has proved to be a successful strategy: for instance, Savelsbergh and Sol (1995) apply a local search heuristic to active columns. It can be used as an alternative to a multiple column generation strategy: instead of introducing several columns of negative reduced cost at a given iteration, return only one but scan the others in the re-optimization phase after updating the dual prices.

Re-optimization can be somewhat formalized using the concept of *base-pattern* of Vanderbeck (2002). If the generating set defines our universe, *base-patterns* are the aggregate states resulting from some sort of projection/mapping onto a smaller size universe. In a sense a base-pattern summarizes the main properties of the columns/generators that are mapped onto it. It can be viewed as a seed for the generation of all the columns that are represented by it. To re-optimize a column, one can compute the best reduced cost column that can be obtained from its base-pattern, assuming that this re-optimization is much cheaper than solving the subproblem.

For example for the capacitated multi-item lot-sizing problem, the setup solution y to the discrete setup subproblem represents a base pattern for all associated production levels x. Then, re-optimizing x for a fixed y involves solving a shortest path subproblem. For the cutting stock problem, a cutting pattern defines a partition of the wide-roll where cut pieces can be replaced by an assortment of smaller items. The re-optimization can be heuristic.

Both examples call upon the concept of *exchange vectors*. An exchange vector is a valid perturbation: added to a feasible column, it gives rise to another feasible column (see Vanderbeck (2002) for a formal definition). Exchange vectors define dual cuts (Valério de Carvalho, 2002); handling exchange vectors implicitly through re-optimization is the equivalent of using a cutting plane procedure rather than setting all cuts a priori in the dual master formulation. It can be called after a few master simplex iterations and not wait until the restricted master LP is re-optimized.

The column generation literature includes discussions on what could be *dominant/redundant* columns (a review is given in Lübbecke and Desrosiers (2004)). Since columns for the master primal are cuts for its dual, one can turn to the well-defined concepts of redundant dual constraints versus facet defining constraints to characterize columns. A more pragmatic concept that can be imported in column generation is that of *lifting*: the idea is to apply post-processing to subproblem solutions to increase column coefficients in master constraints (assuming "$\geq$" constraints) to their maximum feasible value (feasibility being defined by the subproblem constraints).

A sequential lifting procedure consists in enumerating each master constraint i with zero dual price $\pi_i = 0$ and solve the auxiliary subproblem

$$\max\{A_i\,(x,y):\; B\,(x,y) \geq b, (c - \pi\,A)\,(x,y) \leq \zeta(\pi),\; x \geq 0, y \in I\!N\}$$

where $\zeta(\pi)$ is defined in (8). Alternatively, it can be formulated in terms of variables representing variations from the current subproblem solution. Lifting is a practical approach when this auxiliary subproblem is trivial: for instance, in the cutting stock application, it consists in filling-up the knapsack with zero profit items.

9 Primal heuristics

The column generation approach is also a natural setting for deriving good primal solutions to a combinatorial problem: the decomposition principle can be exploited in greedy, local search or math-programming based heuristics. The price coordination mechanism of the Dantzig-Wolfe approach brings the global view that may be lacking when local search or constructive heuristics are applied directly to the original problem formulation (these latter approaches are often qualified as myopic).

There are examples of decomposition based heuristics in the literature that range from the simplest to the latest heuristic paradigm: a standard heuristic is to initialize the master with a set of heuristically generated columns and to solve the MIP master problem restricted to that initial set. Alternatively, on can use the columns generated while solving the master LP (approximately or exactly) to define the restricted master IP. However, there is typically no guarantee that such a static restricted set of columns holds a feasible integer solution. Methods that dynamically generate further columns needed for the integer solution avoid such drawback.

A rounding heuristic can be implemented by iteratively selecting in the master LP solution the branching variable that is closest to its integer part and rounding it. Variables that are currently integer can be fixed first. It is important to use pre-processing to adapt the subproblem to the residual master problem. The definition of branching variable is not tuned to improving dual bounds as in Section 4: on can round λ_g variables as in Vanderbeck (1999) or the underlying original variables as in Elhedhli and Goffin (2002).

A generic greedy heuristic for the master consists in generating iteratively the columns that have the smallest ratio of cost per unit of constraint satisfaction, take it in the solution and reiterate for the residual master problem (this requires generating columns that are proper for the residual problem).

The local search paradigm can be implemented by exchanging columns in a restricted pool over which the restricted integer master program is solved (heuristically or exactly). Examples of meta-heuristic based on decomposition can also be found in the literature. Taillard (1999) proposed an Adaptive Memory Programming heuristic for the vehicle routing problem based on managing a pool of columns representing feasible routes: columns with a history of being present in good solutions are iteratively selected that are suitable for the residual problem; if no more columns are available, remaining customers are being introduced in an existing route if feasible or new routes are generated for them.

