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

Report n. 322

A branch and reduce approach for solving a class of low rank d.c. programs

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/

A branch and reduce approach for solving a class of low rank d.c. programs

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 reduce approach for solving a class of d.c. problems. Seven partitioning rules are analyzed and some techniques aimed to improve the overall performance of the algorithm are proposed. The results of a computational experience are provided in order to point out the performance effectiveness of the proposed techniques.

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 [16]) and an applicative point of view (see for example [3, 6, 15, 17, 19, 20, 26, 27] 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 [1] 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 [2] this class of programs have been approached by means of a branch and reduce method based on Lagrangean cuts. In [23] the particular case of $c(x) = \frac{1}{2}x^T Q x + q^T x$, with $q \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n \times n}$

Notice that $f_B(x)$ is an underestimation function 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:

$$P_B(\alpha,\beta): \begin{cases} \min f_B(x) \\ x \in X \cap B(\alpha,\beta) \end{cases} \tag{3}$$

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

2.1 The main procedures

The following main procedure "*DcBranch*()" can 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, *DepthStep*; 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 $\forall i \in \{1,\ldots,k\}$:

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

compute $v_j := \min_{x \in X}\{A_j x\}$ $\forall j \in \{1,\ldots,m\}$;
$depth := 0$;
Analyze($\tilde{\alpha}, \tilde{\beta}, depth$);
while the stack is nonempty *do*
($f_B(x_B), \alpha, \beta, x_B, r, X, depth$):=Select();
if $f_B(x_B) < UB$ and $\left|\frac{UB - f_B(x_B)}{UB}\right| > \epsilon$ *then*
if *depth* is multiple of *DepthStep* *then*
$(\alpha,\beta) := Resize(\alpha,\beta,I,X)$;
end if;
$\alpha 1 := \alpha$; $\beta 1 := \beta$; $\alpha 2 := \alpha$; $\beta 2 := \beta$;
γ:=Split(α_r, β_r); $\beta 1_r := \gamma$; $\alpha 2_r := \gamma$;

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.

Procedure **Analyze**(inputs: α, β, *depth*)
  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 $\left|\frac{UB - f_B(x_B)}{UB}\right| > \epsilon$ *then*
    $(\alpha, \beta) := CutBounds()$; update $f_B(x)$ over $B(\alpha, \beta)$;
    $X := CutRegion()$;
    $r := \arg\max_{i=1,\ldots,k}\{Err_B(x_B, i)\}$;
    $depth := depth + 1$;
    Append$(f_B(x_B), \alpha, \beta, x_B, r, X, depth)$;
  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 $\epsilon > 0$ 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 sub-procedures named "*CutBounds*()" and "*CutRegion*()" which has been proposed in [2] in order to improve the performance of the method by properly reducing the bounds α, β and the feasible region X by means of the use of duality results. These sub-procedures will be described in details in Subsection 2.3.

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 [17]). 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 [18]). In this light, the tolerance parameter $\epsilon > 0$ is used in order to guarantee the numerical convergence of the algorithm in reasonable time.

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 [25]).

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*()" 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.

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

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

3 Computational results

The procedures 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 $n = 10$ variables,

