Kaisa Joki | Adil M. Bagirov | Napsu Karmitsa | Marko M. Mäkelä

New Proximal Bundle Method for Non-smooth DC Optimization

Turku Centre *for* Computer Science

TUCS Technical Report
No 1130, February 2015

New Proximal Bundle Method for Nonsmooth DC Optimization

Kaisa Joki
University of Turku, Department of Mathematics and Statistics
FI-20014 Turku, Finland
kaisa.joki@utu.fi
Adil M. Bagirov
Faculty of Science and Technology, Federation University Australia
University Drive, Mount Helen, PO Box 663, Ballarat, VIC 3353, Australia
a.bagirov@federation.edu.au
Napsu Karmitsa
University of Turku, Department of Mathematics and Statistics
FI-20014 Turku, Finland
napsu@karmitsa.fi
Marko M. Mäkelä
University of Turku, Department of Mathematics and Statistics
FI-20014 Turku, Finland
makela@utu.fi

TUCS Technical Report
No 1130, February 2015

Abstract

In this paper, we develop a version of the bundle method to locally solve unconstrained difference of convex (DC) programming problems. It is assumed that a DC representation of the objective function is available. Our main idea is to use subgradients of both the first and second components in the DC representation. This subgradient information is gathered from some neighborhood of the current iteration point and it is used to build separately an approximation for each component in the DC representation. By combining these approximations we obtain the so-called nonconvex cutting plane model of the original objective function. We design the proximal bundle method for DC programming based on this approach and prove the convergence of the method to an ε-critical point. This algorithm is tested using some academic test problems.

Keywords: Nonsmooth optimization, Nonconvex optimization, Proximal bundle methods, DC functions, Cutting plane model

TUCS Laboratory
Turku Optimization Group (TOpGroup)

1 Introduction

We design an algorithm for solving a minimization problem of the form

$$\begin{cases} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathbb{R}^n, \end{cases} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a difference of convex (DC) functions. This problem is called the *unconstrained DC programming problem*. The function f defined on $\mathbb{R}^n$ is called a *DC function* if it can be decomposed as the difference of two convex functions:

$$f(\boldsymbol{x}) = f_1(\boldsymbol{x}) - f_2(\boldsymbol{x}), \tag{2}$$

where f_1 and f_2 are convex functions on $\mathbb{R}^n$. In what follows, the functions f_1 and f_2 are called *convex DC components* of f and we assume that

$$\text{dom } f_i = \{\boldsymbol{x} \in \mathbb{R}^n \mid |f_i(\boldsymbol{x})| < +\infty\} = \mathbb{R}^n \quad \text{for } i = 1, 2. \tag{3}$$

Since the convex DC components of f are assumed to be finite on $\mathbb{R}^n$, the function f is also finite on the whole $\mathbb{R}^n$. In addition, the function f is typically nonconvex and it needs not to be differentiable. It is worth noting that if f is nonsmooth, then at least one of the functions f_1 and f_2 is a nonsmooth convex function. Such DC functions constitute a large and interesting subclass of nonsmooth nonconvex functions.

The class of DC functions is very broad. Any twice continuously differentiable function can be represented as a DC function. Moreover, any continuous function can be approximated by the sequence of DC functions [39]. Many optimization problems of potential interest can be expressed into the form of a DC program such as production-transportation planning, location planning, engineering design, cluster analysis, multilevel programming and multi-objective programming [7, 20, 32]. DC optimization algorithms have been proved to be particularly successful for analyzing and solving a variety of highly structured problems. Although in many optimization problems it is not easy to extract DC model of functions, there are some problems where such models can be written explicitly. Such problems include cluster analysis, nonparametric regression using piecewise linear functions and supervised data classification problems [7, 8] to name a few. Moreover, in [13], a special algorithm is developed to find the best DC representation of polynomials.

DC programming problems have been mainly considered in global optimization and some algorithms have been designed to find global solutions to such problems (see, e.g., [2, 21, 22, 39]). However, so far a little attention has been paid to the development of local solution methods for specifically DC programming problems. Such methods are often needed as a part of global optimization solvers, since they typically utilize local methods.

In this paper, our aim is to design a version of the bundle method to locally solve the unconstrained DC programming problem (1). The bundle method and its variations are among the most efficient methods in nonsmooth optimization (see [31] and references therein). To date these methods have been designed based only on the convex model of a function, also in the nonconvex case. To our best knowledge versions of the bundle methods based on the explicitly known structures of a problem have not been studied in depth before.

The novelty of our proximal bundle method is that the DC decomposition of the objective function is utilized explicitly in the model construction. Due to this the main idea is to approximate the subdifferentials of both DC components with a bundle. This means that we are maintaining two separate bundles which consists of subgradients from some neighborhood of the current iteration point. This subgradient information is used to construct separately a classical convex cutting plane model (see, e.g., [26, 29, 34]) for each DC component. Combining these approximations we obtain a piecewise affine nonconvex cutting plane model of the original objective function.

The algorithm of our proximal bundle method is based on the bundle methods introduced in [14, 15, 16] and it especially combines the features of those bundle methods with suitable modifications. Since the trust region approach is used in the form of the classical proximity parameter (see, e.g., [26, 34]), we need not to perform any line search procedure to determine a step size. In addition, the global convergence of the new method is established to an ε-critical point after a finite number of steps. The results of numerical experiments also show the good performance of the new method.

