An Augment-and-Branch-and-Cut Framework for Mixed 0-1 Programming

Adam N. Letchford[1] and Andrea Lodi[2]

[1] Department of Management Science, Lancaster University, Lancaster LA1 4YW, England
A.N.Letchford@lancaster.ac.uk

[2] DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
alodi@deis.unibo.it

Abstract. In recent years the *branch-and-cut* method, a synthesis of the classical *branch-and-bound* and *cutting plane* methods, has proven to be a highly successful approach to solving large-scale integer programs to optimality. This is especially true for *mixed 0-1* and *pure 0-1* problems. However, other approaches to integer programming are possible. One alternative is provided by so-called *augmentation* algorithms, in which a feasible integer solution is iteratively improved (augmented) until no further improvement is possible.
Recently, Weismantel suggested that these two approaches could be combined in some way, to yield an *augment-and-branch-and-cut* (ABC) algorithm for integer programming. In this paper we describe a possible implementation of such a finite ABC algorithm for mixed 0-1 and pure 0-1 programs. The algorithm differs from standard branch-and-cut in several important ways. In particular, the terms *separation*, *branching*, and *fathoming* take on new meanings in the primal context.

1 Introduction

One of the most successful methods for solving Integer and Mixed-Integer Linear Programs (ILPs and MILPs, respectively) is the *branch-and-cut* approach, in which strong cutting planes are used to strengthen the linear programming relaxations at each node of a branch-and-bound tree (see Padberg & Rinaldi [22]; Caprara & Fischetti [5]). Branch-and-cut is currently very popular because it appears to be much more robust than the use of either cutting planes or branching in isolation.

However, although branch-and-cut is popular, research has been and still is being conducted into other approaches to integer programming — based on Lagrangian, surrogate or group relaxation, lattice basis reduction, test sets, and so on (see, e.g., Nemhauser & Wolsey [19]). Of particular interest in the present paper are so-called *augmentation* algorithms, in which a feasible solution is iteratively improved (augmented) until no further improvement is possible (and it can be proved that this is the case). Some recent results on augmentation algorithms can be found in Firla et al. [8], Haus et al. [14], Schulz et al. [23], Thomas [24] and Urbaniak et al. [25].

Recently, Weismantel [26] suggested the possibility of somehow combining elements of augmentation and branch-and-cut algorithms, to yield an *augment-and-branch-and-cut* algorithm for integer programming — with the convenient acronym ABC. In this paper we describe such an ABC algorithm for *mixed 0-1* programs. It is based on a new primal cutting plane algorithm which we presented in [17], combined with some *branching rules* which are specifically tailored to the primal context.

The remainder of the paper is structured as follows. In Section 2 we briefly review the literature on branch-and-cut methods. In Section 3 we review the literature on primal cutting plane methods, including our new algorithm. In Section 4, we examine the issue of *branching* in the primal context. In Section 5 we show how the various components of the algorithm are integrated to give the general-purpose ABC method for mixed 0-1 programs. Preliminary computational results are reported in Section 6, while conclusions are given in Section 7.

2 Basic Concepts of Branch-and-Cut

In this section we briefly review the literature on standard branch-and-cut algorithms. Although much of this material is widely known, it is necessary to include it here so that in subsequent sections we can show how our primal ABC approach departs from the standard approach.

Suppose that a MILP has n integer-constrained variables, p continuous variables and m linear constraints. The vector of integer-constrained variables will be denoted by x and the vector of continuous variables by y. Let us suppose that the MILP takes the form:

$$\max\{c^T x + d^T y : Ax + Gy \leq b, x \in \mathbb{Z}^n_+, y \in \mathbb{R}^p_+\},$$

where c and d are objective coefficient vectors, A and G are matrices of appropriate dimension ($m \times n$ and $m \times p$, respectively) and b is an m-vector of right hand sides.

The feasible region of the *linear programming* (LP) *relaxation* of the MILP is the polyhedron

$$P := \{(x, y) \in \mathbb{R}^{n+p}_+ : Ax + Gy \leq b\},$$

and the convex hull of feasible MILP solutions is the polyhedron

$$P_I := \text{conv}\{x \in \mathbb{Z}^n_+, y \in \mathbb{R}^p_+ : Ax + Gy \leq b\}.$$

We have $P_I \subseteq P$ and in this paper we assume that containment is strict. For simplicity we also assume throughout the paper that the MILP is a *maximization* problem.

Suppose that the vector (x^*, y^*) is an optimal solution to the LP relaxation. If x^* is integral, then (x^*, y^*) is an optimal solution to the MILP and we are done. If not, then the value of the objective function provides an upper bound on the value of the optimum, but further work will be needed to solve the MILP to optimality. We can either *cut* (add extra inequalities) or *branch* (divide the problem into subproblems).

2.1 Cutting

A *cutting plane* (or *cut*) is an extra linear inequality which (x^*, y^*) does not satisfy, but which is satisfied by all solutions to the MILP. That is, the inequality must be valid for P_I but not for P. If we can find a cut, then we can add it to the LP relaxation and re-solve (typically via the dual simplex method), to obtain a new (x^*, y^*). If the new x^* is integral, we are done, otherwise the procedure of adding cuts continues until x^* becomes integral. As cuts are added, we obtain a non-increasing sequence of upper bounds.