δ	k	DS	*Resize*	p1	p2	p3	p4	p5	p6	p7
1	6	1	2^{nd}	*106.65*	71.882	72.01	69.147	**69.108**	72.02	69.294
1	6	1	$2^{nd} - 3^{rd}$	*76.02*	57.363	57.294	55.922	55.922	57.412	**55.098**
1	6	1	$2^{nd} - 4^{th}$	*65.363*	50.696	50.657	48.882	48.873	50.696	**48.363**
1	6	1	$2^{nd} - 5^{th}$	*60.804*	47.333	47.412	44.843	**44.833**	47.324	45.324
1	6	1	$1^{st} - k^{th}$	*57.549*	45.902	45.98	44.245	44.245	45.961	**44.157**
1	6	2	2^{nd}	*142.37*	88.314	88.382	84.088	**84.039**	88.412	85.118
1	6	2	$2^{nd} - 3^{rd}$	*110.85*	73.167	73.314	**70.333**	70.412	73.225	71.118
1	6	2	$2^{nd} - 4^{th}$	*94.471*	64.833	64.833	62.363	**62.304**	64.824	62.775
1	6	2	$2^{nd} - 5^{th}$	*90.578*	61.147	61.363	58.824	**58.735**	61.314	59.029
1	6	2	$1^{st} - k^{th}$	*86.088*	59.48	59.52	**57.863**	57.971	59.588	57.902
1	9	1	2^{nd}	*455.55*	194.13	193.73	198.68	198.65	193.93	**193.63**
1	9	1	$2^{nd} - 3^{rd}$	*268.96*	139.86	**139.82**	143.81	143.76	139.97	140.4
1	9	1	$2^{nd} - 4^{th}$	*192.96*	**116.62**	116.65	117.13	117.12	116.63	116.69
1	9	1	$2^{nd} - 5^{th}$	*159.47*	102.51	102.43	102.45	102.47	**102.35**	103.08
1	9	1	$1^{st} - k^{th}$	*125.06*	84.333	84.441	83.765	83.755	84.294	**83.343**
1	9	2	2^{nd}	*723.59*	251.48	**251.25**	257.31	257.17	251.73	251.74
1	9	2	$2^{nd} - 3^{rd}$	*459.76*	194.62	194.64	198.54	198.65	194.77	**193.88**
1	9	2	$2^{nd} - 4^{th}$	*339.51*	162.8	162.65	166	165.96	162.75	**162.17**
1	9	2	$2^{nd} - 5^{th}$	*280.52*	141.66	**141.55**	145.73	145.9	141.65	142.75
1	9	2	$1^{st} - k^{th}$	*220.32*	115.89	115.92	119.26	119.34	**115.75**	118.15
1	12	1	2^{nd}	*1332.9*	348.79	**348.75**	376.89	376.99	**348.75**	358.7
1	12	1	$2^{nd} - 3^{rd}$	*646.56*	**240.76**	240.92	257.49	257.62	240.85	249.94
1	12	1	$2^{nd} - 4^{nd}$	*433*	191.38	191.37	200.7	200.5	**191.28**	195.48
1	12	1	$2^{nd} - 5^{nd}$	*324.69*	164.95	165.38	171.23	171.51	**164.89**	166.27
1	12	1	$1^{st} - k^{nd}$	*185.9*	**112.92**	113.05	116.2	116.26	113.04	113.29
1	12	2	2^{nd}	*2556.8*	**477.17**	477.76	516.79	517.34	477.82	491.94
1	12	2	$2^{nd} - 3^{rd}$	*1346.5*	350.89	**350.43**	380.7	380.78	350.48	360.36
1	12	2	$2^{nd} - 4^{nd}$	*899.97*	283.71	283.99	305.54	305.3	**283.6**	294.5
1	12	2	$2^{nd} - 5^{nd}$	*661.12*	**241.09**	**241.09**	260.62	260.56	241.13	250
1	12	2	$1^{st} - k^{nd}$	*359.1*	160.77	**160.76**	171.43	171.33	160.92	166.2
