Concavity Cuts for Disjoint Bilinear Programming

Stéphane Alarie, Charles Audet, Brigitte Jaumard, and Gilles Savard

CRPC-TR99788-S
September 1999

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Submitted September 1999

Concavity Cuts for Disjoint Bilinear Programming

Stéphane Alarie
École Polytechnique de Montréal
Departement of Electrical Engineering and Computer Science
GRPR
C.P. 6079, Succursale Centre-Ville
Montréal (Québec), H3C 3A7, Canada
email: alaries@ai.polymtl.ca

Charles Audet
Rice University
Department of Computational and Applied Mathematics
6100 South Main Street - MS 134
Houston, Texas, 77005-1892, USA
email: charlesa@caam.rice.edu

Brigitte Jaumard and **Gilles Savard**
École Polytechnique de Montréal
Department of Mathematics and Industrial Engineering
GERAD & RCM$_2$
C.P. 6079, Succursale Centre-Ville
Montréal (Québec), H3C 3A7, Canada
email: brigitt@crt.umontreal.ca, gilles@crt.umontreal.ca

September 22, 1999

Abstract

We pursue the study of concavity cuts for the disjoint bilinear programming problem. This optimization problem has two equivalent symmetric linear maxmin reformulations, leading to two sets of concavity cuts. We first examine the depth of these cuts by considering the assumptions on the boundedness of the feasible regions of both maxmin and bilinear formulations. We next propose a branch and bound algorithm which make use of concavity cuts. We also present a procedure that eliminates degenerate solutions. Extensive computational experiences are reported. Sparse problems with up to 500 variables in each disjoint sets and 100 constraints, and dense problems with up to 60 variables again in each sets and 60 constraints are solved in reasonable computing times.
Key words: Concavity Cuts; Disjoint Bilinear Programming; Linear Maxmin Programming; Branch and bound Algorithm; Global Optimization.

Acknowledgments: Work of the second author was supported by NSERC (Natural Sciences and Engineering Research Council) fellowship PDF-207432-1998, and by CRPC (Center for Research on Parallel Computing). Work of the third author was supported by NSERC grant #GP0036426 and by FCAR (Fonds pour la Formation des Chercheurs et l'Aide à la Recherche) grant #99ER1048. Work of the fourth author was supported by NSERC grant #OGP0046405 and FCAR grant #93ER0141.

1 Introduction

The disjoint bilinear programming problem can be written as follows:

$$(\text{BILD}) \qquad \begin{array}{ll} \max\limits_{x,u} & c^t x - u^t Q x + u^t d \\ \text{s.t.} & x \in X, \quad u \in U \end{array}$$

where

$$X = \{x \in \mathbb{R}^{n_x} : Ax \le a, x \ge 0\}; \qquad U = \{u \in \mathbb{R}^{n_u} : u^t B \le b^t, u \ge 0\};$$

$$c \in \mathbb{R}^{n_x}; \quad a \in \mathbb{R}^{n_v}; \quad A \in \mathbb{R}^{n_v \times n_x}; \quad Q \in \mathbb{R}^{n_u \times n_x};$$
$$d \in \mathbb{R}^{n_u}; \quad b \in \mathbb{R}^{n_y}; \quad B \in \mathbb{R}^{n_u \times n_y}.$$

Several methods have been proposed in the literature to solve (BILD). All of them, except for those of Thieu [11] and Audet *et al.* [4], assume that the sets X and U are bounded.

A first class of methods corresponds to cutting-plane algorithms. Konno [9] proposes an algorithm of this class with a convergence to an ε-optimal solution although he shows that there always exists an optimal solution which lies at one of the extreme points of X and U. This algorithm consists of two phases which are repeated until an ε-optimal solution is obtained. The first one aims at finding a pair of ε-locally maximum basic feasible solutions, by alternately solving parameterized linear programs and then, if necessary, moving to an adjacent pair of basic feasible solutions (this is called augmented mountain climbing). In the second phase, a concavity cut which exploits the two basic solutions of the first phase is computed. It allows the elimination of solutions with a value which is never more than ε far away from the incumbent one. Very few computational results are available. Konno presents results obtained on a dozen of test problems of small size, the largest of them contains 10 (x) and 13 (u) variables with a set of 22 (X) and 24 (U) constraints. Independently, Vaish and Shetty [14] propose a cutting-plane very similar to Konno's one, but again with no guarantee of finite convergence. Sherali and Shetty [10] later show that finite convergence can be obtained with the addition of disjunctive cuts. Unfortunately, those cuts are quite expensive to compute.

Polyhedral annexation defines a second class of methods. Vaish and Shetty [15] propose an interior approximation algorithm with a finite convergence. They mention that numerical difficulties quickly arise when the size of the problems increases. Gallo and Ülkücü [7] present an algorithm which combines cutting-planes and outer approximation, with however no finite convergence. Thieu [11] develops an interior approximation method which is finite. It exploits a reformulation of (BILD) as a concave optimization one, and includes the solution of a parameterized linear program to solve it when the concave function is bounded below.

Several authors have observed that the (BILD) problem can be reformulated as any of the following two concave optimization problems:

$$\max_{x \in X} f(x), \qquad \max_{u \in U} g(u)$$

where $f : \mathbb{R}^{n_x} \to \mathbb{R}$ and $g : \mathbb{R}^{n_\mu} \to \mathbb{R}$ are the following two piecewise linear convex objective functions

$$f(x) = c^t x + \max_{u \in U} u^t(d - Qx), \qquad g(u) = u^t d + \max_{x \in X}(c^t - u^t Q)x.$$

By taking the dual of the optimization operator, they can be rewritten

$$f(x) = c^t x + \min_{y \in Y(x)} b^t y, \qquad g(u) = u^t d + \min_{v \in V(u)} v^t a,$$

where $Y(x)$ and $V(u)$ are two parameterized polyhedrons:

$$\begin{array}{rcl} Y(x) & = & \{y \in \mathbb{R}^{n_y} : By \geq d - Qx,\ y \geq 0\} \subseteq \mathbb{R}^{n_y}, \\ V(u) & = & \{v \in \mathbb{R}^{n_v} : v^t A \geq c^t - u^t Q,\ v \geq 0\} \subseteq \mathbb{R}^{n_v}. \end{array}$$

This leads to reformulations of the general bilinear programming problem (BILD) as one of the following two equivalent linear maxmin problems:

$$(\text{LMM}_{xy}) \qquad \max_{x \in X}\left(c^t x + \min_{y \in Y(x)} b^t y\right) = \begin{array}{ll} \max\limits_{x} & c^t x + \left(\begin{array}{ll} \min\limits_{y} & b^t y \\ \text{s.t.} & By \geq d - Qx \\ & y \geq 0 \end{array}\right), \\ \text{s.t.} & Ax \leq a \\ & x \geq 0 \end{array}$$

$$(\text{LMM}_{uv}) \qquad \max_{u \in U}\left(u^t d + \min_{v \in V(u)} v^t a\right) = \begin{array}{ll} \max\limits_{x} & u^t d + \left(\begin{array}{ll} \min\limits_{v} & v^t a \\ \text{s.t.} & v^t A \geq c^t - u^t Q \\ & v \geq 0 \end{array}\right). \\ \text{s.t.} & u^t B \leq b \\ & u \geq 0 \end{array}$$

Audet *et al.* [4] exploit these reformulations to build an efficient branch and bound algorithm to solve (BILD). No assumptions are made on the boundedness of the sets X and U. The algorithm consists of two phases. The first one detects if the optimal value of (BILD) is bounded or not. If it is bounded, the second phase (designed for bilinear programs with a bounded optimal value) is applied and finds its optimal solution. The algorithm creates an implicit enumeration search tree, and exploits the complementary slackness conditions of the primal and dual expressions of the functions f and g of the above reformulations. A binary branching is proposed. In one branch, it fixes a variable (or slack variable) to 0, while in the other one the corresponding complementary constraint (or non-negativity constraint) is transformed into an equality. Lower bounds are computed by iteratively solving finitely many parameterized linear programs: optimize iteratively $f(x)$ to obtain u and $g(u)$ to obtain x until both function values agree; this is the mountain climbing procedure of Konno [9]. Upper bounds are obtained with a relaxation of the linear maxmin programs.

The reader interested in a more detailed overview of the different methods of the literature is referred to the surveys written by Al-Khayyal [1] [2] and Floudas and Visweswaran [6], or to Audet [3] for links with other optimization problems.

The aim of this paper is to study the usefulness of incorporating concavity cuts in a branch and bound procedure for the disjoint bilinear program. The paper is organized as follows. We first recall in Section 2 the definition of a concavity cut in the context of linear maxmin problems. Section 3 is devoted to the study of the depth of the cuts for different assumptions on the boundedness of the domains. In Section 4, we present a branch and bound algorithm which includes concavity cuts. If the domain is unbounded, then it first checks boundedness of the optimal value (this is done by solving auxiliary programming problems see Section 4.3). In order to improve the efficiency of the algorithm, we discuss how to remove degenerate solutions (Section 4.4). A small example is provided in Section 4.5 to illustrate the algorithm. Section 5. contains extensive computational experiences on sparse and dense, bounded and unbounded randomly generated test problems. Sparse problems with up to 500 variables in both sets and 100 constraints and dense problems with up to 60 variables again in both sets and 60 constraints are solved in reasonable computing times.

2 Concavity cuts

We recall and extend the definitions of the concavity cuts for the linear maxmin problem. The reader is referred to Tuy [12] and to Horst and Tuy [8] for a reference on the original definitions. The extension here is with respect to the fact that the functions f and g are not defined everywhere but only on a restricted domain which corresponds to a projection.

Let $\hat{x}$ be a non-degenerate vertex of the polyhedron X. We do not consider here the degenerate case, as we propose in Section 4.4 a way to eliminate degenerate solutions.