In order to generate a cutting plane, one faces the following problem:

The Separation Problem: *Given some $(x^*, y^*) \in P$, find an inequality which is valid for P_I and violated by (x^*, y^*), or prove that none exists.*

A famous theorem of Grötschel, Lovász & Schrijver [13] states that, under certain technical assumptions, the separation problem is $\mathcal{NP}$-hard if and only if the original MILP is. However, if (x^*, y^*) is an extreme point of the current LP relaxation, which will be the case if the simplex method is being used, then cuts can be generated fairly easily. (For example, one can use the cuts of Gomory [11], [12]; or the disjunctive cuts of Balas, Ceria & Cornuéjols [1].)

In general, cutting plane algorithms based on 'general-purpose' cuts such as Gomory or disjunctive cuts exhibit slow convergence. This can be alleviated somewhat by adding several cuts in one go before reoptimizing by dual simplex, see for example Balas, Ceria & Cornuéjols [1] and Balas, Ceria, Cornuéjols & Natraj [3]. In general, however, it is preferable to use inequalities which take problem structure into account, especially inequalities which are 'deep' in the sense of inducing facets (or faces of high dimension) of the polyhedron P_I (see Padberg & Grötschel [20]; Nemhauser & Wolsey [19]).

It is often the case that several classes of deep inequalities are known for a given problem, and frequently each class contains an exponential number of members. To use a particular class of inequalities in practice, one needs to solve the following modified separation problem:

The Separation Problem for a Class of Inequalities: *Given some class $\mathcal{F}$ of inequalities which are valid for P_I and some $(x^*, y^*) \in P$, find a member of $\mathcal{F}$ which is violated by (x^*, y^*), or prove that none exists.*

It frequently happens that this modified separation problem is polynomially solvable for some classes of inequalities, and $\mathcal{NP}$-hard for others. Yet, even in the latter case it is frequently possible to devise heuristics for separation which perform reasonably well in practice.

Cutting plane algorithms based on deep inequalities typically converge much more quickly than algorithms based on general-purpose cuts. However, there is one disadvantage: due to the heuristic nature of the separation algorithms, or due to the lack of a complete description of P_I, it may happen that no deep cuts can be found even though x^* is still fractional. (This never happens with

Gomory or disjunctive cuts.) If we want to solve the MILP to optimality, and if we want to avoid using general-purpose cuts, then we will need to branch as described in the next subsection (or use some other solution technique).

Note that the cutting plane procedure yields a (typically good) upper bound on the optimum.

2.2 Branching

Instead of adding cuts to remove an invalid vector (x^*, y^*), one can *branch* instead, i.e., divide the original problem into subproblems. The most common method of branching is to choose an index i such that x_i^* is fractional and to create two subproblems. In one, the constraint $x_i \leq \lfloor x_i^* \rfloor$ is added; in the other, the constraint $x_i \geq \lceil x_i^* \rceil$ is added. (Here, $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote rounding down and rounding up to the nearest integer, respectively.) In this way, x^* is excluded from each of the two branches created. Each of these subproblems can be quickly solved using the dual simplex method.

In the standard *branch-and-bound* method, often attributed to Land & Doig [15], recursive branching leads to a tree-like structure of subproblems. (In the case of mixed 0-1 programs, for example, a given node of the branch-and-bound tree will correspond to fixing a specified subset of the variables to zero and another specified subset of the variables to one.) The tree is then explored, e.g., by breadth-first or depth-first search. Along the way, branches are 'pruned' (removed from consideration) when their associated LP upper bound is lower than the current lower bound (corresponding to the best integer solution found so far). The algorithm terminates when the only remaining node is the root node of the tree.

Note that in general the set of variables for which x_i^* is fractional may be large, and therefore some kind of heuristic rule is needed for choosing the branching variable. A common method is to choose the variable whose fractional part is closest to one-half, but other rules can perform better, particularly if they take specific problem structure into account.

2.3 The Overall Scheme

The two methods of cutting and branching are in a sense complementary. Cutting planes, especially deep ones, can lead to very tight upper bounds, but may not yield a feasible solution for a long time. Branching, on the other hand, allows one to find feasible solutions relatively quickly, but the upper bounds may be too weak to limit the size of the search tree.

The natural solution is to integrate cutting and branching within a single algorithm. This yields the *branch-and-cut* technique, in which cutting planes are used at *each node* of the branch-and-bound tree to strengthen the LP relaxations (see Padberg & Rinaldi [22]; Balas, Ceria & Cornuéjols [2]; Balas, Ceria, Cornuéjols & Natraj [3]; Caprara & Fischetti [5]).

In branch-and-cut it is normal to use inequalities which are valid for P_I as cutting planes. Inequalities of this kind are valid *globally*, i.e., at every node of the branch-and-cut tree. Thus, any violated inequality generated at any node may be used to strengthen the upper bound at any other node of the tree (assuming of course that it is violated). This would not be possible with cuts which are only valid at a particular node of the tree.

