Università degli Studi di Pisa
Dipartimento di Statistica e Matematica
Applicata all'Economia

Report n. 321

Solving a class of low rank d.c. programs via a branch and reduce approach: a computational study

Riccardo Cambini and Francesca Salvi

Pisa, luglio 2009
- Stampato in Proprio –

Via Cosimo Ridolfi, 10 – 56124 PISA – Tel. Segr. Amm. 050 2216231 Segr. Stud. 050 2216317 Fax 050 2216375
Cod. Fisc. 80003670504 - P. IVA 00286820501 - Web http://statmat.ec.unipi.it/

Solving a class of low rank d.c. programs via a branch and reduce approach: a computational study

Riccardo Cambini and Francesca Salvi

Department of Statistics and Applied Mathematics
Faculty of Economics, University of Pisa
Via Cosimo Ridolfi 10, 56124 Pisa, ITALY
e-mail: cambric@ec.unipi.it, francesca.salvi@unifi.it

Abstract

D.C. programs have been widely studied in the recent literature due to their importance in applicative problems. In this paper a branch and reduce approach for solving a class of d.c. problems is considered. Seven partitioning rules and some acceleration devices are analyzed. The results of a computational study are provided in order to point out the performance effectiveness of both partitioning rules and acceleration devices.

Key words: d.c. programming, branch and reduce.
AMS - 2000 Math. Subj. Class. 90C30, 90C26.
JEL - 1999 Class. Syst. C61, C63.

1 Introduction

The so called d.c. programming is one of the main topics in the recent optimization literature. There is no need to recall its relevance from both a theoretical (see for all [9]) and an applicative point of view (see for example [1, 3, 5, 7, 8, 10, 12, 13, 19, 20] and references therein). In this paper the following d.c. program is considered:

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

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

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 [16] the particular case of $c(x) = \frac{1}{2}x^T Q x + q^T x$, with $q \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n \times n}$ symmetric and positive semi-definite,

The following result provides an estimation of the error done by solving the relaxed problem. With this aim the next function will be used:

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

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

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

Then, $f_B(\overline{x}) \leq f(x^*) \leq f(\overline{x})$, that is to say that $0 \leq f(x^*) - f_B(\overline{x}) \leq Err_B(\overline{x})$.

In order to proceed in the iterations of the branch and bound process it will be useful to consider the following further error function:

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

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

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

Procedure **DcBranch**(inputs: P; outputs: Opt, $OptVal$)
fix the tolerance parameter $\epsilon > 0$;
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*
($f_B(x_B), \alpha, \beta, x_B, r, X$):=Select();
if $f_B(x_B) < UB$ and $\left| \frac{UB - f_B(x_B)}{UB} \right| > \epsilon$ *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.

Notice that $2k$ linear programs are needed to determine the starting vectors $\tilde{\alpha}, \tilde{\beta} \in \Re^k$. The sub-procedure named "*Select*()" extracts from the stack the sub-problem 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 "*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 $\epsilon > 0$ is also used, avoiding the study when $\left|\frac{UB - f_B(x_B)}{UB}\right| \leq \epsilon$. 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 $r = \arg\max_{i=1,\ldots,k}\{Err_B(x_B, i)\}$. In this light, notice that condition $\left|\frac{UB - f_B(x_B)}{UB}\right| > \epsilon$ 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.

Finally, it is worth recalling that a necessary condition for the convergence of a branch and bound algorithm is the exhaustiveness of the subdivision process (see for all [10]). In order to guarantee such a convergence, either particular subdivision criteria have to be chosen or a tolerance parameter $\epsilon > 0$ has to be used in order to get a solution "sufficiently close" to the optimum (see for example [11]). In this light, the tolerance parameter $\epsilon > 0$ is used in order to guarantee the numerical convergence of the algorithm in reasonable time.

3 Branch and Reduce Acceleration Devices

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. The aim of these acceleration devices is twofold. In procedure "*DcBranch*()" we can reduce the bounds by means of the optional subprocedure "*Resize*()". In procedure "*Analyze*()" two optional sub-procedures, named "*CutBounds*()" and "*CutRegion*()", can be used in order to cut the bounds and, eventually, the feasible region itself by means of duality results.

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 $\tilde{\beta}_i = \max_{x \in X}\{d_i^T x\}$, $i = 1, \ldots, k$. 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 $X \cap B(\alpha, \beta)$. This produces a "not good" underestimation function $f_B(x)$ and hence an error function $Err_B(x)$ "too big". This affects the performance of the solution method, since the branch iterations are stopped when the error provided by the relaxed problems results to be sufficiently small.

In this light, in order to improve the performance of the algorithm we could periodically recalculate the values of (α, β) with respect to the considered feasi-

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

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 [18]).

As it has been shown, these cuts can be applied to the bounds $\alpha_i \leq d_i^T x \leq \beta_i$, $i = 1, \ldots, k$, 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 $v_j := \min_{x \in X} \{A_j x\}\ \forall j \in \{1, \ldots, m\}$. Notice finally that many solvers automatically provides the K-K-T multipliers corresponding to the optimal solution, making the calculus of the described cuts extremely efficient.

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 $Err_B(x,j)$, $j = 1,\ldots,k$; "$1^{st} - 10^{th}$" means that the set I is composed by all of the ten indices $1,\ldots,10$; "$2^{nd} - 5^{th}$" means that the set I is made by 4 indices corresponding to the errors $Err_B(x,j)$, $j = 1,\ldots,k$, 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 $p1 - p7$.

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.

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

- the "$\omega-$subdivision" process $p1$ proposed and used in [15, 16] 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;

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
$1^{st} - 10^{th}$	*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

Table 2: Average CPU time spent ($k = 10, n = m = 15$)

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

[9] R. Horst, N. V. Thoai, (1999): *D.C. programming: Overview*, Journal of Optimization Theory and Applications, vol. 103, No 1, 1-43

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

[11] F. A. A. Khayyal, H. D. Sherali, (2000): *On finitely terminating branch and bound algorithms for some global optimization problems*, SIAM Journal Optimization, vol. 10, No. 4, 1049-1057

[12] 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

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

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

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

[16] 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, pp. 215-220

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

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

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

[20] 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

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