On the effectiveness of Lagrangean cuts in solving a class of low rank d.c. programs

Riccardo Cambini *
Francesca Salvi
Department of Economics and Management
University of Pisa
Via Cosimo Ridolfi 10
56124 Pisa
Italy

Abstract

D.C. programs have been widely studied in the recent literature due to their importance in applicative problems. In this paper the results of a computational study related to a branch and reduce approach for solving a class of d.c. problems are provided, pointing out the concrete effectiveness of the use of Lagrangean cuts as an acceleration device.

Keywords: *d.c. programming, branch and reduce.*

AMS - 2010 Math. Subj. Class: *90C30, 90C26.*

JEL - 1999 Class. Syst: *C61, C63.*

1. Introduction

The so called d.c. programming, where a d.c. function (that is a function given by the difference of two convex ones) is optimized over a certain feasible region, is one of the main topics in the recent optimization literature. Its relevance from both a theoretical (see for all [11]) and an applicative point of view (see for example [1, 4, 6, 8, 10, 12, 14, 15, 21, 22] and references therein) is widely known. Specifically speaking, in this paper the following d.c. program is considered:

**E-mail:* `riccardo.cambini@unipi.it`

$$P:\begin{cases} min\ f(x)=c(x)-\sum_{i=1}^{k} g_i(d_i^T x) \\ x\in X\subseteq \mathbb{R}^n \end{cases} \tag{1}$$

The set X is a polyhedron given by inequality constraints $Ax \le b$ and/or equality constraints $A_{eq}x = b_{eq}$ and/or box constraints $1 \le x \le u$, where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, l, $u \in \mathbb{R}^n$, $A_{eq} \in \mathbb{R}^{h\times n}$, $b_{eq} \in \mathbb{R}^h$, $d_i \in \mathbb{R}^n$ for all $i = 1, ..., k$. The functions $c:\mathbb{R}^n \to \mathbb{R}$ and $g_i:\mathbb{R}\to\mathbb{R}, i=1,\dots,k$, are convex and continuous. We also assume that there exists $\tilde{\alpha},\tilde{\beta}\in\mathbb{R}^k$ such that $\tilde{\alpha}_i \le d_i^T x \le \tilde{\beta}_i \quad \forall x\in X$ $\forall i=1,\dots,k$.

In [2] this class of problems have been computationally studied with a branch and bound approach, pointing out the effectiveness of partitioning rules and of stack policies for managing the branches. In [3] these problems have been approached with a branch and reduce method, showing the importance of applying acceleration devices at every single algorithm iteration. Particular cases of problem P have been considered in [9, 17, 18].

The aim of this paper is to deepen on the study proposed in [2, 3] analyzing the opportunity of using Lagrangean cuts within the branch process of a branch and reduce solution scheme. It will be pointed out that, in the case "dual-adequate" primitives are available (see [20]), the use of Lagrangean cuts highly improve the performance of the branch and reduce method. It will be also shown that the "ω-subdivision" partitioning rule, which is commonly used in the literature, is not the better choice.

In Section 2 the branch and reduce approach is analyzed and described in details. In Section 3 the theoretical fundamentals needed for Lagrangean cuts are provided. In Section 4 the results of a computational study are provided and discussed in order to point out the concrete effectiveness of Lagrangean cuts.

2. The general branch and bound approach

A branch and bound scheme for the considered class of problems has been already described in [2, 3]. For the sake of completeness, and in order to let the reader understand the computational results provided and discussed in Section 4, let us briefly recall the approach and let us notice that the aim of this paper is to deep on the use of Lagrangean cuts in the branch and reduce solution scheme.

The concave part $-\sum_{i=1}^{k} g_i(d_i^T x)$ of $f(x)$ can be linearized with respect to the functions $d_i^T x, i=1,\dots,k$ (see for example [2, 3, 5, 18]), and then the relaxed convex subproblem can be solved. Given a pair of vectors $\alpha, \beta \in \Re^k$, with $\alpha \le \beta$ let $B(\alpha, \beta)$ the following set:

$$B(\alpha,\beta) = \{x \in \Re^n : \alpha \leqq D^T x \leqq \beta\}$$

where D is the $n \times k$ matrix whose columns are the k vectors $d_1, \dots, d_k$. The concave part $-\sum_{i=1}^{k} g_i(d_i^T x)$ of function $f(x)$ can be linearized over $B(\alpha, \beta)$ as follows:

$$f_B(x) = c(x) - \sum_{i=1}^{k}[\mu_i(d_i^T x - \alpha_i) + g_i(\alpha_i)] = c(x) - \mu^T(D^T x - \alpha) - \sum_{i=1}^{k} g_i(\alpha_i)$$

where for all $i = 1, \dots, k$ it is:

$$\mu_i = \begin{cases} \dfrac{g_i(\beta_i) - g_i(\alpha_i)}{\beta_i - \alpha_i} & \text{if } \alpha_i < \beta_i \\ 0 & \text{if } \alpha_i = \beta_i \end{cases}$$

Function $f_B(x)$ is an underestimation for $f(x)$ over the set $B(\alpha, \beta)$, so that the following relaxed convex subproblem can be defined and used in the branch and bound scheme:

$$P_B(\alpha,\beta): \begin{cases} \min f_B(x) \\ x \in X \cap B(\alpha,\beta) \end{cases} \tag{2}$$

The following theorem estimates the error done by solving the relaxed problem.