There are a few more ingredients which are needed to form a sophisticated branch-and-cut algorithm. We have already mentioned rules for selecting the branching variable, but there are other key components such as pricing, variable fixing, cut pools, and so on (Padberg & Rinaldi [22]). In Section 5 we briefly review these and consider how to adapt them to the primal context.

3 Primal Cutting Plane Algorithms

3.1 The Basic Concept

The goal of a primal cutting plane algorithm is to iteratively move from one corner of P_I to a better one until no further improvement is possible. To begin the method, one must know of a feasible solution to the ILP, preferably a good one, which we will denote by $(\bar{x}, \bar{y})$. Moreover, it must be a solution with a very specific property: it must be an extreme point of both P and P_I. Equivalently, it must be a basic feasible solution to the LP relaxation of the problem.

If no such $(\bar{x}, \bar{y})$ is known, then artificial variables must be used to find one, much as in the ordinary simplex method. However, in practice it is often easy to find one. Indeed, for mixed 0-1 problems, all we need to do is find *any* feasible solution to the MILP: it can be converted into a basic feasible solution by fixing the x variables at their given values and solving an LP to determine the y variables.

Given a suitable $(\bar{x}, \bar{y})$, one then constructs the associated simplex tableau. If $(\bar{x}, \bar{y})$ is dual feasible, it is optimal and the method stops. If not, a primal simplex pivot is made, leading to a new vector (or the same one in the case of degeneracy), which we will denote by (x^*, y^*). If x^* is integral, then we have found an improved MILP solution, and (x^*, y^*) becomes the new $(\bar{x}, \bar{y})$. If on the other hand it is fractional, then a cutting plane is generated which cuts off (x^*, y^*). Then, another attempt is made to pivot from $(\bar{x}, \bar{y})$, leading to a different (x^*, y^*), and so on. The method terminates when $(\bar{x}, \bar{y})$ has been proved to be dual feasible.

In fact, it is possible to compute (x^*, y^*) from the information in the $(\bar{x}, \bar{y})$ tableau without actually performing a pivot. Therefore it is not actually necessary to pivot to (x^*, y^*) in order to determine if it is a feasible MILP solution or not. For this reason, some primal cutting plane algorithms only perform the pivot explicitly when it leads to an augmentation.

3.2 Algorithms from the 1960s

Several primal cutting plane algorithms, for pure ILPs only, appeared in the 1960s (Ben-Israel and Charnes [4], Young [27], [28], Glover [10]). The algorithms in [4] and [27] are extremely complicated and, as shown in [28] and [10], they can be simplified considerably without affecting convergence. The simplest of these algorithms, and in our view the best one, is that of Young [28].

Detailed descriptions of the Young algorithm can be found in Garfinkel & Nemhauser [9], in our paper [17], and also in Firla et al. [8]. The basic idea behind Young's algorithm is to begin with an all-integer tableau and to perform primal simplex pivots whenever the pivot element is equal to 1. In this way one guarantees that the subsequent tableau (and the associated x^*) is also all-integer. If the pivot element is not equal to 1, then a cutting plane is added to the tableau so that the pivot element in the enlarged tableau *is* 1.

Young proved that, with an appropriate lexicographic variable selection rule, his algorithm is finitely convergent. However, the performance of the algorithm in practical computation has been disappointing. The main problem is that one often encounters extremely long sequences of degenerate pivots, without either augmenting or proving dual feasibility.

3.3 Padberg and Hong's Algorithm

The key to obtaining a viable primal cutting plane algorithm is the use of strong (preferably facet-inducing) cutting planes. To our knowledge the first authors to use strong cutting planes in a primal context were Padberg & Hong [21]. (A detailed description of this paper appears in Padberg & Grötschel [20] and in our paper [17].) Padberg and Hong implemented a primal cutting plane algorithm for the *Travelling Salesman Problem* (TSP), based on facet-defining cuts such as *subtour elimination constraints* (SECs) and *2-matching*, *comb* and *chain* inequalities.

The algorithms used by Padberg and Hong to generate violated inequalities — which they called *constraint identification algorithms* — are very similar to what are now known as separation algorithms. However, there was a subtle difference: as well as requiring the inequality to be violated by x^*, they also required that it be *tight* (satisfied as an equation) at $\bar{x}$. The reason for this is that, in the case of 0-1 problems like the TSP, only inequalities which are tight at $\bar{x}$ can help in either augmenting or proving dual feasibility of $\bar{x}$.

Therefore the problem solved by Padberg and Hong's algorithms is as follows (see also [17], [16], [7]):

The Primal Separation Problem for a Class of Inequalities: *Given some class $\mathcal{F}$ of inequalities which are valid for P_I, some $(x^*, y^*) \in P$ and some $(\bar{x}, \bar{y})$ which is an extreme point of both P and P_I, find a member of $\mathcal{F}$ which is violated by (x^*, y^*) and tight for $(\bar{x}, \bar{y})$, or prove that none exists.*

It is easy to show (see Padberg & Grötschel [20]) that a given primal separation can be transformed to the equivalent standard (dual) version. However, the reverse does not hold in general and in the papers Letchford & Lodi [16] and Eisenbrand, Rinaldi & Ventura [7] it is shown that for many classes of inequalities, primal separation is substantially *easier* than standard separation. This gives some encouragement for pursuing the primal approach, especially for (mixed) 0-1 problems.