Let $N_x = \{1, 2, \ldots, n_x\}$ and $\{\hat{x}^i : i \in N_x\}$ be the set of points of X that can be obtained by performing a single simplex pivot from $\hat{x}$. These points are called *neighbors* of $\hat{x}$. Since $\hat{x}$ is assumed non-degenerate, there are exactly n_x neighbors of $\hat{x}$.

For $i \in N_x$, if $\|\hat{x}^i - \hat{x}\|$ is finite, then $\hat{x}^i$ is a vertex of X and the i^{th} feasible neighboring direction is defined to be $\lambda^i = \hat{x}^i - \hat{x}$, otherwise $\hat{x}^i$ is an extreme ray of X and λ^i is defined to be the normalized direction $\frac{\hat{x}^i - \hat{x}}{\|\hat{x}^i - \hat{x}\|}$. Observe that if $Y(\hat{x}) = \emptyset$ or if $Y(\hat{x}^i) = \emptyset$ for some i, then $f(\hat{x}) = \infty$ or $f(\hat{x}^i) = \infty$, and hence (BILD) is unbounded (see Section 3 for details).

Consider a scalar γ satisfying $f(\hat{x}) \leq \gamma$ and $f(\hat{x} + \lambda^i) \leq \gamma$ $(i = 1, 2, \ldots, n_x)$.

Definition 2.1 *For all* $i \in \{1, 2, \ldots, n_x\}$ *the* γ-extension *along the direction* λ^i *is the point*

$\hat{x} + \hat{\theta}_i\lambda^i$ *where* $\hat{\theta}_i \in \overline{I\!R} = I\!R \cup \{\pm\infty\}$ *satisfies*

$$\hat{\theta}_i = \max_{\theta_i}\{\theta_i : f(\hat{x} + \theta_i\lambda^i) \leq \gamma\}.$$

The γ-extension $\hat{x} + \hat{\theta}_i\lambda^i$ necessarily satisfies $\hat{\theta}_i \geq 1$. Moreover, $\hat{\theta}_i$ may be infinite.

Proposition 2.2 *Let $\hat{x}$ be a non-degenerate vertex of X, and λ^i one of its feasible direction. The value of the parameter $\hat{\theta}_i$ of the γ-extension of $\hat{x}$ along λ^i is equal to the optimal value of the linear program*

$$\begin{array}{rl} \max\limits_{\theta_i, y} & \theta_i \\ s.t. & c^t(\hat{x} + \theta_i\lambda^i) + b^t y \leq \gamma \\ & y \in Y(\hat{x} + \theta_i\lambda^i). \end{array}$$

Proof: Let $\hat{x} + \hat{\theta}_i\lambda^i$ be the γ-*extension* along the direction λ^i. The parameter $\hat{\theta}_i$ satisfies:

$$\begin{aligned} \hat{\theta}_i &= \max_{\theta_i}\left\{\theta_i : c^t(\hat{x} + \theta_i\lambda^i) + \min_{y \in Y(\hat{x}+\theta_i\lambda^i)} b^t y \leq \gamma\right\} \\ &= \max_{\theta_i}\left\{\theta_i : \min_{y \in Y(\hat{x}+\theta_i\lambda^i)} b^t y \leq \gamma - c^t(\hat{x} + \theta_i\lambda^i)\right\}. \end{aligned}$$

However, for a fixed value θ_i, imposing that the minimum of $b^t y$ over $Y(\hat{x}+\theta_i\lambda^i)$ is less than or equal to a given threshold is equivalent to imposing that there exists a point $y \in Y(\hat{x}+\theta_i\lambda^i)$ with value $b^t y$ less than or equal to this threshold. ∎

Convexity of the function f ensures that all the points x contained in the convex envelope defined by $\hat{x}$ and the n_x γ-extensions are such that $f(x) \leq \gamma$. Thus, the inequality induced by the hyperplane going through the γ-extensions defines a valid cut.

Definition 2.3 *The* concavity cut *induced by the non-degenerate point $\hat{x}$ of X can be written $\pi_x(x - \hat{x}) \geq 1$, where $\pi_x(x - \hat{x}) = 1$ defines the unique hyperplane going through the γ-extensions $\hat{x} + \hat{\theta}_i\lambda^i$ $(i \in N_x)$.*

The depth of a concavity cut is defined to be the distance that separates the hyperplane from the vertex that induced it. We next define a deep concavity cut.

Definition 2.4 *The concavity cut induced by a non-degenerate extreme point $\hat{x}$ of X is said to be* deep *if for each $i \in N_x$, we have either $\hat{\theta}_i = \infty$ or $f(\hat{x} + \hat{\theta}_i\lambda^i) = \gamma$ or both.*

3 Properties of disjoint bilinear programming

Different assumptions can be made when considering the disjoint bilinear problem (BILD) or one of its linear maxmin reformulations (LMM_{xy}) or (LMM_{uv}).

For (BILD), a usual assumption is that both polyhedrons X and U are bounded. For (LMM_{xy}), it is often assumed that the set $\{(x, y) : x \in X, y \in Y(x)\}$ is bounded, and that the optimal solution lies in that set.

Both the assumptions on (BILD), or those on (LMM_{xy}) are alone sufficient to ensure that the optimal value of the problem (BILD) or (LMM_{uv}) problem is bounded. However, these assumptions are not equivalent. Suppose that the first assumption is violated and there is a solution (x', u') such that x' belongs to X and u' is an extreme ray of U, that yields an unbounded function value, i.e., $f(x') = \infty$. In such a case, it follows from the duality theory of linear programming that the set $Y(x)$ is empty. This may not necessarily affect the second assumption. The set $\{(x, y) : x \in X, y \in Y(x)\}$ still might be non-empty and bounded. This contradicts the second assumption which states that the optimal solution Solutions where $Y(x) = \emptyset$ are not to be considered by (LMM_{xy}). Similar observations can be made for the (LMM_{uv}) problem.

We investigate further these different assumptions below and analyze their consequences on the depth of concavity cuts. Before going on with the discussion, we recall the definition of the FORBIDDANCE decision problem proposed in Audet *et al.* [4].

Definition 3.1 *Given two polyhedrons $X \in \mathbb{R}^{n_x}$ and $P \in \mathbb{R}^{n_x+n_y}$, does there exist a point x in X such that the projection of P on the y-space, i.e., $Y(x) = \{y \in \mathbb{R}^{n_y} : (x, y) \in P\}$, is empty?*

It is shown in [4] that the FORBIDDANCE problem is strongly NP-complete.

3.1 Boundedness assumptions on the linear maxmin reformulation

The first formulation of the linear maxmin problem is due to Falk [5] where both levels of constraints are combined into a single non-empty polytope $P \subseteq \mathbb{R}^{n_x+n_y}$ (the dimensions of the vectors and variables are defined as in the introduction):

$$\text{(LMM)} \qquad \max_x \min_y \{c^t x + b^t y : (x, y) \in P\}.$$

Define $P_x = \{x : (x, y) \in P \text{ for some } y \in \mathbb{R}^{n_y}\}$ to be the projection of P on the y-space, and define $P_y(x) = \{y : (x, y) \in P\}$ to be the set of values that the variable y may take for a fixed x in $\mathbb{R}^{n_x}$. It follows that (LMM) may be rewritten as

$$\max_{x \in P_x} \left(c^t x + \min_{y \in P_y(x)} b^t y \right).$$

It is enough to assume that P is bounded to ensure that the optimal value is bounded. This formulation is however not equivalent to (LMM$_{xy}$). Problem (LMM$_{xy}$) is more general since there may be an $x \in X$ such that $Y(x) = \emptyset$, thus yielding an optimal value of $+\infty$. The dual counterpart of this solution in (BILD) is such that x belongs to X and u is an extreme ray of U along which the function value satisfies $\lim_{k\to\infty} c^t x - ku^t Qx + ku^t d = \infty$.

This behavior is impossible with (LMM) since for any $x \in P_x$, the set $P_y(x)$ is never empty. This is due to the fact that both sets are constructed from the same non-empty polytope P.

3.2 Boundedness assumptions on the bilinear formulation

Different cases must be considered when studying the disjoint bilinear problem (BILD). If both X and U are bounded, then the optimal value of (BILD) is necessarily bounded.

Let $\hat{x}$ be a non-degenerate vertex of X. Recall that the γ-extension induced by $\hat{x}$ along the feasible direction λ^i is obtained by computing the largest value $\hat{\theta}_i$ of θ_i for which the value $f(\hat{x} + \theta_i\lambda^i)$ is less than or equal to γ. Consider the following convex piecewise-linear function

$$\begin{aligned} h :& \mathbb{R} \to \mathbb{R} \\ \theta_i \mapsto& h(\theta_i) = f(\hat{x} + \theta_i\lambda^i) = c^t(\hat{x} + \theta_i\lambda^i) + \max_{u\in U} u^t\Big(d - Q(\hat{x} + \theta_i\lambda^i)\Big). \end{aligned}$$

If the domain U is bounded, then $h(\theta_i)$ is bounded for any value of θ_i. However if U is unbounded, it may happen that for a large enough $\hat{\theta}_i$, the value $h(\hat{\theta}_i)$ is unbounded. This is illustrated by the following small example in $\mathbb{R}$. Consider $\max\limits_{0\le u} u(\theta_i - 2)$. If $\theta_i > \hat{\theta}_i = 2$ then $h(\theta_i) = \infty$. This means that for values of θ_i larger than $\hat{\theta}_i$, the set $Y(\hat{x} + \theta_i\lambda^i)$ is empty. This leads to the following proposition that studies the depth (see Definition 2.4) of concavity cuts.