Theorem 1: *Let us consider problems P and $P_B(\alpha, \beta)$ and let*

$$x^* = \arg \min_{x \in X \cap B(\alpha,\beta)} \{f(x)\} \quad \text{and} \quad \bar{x} = \arg \min_{x \in X \cap B(\alpha,\beta)} \{f_B(x)\}.$$

Then, $f_B(\bar{x}) \le f(x^*) \le f(\bar{x})$, that is to say that $0 \le f(x^*) - f_B(\bar{x}) \le Err_B(\bar{x})$ where :

$$\begin{aligned} Err_B(x) &= f(x) - f_B(x) = \\ &= \mu^T(D^T x - \alpha) - \sum_{i=1}^{k}\left[g_i(d_i^T x) - g_i(\alpha_i)\right] \end{aligned}$$

The following main procedure "*DcBranch*()" can then be proposed. With this aim, let us denote with $A_j, j = 1, \dots, m,$ the *j*-th row of matrix *A*.

Procedure **DcBranch**(inputs: P; outputs: Opt, $OptVal$)
fix the tolerance parameter $\varepsilon > 0$;
initialize the global variables $x_{opt} := []$ and $UB := +\infty$;
initialize the stack;
determine the starting vectors $\tilde{\alpha}, \tilde{\beta} \in \Re^k$ such that $\forall i \in \{1, \ldots, k\}$:

$$\tilde{\alpha}_i = \min_{x \in X} \{d_i^T x\} \quad \text{and} \quad \tilde{\beta}_i = \max_{x \in X} \{d_i^T x\}$$

Optional : compute $v_j := \min_{x \in X} \{A_j x\} \ \forall j \in \{1, \ldots, m\}$;
Analyze $(\tilde{\alpha}, \tilde{\beta})$;
while the stack is nonempty *do*
$\left(f_B(x_B), \alpha, \beta, x_B, r, X\right)$:=Select();
if $f_B(x_B) < UB$ and $\left|\frac{UB - f_B(x_B)}{UB}\right| > \varepsilon$ *then*
# Optional : $(\alpha, \beta) := Resize(\alpha, \beta, I, X)$;
$\alpha 1 := \alpha$; $\beta 1 := \beta$; $\alpha 2 := \alpha$; $\beta 2 := \beta$;
γ:=Split(α_r, β_r); $\beta 1_r := \gamma$; $\alpha 2_r := \gamma$;
Analyze ($\alpha 1$, $\beta 1$); Analyze($\alpha 2$, $\beta 2$);
end if;
end while;
Opt:=x_{opt}; *OptVal* :=*UB*;
end proc.

The sub-procedure named "*Select*()" extracts from the stack the subproblem to be eventually branched. In [2] it has been shown that the way such a stack is implemented greatly affects the overall performance of the algorithm. In this light, in [2] it is pointed out that a priority stack, where problems having the smaller lower bound $f_B(x_B)$ have the biggest priority, is an effective choice. The sub-procedure named "*Split*()" determines a value $\gamma \in (\alpha_r, \beta_r)$ which will be used to divide $B(\alpha, \beta)$ in two hyper-rectangles (this is a generalization of the so called "rectangular partitioning method" [7, 23]). We considered the same 7 different partitioning rules proposed in [2, 3], which are based on the following values:

- $\gamma_1 := d_r^T x_B$;
- $\gamma_2 := \dfrac{\alpha_r + \beta_r}{2}$;
- $\gamma_3 := \arg\max_{y \in [\alpha_r, \beta_r]} \{\mu_r (y - \alpha_r) - (g_r(y) - g_r(\alpha_r))\}$.

In other words, the value $\gamma \in (\alpha_r, \beta_r)$ provided within procedure "*DcBranch*()" by the sub-procedure "*Split*()" can be computed as follows:

p1) $\gamma := \gamma_1$ ("ω – subdivision process");

p2) $\gamma := \gamma_2$ (classical bisection);

p3) $\gamma := \gamma_3$ (maximum error);

p4) $\gamma := \dfrac{\gamma_1 + \gamma_2}{2}$;

p5) $\gamma := \dfrac{\gamma_1 + \gamma_3}{2}$;

p6) $\gamma := \dfrac{\gamma_2 + \gamma_3}{2}$;

p7) $\gamma := \dfrac{\gamma_1 + \gamma_2 + \gamma_3}{3}$.

Notice that in procedure "*DcBranch*()" there is another optional sub-procedure named "*Resize*()" which is aimed to improve the performance of the solution method. Notice also that the calculus of the optional values $v_j, j \in \{1, \ldots, m\}$, is needed just in case the optional sub-procedure "*CutRegion*()" is used within the forthcoming procedure "*Analyze*()".

Procedure "*Analyze*()" studies the current relaxed subproblem, eventually improves the incumbent optimal solution, determines the index *r* corresponding to the maximum error, and finally appends in the stack the obtained results. With these aims, the following further error function is used:

$$Err_B(x,i) = \mu_i(d_i^T x - \alpha_i) - (g_i(d_i^T x) - g_i(\alpha_i))$$

Notice that it yields $Err_B(x) = \sum_{i=1}^{k} Err_B(x,i)$.