3.4 Our Algorithm

In our recent paper [17] we argued that primal cutting plane algorithms were worthy of more attention, and we proposed a generic algorithm for the case of mixed 0-1 problems based on primal separation algorithms. For the sake of brevity we do not describe this in detail here, but the basic structure of the algorithm is as follows:

- Step 1: Find a 'good' initial basic feasible solution $(\bar{x}, \bar{y})$ and construct an appropriate tableau.
- Step 2: If $(\bar{x}, \bar{y})$ is dual feasible, stop.
- Step 3: Perform a primal simplex pivot. If it is degenerate, return to step 2.
- Step 4: Let (x^*, y^*) be the new vector obtained. If (x^*, y^*) is a feasible solution to the MILP, set $(\bar{x}, \bar{y}) := (x^*, y^*)$ and return to step 2.
- Step 5: Call primal separation for known strong inequalities. If any are found, pivot back to $(\bar{x}, \bar{y})$, add one or more cuts to the LP, and return to step 3.
- Step 6: If x^* is integral but (x^*, y^*) is infeasible, call the 'special' cut generating procedure (described below) and return to step 3.
- Step 7: Generate one or more general-purpose cuts (such as Gomory fractional or mixed-integer cuts), add them to the LP, pivot back to $(\bar{x}, \bar{y})$ and return to step 3.

The reason that step 6 is necessary is that we do not require that the entire constraint system $Ax + Gy \leq b$ be present in the initial LP. (For some problems, the constraint system is too large to be handled all at once.) Therefore it is theoretically possible to obtain a vector (x^*, y^*) which has an integral x component, but which is not a feasible MILP solution because it violates a constraint in the system $Ax + Gy \leq b$ which is *not tight* at $(\bar{x}, \bar{y})$.

The 'special' cut generating procedure to handle this exceptional case is as follows:

- 6.1. Find an inequality in the system $Ax + Gy \leq b$ which is violated by (x^*, y^*) yet *not tight* at $(\bar{x}, \bar{y})$, and add it to the LP.
- 6.2. Perform a dual simplex pivot to arrive at a new point $(\hat{x}, \hat{y})$ which is a convex combination of $(\bar{x}, \bar{y})$ and (x^*, y^*). Set $(x^*, y^*) := (\hat{x}, \hat{y})$.
- 6.3. Generate a Gomory mixed-integer cut from a row of the tableau which corresponds to a fractional structural variable and add it to the LP. (We show in [17] that it is guaranteed to be tight at $(\bar{x}, \bar{y})$.)
- 6.4. Perform a dual simplex pivot to return to $(\bar{x}, \bar{y})$ and discard the inequality generated in step 6.2 (which is no longer tight).

In a sense, the mixed-integer cut generated in step 6.3 is a 'rotated' version of the inequality generated in step 6.1 — rotated in such a way as to provide a solution to the primal separation problem.

Computational results given in [17] clearly demonstrate that this algorithm is significantly better than that of Young. Nevertheless, it seems desirable to branch in step 7 instead of adding general-purpose cuts. That is our goal in the next section.

4 Branching in a Primal Context

Branching is desirable in ordinary branch-and-cut when (x^*, y^*) has a fractional x component, but the separation algorithms fail to find any violated cuts. In our primal approach, branching is desirable when $(\bar{x}, \bar{y})$ is not dual feasible, the adjacent point (x^*, y^*) has a fractional x component, and the *primal* separation algorithms fail.

However, branching is problematic in the primal context. Suppose one tried to branch in the standard way, by picking a variable index i such that x_i^* is fractional, and imposing either $x_i = 0$ or $x_i = 1$. Then the current best MILP solution $(\bar{x}, \bar{y})$ would be excluded from one of the two subproblems, and we would lose our starting basis on one of the two branches. Therefore, just as a non-standard separation problem appears in the primal context, a non-standard form of branching is also needed. One wants to branch in such a way that (x^*, y^*) is removed, but $(\bar{x}, \bar{y})$ remains intact on both branches.

We have developed suitable branching rules for 0-1 MILPs, which we now describe.

First, let us assume for simplicity of notation that the 0-1 variables have been complemented so that $\bar{x}$ is a vector of n zeroes, which we denote here by $\mathbf{0}$. Moreover, let us also assume that to save on memory our LP contains only constraints which are tight for $(\bar{x}, \bar{y})$, together with upper bounds of 1 on the x variables. (Our cutting plane algorithm described in Subsection 3.4 is designed to run in this way.) We begin with a fairly naive branching rule:

Naive Branching Rule: *Suppose that there are a pair of variable indices i and j such that $0 < x_i^* < x_j^* \leq 1$. Create two branches, one with the constraint $x_i = 0$ added; the other with the constraint $x_i \geq x_j$ added.*

With this rule, no feasible solution is lost, $(\bar{x}, \bar{y})$ is feasible for both branches, and (x^*, y^*) is removed in both branches. One drawback however is that any feasible solution (x, y) which satisfies $x_i = x_j = 0$ will be valid on both branches. It would be more desirable to branch in such a way that no feasible solution, apart from $(\bar{x}, \bar{y})$ itself, appeared on both branches.