The rest of the paper is organized as follows: In Section 2 the main concepts used throughout the paper are described. Section 3 describes the new cutting plane model for DC functions. A minimization algorithm PBDC (Proximal Bundle method for DC functions) based on such models is studied in Section 4 and its convergence is proved in Section 5. Section 6 presents computational results using academic test problems and Section 7 contains some concluding remarks.

2 Preliminaries

In this section we recall some concepts and basic results from nonsmooth analysis and DC programming. For more details about these subjects we refer to [2, 8, 10, 17, 31, 33, 36]. The following notations will be used throughout the paper.

We denote by $\|\cdot\|$ the norm in the n-dimensional real Euclidean space $\mathbb{R}^n$ and by $\boldsymbol{x}^T\boldsymbol{y}$ the usual inner product of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. The open ball with center $\boldsymbol{x} \in \mathbb{R}^n$ and radius $r > 0$ is denoted by $B(\boldsymbol{x}; r)$. The distance

between a point $\boldsymbol{x} \in \mathbb{R}^n$ and a nonempty set $S \subset \mathbb{R}^n$ is defined as

$$d(\boldsymbol{x}, S) = \inf \{\|\boldsymbol{x} - \boldsymbol{s}\| \mid \boldsymbol{s} \in S\}.$$

Let S be a subset of $\mathbb{R}^n$. A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *Lipschitz continuous* on S, if there exist a constant $L > 0$ such that

$$|f(\boldsymbol{y}) - f(\boldsymbol{x})| \leq L\|\boldsymbol{y} - \boldsymbol{x}\| \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in S.$$

A function $f : \mathbb{R}^n \to \mathbb{R}$ is in turn *locally Lipschitz continuous* at a point $\boldsymbol{x} \in \mathbb{R}^n$, if there exist a constant $L > 0$ and some $\varepsilon > 0$ such that

$$|f(\boldsymbol{y}) - f(\boldsymbol{z})| \leq L\|\boldsymbol{y} - \boldsymbol{z}\| \quad \text{for all } \boldsymbol{y}, \boldsymbol{z} \in B(\boldsymbol{x}; \varepsilon).$$

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *convex*, if

$$f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \text{ and } \lambda \in [0, 1].$$

Since a convex function is locally Lipschitz continuous on $\mathbb{R}^n$ by Rademacher's theorem, it is differentiable almost everywhere. The *subdifferential* (or generalized gradient) of a convex function f at a point $\boldsymbol{x} \in \mathbb{R}^n$ is the set [33]

$$\partial_c f(\boldsymbol{x}) = \left\{\boldsymbol{\xi} \in \mathbb{R}^n \mid f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{\xi}^T(\boldsymbol{y} - \boldsymbol{x}) \text{ for all } \boldsymbol{y} \in \mathbb{R}^n\right\} \tag{4}$$

and each vector $\boldsymbol{\xi} \in \partial_c f(\boldsymbol{x})$ is called a *subgradient* of f at $\boldsymbol{x}$. The subdifferential $\partial_c f(\boldsymbol{x})$ is a nonempty, convex and compact set such that $\partial_c f(\boldsymbol{x}) \subset B(0; L)$, where $L > 0$ is the Lipschitz constant of f at $\boldsymbol{x}$. The subdifferential $\partial_c f(\boldsymbol{x})$ is a generalization of the classical derivative because if f is both convex and differentiable at some point $\boldsymbol{x} \in \mathbb{R}^n$, then the subgradient is unique and equals to the gradient, i.e. $\partial_c f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}$ [33].

We also define the ε-subdifferential of a convex function f which approximates the subdifferential. For $\varepsilon \geq 0$, the *ε-subdifferential* of a convex function f at a point $\boldsymbol{x} \in \mathbb{R}^n$ is given by [33]

$$\partial_\varepsilon f(\boldsymbol{x}) = \left\{\boldsymbol{\xi}_\varepsilon \in \mathbb{R}^n \mid f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{\xi}_\varepsilon^T(\boldsymbol{y} - \boldsymbol{x}) - \varepsilon \text{ for all } \boldsymbol{y} \in \mathbb{R}^n\right\}. \tag{5}$$

Each vector $\boldsymbol{\xi}_\varepsilon \in \partial_\varepsilon f(\boldsymbol{x})$ is called an *ε-subgradient* of f at $\boldsymbol{x}$. The set $\partial_\varepsilon f(\boldsymbol{x})$ contains now in a condensed form the subgradient information from some neighborhood of $\boldsymbol{x}$ as the following theorem shows.

THEOREM 2.1. [8] *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function with Lipschitz constant $L > 0$ at a point $\boldsymbol{x} \in \mathbb{R}^n$. If $\varepsilon \geq 0$, then*

$$\partial_c f(\boldsymbol{y}) \subset \partial_\varepsilon f(\boldsymbol{x}) \quad \textit{for all } \boldsymbol{y} \in B\left(\boldsymbol{x}; \tfrac{\varepsilon}{2L}\right). \tag{6}$$

A DC function f defined by (2) is also locally Lipschitz continuous on $\mathbb{R}^n$, i.e. it is Lipschitz continuous on every bounded set [14]. Now, the *generalized subdifferential* of a locally Lipschitz continuous function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $\boldsymbol{x} \in \mathbb{R}^n$ is defined by [10]

$$\partial f(\boldsymbol{x}) = \operatorname{conv}\{\boldsymbol{\xi} \in \mathbb{R}^n \,|\, \text{there exists } \{\boldsymbol{x}_i\} \subset \mathbb{R}^n \setminus \Omega_f \text{ such that } \boldsymbol{x}_i \to \boldsymbol{x} \text{ and } \nabla f(\boldsymbol{x}_i) \to \boldsymbol{\xi}\},$$

where conv denotes the convex hull of a set and Ω_f is the set where f is not differentiable. Especially $\partial f(\boldsymbol{x}) = \partial_c f(\boldsymbol{x})$ for convex f defined on $\mathbb{R}^n$ [10]. In what follows, we denote the subdifferential of a convex DC component f_i by $\partial f_i(\boldsymbol{x})$ for $i = 1, 2$.