Procedure **Analyze**(inputs: α, β)
  determine the function $f_B(x)$ over $B(\alpha, \beta)$;
  $x_B := \arg\min\{P_B\}$;
  if $f(x_B) < UB$ *then*
    $x_{opt} := x_B$ and $UB := f(x_B)$;
  end if;

if $f_B(x_B) < UB$ and $\left|\frac{UB - f_B(x_B)}{UB}\right| > \varepsilon$ *then*

Optional : $(\alpha, \beta) := CutBounds()$; update $f_B(x)$ over $B(\alpha, \beta)$;

Optional : $X := CutRegion()$;

$r := \arg\max_{i=1,\dots,k}\{Err_B(x_B, i)\}$;

Append $\left(f_B(x_B), \alpha, \beta, x_B, r, X\right)$;

end if;

end proc.

The sub-procedure named "*Append*()" inserts into the stack the studied subproblem. Notice that, since $f_B(x)$ is an underestimation function of $f(x)$, there is no need to study the current relaxed subproblem in the case $f_B(x_B) \geq UB$. For the sake of convenience, the tolerance parameter $\varepsilon > 0$ is also used, avoiding the study when $\left|\frac{UB - f_B(x_B)}{UB}\right| \leq \varepsilon$. The point $x_B := \arg\min\{P_B\}$ can be determined by any of the known algorithms for convex programs, that is any algorithm which finds an optimal local solution of a constrained problem. In order to decrease as fast as possible the error $Err_B(x_B)$, the eventual branch operation is scheduled for the index r such that $r = \arg\max_{i=1,\dots,k}\{Err_B(x_B, i)\}$. In this light, notice that condition $\left|\frac{UB - f_B(x_B)}{UB}\right| > \varepsilon$ implies $Err_B(x_B, r) > 0$ which yields $\alpha_r < \beta_r$. This guarantees that a branch operation with respect to such an index r is possible.

Notice that there are two optional procedures named "*CutBounds*()" and "*CutRegion*()" which will be discussed in the next section and which are aimed to improve the performance of the solution method by properly reducing the bounds α, β and the feasible region X by means of the use of duality results.

It is worth noticing that the very aim of this paper is to emphasize the role of these two optional subprocedures. In other words, the performance behavior of the solution scheme will be studied depending on the use of none, one or both of these optional subprocedures "*CutBounds*()" and "*CutRegion*()".

3. Lagrangean Cuts Acceleration Device

In this section some acceleration techniques are studied in order to improve the performance of the general branch and bound method described in the previous section. Specifically speaking, two optional sub-procedures, named "*CutBounds*()" and "*CutRegion*()", will be provided with the aim to determine their effectiveness among the branch and reduce solution scheme. In this light, Section 4 will point out from a computational point of view whether it is worth using none, one or both of these sub-procedures. Notice also that in [3] these two subprocedures have been both used by default without any computational and explicit motivation. Let us also point out that the results stated in the forthcoming Subsection 3.2 are aimed to deep on the ones given in [3].

3.1 *Resizing the bounds*

As it has been described in the previous section, the solution method starts with the bounds $\tilde{\alpha}, \tilde{\beta} \in \Re^k$, computed by means of the $2k$ linear programs $\tilde{\alpha}_i = \min_{x\in X}\{d_i^T x\}$ and $\tilde{\beta}_i = \max_{x\in X}\{d_i^T x\}$, $i = 1,\dots,k$. Clearly, this starting vectors have the tightest possible values with respect to the feasible region X.

Unfortunately, after some branch iterations the current bounds (α, β) are no more tight with respect to the considered feasible region $X \cap B(\alpha,\beta)$. In order to improve the performance of the algorithm the values of (α, β) are periodically recalculated with respect to the current feasible region $X \cap B(\alpha,\beta)$. Since this could be heavy from a computational point of view, we considered the opportunity to recalculate the values only for a subset I of the indices, that is $I \subseteq \{1,\dots,k\}$. In other words, the sub-procedure call $(\alpha,\beta) := Resize(\alpha,\beta,I,X)$ just recalculates for all $i \in I$ the values:

$$\alpha_i = \min_{x\in X\cap B(\alpha,\beta)} \{d_i^T x\} \quad \text{and} \quad \beta_i = \max_{x\in X\cap B(\alpha,\beta)} \{d_i^T x\}$$

Various subsets I of indices have been considered in a computational test in order to determine the better choice. The obtained computational results will be described in Section 4.

3.2 *Lagrangean Cuts*

Let us now show how to improve the solution algorithm by means of the use of reduction techniques based on duality results. This is a technique already used in [20, 17] and based on known results by Rockafellar [19] and by Minoux [16]. Some of the following results have been already briefly described in [3], while in this section they are deepened on and fully proved. Consider the parametric convex problem

$$C_y = \begin{cases} \min \phi(x) \\ h(x) \le y \\ x \in X \subseteq \mathbb{R}^n \end{cases}$$

where X is a convex set, the functions $\phi : X \to \mathbb{R}$ and $h : X \to \mathbb{R}$ are convex, and y is a real parameter. Let us define also the set $X_y = \{x \in X \subseteq \mathbb{R}^n : h(x) \le y\}$ and the function

$$\psi(y) = \min_{x \in X_y} \phi(x)$$

In [19] Rockafellar proved that function $\psi(y)$ is convex. By means of Theorem 5.4 proved by Minoux in [16] we can then obtain the following result.