Let us consider the issue in more detail. The goal of branching is to force a variable x_i to be integral, even though x_i^* is currently fractional. An ideal branching rule would therefore be $(x_i = 0) \vee (x_i = 1)$, but, as we have seen, this would make $(\bar{x}, \bar{y})$ infeasible on the 'up'-branch. So, suppose that the current

LP feasible region (including any cutting planes added) is of the form: $\{x \in [0,1]^n, y \in \mathbb{R}^p_+ : \bar{A}x + \bar{G}y \leq \bar{b}\}$, where all of the inequalities in the system $\bar{A}x + \bar{G}y \leq \bar{b}$ are currently tight for $(\bar{x}, \bar{y})$. If we *had* imposed $x_i = 1$, then the resulting feasible region would be

$$P^1 := \{x \in [0,1]^n, y \in \mathbb{R}^p_+ : \bar{A}x + \bar{G}y \leq \bar{b}, x_i = 1\}.$$

Given that we do not want to remove $(\bar{x}, \bar{y})$, we consider instead the smallest polyhedron containing both $(\bar{x}, \bar{y})$ and P^1. It is not difficult to show that, when $\bar{x} = \mathbf{0}$, this polyhedron is:

$$P^2 := \text{conv}\{\mathbf{0} \cup P^1\} = \{x \in [0,1]^n, y \in \mathbb{R}^p_+ : \bar{A}x + \bar{G}y \leq \bar{b}, x_j \leq x_i \ \forall j \neq i\}.$$

By definition, this new polyhedron has no extreme points in which x_i is fractional. Moreover, the next time we attempt to pivot from $(\bar{x}, \bar{y})$, we will arrive at a point (x^*, y^*) with $x^*_i = 1$.

Therefore our main branching rule for 0-1 MILPs is as follows:

> **Main Branching Rule:** *Suppose that* $0 < x^*_i < 1$. *Create two branches, one with the constraint* $x_i = 0$ *added; the other with the constraints* $x_j \leq x_i \ \forall j \neq i$ *added.*

Note that $(\bar{x}, \bar{y})$ remains basic after the change.

With this branching rule, just as with the original one, no feasible solution is lost, $(\bar{x}, \bar{y})$ is feasible for both branches, and (x^*, y^*) is removed in both branches. However, in addition, the only vectors which can appear on both branches are those with x component equal to $\mathbf{0}$. This is therefore a more powerful partition, which we would expect to lead to quicker convergence of the algorithm.

Let us consider what happens when several branchings have occurred. Suppose that we wish to 'fix' variables x_i for $i \in N_0$ to zero, and variables x_i for $i \in N_1$ to one. The polyhedron of interest is now the convex hull of $(\bar{x}, \bar{y})$ and the set $\{x \in [0,1]^n, y \in \mathbb{R}^p_+ : \bar{A}x + \bar{G}y \leq \bar{b}, x_i = 0 \ (i \in N_0), x_i = 1 \ (i \in N_1)\}$. By a similar argument to that given above it can be shown that, when $\bar{x} = \mathbf{0}$, the polyhedron in question is

$$\begin{aligned} P^3 := \{x \in [0,1]^n, y \in \mathbb{R}^p_+ : \ & \bar{A}x + \bar{G}y \leq \bar{b}, \\ & x_j = 0 \ \forall j \in N_0, \\ & x_j = x_i \ \forall j \in N_1 \setminus \{i\}, \\ & x_j \leq x_i \ \forall j \notin (N_0 \cup N_1)\}, \end{aligned} \tag{1}$$

where i is an arbitrary index in N_1.

That is, in order to perform further branching in an 'upward' direction it is merely necessary to add equations of the form $x_j = x_i$. Thus the system of inequalities can be easily modified as branching progresses.

Note that the very compact and clean form of polyhedron P^3 is possible because all the constraints in the LP before the branching operation are tight at

$(\bar{x}, \bar{y})$. If it is desired to include non-tight constraints in the LP, then it is still possible to perform the above branching but the non-tight inequalities require some additional 'work'.

Note also that the same kind of argument does not apply to general MILPs (i.e., MILPs in which the integer variables are not restricted to be binary). The branching rule implicitly relies on the fact that any 0-1 vector $\bar{x}$ is an extreme point of the unit hypercube. It is this property which enables us to complement variables in order to get $\bar{x} = \mathbf{0}$. There is no analogous complementation procedure in the case of general MILPs.

To close this section, we must mention one potential problem. If only constraints which are tight are included in the LP (along with the upper bounds on the x variables), then there is a (small) risk that the LP will be *unbounded.* If this happens, however, it will be because the profit can be increased without limit by changing $\bar{y}$ while leaving $\bar{x}$ unchanged. The solution to this is to solve a (typically small) LP to see if it is possible to augment by changing only the y component. If so, then one should augment and continue from there.