Generally a point $\boldsymbol{x}^* \in \mathbb{R}^n$ is a *local minimizer* of the problem (1) if $f(\boldsymbol{x}^*) = f_1(\boldsymbol{x}^*) - f_2(\boldsymbol{x}^*)$ is finite and there exists an $\varepsilon > 0$ such that

$$f_1(\boldsymbol{x}^*) - f_2(\boldsymbol{x}^*) \leq f_1(\boldsymbol{x}) - f_2(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in B(\boldsymbol{x}^*; \varepsilon).$$

Since we made the assumption (3), the finiteness of f is always fulfilled. For a DC function we can also specify the following necessary optimality condition for local optimality and this condition is sufficient in that case when the DC component f_2 is a polyhedral convex function, that is, f_2 is of the form

$$f_2(\boldsymbol{x}) = \max_{i=1,\ldots,m} \left\{ \boldsymbol{a}_i^T \boldsymbol{x} - b_i \right\},$$

where $\boldsymbol{a}_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$ for $i = 1, \ldots, m$.

THEOREM 2.2. [2, 38] *Let f_1 and f_2 be convex functions. If $\boldsymbol{x}^* \in \mathbb{R}^n$ is a local minimizer of $f = f_1 - f_2$, then*

$$\partial f_2(\boldsymbol{x}^*) \subseteq \partial f_1(\boldsymbol{x}^*). \tag{7}$$

Furthermore, the condition (7) *guarantees local optimality if f_2 is a polyhedral convex function.*

The optimality condition (7) above is usually hard to be verified and for that reason it can be relaxed to the form [2, 18, 38]

$$\partial f_2(\boldsymbol{x}^*) \cap \partial f_1(\boldsymbol{x}^*) \neq \emptyset. \tag{8}$$

A point $\boldsymbol{x}^* \in \mathbb{R}^n$ which satisfies (8) is said to be a *critical point* of f. In a similar way, we can define an ε-critical point. Let $\varepsilon \geq 0$, a point $\boldsymbol{x}^* \in \mathbb{R}^n$ is said to be an *ε-critical point* of f defined by (2) if it satisfies [35]

$$\partial_\varepsilon f_2(\boldsymbol{x}^*) \cap \partial_\varepsilon f_1(\boldsymbol{x}^*) \neq \emptyset. \tag{9}$$

It is worth pointing out that the condition (9) reduces to (8), when $\varepsilon = 0$. Because conditions (8) and (9) are only necessary optimality conditions for

the unconstrained DC programming problem (1), a critical or an ε-critical point of f is not necessarily a local minimizer. However, each local minimizer of f can be found among the set of critical points of f and, similarly, also among the set of ε-critical points of f. In what follows, an optimization method is said to be *globally convergent*, if starting from any arbitrary point $\boldsymbol{x}_0 \in \mathbb{R}^n$ it generates a sequence $\{\boldsymbol{x}_k\}$ converging to an ε-critical point $\boldsymbol{x}^*$, that is, $\boldsymbol{x}_k \to \boldsymbol{x}^*$ whenever $k \to \infty$.

In general, when dealing with a DC function f defined by (2), the subdifferentials of DC components f_1 and f_2 cannot be used to derive the subdifferential of f at some point $\boldsymbol{x} \in \mathbb{R}^n$, because from subdifferential calculus it only follows that for a nonsmooth DC function f we have $\partial f(\boldsymbol{x}) \subseteq \partial f_1(\boldsymbol{x}) - \partial f_2(\boldsymbol{x})$ [10]. However, if a DC component f_2 is differentiable at a critical point $\boldsymbol{x}^* \in \mathbb{R}^n$, then $\partial f(\boldsymbol{x}^*) = \partial f_1(\boldsymbol{x}^*) - \partial f_2(\boldsymbol{x}^*)$ and especially

$$\partial f_2(\boldsymbol{x}^*) \subseteq \partial f_1(\boldsymbol{x}^*),$$

since $\partial f_2(\boldsymbol{x}^*) = \{\nabla f_2(\boldsymbol{x}^*)\}$ [10]. This indicates that $\mathbf{0} \in \partial f(\boldsymbol{x}^*)$ and thus the corresponding critical point $\boldsymbol{x}^*$ satisfies also the condition of substationarity. A point $\boldsymbol{x} \in \mathbb{R}^n$ is called *substationary* if $\mathbf{0} \in \partial f(\boldsymbol{x})$. For a locally Lipschitz function substationarity is a necessary condition for local optimality and, in the convex case, it is sufficient for global optimality.