Theorem 2: *Let $\bar{x}$ be the optimal solution of C_0 such that $h(\bar{x}) = 0$ and let $\lambda \in \mathbb{R}, \lambda < 0$, be the corresponding K-K-T multiplier relative to the constraint $h(x) \le 0$. Then, $\psi(y) \ge \psi(0) + \lambda y \quad \forall y \in \mathbb{R}$.*

The following corollary holds.

Corollary 1: *Let UB be an upper bound for the minimum value of $\phi(x)$ in problem C_0. Under the assumptions of Theorem 2 we get:*

$$y < \frac{UB - \psi(0)}{\lambda} \quad \Rightarrow \quad \psi(y) > UB \tag{3}$$

In other words, $\bar{x}$ (optimal solution of C_0) verifies the inequality $h(\bar{x}) \ge \frac{UB - \psi(0)}{\lambda}$.

Proof: From $y < \frac{UB - \psi(0)}{\lambda}$ we get $\psi(0) + \lambda y > UB$ so that (3) follows being $\psi(y) \ge \psi(0) + \lambda y \ \forall y \in \mathbb{R}$. The whole result is stated noticing that for all $x \in X$ such that $h(x) < \frac{UB - \psi(0)}{\lambda}$, that is to say for all $x \in X_y$ such that $y < \frac{UB - \psi(0)}{\lambda}$, it results $\phi(x) \ge \psi(y) > UB$.

By applying Corollary 1 to the convex subproblems $P_B(\alpha, \beta)$ we can obtain the following specific results. In this light, an inequality constraint is defined a "valid cut" if it does not exclude any solutions with values smaller than the incumbent upper bound *UB*.

Theorem 3: *Consider Problem P and its convex relaxation $P_B(\alpha, \beta)$, described in (1) and (2), respectively. Let x_B be the optimal solution of $P_B(\alpha, \beta)$ with* *value $f_B(x_B)$. Let also UB, $UB \geq f_B(x_B)$, be the value of the current incumbent optimal solution x_{opt}. Then, the following valid cuts hold for the active inequality constraints corresponding to x_B and having a strictly negative K-K-T multiplier:*

	Active Constraint	K-K-T Multiplier	Indices	Valid Cut
1.	$d_i^T x - \beta_i \leq 0$	$\mu_i < 0$	$i = 1,\dots,k$	$d_i^T x \geq \beta_i + \frac{UB - f_B(x_B)}{\mu_i}$
2.	$\alpha_i - d_i^T x \leq 0$	$\lambda_i < 0$	$i = 1,\dots,k$	$d_i^T x \leq \alpha_i - \frac{UB - f_B(x_B)}{\lambda_i}$
3.	$A_i x - b_i \leq 0$	$\mu_i < 0$	$i = 1,\dots,m$	$A_i^T x \geq b_i + \frac{UB - f_B(x_B)}{\mu_i}$
4.	$v_i - A_i x \leq 0$	$\lambda_i < 0$	$i = 1,\dots,m$	$A_i x \leq v_i - \frac{UB - f_B(x_B)}{\lambda_i}$
5.	$e_i^T x - u_i \leq 0$	$\mu_i < 0$	$i = 1,\dots,n$	$e_i^T x \geq u_i + \frac{UB - f_B(x_B)}{\mu_i}$
6.	$l_i - e_i^T x \leq 0$	$\lambda_i < 0$	$i = 1,\dots,n$	$e_i^T x \leq l_i - \frac{UB - f_B(x_B)}{\lambda_i}$

Proof: Consider the constraints of type 1. The result follows directly from Corollary 1 assuming $h(x) = d_i^T x - \beta_i$ and noticing that $\psi(0) = f_B(x_B)$. The other cases are analogous.

The previous theorem suggests some valid inequalities which could be helpful in improving the algorithm performance by cutting off an "useless" part of the feasible region. With this aim, the convex subproblems $P_B(\alpha,\beta)$ have to be solved with an algorithm providing both the optimal solution and the corresponding K-K-T multipliers (such a kind of algorithms have been called "dual-adequate" in [20]).

As it has been shown, these cuts can be applied to the bounds $\alpha_i \leq d_i^T x \leq \beta_i, i = 1,\dots,k,$ thus improving the convex relaxation function $f_B(x)$ and the related error function $Err_B(x)$. They can also be used in reducing the feasible region X, that is to say the constraints $v \leq Ax \leq b$ and $l \leq x \leq u$;

this does not affect the error by itself, but it improves the effectiveness of the "*Resize*()" optional sub-procedure. These cuts are concretely described in the following sub-procedures "*CutBounds*()" and "*CutRegion*()". Notice that the use of "*CutRegion*()" optional sub-procedure requires in procedure "*DcBranch*()" the computation of the preliminary values $v_j := \min_{x \in X}\{A_j x\}$ $\forall j \in \{1,\dots,m\}$. Let us conclude recalling that the aim of this paper is to study the computational role of these two optional subprocedures. In this light, the performance of the branch and bound method will be analyzed depending on the use of none, one or both of subprocedures "*CutBounds*()" and "*CutRegion*()".