2	6	1	2^{nd}	*56.822*	42.248	36.109	40.297	**35.03**	38.812	37.733
2	6	1	$2^{nd} - 3^{rd}$	*39.208*	33.861	28.208	31.812	**28.109**	30.851	30.129
2	6	1	$2^{nd} - 4^{th}$	*32.465*	29.842	24.95	27.337	**24.545**	27.366	26.614
2	6	1	$2^{nd} - 5^{th}$	*30.287*	27.95	23.406	25.594	**22.97**	25.673	24.861
2	6	1	$1^{st} - k^{th}$	*29.168*	26.861	22.812	24.901	**22.663**	25.04	24.178
2	6	2	2^{nd}	*78.149*	53.356	44.614	50.178	**42.941**	49.188	46.307
2	6	2	$2^{nd} - 3^{rd}$	*56.545*	43.317	36.139	40.475	**35.495**	39.713	37.723
2	6	2	$2^{nd} - 4^{th}$	*48.881*	37.327	32.653	36.04	**31.584**	35.188	34.149
2	6	2	$2^{nd} - 5^{th}$	*46.624*	34.881	30.287	33.465	**29.485**	32.376	31.733
2	6	2	$1^{st} - k^{th}$	*44.158*	34.069	29.574	32.98	**29.475**	31.495	31.337
2	9	1	2^{nd}	*189.56*	82.455	**68.723**	83.347	71.98	74.822	75.208
2	9	1	$2^{nd} - 3^{rd}$	*106.64*	59.762	**50.495**	58.564	51.208	55	54.455
2	9	1	$2^{nd} - 4^{th}$	*74.376*	49.515	**42.297**	47.465	42.416	46.149	44.851
2	9	1	$2^{nd} - 5^{th}$	*60.644*	43.901	**37.416**	41.822	37.653	40.941	39.891
2	9	1	$1^{st} - k^{th}$	*47.05*	36.832	**31.059**	34.307	31.436	34.248	32.485
2	9	2	2^{nd}	*334.19*	110.56	**92.426**	113.24	93.04	100.78	99.545
2	9	2	$2^{nd} - 3^{rd}$	*192.26*	83.554	**70.495**	83.485	71.178	75.743	76.584
2	9	2	$2^{nd} - 4^{th}$	*140.05*	68.792	**58.099**	69.178	59.762	63.03	63.366
2	9	2	$2^{nd} - 5^{th}$	*108.19*	60.238	**51.644**	59.782	52.327	56.218	55.842
2	9	2	$1^{st} - k^{th}$	*82.238*	50.099	**43.03**	48.376	43.891	46.327	44.881
2	12	1	2^{nd}	*629.94*	175.54	**135.09**	184.67	145.56	151.32	156.36
2	12	1	$2^{nd} - 3^{rd}$	*279.05*	116.65	**92.238**	116.71	96.713	102.23	104.12
2	12	1	$2^{nd} - 4^{th}$	*173.2*	91.604	**74.208**	90.139	76.356	82.297	82.168
2	12	1	$2^{nd} - 5^{th}$	*123.87*	78.792	**63.842**	74.96	64.347	71.149	69.396
2	12	1	$1^{st} - k^{th}$	*69.644*	56.03	45.455	50.95	**44.446**	49.475	47.356
2	12	2	2^{nd}	*1388.5*	248	**191.94**	267.68	203.05	212.08	223.61
2	12	2	$2^{nd} - 3^{rd}$	*648.9*	171.82	**133.98**	183.98	144.34	150.43	156.81
2	12	2	$2^{nd} - 4^{th}$	*389.48*	137.41	**111.37**	143.1	116.04	120.83	125.08
2	12	2	$2^{nd} - 5^{th}$	*271.56*	116.53	**94.861**	116.08	97.337	104.92	104.92
2	12	2	$1^{st} - k^{th}$	*125.46*	76.356	**63.693**	74.604	65.257	70.495	67.436