Proposition 3.2 *If the set U is bounded, then the concavity cuts with an extreme point of X are deep, and symmetrically, if X is bounded, the concavity cuts generated with an extreme point of U are deep.*

There are three possible types of γ-extensions:

(a) finite and deep with $h(\hat{\theta}_i) = \gamma$;

(b) infinite (thus deep) with $h(\theta_i) \le \gamma$ for non-negative values of θ_i;

(c) finite and not deep with $h(\hat{\theta}_i) < \gamma$ and $h(\theta_i) = \infty$ for every $\theta_i > \hat{\theta}_i$.
It occurs when the domain U is unbounded.

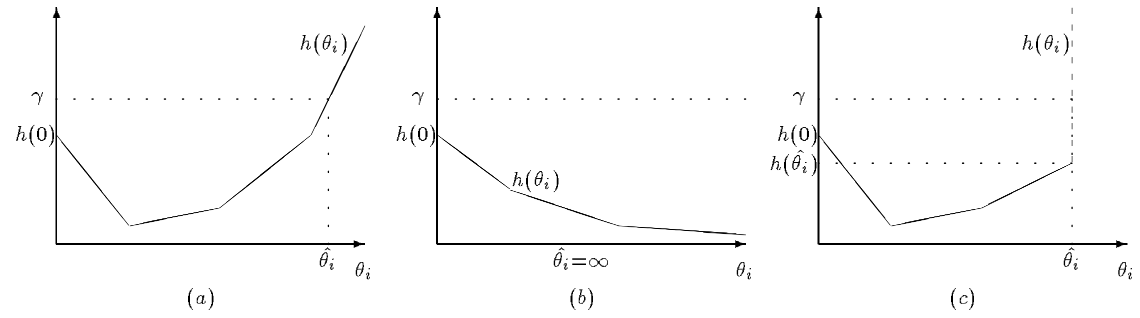

Figure 1: Evaluation of γ-extensions.

These three cases are illustrated in Figure 1.

Consider again the case where X is bounded and U is unbounded. Audet *et al.* [4] show that the optimal value of (BILD) is unbounded if and only if that of the auxiliary bilinear program

$$\begin{array}{rl} \max\limits_{x \in X,\ u} & u^t(d - Qx) \\ \text{s.t.} & u^tB \le 0, \\ & u^t\mathbf{1} \le 1, \\ & u \ge 0, \end{array}$$

is strictly positive, where $\mathbf{1}$ denotes the vector whose elements are all ones. The same authors propose to use their algorithm (for bilinear problems with a bounded optimal value) to solve this auxiliary bilinear problem. However, it is time consuming as the optimal solution of the auxiliary program is highly degenerate when the optimal value of (BILD) is bounded. Indeed, all the constraints (except $u^t\mathbf{1} \le 1$) go through the optimal solution $u = 0$. Moreover, observe that the feasible region with respect to u is a truncated cone vertexed at the optimal solution. We suggest an easier way to detect an unbounded optimal value.

Proposition 3.3 *Let X be a bounded polytope. The optimal value of* (BILD) *is unbounded if and only if the optimal value of*

$$(F_U) \qquad \max_{x \in X,\ u \in K_U} u^t(d - Qx)$$

is strictly positive, where K_U is the truncated cone $\{u \ge 0 : u^tB \le 0, 1 \le u^t\mathbf{1} \le 2\}$.

Proof: It is shown in [4] that when X is bounded, the optimal value of (BILD) is unbounded if and only if FORBIDDANCE (existence of an $x \in X$ such that $Y(x) = \emptyset$) is verified. Moreover, FORBIDDANCE is verified if and only if the optimal value of the linear maxmin problem

$$\max_{x \in X} \min_{y \in Y(x)} 0 = \max_{x \in X} \left(\begin{array}{rl} \min\limits_{y} & 0 \\ \text{s.t.} & By \ge d - Qx, \\ & y \ge 0, \end{array} \right)$$

is unbounded. The corresponding bilinear programming problem is

$$\begin{array}{rl} \max\limits_{x\in X,\ u} & u^t(d-Qx) \\ \text{s.t.} & u^tB \le 0, \\ & u \ge 0. \end{array}$$

The feasible region of the variable u is a cone. The result follows by observing that for fixed x, the function $u^t(d-Qx)$ is linear with respect to the variable u, and thus, if there is an extreme ray along which the objective value goes to infinity, then it must be strictly positive at the intersection of the ray and the plane defined by $u^t\mathbf{1} = 1$ (since it is null at the origin).

■

A symmetric result can be obtained when only the set X is unbounded using the following auxiliary disjoint bilinear programming problem:

(F_X) $$\max_{x\in K_X,\ u\in U} (c-u^tQ)x,$$

where K_X is the truncated cone $\{x \ge 0 : Ax \le 0, 1 \le \mathbf{1}^t x \le 2\}$.

When both domains are unbounded, a third auxiliary problem, denoted (F_{XU}), must be solved to verify if the optimal value of the problem (BILD) is bounded. It is defined as a disjoint bilinear problem where the constraint set is made of the two truncated cones K_X and K_U.

(F_{XU}) $$\max_{x\in K_X,\ u\in K_U} -u^tQx.$$

Proof is omitted since it is essentially identical to that appearing in [4].

3.3 Difference of convex sets

We recall that a set $M \subseteq \mathbb{R}^{n_x}$ is equal to the difference of two convex sets, i.e., is a *d.c.* set for short, if it can be expressed as $D \setminus C$ where D and C are respectively close and open convex sets in $\mathbb{R}^{n_x}$. The *d.c.* sets have been well studied, see, e.g., Tuy [13]. In this subsection, we present links between the FORBIDDANCE problem and *d.c.* sets.

It can be observed that solving the FORBIDDANCE problem corresponds to finding if there exists a solution belonging to a *d.c.* set. Let X, P and $Y(x) = \{y : (x,y) \in P\}$ define the FORBIDDANCE problem. Without any loss of generality, we assume that the set P is full dimensional.

Proposition 3.4 *The answer to* FORBIDDANCE *is* NO *if and only if* $\overset{\circ}{M}$ *is empty, with* $M = X \setminus \overset{\circ}{P}_x$*, where* $P_x = \{x \in \mathbb{R}^{n_x} : (x,y) \in P \text{ for some } y\}$ *and where the circle above a set denotes the largest open set contained in it.*

Proof: Assume that the answer to FORBIDDANCE is NO, i.e., that $Y(x) \neq \emptyset$ for all $x \in X$. It follows that for any $x \in X$, there exists a y such that $(x, y) \in P$, and therefore $X \subseteq P_x$. Consequently, $M = X \setminus \overset{\circ}{P_x} \subseteq P_x \setminus \overset{\circ}{P_x}$ and thus $\overset{\circ}{M} = \emptyset$.

Conversely, assume that $\overset{\circ}{M} \neq \emptyset$. Choose x in a ball entirely contained in the open set $\overset{\circ}{M}$ in such a way that x is not on the boundary of P_x. Definition of M implies that $x \in X$ and $x \notin \overset{\circ}{P_x}$, therefore $(x, y) \notin P$ for any y, thus $Y(x) = \emptyset$. ∎

4 Solving disjoint bilinear programming problems

We next recall the *mountain climbing* procedure of Konno [9] to find a first feasible solution. Consider a point $x^0 \in \mathbb{R}^{n_x}$. The value $f(x^0)$ is evaluated, yielding in the process a vector u^1 in U (even if the vector x^0 does not belong to X). Then, the objective function value $g(u^1)$ is evaluated, yielding in the process a vector x^1 in X. These steps are reiterated, creating (x^{i+1}, u^{i+1}) from (x^i, u^i) until the objective function value $f(x^{i+1})$ coincides with $g(u^{i+1})$. This process is finite, as an increasing sequence of objective function values is generated where each solution corresponds either to a vertex or to an extreme ray of a polyhedron. In the latter case, the process is stopped and the optimal value of the problem is unbounded. Solutions (x', u') obtained as an output of a mountain climbing procedure with x^0 as a starting point, will be denoted by $(x', u', \gamma') \leftarrow MC(x^0)$, with $\gamma' = f(x') = g(u')$ and $x' \in X$, $u' \in U$. Similarly, $(x', u', \gamma') \leftarrow MC(u^0)$ will refer to the symmetrical case when starting a mountain climbing procedure with a point $u^0 \in \mathbb{R}^{n_u}$.

This section is divided into five parts. We first present the basic algorithm that finds the optimal solution of any instance of (BILD) with a bounded optimal value following by the presentation of the cut's keeping strategy. We then show in the third part how to use the basic algorithm to solve problems when it is not assumed that the optimal value is bounded, by answering FORBIDDANCE questions. In the fourth part, we add a feature to the basic algorithm that allows removal of degenerate solutions. Finally, the algorithm is illustrated on a small example.

4.1 Basic algorithm for finite valued instances

A common drawback of cutting-plane algorithms lies in a slow convergence with cuts which are almost parallel and eliminate a very small part of the domain. When the domain is unbounded, it may happen that the cuts are not deep enough to completely eliminate it. We present an algorithm, called CBA, that combines concavity cuts with the branch and bound algorithm BB of Audet *et al.* [4].

We propose to perform first a preprocessing phase which generates concavity cuts. It corresponds to the algorithm CC described below. Concavity cuts are iteratively added to the sets X and/or U until one of the following stopping conditions is satisfied.

C1. A solution with a positive unbounded value is found.

C2. Either X or U is reduced to the empty set: the incumbent solution is the optimal solution.

C3. There has been more than m_x attempts to add a cut in X, and more than m_u in U.

C4. The depth of the last κ^{th} consecutive cuts in X was less than ε_x and the depth of the last κ^{th} consecutive cuts in U was less than ε_u.

C5. No cuts were added to X and to U in the last iteration.

Where m_x, m_u and κ are integers, and $\varepsilon_x > 0$ and $\varepsilon_u > 0$ are real numbers. We now describe the algorithm CC which iterates until one of these conditions is met.