For a DC function f it is also possible to formulate the following global optimality condition. Unfortunately, this condition is rather difficult to utilize in solution methods for DC functions.

THEOREM 2.3. [19] *Let f_1 and f_2 be convex functions. A DC function $f = f_1 - f_2$ attains its global minimum at a point $\boldsymbol{x}^* \in \mathbb{R}^n$, if and only if*

$$\partial_\varepsilon f_2(\boldsymbol{x}^*) \subseteq \partial_\varepsilon f_1(\boldsymbol{x}^*) \quad \textit{for all } \varepsilon \geq 0.$$

3 Cutting plane model for DC functions

In this section we propose a new cutting plane model, which is used to find a search direction in our bundle algorithm PBDC. The distinctive feature of the new model is that it utilizes explicitly a DC decomposition of the objective function f defined by (2). This differs from other bundle methods, which usually do not require some specific structure of a nonconvex objective function.

The main idea is to use subgradients of both the DC components f_1 and f_2. This subgradient information is gathered from some neighborhood of the current iteration point and it is used to construct separately an approximation for each convex DC component. Then by combining these approximations we obtain a cutting plane model of the original objective function. For this reason, we particularly assume that a DC decomposition of the objective

function f is available. It is worth noting that in general this DC decomposition is not unique, but our objective function has infinite number of different DC decompositions.

Characteristic to bundle methods is also to assume that at each point $\boldsymbol{x} \in \mathbb{R}^n$ we can compute the function value $f(\boldsymbol{x})$ and one unspecified subgradient $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$. Now we make similar assumptions because we require that at each point $\boldsymbol{x} \in \mathbb{R}^n$ we can evaluate the function values of the DC components $f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{x})$. Moreover, our method requires at each point $\boldsymbol{x} \in \mathbb{R}^n$ two subgradient calculations, namely $\boldsymbol{\xi}_1 \in \partial f_1(\boldsymbol{x})$ and $\boldsymbol{\xi}_2 \in \partial f_2(\boldsymbol{x})$.

We denote by $\boldsymbol{x}_k$ the current iteration point (or stability center) which corresponds to the best estimate of the minimum found so far by the algorithm. Furthermore, we have at our disposal two collections of auxiliary points $\boldsymbol{y}_j$ from previous iterations. One collection is for the DC component f_1 together with subgradients $\boldsymbol{\xi}_{1,j} \in \partial f_1(\boldsymbol{y}_j)$ for $j \in J_1^k$ and, similarly, the other is for the DC component f_2 together with subgradients $\boldsymbol{\xi}_{2,j} \in \partial f_2(\boldsymbol{y}_j)$ for $j \in J_2^k$. Here J_1^k and J_2^k are nonempty sets of indices for the DC components f_1 and f_2, respectively, and these index sets need not to be similar. This means that at each iteration we maintain two separate bundles containing available information and these sets are denoted by

$$\mathcal{B}_i^k = \left\{ \left(\boldsymbol{y}_j,\, f_i(\boldsymbol{y}_j),\, \boldsymbol{\xi}_{i,j} \right) \,\middle|\, j \in J_i^k \right\} \quad \text{for } i = 1, 2,$$

where the subscript tells the DC component in question. The current iteration point $\boldsymbol{x}_k$ is assumed to be included in both bundles $\mathcal{B}_1^k$ and $\mathcal{B}_2^k$ with a suitable index and this is an important requirement in future considerations.

In what follows, we assume that for a point $\boldsymbol{x}_0 \in \mathbb{R}^n$ the set

$$\mathcal{F}_0 = \{\boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0)\} \tag{10}$$

is compact, where $\boldsymbol{x}_0$ is the starting point used in our algorithm. In addition, (overestimated) Lipschitz constants of the convex DC components f_1 and f_2 are assumed to be known on the set $\mathcal{F}_\varepsilon = \{\boldsymbol{x} \in \mathbb{R}^n \mid d(\boldsymbol{x}, \mathcal{F}_0) \leq \varepsilon\}$ with some $\varepsilon > 0$. We denote these constants by $L_1 > 0$ and $L_2 > 0$, respectively. It is easy to show that in this case the original function f is Lipschitz continuous on the set $\mathcal{F}_\varepsilon$ with a constant $L_1 + L_2$.

The idea is to utilize the DC decomposition of the objective function f in the model construction. Since DC components f_i for $i = 1, 2$ are convex, we can first construct a convex piecewise linear approximation of the DC component f_i by

$$\hat{f}_i^k(\boldsymbol{x}) = \max_{j \in J_i^k} \left\{ f_i(\boldsymbol{y}_j) + (\boldsymbol{\xi}_{i,j})^T (\boldsymbol{x} - \boldsymbol{y}_j) \right\} \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^n.$$

This approximation is the classical cutting plane model used in convex bundle methods (see, e.g., [26, 29, 31, 34]) and it can be rewritten in an equivalent

form

$$\hat{f}_i^k(\boldsymbol{x}) = \max_{j \in J_i^k} \left\{ f_i(\boldsymbol{x}_k) + (\boldsymbol{\xi}_{i,j})^T(\boldsymbol{x} - \boldsymbol{x}_k) - \alpha_{i,j}^k \right\},$$

when we use the current iteration point $\boldsymbol{x}_k$ and the *linearization error*

$$\alpha_{i,j}^k = f_i(\boldsymbol{x}_k) - f_i(\boldsymbol{y}_j) - (\boldsymbol{\xi}_{i,j})^T(\boldsymbol{x}_k - \boldsymbol{y}_j) \quad \text{for all } j \in J_i^k.$$