Table 1: Average number of relaxed subproblems solved ($n = m = 10$)

- in the case $k = 6$ the "Resize" operations giving the best performances are the ones with $I = \{2, 3\}$, in the case $k = 9$ we should choose $I = \{2, 3, 4\}$, while for $k = 12$ it seems that the best results are obtained with $I = \{2, 3, 4, 5\}$; in other words, as bigger is the value of k as more "Resize" operations should be done to improve the performance of the branch and reduce method.

4 The DCA approach

In the recent literature a solution algorithm for d.c. problems have been proposed by L.T. Hoai An and T.Q. Phong. Many papers dealing with such an algorithm appeared in the literature of global optimization from both a theoretical and an applicative point of view [7, 8, 9, 10, 11, 12, 13, 14]. The interest of the literature of global optimization in that algorithm deserves a comparison with the branch and reduce approach proposed in this paper, taking into account that the so called "DCA" method guarantees that just a local optimum is found.

The "DCA" method deals with a general unconstrained d.c. optimization problem having an objective function of the kind $f(x) = c(x) - g(x)$, with c and g convex functions. Constrained problems are suggested to be managed by adding in the objective function a proper penalty function. The method is based on conjugacy and duality theory of d.c. programming. Such a general method can be specified for the class of programs considered in this paper as described in the following procedure "$DCA()$".

Procedure **DCA**(inputs: P; outputs: Opt, $OptVal$)
fix the tolerance parameter $\epsilon > 0$;
determine the starting vectors $\tilde{\alpha}, \tilde{\beta} \in \Re^k$ such that $\forall i \in \{1, \ldots, k\}$:

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

determine a starting feasible point x^0 and set $t := 0$;
repeat
$t := t + 1$;
choose $w \in \partial g(x^{t-1})$ and set $x^t := \arg\min_{x \in X}\{c(x) - w^T x\}$
until $\frac{\|x^t - x^{t-1}\|}{\|x^{t-1}\|} \leq \epsilon$ or $\frac{|f(x^t) - f(x^{t-1})|}{|f(x^{t-1})|} \leq \epsilon$
$Opt := x^t$ and $OptVal := f(x^t)$;
end proc.

Notice that in the case $g(x)$ is differentiable then $\partial g(x^{t-1}) = \{\nabla g(x^{t-1})\}$ and hence in procedure "$DCA()$" we have $w = \nabla g(x^{t-1})$.

Procedure $DCA()$" has been computationally compared with the procedure "$DcBranch()$" previously proposed. Since "$DCA()$" guarantees just a local optimum, it has been compared also with the "fmincon" MatLab function. We considered problems with $n = 15$ variables, $m = 15$ inequality constraints, box constraints $l \leq x \leq u$, no equality constraints and objective function $f(x) = \frac{1}{2}x^T Q x + q^T x - \sum_{i=1}^{k} \lambda_i (d_i^T x + d_i^0)^4$, with $k = 10$ and $Q \in \mathbb{R}^{n \times n}$ symmetric and positive semi-definite. The problems have been randomly generated as described in Section 3. For the various instances 5000 randomly generated problems have been solved. The average CPU time needed to solve

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

[6] 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

[7] L. T. Hoai An, (2000): *An efficient algorithm for globally minimizing a quadratic function under convex quadratic constraints*, Mathematical Programming Series A 87, 401-426

[8] L. T. Hoai An et al, (2002): *Towards Tikhonov regularization of non linear ill posed problems: a d.c. programming approach*, C. R. Acad. Sci. Paris, Ser. I 335, 1073-1078

[9] L. T. Hoai An, M. T. Belghiti, P. D. Tao, (2007): *A new efficient algorithm based on d.c. programming and DCA for clustering*, Journal of Global Optimization, 37, 593-608

[10] L. T. Hoai An, N. T. Phuc, P. D. Tao, (2007): *A continuous d.c. programming approach to the strategic supply chain design problem from qualified partner set*, European Journal of Operational Research, 183, 1001-1012

[11] L. T. Hoai An, P. D. Tao, (1997): *Solving a class of linearly constrained indefinite quadratic problems by d.c. algorithms*, Journal of Global Optimization, 11, 253-285

[12] L. T. Hoai An, P. D. Tao, (2002): *D.C. programming approach for multicommodity network optimization problems with step increasing cost functions*, Journal of Global Optimization, 22, 205-232

[13] L. T. Hoai An, P. D. Tao, (2008): *A continuous approach for the concave cost supply problem via d.c. programming and DCA*, European Journal of Operational Research, 156, 325-338

[14] L. T. Hoai An, P. D. Tao, D. N. Hao, (2003): *Solving an Inverse Problem for an Elliptic Equation by d.c. programming*, Journal of Global Optimization, 25, 407-423

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

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

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

[18] 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

[19] 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

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

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

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