Procedure **CutBounds**(outputs: α, β)
    for all $i \in \{1,\dots,k\}$ *do*
        let λ_i be the KKT multiplier corresponding to $d_i^T x \le \beta_i$;
        if $\lambda_i < 0$ *then* set $\alpha_i := max\left\{\alpha_i, \beta_i + \frac{UB - f_B(x_B)}{\lambda_i}\right\}$ *end if*;
        let μ_i be the KKT multiplier corresponding to $d_i^T x \ge \alpha_i$;
        if $\mu_i < 0$ *then* set $\beta_i := min\left\{\beta_i, \alpha_i - \frac{UB - f_B(x_B)}{\mu_i}\right\}$ *end if*;
    end for;
end proc.

Procedure **CutRegion**(outputs: X)
    for all $i \in \{1,\dots,m\}$ *do*
        let λ_i be the KKT multiplier corresponding to $A_i x \le b_i$;
        if $\lambda_i < 0$ *then* set $l_i := max\left\{v_i, b_i + \frac{UB - f_B(x_B)}{\lambda_i}\right\}$ *end if*;
        let μ_i be the KKT multiplier corresponding to $A_i x \ge v_i$;
        if $\mu_i < 0$ *then* set $b_i := min\left\{b_i, v_i - \frac{UB - f_B(x_B)}{\mu_i}\right\}$ *end if*;
    end for;
    for all $i \in \{1,\dots,n\}$ *do*
        let λ_i be the KKT multiplier corresponding to $x_i \le u_i$;
        if $\lambda_i < 0$ *then* set $l_i := max\left\{l_i, u_i + \frac{UB - f_B(x_B)}{\lambda_i}\right\}$ *end if*;
        let μ_i be the KKT multiplier corresponding to $x_i \ge l_i$;
        if $\mu_i < 0$ *then* set $u_i := min\left\{u_i, l_i - \frac{UB - f_B(x_B)}{\mu_i}\right\}$ *end if*;

end for;

end proc.

4. Computational results

The procedures and the acceleration devices described in the previous section have been implemented in order to study their concrete effectiveness. This has been done in a MatLab R2009a environment on a computer having 6 Gb RAM and two Xeon dual core processors at 2.66 GHz. We considered problems with n = 15 variables, m = 15 inequality constraints, box constraints $l \le x \le u$ and no equality constraints. For the sake of convenience, we considered the class of functions $f(x) = \frac{1}{2}x^T Q x + q^T x - \sum_{i=1}^{k} \lambda_i (d_i^T x + d_i^0)^4$ with k = 10 and $Q \in \mathbb{R}^{n \times n}$ symmetric and positive semi-definite. The problems have been randomly generated; in particular, matrices and vectors $A \in \mathbb{R}^{m \times n}, Q \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^m, q, l, u \in \mathbb{R}^n$, $d_i \in \mathbb{R}^n, \lambda_i > 0, d_i^0 \in \mathbb{R}, i = 1, \ldots, k,$ have been generated with components in the interval [–10, 10] by using the "randi()'' MatLab function (integers numbers generated with uniform distribution). Within the procedures, the problems have been solved with the "linprog()'', "quadprog()'' and "fmincon()'' MatLab functions which provide both the optimal solution and the K-K-T multipliers. For the various instances 100 randomly generated problems have been solved. The average numbers of relaxed problems solved and the average CPU time needed to solve the problems are given as results of the test in Table 1 and Table 2, respectively. The two tables are organized as follows:

- the first column "*Resize*'' concerns the use of sub-procedure "*Resize*()''; "*None*'' means that such a sub-procedure is not used at all; "1^{st}'' means that sub-procedure "*Resize*()'' is used with the set of indices I made by just the index i corresponding to the biggest error $Err_B(x, j), j = 1, \ldots, k$; "$2^{nd}$'' means that sub-procedure "*Resize*()'' is used with I given by just the index i corresponding to the second biggest error $Err_B(x, j)$, j = 1,...k; "1^{st} – 10^{th}'' means that the set I is composed by all of the ten indices 1, ..., 10; "2^{nd} – 5^{th}'' means that the set I is made by 4 indices corresponding to the errors $Err_B(x, j)$, j = 1,...k, from the second biggest one to the fifth biggest one; the other cases are analogous;
- the second column "*LC*'' concerns the use of the Lagrangean cuts: "*None*'' means that neither "*CutBounds*()'' nor "*CutRegion*()'' are used; "*CB*'' means that only the sub-procedure "*CutBounds*()'' is

used; "*CB* + *CR*" means that both "*CutBounds*()" and "*CutRegion*()" are used;

- Columns 3 – 9 report the use of the 7 partitioning rules *p*1 – *p*7.

The rows of the tables are divided into 5 groups:

- the first one (row 1) regards the use of no acceleration devices at all;
- the second one (rows 2 – 3) regards the use of Lagrangean cuts and no "*Resize*()" ;
- the third one (rows 4 – 14) regards the use of "*Resize*()" and no Lagrangean cuts;
- the fourth one (rows 15 – 25) regards the use of "*Resize*()" and just "*CutBounds*()";
- the last one (rows 26 – 36) regards the use of "*Resize*()" and both "*CutBounds*()" and "*CutRegion*()";

In each row the better performance is emphasized in bold, while the worst performance is expressed in italics.