5 The Overall ABC Algorithm

At this point we have the main ingredients for the ABC algorithm: the primal separation component and the branching rules. However, some more details are necessary in order to specify how the overall algorithm works.

Fathoming of Nodes: Obviously we need some way of pruning the branching tree and, in particular, of fathoming a node. It is not difficult to see, from the properties of the primal simplex method, that in ABC a node can be fathomed when $(\bar{x}, \bar{y})$ is dual feasible.

Cut Pool: In order to keep the size of the basis small, it is normal in standard branch-and-cut to delete inequalities from the LP whenever their slack exceeds some small positive value. However, to avoid wasting time by separating the same inequality more than once, it is common to store these constraints in a so-called *cut pool* (e.g., Padberg & Rinaldi [22]). The natural primal analogue of this is as follows: tight cuts are kept in the LP and non-tight cuts are kept in the pool. Whenever $(\bar{x}, \bar{y})$ is augmented, constraints which are no longer tight can be put into the pool and any constraints in the pool which have become tight can be put into the LP.

Handling Augmentations: At first sight it would appear that every time an improved feasible solution $(\bar{x}, \bar{y})$ is found, it will be necessary to discard the branch-and-cut tree and begin branching and cutting from scratch. In fact, this is *not* necessary. It is possible to work with a single tree. When a node is fathomed, it means that no feasible solution exists with objective value greater than $(\bar{x}, \bar{y})$ when the associated variables are fixed. Given that the new $(\bar{x}, \bar{y})$ has a greater objective value than the old one, this remains true after the augmentation. Hence, it is necessary only to construct a new basis at the root node, which can be done using the cuts which are now tight at the new $(\bar{x}, \bar{y})$.

Pricing: When n is very large, it is normal practice in standard branch-and-cut to begin with only a subset of the variables and to include other variables only when needed. This is done by *pricing*, i.e., computing the LP reduced costs of the remaining variables and adding the variables whose reduced costs are positive (Padberg & Rinaldi [22]). This can be done in the primal approach as well.

Handling Problems with m Huge: As mentioned, for many important problems, the number of constraints m needed to define P is exponential in the problem input, but optimization over P is still possible because the (standard) separation problem for these constraints is solvable in polynomial time. These problems can be dealt with in the primal context also, because (as explained in Section 4) we only keep tight constraints in the LP.

Finally, the reader will have noticed that up to now there has been no mention of upper bounds in the ABC context. This is because, strictly speaking, they are not needed: to fathom a node of the tree, it is only necessary to prove dual feasibility. Nevertheless, there are reasons for thinking that some kind of upper bounding mechanism might be desirable. The main one is this: if for some reason the ABC algorithm has to be interrupted before optimality has been achieved, then an upper bound can be used to assess the quality of the final $(\bar{x}, \bar{y})$.

The simplest way to produce an upper bound, based on the idea of Padberg and Hong, is to solve the final LP at the root node to optimality. This LP can be solved in a relatively small number of primal simplex pivots, because $(\bar{x}, \bar{y})$ can be used as a starting basis. However, note that the resulting upper bound is unlikely to be better than the upper bound which would be obtained from a dual approach (assuming that similar inequalities and separation algorithms are used).

Another reason for wanting an upper bounding mechanism is to somehow eliminate variables from the problem entirely. In standard branch-and-cut, this is done as follows. Any variable with a reduced cost greater than the difference between the current upper and lower bounds may be eliminated from the problem (i.e., fixed at zero). This is called *reduced cost fixing*. In the ABC context we can do something similar, at least at the root node, by using the reduced costs from the optimal solution to the LP relaxation. However, again our feeling is that this might lead to less powerful fixing than is achievable in the dual context.

6 Preliminary Computational Results

We implemented a first version of an ABC algorithm as described in the previous sections. For the LP solution, we used the CPLEX 7.0 callable library of ILOG.

We tested the algorithm on the same set of 50 *multi-dimensional 0-1 knapsack* instances we already considered in [17] and [18]. Specifically, the problems are of the form $\max\{c^T x : Ax \leq b, x \in \{0,1\}^n\}$, where $c \in Z_+^n$, $A \in Z_+^{m \times n}$ and $b \in Z_+^m$, which were randomly generated as follows. For any pair (n, m) with $n \in \{5, 10, 15, 20, 25\}$ and $m \in \{5, 10\}$, we constructed 5 random instances whose objective function coefficients are integers generated uniformly between 1 and 10. Moreover, for the instances with $m = 5$, the left-hand side coefficients are also integers generated uniformly between 1 and 10, while for the instances with $m = 10$, the left-hand side coefficients have a 50% chance of being an integer generated uniformly between 1 and 10, but also have a 50% chance of being zero. That is, these instances are sparse. In all cases the right hand side of each constraint was set to half the sum of the left hand side coefficients[1].