It is worth noting that in most nonconvex bundle methods (see, e.g., [25]) we need to use *subgradient locality measures* instead of linearization errors, since for a nonconvex funtion the linearization error can be negative. However, in our case the linearization error is always nonnegative, that is, $\alpha_{i,j}^k \geq 0$ and therefore we can use it as it is. This property follows from the convexity of f_i and also implies that

$$\hat{f}_i^k(\boldsymbol{x}) \leq f_i(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^n \text{ and } i = 1, 2. \tag{11}$$

Since the linearization error is dependent on the current iteration point, it has to be updated every time when we obtain a new iteration point $\boldsymbol{x}_{k+1}$ differing from the current iteration point $\boldsymbol{x}_k$. However, the linearization error can be updated by using the formula

$$\alpha_{i,j}^{k+1} = \alpha_{i,j}^k + f_i(\boldsymbol{x}_{k+1}) - f_i(\boldsymbol{x}_k) - \boldsymbol{\xi}_{i,j}^T(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$$

and therefore it is sufficient to store only the subgradients $\boldsymbol{\xi}_{i,j} \in \partial f_i(\boldsymbol{y}_j)$ and the linearization errors $\alpha_{i,j}^k$. Especially we do not need to keep track of the auxiliary points $\boldsymbol{y}_j$ together with the function values $f_i(\boldsymbol{y}_j)$ and thus from now on the bundles $\mathcal{B}_i^k$ will consist of a set of pairs $(\boldsymbol{\xi}_{i,j},\, \alpha_{i,j}^k)$, that is,

$$\mathcal{B}_i^k = \left\{ \left(\boldsymbol{\xi}_{i,j},\, \alpha_{i,j}^k \right) \,\middle|\, j \in J_i^k \right\} \quad \text{for } i = 1, 2.$$

To approximate the original objective function f we substitute in the formula (2) the functions f_i with their cutting plane models $\hat{f}_i$. Thus, our piecewise linear *nonconvex cutting plane model* of f is defined by

$$\hat{f}^k(\boldsymbol{x}) = \hat{f}_1^k(\boldsymbol{x}) - \hat{f}_2^k(\boldsymbol{x})$$

and it can be rewritten in an equivalent form

$$\hat{f}^k(\boldsymbol{x}_k + \boldsymbol{d}) = f(\boldsymbol{x}_k) + \max_{j \in J_1^k} \left\{ (\boldsymbol{\xi}_{1,j})^T \boldsymbol{d} - \alpha_{1,j}^k \right\} + \min_{j \in J_2^k} \left\{ -(\boldsymbol{\xi}_{2,j})^T \boldsymbol{d} + \alpha_{2,j}^k \right\}, \tag{12}$$

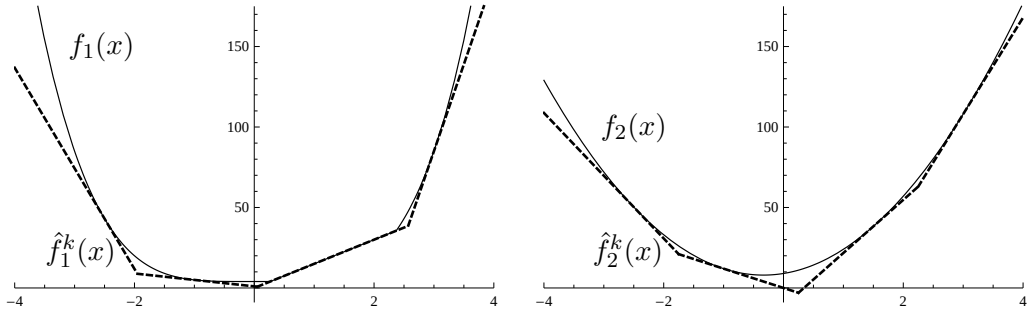

Figure 1: The convex cutting plane models of the DC components

where the new variable $\boldsymbol{d} = \boldsymbol{x} - \boldsymbol{x}_k$ is the search direction ('displacement') at the current iteration point $\boldsymbol{x}_k$. Moreover, we are going to denote by

$$\Delta_1^k(\boldsymbol{d}) = \max_{j \in J_1^k} \left\{ (\boldsymbol{\xi}_{1,j})^T \boldsymbol{d} - \alpha_{1,j}^k \right\} \text{ and } \Delta_2^k(\boldsymbol{d}) = \min_{j \in J_2^k} \left\{ -(\boldsymbol{\xi}_{2,j})^T \boldsymbol{d} + \alpha_{2,j}^k \right\} \quad (13)$$

the piecewise affine functions in the approximation (12).

Figure 1 illustrates the convex cutting plane models of the DC components $f_1(x) = \max\{x^4 + 4,\, 15x\}$ and $f_2(x) = 9x^2 + 6x + 9$, when the linearizations of both models are formed at points $x = -2.5$, -1, 1.5 and 3. The overall approximation of the objective function $f = f_1 - f_2$ is in turn shown in Figure 2 and the model $\hat{f}^k(x)$ describes quite well the actual behavior of f.