Table 1

Average number of relaxed subproblems solved (k = 10, n = m = 15)

Resize	LC	p1	p2	p3	p4	p5	p6	p7
None	*None*	*3116.50*	860.29	563.52	876.58	**559.45**	684.73	675.89
None	*CB*	*3017.70*	837.68	**540.61**	856.15	542.33	663.17	655.630
None	*CB + CR*	*2987.30*	755.89	**485.41**	792.72	500.47	598.60	601.66
1^{st}	*None*	*2138.80*	676.90	**470.07**	815.14	630.90	580.01	650.64
2^{nd}	*None*	*866.22*	437.18	314.18	418.01	**298.50**	380.52	357.80
$2^{nd}-3^{rd}$	*None*	*581.38*	343.81	257.49	310.50	**234.17**	297.50	273.12
$2^{nd}-4^{th}$	*None*	*473.47*	298.42	230.79	265.33	**199.38**	265.57	239.75
$2^{nd}-5^{th}$	*None*	*452.50*	279.42	217.06	242.24	**184.51**	247.45	221.58
$2^{nd}-6^{th}$	*None*	*428.29*	267.90	208.81	232.00	**173.68**	237.68	211.15
$2^{nd}-7^{th}$	*None*	*427.08*	260.53	204.24	223.04	**169.25**	231.10	205.50

Contd...

$2^{nd}-8^{th}$	*None*	*424.90*	256.46	201.77	219.06	**166.36**	227.62	202.27
$2^{nd}-9^{th}$	*None*	*426.41*	255.41	200.44	217.53	**165.53**	226.03	200.50
$2^{nd}-10^{th}$	*None*	*425.56*	255.24	199.87	217.14	**164.81**	225.71	199.90
$1^{st}-10^{th}$	*None*	*419.10*	249.16	193.97	210.68	**162.84**	219.18	195.69
1^{st}	*CB*	*2129.5*	670.90	**465.91**	821.51	638.19	579.06	653.18
2^{nd}	*CB*	*742.94*	410.96	286.21	396.80	**279.98**	356.27	335.36
$2^{nd}-3^{rd}$	*CB*	*442.44*	315.16	229.98	287.02	**215.59**	268.81	251.17
$2^{nd}-4^{th}$	*CB*	*324.49*	268.07	200.03	241.66	**181.07**	236.45	217.58
$2^{nd}-5^{th}$	*CB*	*286.58*	248.45	185.98	218.24	**163.99**	216.20	195.98
$2^{nd}-6^{th}$	*CB*	*255.76*	233.67	178.17	205.11	**152.5**	205.59	184.19
$2^{nd}-7^{th}$	*CB*	*236.42*	225.08	172.95	196.09	**147.77**	199.58	177.58
$2^{nd}-8^{th}$	*CB*	*230.78*	221.34	169.68	191.41	**143.10**	195.13	173.34
$2^{nd}-9^{th}$	*CB*	*228.18*	218.35	167.06	187.78	**142.59**	190.93	170.42
$2^{nd}-10^{th}$	*CB*	*225.46*	217.13	164.62	187.22	**141.29**	191.14	169.82
$1^{st}-10^{th}$	*CB*	*223.81*	214.21	159.67	182.60	**141.06**	187.50	169.42
1^{st}	*CB + CR*	*2163.60*	671.38	**473.12**	883.89	689.77	589.46	695.28
2^{nd}	*CB + CR*	*601.28*	318.22	**211.39**	324.73	226.15	263.38	265.93
$2^{nd}-3^{rd}$	*CB + CR*	*311.86*	222.35	**154.97**	215.70	160.26	187.53	183.71
$2^{nd}-4^{th}$	*CB + CR*	*209.60*	181.00	**127.97**	167.32	128.38	152.75	147.81
$2^{nd}-5^{th}$	*CB + CR*	*167.30*	157.29	112.52	144.26	**109.72**	133.21	127.88
$2^{nd}-6^{th}$	*CB + CR*	*142.94*	143.78	102.70	129.89	**98.911**	121.29	115.52
$2^{nd}-7^{th}$	*CB + CR*	*131.60*	134.72	96.27	120.16	**92.13**	113.300	107.72
$2^{nd}-8^{th}$	*CB + CR*	*126.38*	129.36	91.90	115.05	**87.50**	108.02	102.88
$2^{nd}-9^{th}$	*CB + CR*	*122.68*	126.83	88.98	112.18	**86.10**	104.98	100.72
$2^{nd}-10^{th}$	*CB + CR*	*121.59*	125.64	87.35	111.23	**84.76**	104.31	98.87
$1^{st}-10^{th}$	*CB + CR*	*119.65*	126.56	87.93	111.16	**87.10**	106.96	99.79

Table 2

Average CPU time spent (k = 10, n = m = 15)