10 Automated reformulation

Implementations of column generation based solution methods that are reported in the literature have been developed for a specific application. These algorithms are building on general purpose features such as the ideas reviewed herein. They are therefore amenable to a generic solver for decomposable MIPs. If such generic solvers were not proposed, it is because of the belief that there remain mandatory manual steps in the process of implementing a column generation code: the user must define the reformulation, the form taken by a column and its reduced cost, the user must say how the subproblem is modified after branching; he must define the form of intermediate Lagrangian bounds for its problem. Once the user has all that defined then he can use the *tool-box* offered by Abacus (Thienel, 1995), BCP (Ladányi, Ralphs and Trotter, 1998), Maestro (Chabrier, 2003), or Minto (Savelsbergh and Nemhauser, 1991) to ease the implementation. Then, generic parts of the code can easily be adapted for re-use in other applications.

However, it is possible to automatize the reformulation, letting a code apply the Dantzig-Wolfe decomposition based on the original formulation and the user indication of what constraints must be dualized. All further modifications resulting from branching or adding cuts in the master for instance can be taken into account automatically too. Vanderbeck (2003) presents a framework for representing a MIP in an aggregate/implicit way from which the column generation reformulation can easily be generated automatically along with the subproblem and Lagrangian bound expressions. A prototype “black-box” column generation generic solver for MIP, named *BaPCod*, was developed along these principles. The underlying idea is simple: start with the most general form assumed by a decomposable MIP, write the formulas for its D-W reformulation, and code them. All simpler problems will be special cases.

Acknowledgment

The author received support from INRIA via a Collaborative Research Action (ARC) between INRIA Rhones-Alpes, HEC Geneva and the University of Bordeaux 1. We are very grateful for the patience of the anonymous referee in listing the typos of the original draft and making constructive remarks.

References

Briant, O., C. Lemaréchal, Ph. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck (2004). Comparison of bundle and classical column generation. *Working paper*, in preparation.

Briant, O. and D. Naddef (2004). The optimal diversity management problem. *Operations Research*, 52:515-526.

Caprara, A., M. Fischetti, and P. Toth (2000). Algorithms for the set covering problem. *Annals of Operations Research*, 98:308-319.

Chabrier, A. (2003). Génération de colonnes et de coupes utilisant des sous-problèmes de plus court chemin. *Thèse de doctotat*, Université d'Angers.

Cheney, E. and A. Goldstein (1959). Newton's method for convex programming and Tchebycheff approximations. *Numerische Math.*, 1:253-268.

Crowder, H.P., E.L. Johnson and M.W. Padberg (1983). Solving large-scale zero-one linear programming problems. *Operations Research*, 31: 803-834.

Desaulniers, G., J. Desrosiers and M.M. Solomon (2001). Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In C.C. Ribeiro and P. Hansen, editors. *Essays and surveys in metaheuristics*, Kluwer, Boston, pp309-324.

du Merle, O. and J.-Ph. Vial (2002). Proximal ACCPM, a cutting plane method for column generation and lagrangian relaxation: application to the p-median. HEC/Logilab *Technical Report* , 2002.23.

du Merle, O. , D. Villeneuve, J. Desrosiers and P. Hansen (1999). Stabilized column generation. *Discrete Math.*, 194, 229-237.

Elhedhli, S. and J.-L. Goffin (2002). Efficient Production-Distribution System Design. *Working paper*, McGill University, Canada.

Gamache, M., F. Soumis, G. Marquis and J. Desrosiers (1999). A column generation approach for large scale aircrew rostering problems. *Operations Research*, 47:247-263.

Geoffrion, A.M. (1974). Lagrangian relaxation for integer programming. *Mathematical Programming Studies*, 2:82-114.

Goffin, J.-L. and J.-Ph. Vial (2002). Convex non-differentiable optimization: a survey focused on the analytic center cutting plane method, *Optimization Methods and Software*, 17:805-867.

Guignard, M. (2004). Lagrangean Relaxation. In M. Resende and P. Pardalos, editors, *Handbook of Applied Optimization*, Oxford University Press.

Kelley, J. E. (1960). The cutting plane method for solving convex programs. *Journal of the SIAM*, 8:703-712.

Kim, S., K.N. Chang, and J.Y. Lee (1995). A descent method with linear programming subproblems for nondifferentiable convex optimization. *Mathematical Programming*, 71:17-28.