Algorithm CC (iterate until one of the above 5 conditions is satisfied)

Step a. Find a starting point (x^0, u^0)**.** If there are still no cuts in X and U, initialize $(x^0, u^0) \leftarrow 0$, otherwise let (x^0, u^0) be the *reflection* (defined below) of the last vertex (x', u') through the last cut.

Step b. Find a feasible solution (x', u')**.** Perform two sets of mountain climbing iterations: let $(x', u', \gamma') \leftarrow MC(x^0)$ and $(x'', u'', \gamma'') \leftarrow MC(u^0)$. Select the one with largest value: if $\gamma'' > \gamma'$ reset (x', u', γ') to (x'', u'', γ'').

Step c. Incumbent Update. If $\gamma' > \hat{\gamma}$ reset $(\hat{x}, \hat{u}, \hat{\gamma})$ to (x', u', γ'). Stop if $\hat{\gamma} = \infty$: the optimal value of (BILD) is unbounded. Go to Step **d**.

Step d. Improve the current feasible solution in X**.** Apply the appropriate case.
Case d1: x' **is non-degenerate:** If there is a neighbor x'' of x' such that $f(x'') > f(x')$ reset $(x', u', \gamma') \leftarrow MC(x'')$ and go back to Step **c**. Else, go to Step **e**.
Case d2: x' **is degenerate:** Do not add any concavity cut to X. Go to Step **e**.

Step e. Improve the current feasible solution in U**.** Apply the appropriate case.
Case e1 : u' **is non-degenerate:** If there is a neighbor u'' of u' such that $g(u'') > g(u')$ reset $(x', u', \gamma') \leftarrow MC(u'')$ and go back to Step **c**. Else, go to Step **f**.
Case e2: u' **is degenerate:** Do not add any concavity cut to U. Go to Step **f**.

Step f. Addition of a concavity cut in X **vertexed at** x'**.** Go to Step **g** if no cut is required to be added to X (see Case **d2**). Else, compute the $\hat{\gamma}$-extensions in the directions of all neighbors of x'. Add to X the concavity cut that goes through all $\hat{\gamma}$-extensions. Stop if $X = \emptyset$.

Step g. Addition of a concavity cut in U vertexed at u'. Stop if no cut is required to be added to U (see Case **e2**). Else, compute the $\hat{\gamma}$-extensions in the directions of all neighbors of u'. Add to U the concavity cut that goes through all $\hat{\gamma}$-extensions. Stop if $U = \emptyset$. ∎

In Step **a**, the initial point x^0 is obtained by the reflection of the last vertex x through the last concavity cut π_x added to X, that is x^0 is heuristically set to $x + n_x\left(\delta_x + \frac{1}{\delta_x + 0.1}\right)\pi_x$, where δ_x is the depth of the cut π_x with respect to x. When no cuts have been added to X during the last iteration, x^0 is set to x. The procedure to initialize the point u^0 is similar.

The underlying idea of the reflection is to initialize the next mountain climbing procedure with a starting point far from the last vertex x. The point $x + \delta_x \pi_x$ lies on the concavity cut defined by the normalized vector π_x, and the starting point x^0 is chosen further in the same direction, in the other half-space created by the cut. The closer x is to the cut, the further on the other half-space the starting point x^0 is to the cut.

It is understood that Steps d and e are processed only when the current number of cuts does not exceed the parameters m_x and m_u.

We next show that whenever a cut is added to X or U, a local optimal solution is eliminated from the remaining optimization domain. We start with the following lemma.

Lemma 4.1 *If the feasible solutions x' and u' obtained by mountain climbing iterations are non-degenerate and are not exchanged with a neighbor (Cases* **d1** *and* **e1***) then they are local optimal solutions of the current linear maxmin problems.*

Proof: Convexity of the functions f and g ensures that

$$\begin{aligned} f(\tilde{x}) \le f(x') &\quad \text{for all } \tilde{x} \text{ in the convex envelope of } \{x', x'' : x'' \text{is a neighbor of } x'\}, \\ g(\tilde{u}) \le g(u') &\quad \text{for all } \tilde{u} \text{ in the convex envelope of } \{u', u'' : u'' \text{is a neighbor of } u'\}. \end{aligned}$$

The proof follows directly from the fact that the linear maxmin problems correspond to the maximization of the convex functions f and g. ∎

Proposition 4.2 *A local optimal solution is eliminated whenever a concavity cut is added in either Steps* **f** *or* **g**.

Proof: A concavity cut added in Step **f** eliminates the current non-degenerate solution processed in Case **d1**, and that of Step **g** eliminates the current non-degenerate solution considered in Case **e1**. Lemma 4.1 guarantees that the solution is indeed a local optimal one. ∎

The preprocessing phase, i.e. algorithm CC, yields the incumbent solution denoted $(\hat{x}, \hat{u})$ with objective function value $\hat{\gamma}$. If this first phase was not able to conclude the optimality of this solution (the stopping condition of CC is **C3**, **C4** or **C5**) , then a second algorithm BB (see [4] for the details) is applied to either show that the incumbent solution is indeed optimal, or to find the optimal solution.

When the concavity cuts are not enough (algorithm CC) to solve (BILD), it is usually not worthwhile to keep all the cuts generated in X or U as it would increase the number of additional dual variables y and v in the minimization subproblems of the linear maxmin reformulations. Therefore, it is useful to define a selection rule to retain the most efficient cuts. We present in the next subsection how to combine algorithm CC to BB.

4.2 Combing concavity cuts with a branch and bound algorithm

We present a rule to identify which concavity cuts produced by CC should be kept and which should be eliminated. The rule relies on the value of the cosine of the angle between two cuts. Recall that, if π and τ are two normal vectors, then the cosine of the angle between these vectors is $\frac{\pi \cdot \tau}{||\pi|| \cdot ||\tau||}$.

For both domains X and U, the following iterative strategy is used to determine which cuts are kept. Let $\{\pi_1, \pi_2, \ldots, \pi_m\}$ be the finite set of cuts produced by algorithm CC in one of the domains.

Initial cuts: Among all cuts eliminating the incumbent solution, conserve the furthest one from the incumbent (using the Euclidean distance). Among all cuts that do not eliminate the incumbent solution, keep the closest one to the incumbent. Initialize Ω to be the set containing these cuts.

Additional cuts: For each index i such that π_i is not in Ω, compute the sum $S_i = \sum\limits_{\tau \in \Omega} \frac{\tau \cdot \pi_i}{||\tau|| \cdot ||\pi_i||}$. If the largest sum S_i is less than half the cardinality of Ω, then add the corresponding cut π_i to Ω and repeat this step. Otherwise stop, no more cuts are kept.

The initial cuts are chosen to eliminate a large part of the domain, and the additional cuts are selected to maintain a wide variety of cuts. With this recursive rule, a cut is added only if the average of the cosines of the angles between this cut and all others already added is less than $\frac{1}{2}$. It is added if the average angle is more than $\frac{\pi}{3}$.

Having determined which cuts to keep, the next step consists in fixing the dual variables v and y (dual with respect to the primal concavity cuts) to zero. Indeed, complementary slackness conditions of linear programming ensure that at optimality, either the concavity cut will be satisfied as an equality, or the corresponding dual variable will be zero. In our case, since we are looking for a solution that has a (strictly) larger objective value than the incumbent value $\hat{\gamma}$, the concavity cuts can be treated as strict inequalities. It follows that in the linear maxmin formulations, the added dual variables can be fixed to zero. Let $\hat{X}$ and $\hat{U}$ be the domains reduced by the concavity cuts. The bilinear programming problem considers in algorithm BB is therefore

$$\max_{x \in \hat{X}, u \in \hat{U}} c^t x - u^t Q x + u^t d,$$

and the equivalent linear maxmin problems are

$$\max_{x \in \hat{X}} \left(c^t x + \min_{y \in Y(x)} b^t y \right) \text{ and } \max_{u \in \hat{U}} \left(u^t d + \min_{v \in V(u)} v^t a \right).$$

The sets $Y(x)$ and $V(u)$ are identical to those of LMM_{xy} and LMM_{uv}. The branching done in algorithm BB only occurs on the original constraints of X and U and corresponding dual variables.

We now show that the algorithm is finite and exact.

Theorem 4.3 *The algorithm CBA solves in finite time any instance of* (BILD) *having a finite optimal value.*

Proof: Upon execution of the algorithm, two scenarios are possible: either BB is called or it is not called. In both cases, either the incumbent solution obtained by CC is the optimal solution, or the true optimal solution is not eliminated by the finite number of concavity cuts generated by CC (see, e.g., Horst and Tuy [8]).

Since each cut computed in CC requires a finite amount of time, and the total number of cuts is bounded above by the parameters m_x and m_u, it follows that the total time required by CC is finite. If BB is not called, then the incumbent solution provided by CC is the optimal solution [8]. If BB is called, then it must solve a bilinear programming problem having a finite number of variables and constraints, and thus, as show in Audet *et al.* [4], the branch and bound algorithm BB solves it in finite time. ■

4.3 General algorithm

Algorithm CBA can be applied to solve either an instance of (BILD), or one of the auxiliary problems $(F_X), (F_U)$ or (F_{XU}) with some minor modifications.

Let us call CBA$^+$ the modified version of algorithm CBA which is used to solve the auxiliary bilinear problems. The differences are as follows. The incumbent value $\hat{\gamma}$ is first set at 0. As soon as it is updated in either CC or BB, i.e., when a feasible solution with a strictly positive objective value is obtained, algorithm CBA$^+$ stops. The answer to the

FORBIDDANCE question is then YES, otherwise it is NO. Also, note that fixing the incumbent value to 0 entails that all $\hat{\gamma}$-extensions are evaluated with a value of $\hat{\gamma} = 0$. This yields deeper concavity cuts than selecting the true incumbent value.

Table 1 summarizes the sequence in which the instances are solved. For example, when U is bounded and X is unbounded, the FORBIDDANCE question is answered through the solution of (F_X) as described above by CBA$^+$. If the answer is *yes*, then the optimal value of (BILD) is unbounded, otherwise it is obtained by applying the algorithm CBA to (BILD).