The search direction $\boldsymbol{d}_t^k$ can now be computed by solving the problem

$$\begin{cases} \text{minimize} & P^k(\boldsymbol{d}) = \Delta_1^k(\boldsymbol{d}) + \Delta_2^k(\boldsymbol{d}) + \frac{1}{2t}\|\boldsymbol{d}\|^2 \\ \text{subject to} & \boldsymbol{d} \in \mathbb{R}^n, \end{cases} \quad (14)$$

where $t > 0$ is the proximity parameter used in most bundle methods. The term $\frac{1}{2t}\|\boldsymbol{d}\|^2$ is a stabilizing term and it is used to guarantee the existence of the solution $\boldsymbol{d}_t^k$ and also to keep the approximation local enough [29]. Note that even though the problem (14) is quadratic, it is still a nonsmooth nonconvex DC optimization problem because $\Delta_1^k(\boldsymbol{d}) + \frac{1}{2t}\|\boldsymbol{d}\|^2$ is convex, while

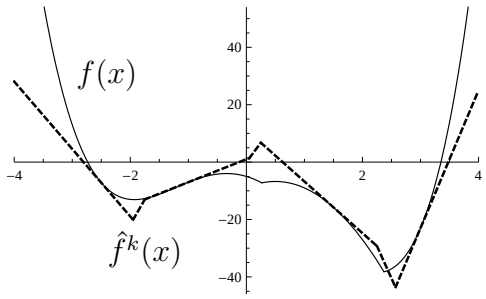

Figure 2: The nonconvex cutting plane model of the objective function

$\Delta_2^k(\boldsymbol{d})$ is a polyhedral concave function. Nevertheless, in the objective function $P^k(\boldsymbol{d})$ the DC component $-\Delta_2^k(\boldsymbol{d})$ is polyhedral convex and in this case the problem (14) can be solved globally quite easily by using an approach described in [1, 2, 36] and also utilized in [14]. In the approach we start with noticing that for all $\boldsymbol{d} \in \mathbb{R}^n$ the objective function $P^k(\boldsymbol{d})$ can be rewritten as

$$P^k(\boldsymbol{d}) = \min_{i \in J_2^k} \left\{ P_i^k(\boldsymbol{d}) = \Delta_1^k(\boldsymbol{d}) - (\boldsymbol{\xi}_{2,i})^T \boldsymbol{d} + \alpha_{2,i}^k + \frac{1}{2t}\|\boldsymbol{d}\|^2 \right\}$$

and hence the problem (14) takes the form

$$\min_{\boldsymbol{d} \in \mathbb{R}^n} \min_{i \in J_2^k} \left\{ P_i^k(\boldsymbol{d}) \right\} = \min_{i \in J_2^k} \min_{\boldsymbol{d} \in \mathbb{R}^n} \left\{ P_i^k(\boldsymbol{d}) \right\}. \tag{15}$$

Due to this we can change the order of minimizations and solve first for each $i \in J_2^k$ the subproblem

$$\begin{cases} \text{minimize} & P_i^k(\boldsymbol{d}) = \Delta_1^k(\boldsymbol{d}) - (\boldsymbol{\xi}_{2,i})^T \boldsymbol{d} + \alpha_{2,i}^k + \frac{1}{2t}\|\boldsymbol{d}\|^2 \\ \text{subject to} & \boldsymbol{d} \in \mathbb{R}^n, \end{cases} \tag{16}$$

which is the inner problem in (15). In what follows, we denote by $\boldsymbol{d}_t^k(i)$ the subproblem minimizer and the global minimizer $\boldsymbol{d}_t^k$ of (14) can be obtained by selecting the 'best' solution from among all the subproblem minimizers. In other words, the search direction $\boldsymbol{d}_t^k$ is given by

$$\boldsymbol{d}_t^k = \boldsymbol{d}_t^k(i^*) \quad \text{where } i^* = \arg\min_{i \in J_2^k} \left\{ P_i^k\Big(\boldsymbol{d}_t^k(i)\Big) \right\}.$$

Thus, the global minimization of the problem (14) requires the solution of $|J_2^k|$ subproblems. For this reason, the size of the bundle $\mathcal{B}_2^k$ has to be bounded by an appropriate upper bound: the larger the bundle $\mathcal{B}_2^k$ is, the more time-consuming it is to obtain the global solution. Especially each of the subproblems (16) is a nonsmooth strictly convex quadratic problem of the type usually encountered in bundle methods. In addition, the nonsmoothness does not cause any difficulties, because for each $i \in J_2^k$ the subproblem (16) can be reformulated as a smooth quadratic subproblem

$$\begin{cases} \text{minimize} & v + \frac{1}{2t}\|\boldsymbol{d}\|^2 \\ \text{subject to} & (\boldsymbol{\xi}_{1,j} - \boldsymbol{\xi}_{2,i})^T \boldsymbol{d} - (\alpha_{1,j}^k - \alpha_{2,i}^k) \le v \quad \text{for all } j \in J_1^k \\ & v \in \mathbb{R},\ \boldsymbol{d} \in \mathbb{R}^n. \end{cases} \tag{17}$$

By duality it is equivalent to the quadratic subproblem

$$\begin{cases} \text{minimize} & \frac{1}{2}t\|\sum_{j \in J_1^k} \lambda_j \boldsymbol{\xi}_{1,j} - \boldsymbol{\xi}_{2,i}\|^2 + \sum_{j \in J_1^k} \lambda_j \alpha_{1,j}^k - \alpha_{2,i}^k \\ \text{subject to} & \sum_{j \in J_1^k} \lambda_j = 1 \\ & \lambda_j \ge 0 \quad \text{for all } j \in J_1^k, \end{cases} \tag{18}$$

which is usually easier to solve than the corresponding primal subproblem. The following theorem shows that the optimal primal solution $(v_t^k(i),\, \boldsymbol{d}_t^k(i))$ is related to the optimal dual solutions $\lambda_{t,j}^k(i)$ for $j \in J_1^k$.

