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

Report n. 320

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

Riccardo Cambini and Francesca Salvi

Pisa, luglio 2009
- Stampato in Proprio –

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

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

Various classes of d.c. programs have been studied in the recent literature due to their importance in applicative problems. In this paper we consider a branch and bound approach for solving a class of d.c. problems. Both stack policies and partitioning rules are analyzed, pointing out their performance effectiveness by means of the results of a computational experience.

Key words: d.c. programming, branch and bound.
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 [7]) and an applicative point of view (see for example [1, 5, 6, 8, 10, 11, 14, 15] 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 [13] the particular case of $c(x) = \frac{1}{2}x^T Qx + q^T x$, with $q \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n\times n}$ symmetric and positive semi-definite, has been studied in the following form (where $d_i^T x = y_i$ for all $i = 1, \ldots, k$):

$$P_Q : \begin{cases} min\ f(x) = \frac{1}{2}x^T Qx + q^T x - \sum_{i=1}^{k} g_i(y_i) \\ (x, y) \in X \times Y \subseteq \mathbb{R}^n \times \mathbb{R}^k \\ Y = \{y \in \mathbb{R}^k : y_i = d_i^T x, x \in X\} \end{cases} \tag{2}$$

The concave case $Q = 0$ of problem (2) has been also analyzed in [12].