	U Bounded		U Unbounded	
	Problem	Algorithm	Problem	Algorithm
X Bounded	BILD	CBA	F_U	CBA$^+$
			BILD	CBA
X Unbounded	F_X	CBA$^+$	F_X	CBA$^+$
			F_U	CBA$^+$
	BILD	CBA	F_{XU}	CBA$^+$
			BILD	CBA

Table 1: Sequence of bilinear programs to be solved

Theorem 4.4 *Given any instance of* (BILD)*, the sequence of bilinear problems (*FORBIDDANCE *ones and the original one) described in Table 1 can be solved in finite time with both algorithms CBA and CBA$^+$ yielding an optimal solution, with possibly an unbounded value.*

Proof: Unboundedness of the optimal value is detected by solving the bilinear formulation of FORBIDDANCE problems [4]. Therefore, by Theorem 4.3, the FORBIDDANCE question is answered in finite time, and if the answer is *no* then the optimal value of the original (BILD) is bounded and obtained in finite time by the algorithm. ■

4.4 Degeneracy removal procedure

Computational difficulties arise when a degenerate extreme point is produced by the mountain climbing method. Such a point has therefore more neighbors than the dimension of the space. In the algorithm CC described above, concavity cuts are not added when degeneracy is encountered. In the literature (see e.g., Horst and Tuy [8]), authors have handled such a case with the addition of a concavity cut induced by the degenerate point using the γ-extensions in all neighboring directions. However, with such an approach, the NP-complete problem of enumerating all neighbors of a degenerate vertex must be solved.

We propose a method to eliminate the degenerate point which does not require the computation of all its neighbors. The underlying idea consists in adding a concavity cut induced by one of its non-degenerate neighbors (it is not necessary to enumerate them all). We present substitutes for Steps **d** and **e** that may remove the degeneracy. Replace Case **d2** of algorithm CC by the following cases.

Case d2': x' is degenerate and $\gamma' < \hat{\gamma}$:

If it is not possible to find a non-degenerate finite neighbor of x' by applying a single simplex iteration for each non-basic variable, then do not add concavity cuts to X. Otherwise, let $x'' \neq x'$ be such a finite non-degenerate neighbor, and proceed in one of the two ways.

i- If $f(x'') > \hat{\gamma}$, reset $(x', u', \gamma') \leftarrow MC(x'')$ and go back to Step **c**.

ii- If $f(x'') \leq \hat{\gamma}$, consider the neighbors of x''. If there is a neighbor x''' of x'' such that $f(x''') > \hat{\gamma}$ reset $(x', u', \gamma') \leftarrow MC(x''')$ and go back to Step **c**. Otherwise reset $x' \leftarrow x''$ and go to Step **e**.

Case d2" : x' is degenerate and $\gamma' = \hat{\gamma}$:

Do not add any concavity cut to X. Go to Step **e**. ■

Case **e2** is replaced by Cases **e2'** and **e2"**, having the same structure as Cases **d2'** and **d2"** but switching the roles of the variables.

4.5 Illustrative example

Consider the following example in which both variables x and u are in $\mathbb{R}^2$.

$$\begin{array}{lll}
\max\limits_{x,u} & -5x_1 - 6x_2 \; - 3u_1x_2 + 2u_2x_1 + 4u_2x_2 & -10u_1 - 15u_2 \\
\text{s.t.} & -2x_1 + x_2 \leq 0 & -47u_1 + 16u_2 \leq 16 \\
& -x_1 + x_2 \leq 1 & -19u_1 + 9u_2 \leq 16 \\
& x_2 \leq 4 & -u_1 + 3u_2 \leq 16 \\
& x_1 + x_2 \leq 12 & u_1 - 4u_2 \leq 2 \\
& 3x_1 - 2x_2 \leq 16 & u \geq 0 \\
& x_1 - 4x_2 \leq 2 & \\
& x \geq 0 &
\end{array}$$

The domain X is bounded and U is unbounded. These domains are illustrated on the left and top-right part of Figure 2. Both vertices $(0,0)$ and $(8,4)$ of X are degenerate. Near each vertex of X and U is written the objective function value of the vertex, and below is the corresponding solution in the other variable. For example, at the vertex $(6,1)$ of X, we have that $f(6,1) = -35$ and $u = (0,1)$ is an optimal solution of the parameterized linear program appearing in the function f. The bottom right part of Figure 2 displays the truncated cone K_U needed to answer the FORBIDDANCE question (F_U).

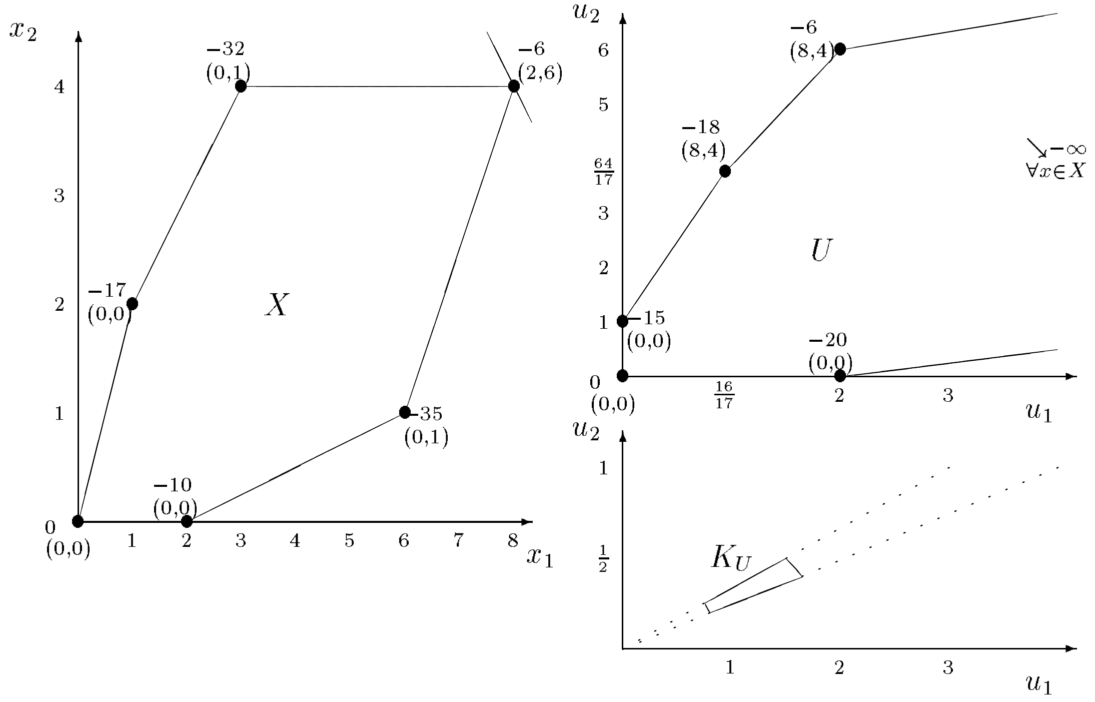

Figure 2: A Small Example

Prior to solving this problem, the FORBIDDANCE question must be answered in order to detect boundedness of the optimal value. The auxiliary bilinear problem (F_U) must now be solved. The mountain climbing iterations in Step **b** yield the non-degenerate vertices

$$x' = (6,1) \text{ and } u' = \left(\frac{3}{4}, \frac{1}{4}\right) \text{ with objective value } \gamma' = -9.5.$$

The $\hat{\gamma}$-extensions (with $\hat{\gamma}$ set to 0) in the direction in the neighbors $(2,0)$ and $(8,4)$ of x' are both infinite, thus the corresponding cut reduces X to the empty set. Therefore, the answer to forbiddance is NO, and the optimal value of (BILD) is bounded.

The original problem (BILD) must now be solved. A first iteration of CC is started. The mountain climbing iterations in Step **b** yield

$$x' = (0,0) \text{ and } u' = (0,0) \text{ with objective value } \gamma' = 0.$$

The incumbent value $\hat{\gamma}$ is set at 0 in c. Since x' is a degenerate vertex of X and $\gamma' = \hat{\gamma}$, Case **d2"** applies and no cuts are added to X. However, u' is non-degenerate and its neighbors are $(2,0)$ and $(0,1)$. Since $g(2,0) = -20 < \hat{\gamma} = 0$ and $g(0,1) = -15 < \hat{\gamma} = 0$, Case **e1** applies and then a concavity cut vertexed at u' added to U in Step **g**, yields

$$\frac{17}{64}u_2 \geq 1.$$

A second iteration of CC is initiated. The mountain climbing procedure of Step **b** is called again, leading to

$$x' = (8,4) \text{ and } u' = (2,6) \text{ of objective value } \gamma' = -6.$$

Again, x' is a degenerate vertex of X, but $\gamma' < \hat{\gamma}$, thus Case **d2'** of the degeneracy removal procedure applies. A concavity cut vertexed at the non-degenerate neighbor $(6,1)$ of x' is evaluated. The $\hat{\gamma}$-extensions in the directions of the neighbors $(2,0)$ and $(8,4)$ of the current solution $(6,1)$ are respectively $(-25.75, -6.9375)$ and $(8.24, 4.36)$. The value of $\hat{\gamma}$ used is 0 and not -6, and thus deeper cuts are generated. The concavity cut that goes through these points $-.127081x_1 + .38234x_2 \geq .619854$ eliminates all these points. Thus, the degenerate solution has been removed. For the variable u, one neighbor of $(2,6)$ is the degenerate vertex $(\frac{16}{17}, \frac{64}{17})$, and the other is at infinity in the direction $(1,3)$. The $\hat{\gamma}$-extension of the first is $(-.9605, -.25)$ and that of the second is at infinity. This yields the concavity cut $+.063333u_1 - .19u_2 \geq -.01333167$. Figure 3 illustrates the current reduced feasible regions. The number next to each vertex is the objective function value. For each point of X, the corresponding optimal solution in U is always $(11.0834, \frac{64}{17})$. For each point of U, the corresponding optimal solution in X is always $(.97217, 1.94434)$.