Resize	LC	p1	p2	p3	p4	p5	p6	p7
None	*None*	*183.220*	48.699	34.425	48.812	**32.525**	40.645	39.376
None	*CB*	*182.590*	48.056	33.721	48.164	**32.005**	39.921	38.697
None	*CB + CR*	*194.790*	47.960	33.753	48.588	**32.312**	39.971	38.954
1^{st}	*None*	*170.320*	47.481	**33.967**	58.466	46.302	41.762	47.287
2^{nd}	*None*	*65.354*	31.199	23.590	29.854	**22.106**	28.196	26.326
$2^{nd}-3^{rd}$	*None*	*54.242*	30.095	23.400	27.414	**21.257**	26.729	24.549
$2^{nd}-4^{th}$	*None*	*52.412*	30.957	24.709	27.814	**21.414**	28.225	25.527
$2^{nd}-5^{th}$	*None*	*57.920*	33.597	26.786	29.370	**22.912**	30.340	27.242
$2^{nd}-6^{th}$	*None*	*62.205*	36.649	29.224	31.950	**24.433**	33.072	29.477
$2^{nd}-7^{th}$	*None*	*69.250*	39.922	31.903	34.324	**26.590**	35.928	32.001
$2^{nd}-8^{th}$	*None*	*75.966*	43.449	34.845	37.238	**28.869**	39.136	34.785
$2^{nd}-9^{th}$	*None*	*83.389*	47.358	37.854	40.454	**31.370**	42.509	37.749
$2^{nd}-10^{th}$	*None*	*90.310*	51.192	40.799	43.695	**33.754**	45.893	40.632
$1^{st}-10^{th}$	*None*	*96.307*	53.690	42.587	45.827	**36.080**	47.875	42.996
1^{st}	*CB*	*175.230*	47.334	**33.932**	59.480	47.275	41.973	47.840
2^{nd}	*CB*	*57.552*	29.567	21.825	28.566	**20.939**	26.691	24.908
$2^{nd}-3^{rd}$	*CB*	*42.407*	27.740	21.205	25.494	**19.750**	24.336	22.760
$2^{nd}-4^{th}$	*CB*	*36.795*	27.916	21.646	25.411	**19.563**	25.248	23.278
$2^{nd}-5^{th}$	*CB*	*37.654*	29.952	23.177	26.546	**20.526**	26.598	24.221
$2^{nd}-6^{th}$	*CB*	*38.103*	32.037	25.147	28.378	**21.594**	28.684	25.818
$2^{nd}-7^{th}$	*CB*	*39.356*	34.591	27.293	30.318	**23.412**	31.184	27.826
$2^{nd}-8^{th}$	*CB*	*42.356*	37.625	29.589	32.681	**25.016**	33.706	30.042
$2^{nd}-9^{th}$	*CB*	*45.795*	40.639	31.890	35.103	**27.205**	36.110	32.275
$2^{nd}-10^{th}$	*CB*	*48.896*	43.599	33.899	37.778	**29.111**	39.054	34.745

Contd...

$1^{st}-10^{th}$	*CB*	*52.573*	46.135	35.279	39.805	**31.378**	40.970	37.267
1^{st}	*CB* + *CR*	*194.130*	50.693	**36.839**	68.902	54.776	45.647	54.607
2^{nd}	*CB* + *CR*	*50.069*	25.522	**18.329**	25.765	18.739	22.146	21.934
$2^{nd}-3^{rd}$	*CB* + *CR*	*31.898*	21.927	**16.279**	21.198	16.328	19.128	18.539
$2^{nd}-4^{th}$	*CB* + *CR*	*25.233*	21.049	15.673	19.461	**15.386**	18.325	17.594
$2^{nd}-5^{th}$	*CB* + *CR*	*23.276*	21.145	15.771	19.321	**15.120**	18.309	17.518
$2^{nd}-6^{th}$	*CB* + *CR*	*22.466*	21.836	16.210	19.742	**15.357**	18.865	17.872
$2^{nd}-7^{th}$	*CB* + *CR*	*23.096*	22.994	16.982	20.386	**15.932**	19.642	18.566
$2^{nd}-8^{th}$	*CB* + *CR*	*24.561*	24.420	17.947	21.575	**16.628**	20.696	19.591
$2^{nd}-9^{th}$	*CB* + *CR*	*26.059*	26.183	18.997	22.996	**17.873**	22.034	21.000
$2^{nd}-10^{th}$	*CB* + *CR*	*28.073*	28.001	20.177	24.659	**18.992**	23.671	22.229
$1^{st}-10^{th}$	*CB* + *CR*	*29.924*	29.268	20.992	26.240	**20.860**	25.217	23.647

It is worth to point out the following obtained computational results:

- the "ω–subdivision" process $p1$ proposed and used in [9, 17, 18] is generally the worst partitioning rule from both the average number of iterations and the average CPU time points of view;
- the partitioning rule $p5$ is generally the one providing the best performance;
- the use of "*Resize*()" sub-procedure is fundamental for having a good performance; Lagrangean cuts without any "Resize" operation results to be not effective;
- the use of "*CutRegion*()" sub-procedure greatly amplifies the effectiveness of "*Resize*()" sub-procedure;
- the use of both "*CutBounds*()" and "*CutRegion*()" sub-procedures improves the algorithm performance;
- the use of "*Resize*()" sub-procedure with respect to just the index corresponding to the biggest error (1^{st}) is useless;
- the "Resize" operations are quite heavy from a computational point of view (two LPs to be solved for each index in I); having a big set I decreases the average number of convex subproblems solved, but may be too expensive from a CPU time point of view;

- the best performance with respect to the average CPU time spent is given by partitioning rule $p5$ in the 28^{th} row, that is when both "*CutBounds*()" and "*CutRegion*()" sub-procedures are used and "*Resize*()" is done in the 4 indices corresponding to the errors $Err_B(x,j),\ j=1,\dots,k$, from the second biggest one to the fifth biggest one; the improvement gain with respect to the partitioning rule $p1$ in the first row (solution algorithm considered in [18]) is about 92%, while the improvement with respect to the partitioning rule $p3$ in the first row (algorithm in [9]) is about 56%.