THEOREM 3.1. *For each $i \in J_2^k$, the problems* (17) *and* (18) *are equivalent, and they have unique solutions $(v_t^k(i),\, \boldsymbol{d}_t^k(i))$ and $\lambda_{t,j}^k(i)$ for $j \in J_1^k$, respectively, such that*

$$\boldsymbol{d}_t^k(i) = -t\left(\sum_{j \in J_1^k} \lambda_{t,j}^k(i)\boldsymbol{\xi}_{1,j} - \boldsymbol{\xi}_{2,i}\right)$$

$$v_t^k(i) = -\frac{1}{t}\|\boldsymbol{d}_t^k(i)\|^2 - \sum_{j \in J_1^k} \lambda_{t,j}^k(i)\alpha_{1,j}^k + \alpha_{2,i}^k.$$

Proof. See [31], pp. 115–117. □

Next, we look closer the value $\Delta_1^k(\boldsymbol{d}) + \Delta_2^k(\boldsymbol{d})$. As in [14], it can be seen as an approximation to the function

$$h_k(\boldsymbol{d}) = f(\boldsymbol{x}_k + \boldsymbol{d}) - f(\boldsymbol{x}_k)$$

and at $\boldsymbol{d} = \boldsymbol{0}$ the value $\Delta_1^k(\boldsymbol{d}) + \Delta_2^k(\boldsymbol{d})$ interpolates f. In a similar way, the separate values $\Delta_1^k(\boldsymbol{d})$ and $\Delta_2^k(\boldsymbol{d})$ can be interpreted as a predicted change of the DC components f_1 and $-f_2$, respectively. Some simple properties of $\Delta_1^k(\boldsymbol{d})$ and $\Delta_2^k(\boldsymbol{d})$ are presented below.

LEMMA 3.2. *The following properties hold:*

(i) $\Delta_1^k(\boldsymbol{d}) \leq f_1(\boldsymbol{x}_k + \boldsymbol{d}) - f_1(\boldsymbol{x}_k)$;

(ii) $\Delta_2^k(\boldsymbol{d}) \geq -f_2(\boldsymbol{x}_k + \boldsymbol{d}) - (-f_2(\boldsymbol{x}_k))$;

(iii) For any $t > 0$ we have $\Delta_1^k(\boldsymbol{d}_t^k) + \Delta_2^k(\boldsymbol{d}_t^k) \leq -\dfrac{1}{2t}\|\boldsymbol{d}_t^k\|^2 \leq 0.$

Proof. (i) Since f_1 is convex, the inequality (11) holds and we have

$$\hat{f}_1^k(\boldsymbol{x}_k + \boldsymbol{d}) = \max_{j \in J_1^k} \left\{ f_1(\boldsymbol{x}_k) + (\boldsymbol{\xi}_{1,j})^T\boldsymbol{d} - \alpha_{1,j}^k \right\} \leq f_1(\boldsymbol{x}_k + \boldsymbol{d}).$$

From this it is easy to derive that

$$\max_{j \in J_1^k} \left\{ (\boldsymbol{\xi}_{1,j})^T\boldsymbol{d} - \alpha_{1,j}^k \right\} \leq f_1(\boldsymbol{x}_k + \boldsymbol{d}) - f_1(\boldsymbol{x}_k),$$

which confirms the claim.

(ii) The property follows by the same way as in (i). The only difference is that the last inequality is multiplied by -1 and after that maximization is converted to minimization.

(*iii*) We notice first that for the problem (14) the vector $\boldsymbol{d} = \mathbf{0}$ yields the objective function value

$$\Delta_1^k(\mathbf{0}) + \Delta_2^k(\mathbf{0}) + \frac{1}{2t}\|\mathbf{0}\|^2 = \max_{j \in J_1^k} \left\{-\alpha_{1,j}^k\right\} + \min_{j \in J_2^k} \left\{\alpha_{2,j}^k\right\} = 0,$$

where the last equality follows from the requirement that the current iteration point $\boldsymbol{x}_k$ is included in both bundles $\mathcal{B}_i^k$, $i = 1, 2$. In other words, for both f_i, $i = 1, 2$, we have some $\bar{\jmath} \in J_i^k$ such that $\alpha_{i,\bar{\jmath}}^k = 0$. Moreover, for all the other $j \in J_i^k$, the linearization error $\alpha_{i,j}^k$ is nonnegative, that is, $\alpha_{i,j}^k \geq 0$. Thus, the optimal objective function value of the problem (14) is always smaller than or equal to zero, that is,

$$\Delta_1^k(\boldsymbol{d}_t^k) + \Delta_2^k(\boldsymbol{d}_t^k) + \frac{1}{2t}\|\boldsymbol{d}_t^k\|^2 \leq 0$$

and this yields the result. □

From Lemma 3.2 we see, for example, that when $\Delta_1^k(\boldsymbol{d})$ is nonpositive, the real descent in the value of f_1 is always less than or equal to the predicted one. On the other hand, if $\Delta_1^k(\boldsymbol{d})$ is positive, then the real increase in f_1 is always greater than or equal to the predicted change $\Delta_1^k(\boldsymbol{d})$. Thus, the cutting plane model of f_1 yields approximations, which are too optimistic in the case of minimization.