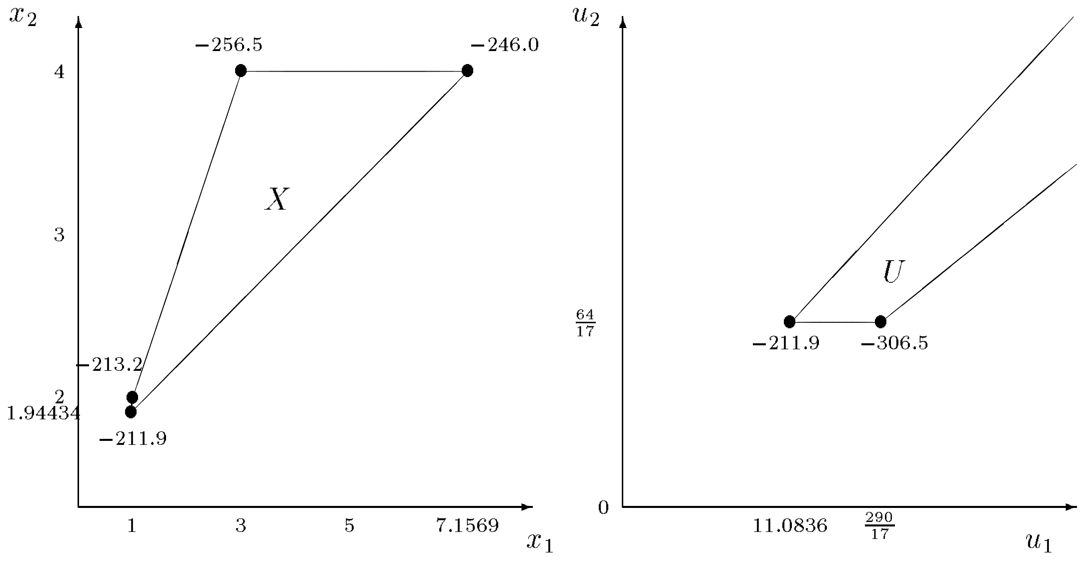

Figure 3: Example after two iterations

A third iteration is initiated. The mountain climbing steps of the third iteration yield the solution $x' = (.97217, 1.94434)$ and $u' = (11.0834, \frac{64}{17})$ having objective value -211.885. The concavity cut vertexed at x' eliminates the whole domain X, and thus the algorithm stops. The optimal solution is $x^* = (0,0), u^* = (0,0)$ and the optimal value is $\gamma^* = 0$.

5 Numerical results

In this section, the algorithm CBA is extensively tested and its performance is discussed on a large number of randomly generated instances. All computational experiments are conducted on a SUN ULTRA 1/167 station under Solaris 2.5-06 with 256M of RAM. The algorithm is coded in *C* (compiler gcc) and uses the CPLEX 5.0 library to solve the linear programs. Audet *et al.* [4]'s implementation of the branching part BB is used.

Three types of instances are considered: those in which both domains X and U are bounded; those in which X is bounded and U is unbounded; and those in which both domains are unbounded. A subsection is devoted to each type.

n	D (%)		BB		CBA						
					CC		BB				total
			time	nodes	time	cuts	needed	kept cuts	time	nodes	time
20	30	μ	162.9	2808.4	1.4	8.3	0	-	-	-	1.4
		σ	77.8	1423.3	0.9	4.8		-	-	-	0.9
	40	μ	713.8	12276.5	159.4	172.4	3	89.0	1644.7	11418.0	652.8
		σ	237.6	4204.3	243.2	225.7		44.0	1045.1	8295.0	1149.6
30	20	μ	>6014.7[1]	>39186.4	10.7	21.3	0	-	-	-	10.7
		σ	>1549.2	>13516.3	10.2	17.4		-	-	-	10.2
40	10	μ	>3835.2[2]	>28437.4	3.1	4.9	0	-	-	-	3.1
		σ	>2939.2	>33736.6	3.4	4.3		-	-	-	3.4
50	5	μ	>1619.4[3]	>6552.2	1.4	2.3	0	-	-	-	1.4
		σ	>2372.8	>8480.8	0.7	1.3		-	-	-	0.7

$m_x = m_u = 250$, $\varepsilon_x = \varepsilon_u = .1$ and $\kappa = 5$

[1] Four problems were stopped after two hours of running.
[2] Three problems were stopped after two hours of running.
[3] One problem was stopped after two hours of running.

Table 2: Comparison between BB and CBA (X and U bounded)

In all tables of this section, the problems are generated by the same problem generator used in [4]. The mean value μ and standard deviation σ of 10 randomly generated problems of density parameter D are given for each series. Unless otherwise specified, the coefficient n gives the instance dimensions: e.g. $n = 10$ means that $n_x = n_u = n_y = n_v = 10$. The computing time in seconds is given in column *time*, the number of cuts generated by algorithm CC appears in column *cuts* and the number of nodes required by algorithm BB is displayed in column *nodes*. The parameter *needed* gives the number of instances out of the ten that required use of algorithm BB. The parameter *kept cuts* defines the number of cuts generated by CC that are kept. These cuts are additional constraints in the polyhedrons X and U, and give indication on the size of the instances actually solved by BB. The means and standard deviations in the BB columns are computed *only* with respect to instances solved by BB. The last column summarizes the overall performance of CBA. Finally, the parameters for the stopping criteria of Section 4.1 (m_x, m_u, ε_x, ε_u and κ) are specified for each table.

Before analyzing the behavior of CBA, we show that the cut preprocessing phase improves the overall performance of BB. Comparative experience are described in Table 2. In four series of problems out of five, the results show that cut algorithm CC is sufficient to find an optimal solution with a proof of its optimality. Moreover, only very few cuts are needed; this leads to a significant reduction of computing time (seconds vs. hours). For the fifth one, 3 out of 10 problems require the use of BB but the average computing time is reduced. However for the three problems where BB is called, the computing time is higher even if the number of nodes explored is lower. This is due to the increase in the size of the problem that now contains additional cuts.

5.1 Bounded Domains

Table 3 presents the behavior of the algorithm on small instances with X and U bounded and for which algorithm CC succeeded in solving all ten instances of each series.

D		$n = 10$		$n = 20$		$n = 30$		$n = 40$		$n = 50$		$n = 60$	
(%)		time	cuts	time	cuts	time	cuts	time	cuts	time	cuts	time	cuts
5	μ	0.04	1.0	0.15	1.2	0.3	1.3	0.6	1.7	1.4	2.3	2.4	2.8
	σ	0.01	0.0	0.05	0.6	0.1	0.5	0.1	0.7	0.7	1.3	0.7	0.8
10	μ	0.05	1.1	0.18	1.5	0.7	2.9	3.1	4.9	6.5	5.6	81.2	23.1
	σ	0.01	0.3	0.06	0.7	0.3	1.3	3.4	4.3	2.8	2.3	99.8	22.2
15	μ	0.06	1.2	0.40	3.4	2.0	5.8	21.9	20.1				
	σ	0.01	0.4	0.30	2.4	1.6	3.9	18.7	14.7				
20	μ	0.06	1.5	0.42	3.0	10.7	21.3						
	σ	0.02	0.5	0.25	1.8	10.2	17.4						
30	μ	0.07	1.6	1.35	8.3								
	σ	0.04	0.8	0.85	4.8								
50	μ	0.19	3.6										
	σ	0.10	2.1										
100	μ	0.43	8.4										
	σ	0.32	5.8										

$m_x = m_u = 250$, $\varepsilon_x = \varepsilon_u = .1$ and $\kappa = 5$

Table 3: Complete solutions by CC (X and U bounded)

For larger or denser problems, algorithm BB is more often needed to find the optimal solution or to prove its optimality as illustrated in Table 4. Tables 3 and 4 confirm that the difficulty increases with n and D. There appears to be threshold values for n and D, under which the algorithm CC can quickly solve any of these randomly generated instances. When these parameters exceed these thresholds, solution time is still reasonable as long as algorithm BB is not called. However, when algorithm BB is called, computing time increases very rapidly. Table 4 illustrates this observation for instances where $n \geq 30$, and where the density parameter D is greater than the corresponding value of Table 3.

n	D		CC		BB				CBA
	(%)		time	cuts	needed	kept cuts	time	nodes	time
20	40	μ	159.4	172.4	3	89.0	1644.7	11418.0	652.8
		σ	243.2	225.7		44.0	1045.1	8295.0	1149.6
	50	μ	192.0	221.8	2	101.5	1722.9	9773.0	536.5
		σ	222.3	205.7		6.3	1024.5	5532.4	976.8
	60	μ	375.1	370.4	7	85.6	3296.6	22418.0	2682.7
		σ	242.1	209.4		40.5	1596.6	9550.9	2232.3
	70	μ	376.0	388.2	6	82.2	3135.6	21028.5	2257.4
		σ	218.9	180.4		44.8	1137.5	3239.8	2001.0
30	30	μ	389.5	184.6	2	84.0	>6 hours	-	-
		σ	522.6	209.3		2.8	-	-	-
40	20	μ	2105.8	405.9	8	96.1	>6 hours	-	-
		σ	1104.3	194.8		23.2	-	-	-
50	15	μ	2299.7	253.6	4	106.8	>6 hours	-	-
		σ	2353.7	227.9		16.0	-	-	-
60	15	μ	7383.2	449.5	9	99.4	>6 hours	-	-
		σ	2771.9	153.4		19.3	-	-	-

$m_x = m_u = 250$, $\varepsilon_x = \varepsilon_u = .1$ and $\kappa = 5$

Table 4: Solutions where BB is needed (X and U bounded)

For each instance of Table 4 where BB is required, the stopping criterion observed of CC is always **C3**: the number of cuts exceeds m_x and m_u. For all instances, CC has however found the optimal solution and BB is only called to rigourously confirm its optimality.