Cutting. For the cutting part of the algorithm we used the same policy developed in [17]: we generate primally violated *lifted cover* inequalities, heuristically separated as described in [17] since their separation has been shown to be NP-hard (see, [16] for details), and when we are not able to find any of them we resort to generating a round of Gomory fractional cuts strengthened as in Letchford & Lodi [18]. After 25 consecutive rounds of Gomory fractional cuts, in order to avoid numerical problems, we branch.

Branching. The branching tree is explored in a *depth-first* search. Differently from what we described in Section 4 we do not complement the current $\bar{x}$, thus we distinguish between a *left-branch*, which is fixing a variable to the same value assumed in the incumbent solution, and a *right-branch* which corresponds to implicitly imposing the other value through the addition of a set of inequalities as described in Section 4.

There are two interesting things to point out. First, the choice of exploring the tree in depth-first manner implies that just two sets must be maintained during the search: we call N_{left} (resp. N_{right}) the set of the variables which are fixed according to (resp. at the contrary of) their value in the incumbent solution. These sets correspond to sets N_0 and N_1 of (1), and an augmentation is simply handled by moving the variables which are currently contained in N_{right} to N_{left} (and, obviously, by manipulating some of the constraints added in the right-branches). Second, as alluded to in Section 4, only in the case of the *first* right-branch a set of constraints must be added, and specifically $n - |N_{left}| - 1$ constraints. In the following right-branches, it is enough to change an **inequality** (previously introduced in the first right-branch) into an **equality**. A straightforward example of this behavior is the following.

Example. Assume that the incumbent solution is such that $\bar{x}_i = 1$ and $\bar{x}_j = 0$. Since the first right-branch, at node h, has been performed on variable x_i, a constraint $x_j + x_i \leq 1$ has been added at node h. If at node k (for which node h is a father), we want to explore the right-branch associated to variable x_j, it is enough to transform the previously added constraint into $x_j + x_i = 1$.

[1] This is well-known to lead to non-trivial instances of the multi-dimensional 0-1 knapsack problem.

Other implementation details and further advances will be discussed in following studies. Concerning the results on the multi-dimensional knapsack instances, they are reported in Table 1.

Table 1. ABC algorithm. Preliminary results on multi-dimensional knapsack instances.

m	n	Aug.	nodes to Opt.	nodes	primal LCIs	overall cuts
	5	2.0	1.0	1.0	4.4	5.2
	10	4.8	1.0	1.0	7.8	16.8
5	15	8.6	26.4	48.6	25.2	736.0
	20	10.0	2.4	189.4	42.6	2503.2
	25	13.8	682.0	765.0	69.0	2196.6
	5	0.8	0.6	1.0	5.0	5.0
	10	4.6	1.0	1.0	9.0	10.0
10	15	7.0	1.8	8.6	13.0	172.6
	20	10.0	8.6	97.4	34.0	669.6
	25	13.2	597.8	1521.0	77.0	8815.0

Table 1 reports for each pair (m, n), the average results on 5 instances of the number of augmentation performed by the algorithm (Aug.) starting by the trivial solution with all the variables set to 0, the number of branching nodes to find the optimal solution (nodes to Opt.), and the number of branching nodes to prove optimality (nodes). The last two columns of the table refer to cuts by reporting the average number of cuts added (overall cuts) and the average number of primal lifted cover inequalities separated (primal LCIs).

With respect to the algorithm outlined in Section 3.4, we resort to generating rounds of primally-violated Gomory fractional cuts as soon as primal separation fails, and not only when an integer infeasible point is encountered. Each round of Gomory cuts contains at most 25 cuts tight to $\bar{x}$. Moreover, step 1. of the algorithm above is disregarded, in the sense that we start with the trivial 0-solution, and we do not apply during the search any heuristic to improve the current solution.

The results obtained show that an augment-and-branch-and-cut algorithm is a viable way of solving 0-1 ILPs (and MILPs) provided that all the sophisticated techniques developed for standard branch-and-cut algorithms were also implemented in this context. Indeed, a comparison with a general-purpose branch-and-cut framework like CPLEX 7.0 is totally unfair at the moment due to the much larger arsenal of cuts and to the great level of software engineering development of the current version of CPLEX. Just to give an idea, however, by avoiding in CPLEX presolve, primal heuristic, and cut generation but including cover and Gomory fractional inequalities and performing a depth-first search, the average number of nodes required for the 5 instances with $m = 5$ and $n = 25$ is 80.2 with respect to 765.0 reported in Table 1, i.e., almost ten times fewer.

Finally, we also preliminary tested our ABC implementation on three of the very famous instances proposed by Crowder, Johnson & Padberg [6], namely `p0033`, `p0040` and `p0201`. For these instances we start by the first integer solution given by CPLEX. The algorithm works quite well on the two smallest instances: it is able to prove optimality for the starting solution of `p0040` without any branching and just 2 primal cuts, while the initial solution of `p0033` is augmented twice and solved with 367 nodes. On `p0201`, instead, degeneracy becomes a severe problem. More than 95% of the time is spent performing degenerate pivots, and in this situation we resort to branching so that the number of nodes becomes huge. This suggests that the method could be improved by some form of anti-stalling device, or by periodically 'purging' the LP of unnecessary non-binding constraints. Further progress on this issue will be discussed in future studies.