5. Conclusion

In this paper a computational experience regarding a branch and reduce approach for solving a class of low rank d.c. optimization programs is provided. It is shown that, in the case "dual-adequate" primitives are available, Lagrangean cuts highly improve the overall performance of the branch and reduce scheme, obtaining results better than the ones in [9, 18]. In particular, it is worth using the Lagrangean cuts for both the bounds and the feasible region, and in combination with some "Resize" operations. It is also pointed out that the partitioning rule $p5$ should be preferred to the "ω-subdivision" commonly used in the literature, and that the "*Resize*()" sub-procedure should be applied to set of indices I not containing the index corresponding to the maximum error.

References

[1] I. Bomze, M. Locatelli, (2004): Undominated d.c. Decompositions of Quadratic Functions and Applications to Branch-and-Bound Approaches, *Computational Optimization and Applications*, 28, 227-245

[2] R. Cambini, F. Salvi, (2010): Solving a class of low rank d.c. programs via a branch and bound approach: a computational experience, *Operations Research Letters*, 38 (5), 354-357

[3] R. Cambini, F. Salvi, (2009): A branch and reduce approach for solving a class of low rank d.c. programs, *Journal of Computational and Applied Mathematics*, 233, 492-501

[4] R. Cambini, C. Sodini, (2002): A finite algorithm for a particular d.c. quadratic programming problem, *Annals of Operations Research*, 117, 33-49

[5] R. Cambini, C. Sodini, (2005): Decomposition methods for solving nonconvex quadratic programs via branch and bound, *Journal of Global Optimization*, 33, 313-336

[6] R. Cambini, C. Sodini, (2008): A computational comparison of some branch and bound methods for indefinite quadratic programs, *Central European Journal of Operations Research,* 16, 139-152

[7] J. E. Falk, R. M. Soland, (1969): An algorithm for separable nonconvex programming problems, *Management Science*, 15, 550-569

[8] C.A. Floudas, P. M. Pardalos, (1999): Handbook of Test Problems in Local and Global Optimization, Nonconvex Optimization and Its Applications, vol. 33, Springer Berlin

[9] X. Honggang, X. Chengxian, (2005): A branch and bound algorithm for solving a class of D-C programming Applied Mathematics and Computation, 165, 29-302

[10] R. Horst, P. M. Pardalos, (1995): Handbook of Global Optimization, Nonconvex Optimization and Its Applications, vol. 2, Kluwer Academic Publishers, Dordrecht

[11] R. Horst, N. V. Thoai, (1999): D.C. programming: Overview, *Journal of Optimization Theory and Applications*, 103, 1-43

[12] R. Horst, H. Tuy, (1990): Global optimization deterministic approaches, Springer-Verlag

[13] F. A. A. Khayyal, H. D. Sherali, (2000): On finitely terminating branch and bound algorithms for some global optimization problems, *SIAM Journal Optimization,* 10, 1049-1057

[14] H. Konno, P.T. Thach, H. Tuy, (1997): Optimization on low rank nonconvex structures, Nonconvex Optimization and Its Applications, vol. 15, Kluwer Academic Publishers, Dordrecht

[15] H. Konno, A. Wijayanayake, (2002): Portfolio optimization under d.c. transaction costs and minimal transaction unit constraints, *Journal of Global Optimization,* 22, 137-154

[16] M. Minoux, (1986): Mathematical Programming Theory and Algorithms, Wiley-Intersciences Publication

[17] J. Parker, N. V. Sahinidis, (1998): A Finite Algorithm for Global Minimization of Separable Concave Programs, *Journal of Global Optimization,* 12, 1-36

[18] T.Q. Phong, L.T. Hoai An, P.D. Tao, (1995): Decomposition branch and bound method for globally solving linearly constrained

indefinite quadratic minimization problems, Operations Research Letters, 17, 215-220

[19] R.T. Rockafellar, (1972): Convex Analysis, Princeton University Press, second edition

[20] H.S. Ryoo, N. V. Sahinidis, (1996): A branch-and-reduce approach to global optimization, *Journal of Global Optimization*, 8, 107-138

[21] H.S. Ryoo, N. V. Sahinidis, (2003): Global optimization of multiplicative programs, *Journal of Global Optimization*, 26, 387-418

[22] H. Tuy, (1996): A general d.c. approach to location problems, State of the art in global optimization, edited by C.A. Floudas, P. M. Pardalos, Nonconvex Optimization and Its Applications, vol. 7, pp. 413-432, Kluwer Academic Publishers, Dordrecht

[23] H. Tuy, (1998): Convex Analysis and Global Optimization, Nonconvex Optimization and its Applications, vol. 22, Kluwer Academic Publishers, Dordrecht

Received ?