As computational time grows significantly when algorithm BB is called, one might think that we should avoid to use it by raising the upper limit on the number of concavity cuts allowed. Consider for example the series $n = 20$ and $D = 70\%$. Four of them are solved by CC, requiring from 77 to 477 cuts. Raising the limit on the total number of iterations m_x and m_u to 1000 yields the following. Out of the six instances previously unsolved by CC, three more can now be solved. They require 702, 1167 and 1615 cuts. The three other instances remain unsolved after 1000 iterations, for a total of 2000 cuts. However the mean computational time of CC for these three instances is now approximately 42,500 seconds with a standard deviation of 4775. Therefore, an upper bound on the number of cuts is justified.

Column *kept cuts* of Table 4 indicates that a significant proportion of the cuts generated by CC differ one from the others, according to the cosine criterion of Section 4.2. Column *kept cuts* shows that approximately 18% of the cuts (about 90 out of the 500 cuts generated as BB is required) are kept, regardless of parameters n and D.

Table 5 illustrates the behavior of algorithm CBA for large instances with low density. We observe that only CC phase of CBA is required to solve all instances.

D (%)		$n_x = n_u = 50$ time	cuts	$n_x = n_u = 100$ time	cuts	$n_x = n_u = 250$ time	cuts	$n_x = n_u = 500$ time	cuts
1	μ	0.8	1.0	2.8	1.3	69.3	5.4	2400.7	22.1
	σ	0.2	0.0	0.7	0.5	21.8	1.5	1104.7	8.9
5	μ	3.1	2.9	64.0	8.8	$>$ 6 hours	-		
	σ	1.0	0.8	47.7	5.0	-	-		

$m_x = m_u = 250$, $\varepsilon_x = \varepsilon_u = .1$ and $\kappa = 5$

Table 5: Complete solutions by CC for $n_y = n_v = 100$ (X and U bounded)

For the problems with $n_x = n_u = 250$ and $D = 5\%$, two instances are solved in less than six hours: one takes almost 5 hours and 108 cuts, and the other takes around one hour and only 31 cuts. The size of the problems in Table 5 is significantly larger than those solved by algorithm BB alone [4].

The initial problems, randomly generated, are rarely degenerate when both domains are bounded. The cuts added by algorithm CC usually does not increase degeneracy as the cuts rarely pass through any vertex of the polyhedra X and U. Hence, the degeneracy removal procedure is not often applied for the bounded case.

5.2 Bounded and Unbounded Domains

We now study the case where only one of the polyhedra is bounded. Without any loss of generality, we study the case in which X is bounded and U is unbounded.

In the following tables, we detail the behavior of CBA on the forbiddance problems (F_U), (F_X) and (F_{XU}) and the original instance (BILD). The overall computing time is given in the additional column *total*.

Table 6 details the performance of CBA when the optimal value is bounded, i.e., when the answer to the forbiddance problem is NO. Before solving (BILD) it must be shown that the optimal value of (F_U) is less than or equal to zero.

As before, most of the computing time is spent in CC. In all cases considered, except for $n = 20$ and $D = 70\%$, both the computing time and the number of cuts for CC are significantly less for the forbiddance problem (F_U) than for the original problem (BILD). However, the observed behavior nicely complements that of BB, since Audet *et al.* [4] observed that BB is significantly faster when solving (BILD) rather than (F_U).

The exception when $n = 20$ and $D = 70\%$ is due to the fact that CC generates deeper cuts while solving instances of (BILD) than while solving instances of (F_U). Hence, Algorithm CC is prematurely stopped by criterion **C4** when solving (F_U).

Another unusual result in Table 6 is the large computing time of BB when $n = 20$

n	D	Problem		CC		BB				CBA	
	(%)			time	cuts	needed	kept cuts	time	nodes	time	(total)
20	10	F_U	μ	0.6	8.9	3	2.3	201.1	11237.7	61.0	
			σ	0.7	9.2		2.5	330.2	18532.4	183.5	
		$BILD$	μ	3.6	22.4	1	28.0	3.5	167.0	4.0	65.0
			σ	9.4	47.0		0.0	0.0	0.0	10.5	182.4
	30	F_U	μ	2.5	17.9	6	3.3	1095.0	20601.2	659.6	
			σ	1.4	9.7		2.2	1056.8	19445.7	970.1	
		$BILD$	μ	71.1	168.4	10	19.8	565.5	19497.8	636.6	1296.2
			σ	29.3	48.1		14.6	559.5	16488.6	573.7	958.1
	50	F_U	μ	19.5	50.6	4	17.0	2741.2	36627.5	1115.9	
			σ	36.2	78.1		14.9	1297.5	18403.0	1620.9	
		$BILD$	μ	64.0	148.8	10	17.9	962.2	26127.5	1026.3	2142.1
			σ	33.3	56.3		10.8	718.2	24975.6	707.9	1980.4
	70	F_U	μ	72.4	166.1	7	11.7	2001.5	26330.9	1473.4	
			σ	29.8	63.1		7.0	935.1	12164.4	1252.6	
		$BILD$	μ	45.8	112.0	10	16.6	1677.8	40839.9	1723.5	3196.9
			σ	31.3	48.6		19.4	677.2	19538.9	673.7	1452.3
30	5	F_U	μ	0.5	4.1	3	1.7	34.8	1162.3	10.9	
			σ	0.5	4.0		1.5	31.5	1071.6	22.5	
		$BILD$	μ	6.3	25.4	1	47.0	0.3	1.0	6.3	17.2
			σ	17.5	60.8		0.0	0.0	0.0	17.6	26.3
	10	F_U	μ	3.1	19.9	9	2.9	1687.4	36666.9	1521.8	
			σ	0.7	4.8		1.2	1955.7	46345.5	1919.5	
		$BILD$	μ	35.1	88.6	10	14.3	37.6	33537.6	73.0	1594.7
			σ	15.9	26.2		5.4	25.1	45177.1	36.8	1913.1
40	5	F_U	μ	3.7	17.3	8	5.0	1322.8	28544.9	1061.9	
			σ	1.7	7.5		5.3	929.4	20562.7	992.3	
		$BILD$	μ	55.9	107.2	9	18.0	5.6	22270.4	60.9	1122.8
			σ	39.8	46.3		15.2	8.8	22986.6	39.4	1000.7

$m_x = m_u = 100$, $\varepsilon_x = \varepsilon_u = .1$ and $\kappa = 15$

Table 6: Complete solution by CBA (X bounded and U unbounded)

and $D = 50\%$. Additional numerical experiments were performed by generating 15 other instances. Out of these, only 4 are not solved by CC alone. For these 4 instances, BB required 1298.6 seconds (with a standard deviation of 635.9) instead of 2741.2 seconds as reported in Table 6.

Degeneracy occurs frequently in (F_U) for low density problems (e.g. $n = 20$ and $D = 10\%$, or $n = 30$ and $D = 5\%$). Recall from Proposition 3.3 that the feasible domain of (F_U) is composed of X and K_U, the truncated cone generated by the extreme rays of the set U. Therefore, when the density of these randomly generated problems is low, the set U often contains less than n_u extreme rays, and thus the cone K_U is not full-dimensioned, which explains the unfortunate degeneracy (for example if $U = \{(u_1, u_2) : u_1 \leq 1, u \geq 0\}$ then $K_U = \{(u_1, u_2) : u1 = 0, u_2 \geq 0\}$). The degeneracy removal procedure cannot provide any cuts in K_U. Nevertheless, most of these instances are solved by CC by adding concavity cuts to X, and frequently invoking the degeneracy removal procedure on this last set.

For denser problems, the degeneracy removal procedure is called a comparable number of times for both X and K_U, and the total number of cuts added to X and K_U are also similar. This follows from the fact that K_U is now full-dimensional, and degenerate solutions arise less frequently. For instances of (BILD), degenerate solutions are rarely encountered, but each time it occurs, the degeneracy removal procedure succeeds in eliminating it.

We conclude the discussion by observing that the same stopping criterion for both problems (F_U) and (BILD) are usually reached. For low-density instances, CC generally stops with criterion **C4**, i.e., several consecutive cuts having low depth are generated. CC stops less frequently with criterion **C5** as the degeneracy removal procedure fails on both domains. As the density increases, the active stopping criterion becomes **C3**, i.e., the maximum number of iterations is reached.

Table 7 addresses problems of the same size as in Table 6, but for which the optimal values are unbounded. For all these instances, only the forbiddance problem (F_U) is solved. Moreover, the solution process stops as soon as a feasible solution with a positive value is obtained.

n	D		CC		BB				CBA
	(%)		time	cuts	needed	kept cuts	time	nodes	time
20	10	μ	0.04	0.2	0	-	-	-	0.04
		σ	0.03	0.4		-	-	-	0.03
	30	μ	0.4	2.6	1	2.0	2.8	43.0	0.7
		σ	0.7	5.4		0.0	0.0	0.0	1.6
	50	μ	0.4	2.2	0	-	-	-	0.4
		σ	0.7	3.7		-	-	-	0.7
	70	μ	0.7	3.4	0	-	-	-	0.7
		σ	0.8	4.0		-	-	-	0.8
30	5	μ	0.2	1.1	0	-	-	-	0.2
		σ	0.3	3.1		-	-	-	0.3
	10	μ	0.4	2.5	1	1.0	0.9	7.0	0.5
		σ	0.7	4.8		0.0	0.0	0.0	1.0
40	5	μ	0.08	0.1	0	-	-	-	0.08
		σ	0.06	0.3		-	-	-	0.06

$m_x = m_u = 100$, $\varepsilon_x = \varepsilon_u = .1$ and $\kappa = 15$

Table 7: Detection of unboundedness by CBA (X bounded and U unbounded)

All 70 instances considered in Table 7 are quickly solved, and only 2 of them required the BB part of the algorithm, **C4** criterion being reached for these two problems.