Ladányi, L. , T.K. Ralphs, and L.E. Trotter Jr. (1998). Branch, cut, and price: sequential and parallel. In M. Junger and D. Naddef, editors. *Computational Combinatorial Optimization*, Springer.

Lemaréchal, C. (1998). Lagrangian Relaxation. In M. Junger and D. Naddef, editors. *Computational Combinatorial Optimization*, Springer.

Lübbecke, M.E. (2001). Engine Scheduling by Column Generation. *PhD thesis*, Braunschweig University of Technology, Cuvillier Verlag, Göttingen.

Lübbecke, M.E. and J. Desrosiers (2004). Selected topics in column generation. To appear in *Operations Research.*

Marsten, R.E., W.W. Hogan and J.W. Blankenship (1975). The `BOXSTEP` method for large-scale optimization. *Operations Research*, 23: 389-405.

Martin, R.K. (1987). Generating alternative mixed-integer programming models using variable redefinition. *Operations Research*, 35(6):820-831.

Michel, S. (2003). Optimisation des tournées de véhicules avec gestion des stocks. *Rapport de stage* de DEA, Université Bordeaux 1.

Monneris, K. (2002). Problème de production par lots: méthode de branch-and-price. *Rapport de stage* de DEA, Université Bordeaux 1.

Neame, P.J. (1999). Nonsmooth dual methods in integer programing. *PhD thesis*, Depart. of Math. and Statistics, The University of Melbourne.

Nemhauser, G.L. and L.A. Wolsey (1988). *Integer and Combinatorial Optimization.* John Wiley & Sons, Inc.

Perrot, N. (2004). Advanced IP column generation strategies for the cutting stock stock problem and its variants. *PhD thesis*, University Bordeaux 1.

Poggi de Aragão, M. and E. Uchoa (2003). Integer program reformulation for robust branch-and-cut-and-price. In *Proceedings of the Conference Mathematical Program in Rio: A Conference in Honour of Nelson Maculan*, pp 56-61.

Rousseau, L.-M., M. Gendreau and D. Feillet (2003). Interior point stabilization for column generation. *Working paper.*

Ryan, D. M. and B. A. Foster (1981). An integer programming approach to scheduling. In A. Wren (editor), *Computer Scheduling of Public Tansport Urban Passenger Vehicle and Crew Scheduling*, North-Holland, Amsterdan, 269-280.

Savelsbergh, M.W.P. (1994). Preprocessing and probing techniques for mixed integer programming problems. *ORSA J. on Comp.*, 6:445-454.

Savelsbergh, M.W.P. and G.L. Nemhauser (1991). *Functional description of MINTO, a Mixed INTeger Optimizer.* COC-91-03A, Georgia Tech.

Savelsbergh, M.W.P. and M. Sol (1995). The general pickup and delivery problem. *Transportation Science*, 29(1):17-29.

Taillard, E.D. (1999). A heuristic column generation method for the heterogenous VRP. *Operations Research - Rech. Opération.*, 33, 1-14.

Thienel, S. (1995). ABACUS - A Branch-And-CUt System. *Doctoral Thesis*, Universität zu Köln.

Thizy, J.-M. (1991). Analysis of Lagrangian Decomposition for the Multi-Item Capacitated Lot-Sizing Problem. *INFOR*, 29(4), pp.271-283.

Valério de Carvalho, J. M. (2002). Using Extra Dual Cuts to Accelerate Column Generation. To appear in *INFORMS Journal on Computing.*

Vanderbeck, F. (1999). Computational Study of a Column Generation algorithm for Bin Packing and Cutting Stock problems *Math. Prog.* A, 86:565-594.

Vanderbeck, F. (2000). On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, Vol. 48, No. 1., pp111-128.

Vanderbeck, F. (2002). A Generic View at the Dantzig-Wolfe Decomposition Approach in Mixed Integer Programming, *Working Paper* no U-02.22, MAB, Université Bordeaux 1.

Vanderbeck, F. (2003). Automated Dantzig-Wolfe re-formulation or how to exploit simultaneously original formulation and column generation re-formulation. *Working Paper*, no U-03.24, MAB, Université Bordeaux 1.

van Vyve, M. (2003). A solution approach of production planning problems based on compact formulations of lot-sizing models. *PhD thesis*, Université Catholique de Louvain.

Vignac, B. (2003). Accélération de la convergence de l'algorithme de génération de colonne. *Rapport de stage* de DEA, Université Bordeaux 1.

Villeneuve, D., J. Desrosiers, M.E. Lübbecke and F. Soumis (2003). On Compact Formulations for Integer Programs Solved By Column Generation. To appear in Annals of Operations Research.

Wentges, P. (1997). Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming. *Int Trans. Opl. Res.*, 4(2), 151-162.