7 Conclusion

We have examined how to perform separation and branching within the primal context and we have seen that, just as Weismantel suggested, it is possible to integrate augmentation, branching and cutting within a single framework, at least for (mixed) 0-1 problems. We have also shown that most of the components of a sophisticated branch-and-cut algorithm have a primal counterpart. Moreover, we have implemented and computationally tested the first version of an ABC algorithm which is a completely new approach to integer programming. The effectiveness of such an approach clearly needs to be proved on harder instances and future (actually, current) work will be devoted to obtaining sophisticated ABC algorithms and ad hoc implementations for specific classes of problems (e.g., for the TSP).

References

1. E. Balas, S. Ceria & G. Cornuéjols (1993) A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Math. Program.* 58, 295–324.
2. E. Balas, S. Ceria & G. Cornuéjols (1996) Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Mgt. Sci.* 42, 1229–1246.
3. E. Balas, S. Ceria, G. Cornuéjols & N. Natraj (1996) Gomory cuts revisited. *Oper. Res. Lett.* 19, 1–9.
4. A. Ben-Israel & A. Charnes (1962) On some problems of diophantine programming. *Cahiers du Centre d'Études de Recherche Opérationelle* 4, 215–280.
5. A. Caprara & M. Fischetti (1997) Branch-and-Cut Algorithms. In M. Dell'Amico, F. Maffioli & S. Martello (eds.) *Annotated Bibliographies in Combinatorial Optimization*, pp. 45-64. New York, Wiley.
6. H. Crowder, E.L. Johnson & M.W. Padberg (1983) Solving large-scale zero-one linear programming problems. *Oper. Res.* 31, 803–834.
7. F. Eisenbrand, G. Rinaldi & P. Ventura (2001) 0/1 primal separation and 0/1 optimization are equivalent. *Working paper*, IASI, Rome.
8. R.T. Firla, U.-U. Haus, M. Köppe, B. Spille & R. Weismantel (2001) Integer pivoting revisited. *Working paper*, Institute of Mathematical Optimization, University of Magdeburg.

9. R.S. Garfinkel & G.L. Nemhauser (1972) *Integer Programming.* New York: Wiley.
10. F. Glover (1968) A new foundation for a simplified primal integer programming algorithm. *Oper. Res.* 16, 727–740.
11. R.E. Gomory (1958) Outline of an algorithm for integer solutions to linear programs. *Bulletin of the AMS* 64, 275–278.
12. R.E. Gomory (1960) An algorithm for the mixed-integer problem. *Report RM-2597*, Rand Corporation, 1960 (Never published).
13. M. Grötschel, L. Lovász & A.J. Schrijver (1988) *Geometric Algorithms and Combinatorial Optimization.* Wiley: New York.
14. U.-U. Haus, M. Köppe & R. Weismantel (2000) The integral basis method for integer programming. *Math. Meth. of Oper. Res.* 53, 353–361.
15. A.H. Land & A.G. Doig (1960) An automatic method for solving discrete programming problems. *Econometrica* 28, 497–520.
16. A.N. Letchford & A. Lodi (2001) Primal separation algorithms. *Technical Report* OR/01/5. DEIS, University of Bologna.
17. A.N. Letchford & A. Lodi (2002) Primal cutting plane algorithms revisited. *Math. Methods of Oper. Res.*, to appear.
18. A.N. Letchford & A. Lodi (2002) Strengthening Chvátal-Gomory Cuts and Gomory fractional cuts. *Oper. Res. Letters*, to appear.
19. G.L. Nemhauser & L.A. Wolsey (1988) *Integer and Combinatorial Optimization.* New York: Wiley.
20. M.W. Padberg & M. Grötschel (1985) Polyhedral computations. In E. Lawler, J. Lenstra, A. Rinnooy Kan, D. Shmoys (eds.). *The Traveling Salesman Problem*, John Wiley & Sons, Chichester, 307–360.
21. M.W. Padberg & S. Hong (1980) On the symmetric travelling salesman problem: a computational study. *Math. Program. Study* 12, 78–107.
22. M.W. Padberg & G. Rinaldi (1991) A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problems. *SIAM Rev.* 33, 60–100.
23. A. Schulz, R. Weismantel & G. Ziegler (1995) 0-1 integer programming: optimization and augmentation are equivalent. In: *Lecture Notes in Computer Science*, vol. 979. Springer.
24. R. Thomas (1995) A geometric Buchberger algorithm for integer programming. *Math. Oper. Res.* 20, 864–884.
25. R. Urbaniak, R. Weismantel & G. Ziegler (1997) A variant of Buchberger's algorithm for integer programming. *SIAM J. on Discr. Math.* 1, 96–108.
26. R. Weismantel (1999) Private communication.
27. R.D. Young (1965) A primal (all-integer) integer programming algorithm. *J. of Res. of the National Bureau of Standards* 69B, 213–250.
28. R.D. Young (1968) A simplified primal (all-integer) integer programming algorithm. *Oper. Res.* 16, 750–782.