5.3 Two Unbounded Domains

In this last section, we now consider the case where both domains X and U are unbounded. These problems are considerably more difficult than the previous instances since up to three auxiliary forbiddance problems must be solved prior to the execution of the algorithm on the original instance.

For each instance of Table 8, all three forbiddance problems (F_X), (F_U) and (F_{XU}) yielded negative optimal values, thus guaranteeing boundedness of the optimal value of (BILD).

n	D	Problem		CC		BB				CBA	
	(%)			time	cuts	needed	kept cuts	time	nodes	time	(total)
20	10	F_X	μ	0.6	9.1	4	3.2	0.4	25.5	0.8	
			σ	0.5	6.3		3.4	0.5	41.2	0.5	
		F_U	μ	0.8	11.7	5	7.0	110.0	6706.8	55.8	
			σ	0.6	8.2		3.4	245.1	14954.9	173.7	
		F_{XU}	μ	0.0	0.0	10	0.0	478.9	31738.3	478.9	
			σ	0.0	0.0		0.0	459.6	27932.8	459.6	
		BILD	μ	2.0	24.5	5	11.0	0.3	30916.6	2.1	537.6
			σ	1.7	20.7		7.7	0.2	34374.7	1.9	479.7
	30	F_X	μ	2.9	21.0	8	3.4	326.2	5795.5	263.8	
			σ	0.8	4.2		2.7	429.3	7161.9	403.2	
		F_U	μ	2.3	16.9	9	2.1	48.9	4131.3	46.3	
			σ	0.8	6.2		0.9	77.9	5769.2	75.3	
		F_{XU}	μ	0.1	0.2	10	0.2	3384.7	75925.7	3384.7	
			σ	0.1	0.4		0.4	3032.0	61055.1	3032.1	
		BILD	μ	26.1	89.8	10	6.1	227.2	79200.1	253.3	3948.1
			σ	15.2	35.4		4.7	231.2	59489.1	234.9	2812.6
	50	F_X	μ	4.7	21.0	2	8.0	318.2	4135.0	68.4	
			σ	4.4	15.6		1.4	315.8	4160.6	173.9	
		F_U	μ	8.6	36.6	5	12.8	561.6	8519.6	289.4	
			σ	4.4	16.5		3.8	730.8	11504.3	571.9	
		F_{XU}	μ	3.0	12.8	3	1.3	1948.2	41488.7	587.5	
			σ	5.8	20.0		1.5	761.8	6626.3	1006.0	
		BILD	μ	27.0	81.7	10	6.8	860.8	26645.4	987.8	1933.1
			σ	19.5	39.1		6.7	1120.0	24210.4	1112.4	1741.4
	70	F_X	μ	11.9	45.7	9	10.6	320.3	3525.3	299.1	
			σ	2.3	7.3		5.6	549.3	5227.9	528.7	
		F_U	μ	11.8	45.8	9	13.7	459.8	7140.7	425.7	
			σ	2.0	7.0		5.6	531.0	5829.4	522.2	
		F_{XU}	μ	67.0	146.3	7	16.0	1395.9	25300.4	1044.1	
			σ	42.3	83.0		5.8	621.8	4804.2	877.1	
		BILD	μ	27.1	79.0	10	8.8	1078.9	34367.4	1105.9	2874.8
			σ	24.0	48.7		13.0	665.6	14656.9	646.7	1122.4

$m_x = m_u = 100$, $\varepsilon_x = \varepsilon_u = .1$ and $\kappa = 15$ for F_{XU} and *BILD*
$m_x = m_u = 25$, $\varepsilon_x = \varepsilon_u = .1$ and $\kappa = 15$ for F_X and F_U

Table 8: Solutions by CBA (X and U unbounded)

Most of the instances of Table 8, including the forbiddance ones, are not solved by CC (as shown by the parameters *needed*). The total computing time strongly depends on the number of times CC fails to solve (F_{XU}). This explains why the total time when $D = 30\%$ is higher that the time for the denser problems.

The previous analysis of the degeneracy is again valid for both K_X and K_U. Degenerated solutions of (BILD) is only observed for $D = 10\%$.

Problem (F_{XU}) has degenerate solutions in both domains K_X and K_U. When D is less than or equal to 30%, the degeneracy removal procedure fails systematically on both domains, and CC always terminates because of the degeneracy of both truncated cones K_X and K_U. When $D = 50\%$, domains K_X and K_U are often degenerate. The degeneracy removal procedure is however able to provide a concavity cut half of the time and most of the instances are solved by CC in that case. For $D = 70\%$, only few degenerate solutions are observed. When one appears, the degeneracy removal procedure always provides a cut. However, many instances are not solved by CC for this value of D, as the stopping criterion **C3** is then reached.

The stopping criteria for CC varies from one instance to the other. For $D = 70\%$, the criterion **C3** is activated most of the time. For problem (BILD), CC reaches the criterion **C4** for all values of D, excepted for $D = 10\%$ where (BILD) is solved by CC half of the time.

In Table 9, we solved problems with unbounded optimal values of the same size and density as in Table 8.

n	D (%)	Problem		CC time	CC cuts	BB needed	BB kept cuts	BB time	BB nodes	CBA time
20	10	F_X	μ	0.1	1.7	0	-	-	-	0.1
			σ	0.2	2.3		-	-	-	0.2
	30	F_X	μ	0.7	5.7	2	2.0	6.5	105.0	2.0
			σ	0.7	6.4		0.0	8.5	144.2	4.4
		F_U	μ	1.1	8.0	1	1.0	1.5	25.0	1.8
			σ	1.3	9.9		0.0	0.0	0.0	2.4
	50	F_X	μ	0.6	3.2	0	-	-	-	0.6
			σ	1.3	7.0		-	-	-	1.3
	70	F_X	μ	0.2	1.0	0	-	-	-	0.2
			σ	0.4	1.7		-	-	-	0.4

$m_x = m_u = 25$, $\varepsilon_x = \varepsilon_u = .1$ and $\kappa = 15$

Table 9: Unbounded Solutions by CBA (X and U unbounded)

Unboundedness of the optimal solution is always detected while solving the forbiddance problem (F_X) except for two instances where it was detected by (F_U). For these two instances, CC showed in less than 1.4 seconds that the optimal value of (F_X) is negative, and thus allowing the possibility that the optimal solution of the original problem be bounded.

Then CC is called to solve (F_U). For the first of these two instances, CC succeeds in finding a positive feasible objective value, and for the second one, CC fails to solve (F_U) as the stopping criterion **C4** is reached. Algorithm BB is called and successfully finds a positive objective value.

For all the other instances except two, CC yields a positive valued feasible solution. For the two others, BB is called since the stopping criterion **C4** is reached once again.

References

[1] AL-KHAYYAL F.A. (1990), " Jointly Constrained Bilinear Programs and Related Problems: an Overview", *Computers and Mathematics with Applications* 19(11) (1990) 53-62.

[2] AL-KHAYYAL F.A.(1992), "Generalized Bilinear Programming, Part I: Models, Applications and Linear Programming Relaxation," *European Journal of Operational Research* 60, 306–314.

[3] AUDET C.(1997), "Optimisation globale structurée : propriétés, équivalences et résolution," Ph.D. Thesis, École Polytechnique de Montréal, available at http://www.crt.umontreal.ca/~charlesa/These.ps.

[4] AUDET C., HANSEN P., JAUMARD B. and SAVARD G.(1999), "A Symmetrical Linear Maxmin Approach to Disjoint Bilinear Programming," *Mathematical Programming* 85, 573–592.

[5] FALK J.E.(1973), "A Linear Max-Min Problem," *Mathematical Programming* 5, 169–188.

[6] FLOUDAS C.A. and VISWESWARAN V.(1995), *Quadratic Optimization* in *Handbook of Global Optimization*, HORST R. and PARDALOS P.M. (eds.), Kluwer Academic Publishers, Boston, 217–269.

[7] GALLO G. and ÜLKÜCÜ A.(1977), "Bilinear Programming: an Exact Algorithm," *Mathematical Programming* 12, 173–194.

[8] HORST R. and TUY H.(1996), *Global Optimization (Deterministic Approaches), third edition* Springer-Verlag Berlin New-York.

[9] KONNO H.(1976), "A Cutting Plane Algorithm for Solving Bilinear Programs," *Mathematical Programming* 11, 14–27.

[10] SHERALI H.D. and SHETTY C.M.(1980), “A Finitely Convergent Algorithm for Bilinear Programming Problem using Polar Cuts and Disjunctive Face Cuts,” *Mathematical Programming* 19, 14–31.

[11] THIEU T.V.(1988), “A Note on the Solution of Bilinear Problems by Reduction to Concave Minimization,” *Mathematical Programming* 41, 249–260.

[12] TUY H.(1964), “Concave Programming under Linear Constraints,” *Soviet Mathematics Doklady Academii Nauk SSSR* 5 [English version :*Soviet Mathematics* 5 (1964) 1437–1440].

[13] TUY H.(1998), “Convex Analysis and Global Optimization,” Kluwer Academic Publishers, Boston, London.

Soviet Mathematics Doklady Academii Nauk SSSR 5 [English version :*Soviet Mathematics* 5 (1964) 1437–1440].

[14] VAISH H. and SHETTY C.M.(1976), “The Bilinear Programming Problem,” *Naval Research Logistics Quarterly* 23, 303–309.

[15] VAISH H. and SHETTY C.M.(1977), “A Cutting Plane Algorithm for the Bilinear Programming Problem,” *Naval Research Logistics Quarterly* 24, 83–94.