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

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

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.

```
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 $\forall i \in \{1, \ldots, k\}$:

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

  Analyze($\tilde{\alpha}, \tilde{\beta}$);
  while the stack is nonempty do
    $(f_B(x_B), \alpha, \beta, x_B, r)$:=Select();
    if $f_B(x_B) < UB$ and $\left|\frac{UB - f_B(x_B)}{UB}\right| > \epsilon$ then
      $\alpha1 := \alpha$; $\beta1 := \beta$; $\alpha2 := \alpha$; $\beta2 := \beta$;
      $\gamma$:=Split($\alpha_r, \beta_r$); $\beta1_r := \gamma$; $\alpha2_r := \gamma$;
      Analyze($\alpha1, \beta1$); Analyze($\alpha2, \beta2$);
    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 subproblem to be eventually branched; how this can be efficiently done will be discussed later. 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" [4, 16]). The way γ is determined will be the core of the following Section 3. 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.

$i = 1, \dots, k$, 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. The test counted 250 random instances generated and solved for functions f_1 and f_2, and 1000 random instances generated and solved for function f_3. The average numbers of relaxed problems solved and the average CPU time needed to solve the problems are given as results of the test.

3.1 Stack management

As it is very well known, a stack can be managed with various policies, such as FIFO, LIFO, Priority , and so on. Such a kind of methods have been approached in the literature in different ways, for example in [2] a recursive procedure has been proposed, while in [13] a priority stack ordered with respect to the value $f_B(x_B)$ is used. For this very reason, it is worth to preliminarily verify the more efficient policy to be used for managing the stack. For the sake of convenience, we considered the same partitioning rule used in [2, 13], that is the ω-subdivision. We considered three different opportunities:

- priority stack: as smaller is the value of $f_B(x_B)$ as bigger is the priority;
- LIFO: the last appended subproblem is first considered;
- recursive: a LIFO policy is implicitly implemented by means of a recursive procedure.

These policies have been computationally tested and the results are summarized in Table 1.

The obtained results point out that, for all the three functions and for all the considered values of k, the more efficient policy is the priority one while the worst is the recursive, for both the average of the number of relaxed problems solved and of the CPU time spent. As a consequence, from now on the study will be based on the priority stack previously described.

3.2 Partitioning Rules

In the literature various kinds of partitioning criteria have been studied. Specifically speaking, Phong, Hoai-An and Tao in [13] compared three different partitioning rules in the case of indefinite quadratic programs, named as "Exhaustive bisection", "ω-subdivision" and "Adaptive bisection". Their brief computational tests suggested that the better performance is provided by the "ω-subdivision". Cambini and Sodini in [2, 3] used the "ω-subdivision" to solve indefinite quadratic programs with recursive procedures.

In this paper we propose to compare the "ω-subdivision" with 6 more different partitioning criteria. With this aim let us define the following three values:

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

	k	p1	p2	p3	p4	p5	p6	p7
f_1	3	**29.776** (**0.441**)	*88.224* (1.2126)	*88.224* (1.338)	63.288 (0.88548)	63.288 (0.9758)	*88.224* (*1.3398*)	71.368 (1.0943)
f_1	4	**68.48** (**0.98296**)	*141.32* (1.964)	*141.32* (*2.1698*)	105.24 (1.4795)	105.24 (1.6331)	*141.32* (2.1673)	116.57 (1.8038)
f_1	5	**145.63** (**2.0848**)	*217.67* (3.0506)	*217.67* (3.3728)	163.96 (2.3204)	163.96 (2.5684)	*217.67* (*3.3743*)	179.66 (2.81)
f_1	6	311.22 (4.5413)	*315.81* (4.4338)	*315.81* (4.9146)	**244.34** (**3.4806**)	**244.34** (3.8486)	*315.81* (*4.9182*)	266.12 (4.1858)
f_1	7	*596.18* (*8.8948*)	439.61 (6.228)	439.61 (6.9066)	**347.7** (**4.9814**)	**347.7** (5.5175)	439.61 (6.9101)	368.48 (5.8371)
f_1	8	*1213.8* (*18.533*)	617.13 (8.7311)	617.13 (9.7007)	**504.46** (**7.2368**)	**504.46** (8.0324)	617.13 (9.7027)	532.6 (8.4477)
f_1	9	*2362.8* (*37.4*)	841.94 (12.037)	841.94 (13.382)	**704.56** (**10.207**)	**704.56** (11.337)	841.94 (13.394)	732.17 (11.742)
f_2	3	**19.448** (**0.30832**)	*103.82* (1.461)	*103.82* (*1.636*)	73.296 (1.0431)	72.792 (1.1634)	*103.82* (1.635)	82.2 (1.3081)
f_2	4	**40.6** (**0.61496**)	*163.07* (2.4006)	161.33 (*2.6659*)	116.34 (1.7392)	113.62 (1.8904)	159.43 (2.6284)	128.86 (2.1428)
f_2	5	**84.76** (**1.2641**)	*230.64* (3.5162)	225.13 (*3.8518*)	168.5 (2.5925)	159.93 (2.748)	224.52 (3.8438)	182.75 (3.1427)
f_2	6	**176.86** (**2.6324**)	*314.31* (4.8539)	302.32 (5.2284)	233.62 (3.629)	218.93 (3.8284)	302.73 (*5.2585*)	250.62 (4.3727)
f_2	7	354.02 (5.3502)	*411.86* (6.3712)	389.94 (6.8126)	311.29 (**4.8497**)	**286.73** (5.0589)	391.95 (*6.8782*)	326.19 (5.7431)
f_2	8	*699.47* (*10.858*)	525.55 (8.0875)	490.36 (8.5968)	404.26 (**6.2725**)	**365.98** (6.4609)	495.92 (8.7032)	412.3 (7.2688)
f_2	9	*1348.2* (*21.471*)	659.22 (10.047)	605.57 (10.577)	527.26 (8.0937)	**460.19** (**8.087**)	618.02 (10.798)	520.99 (9.139)
f_3	3	17.136 (0.21798)	*20.426* (*0.25029*)	**8.444** (**0.17316**)	18.204 (0.22805)	10.094 (0.19558)	12.804 (0.2346)	12.696 (0.23309)
f_3	4	32.284 (0.38698)	*36.314* (*0.42518*)	**12.878** (**0.25615**)	32.64 (0.38927)	16.422 (0.3063)	21.016 (0.37464)	21.142 (0.37588)
f_3	5	56.376 (*0.65404*)	*56.752* (0.65117)	**18.43** (**0.35798**)	52.04 (0.60365)	24.868 (0.45301)	30.854 (0.54458)	31.942 (0.56076)
f_3	6	*96.56* (*1.1023*)	86.806 (0.9855)	**25.68** (**0.48806**)	81.116 (0.92544)	36.752 (0.6563)	45.186 (0.78876)	47.898 (0.83045)
f_3	7	*162.7* (*1.8608*)	131.11 (1.4817)	**35.182** (**0.65552**)	125.77 (1.426)	53.078 (0.9338)	65.236 (1.1302)	70.07 (1.2038)
f_3	8	*272.18* (*3.1666*)	195.42 (2.2193)	**47.456** (**0.8719**)	190.98 (2.1732)	76.076 (1.3273)	90.374 (1.5674)	100.24 (1.7241)
f_3	9	*462.53* (*5.5116*)	295.46 (3.3771)	**64.146** (**1.1626**)	295.4 (3.3876)	109.59 (1.906)	128.22 (2.2248)	146 (2.5178)

Table 2: Partitioning rules comparison

These results suggest the following remarks:

- it is quite obvious that the efficiency of the method depends on the considered classes of objective functions;
- the ω-subdivisions are effective just for small values of k ($k \leq 6$) and for functions f_1 and f_2;

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

[5] 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

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

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

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

[9] 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

[10] 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

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

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

[13] 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

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

[15] 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

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