When we consider the function $-f_2$, we get somewhat opposite results. First of all, if the value $\Delta_2^k(\boldsymbol{d})$ is nonpositive, then the real decrease in the function $-f_2$ is always greater than or equal to the predicted one. This means that the model $-\hat{f}_2$ of the function $-f_2$ actually underestimates the real descent. In addition, when the value $\Delta_2^k(\boldsymbol{d})$ is positive, we can be sure that the real increase in the value of $-f_2$ is always less than or equal to the predicted one.

Lemma 3.2 also ensures that the overall approximation $\Delta_1^k(\boldsymbol{d}_t^k)+\Delta_2^k(\boldsymbol{d}_t^k)$ of the change is always nonpositive for the solution $\boldsymbol{d}_t^k$. For this reason, it can be used as a predicted descent of the objective function f at the current iteration point $\boldsymbol{x}_k$. It is also worth pointing out that one of the approximations $\Delta_i^k(\boldsymbol{d}_t^k)$ for $i = 1, 2$ can be positive if the other one is negative enough. This is due to the fact that only the overall approximation needs to be nonpositive. Thus, it is possible that one of the DC components increases while the other one decreases.

4 Minimization algorithm

In this section we describe a new proximal bundle method (PBDC) for nonsmooth DC optimization. The main novelty of our bundle method is that it uses the new cutting plane model introduced in the previous section. Due

to this we utilize explicitly the DC decomposition of the objective function f. The PBDC algorithm is based on the bundle methods introduced in [14, 15, 16] and it especially combines the features of the 'main iterations' of those bundle methods with suitable modifications.

The core of PBDC is the 'main iteration', which consists of a sequence of steps where the current iteration point remains unchanged. The basic idea in the 'main iteration' is that we repeatedly solve the problem (14) globally and use the solution obtained to update appropriately either the proximity parameter or the bundles. The exit from the 'main iteration' happens either because we end up in a better iteration point or because the current solution satisfies the approximate stopping condition.

If we find a better solution candidate during the 'main iteration', then the sufficient descent condition is satisfied and the value of the objective function decreases. In this case we update the current iteration point and start a new 'main iteration'. However, if the approximate stopping condition is fulfilled, then the execution of the overall bundle algorithm stops with the current iteration point as the final solution.

The execution of the PBDC method starts with the initialization of the algorithm. In this step we need to choose a starting point $\boldsymbol{x}_0 \in \mathbb{R}^n$ and set the following global parameters:

- the criticality tolerance $\delta > 0$ and the proximity measure $\varepsilon > 0$
- the descent parameter $m \in (0, 1)$
- the decrease parameter $r \in (0, 1)$ and the increase parameter $R > 1$.

After this the initial auxiliary point $\boldsymbol{y}_1$ is set equal to $\boldsymbol{x}_0$ in both bundles. This means that for the DC components f_i $(i = 1, 2)$ the initial bundle $\mathcal{B}_i^0$ contains only one element $(\boldsymbol{\xi}_i(\boldsymbol{y}_1),\ \alpha_{i,1}^0)$ where $\boldsymbol{\xi}_i(\boldsymbol{y}_1) \in \partial f_i(\boldsymbol{y}_1)$ and $\alpha_{i,1}^0 = 0$. It is also worth noting that the PBDC algorithm requires (overestimated) Lipschitz constants $L_1 > 0$ and $L_2 > 0$ of the DC components f_1 and f_2 on the set $\mathcal{F}_\varepsilon = \{\boldsymbol{x} \in \mathbb{R}^n \mid d(\boldsymbol{x}, \mathcal{F}_0) \leq \varepsilon\}$, where $\mathcal{F}_0$ is the set defined by (10). In addition, the following local parameters are set each time the 'main iteration' is entered:

- the safeguard parameters $t_{\min}$ and $t_{\max}$, $0 < t_{\min} < t_{\max}$
- the local proximity measure $\theta > 0$.

Due to the importance of the 'main iteration' we first describe in detail its algorithm and only after that we present the overall PBDC algorithm. For the sake of notational simplicity we do not index the 'main iteration' algorithm. Also the superscript k is omitted (except for $\boldsymbol{x}_k$), since it indicates only the values with respect to the current iteration point $\boldsymbol{x}_k \in \mathbb{R}^n$ and this point remains unchanged during the 'main iteration'. In what follows, $\boldsymbol{\xi}_1(\boldsymbol{x}_k) \in \partial f_1(\boldsymbol{x}_k)$ and $\boldsymbol{\xi}_2(\boldsymbol{x}_k) \in \partial f_2(\boldsymbol{x}_k)$ are the subgradients computed at the current iteration point $\boldsymbol{x}_k$.

ALGORITHM 4.1. Main Iteration

Step 0. *(Stopping condition)* If $\|\boldsymbol{\xi}_1(\boldsymbol{x}_k) - \boldsymbol{\xi}_2(\boldsymbol{x}_k)\| < \delta$ then STOP (criticality achieved)

Step 1. *(Parameter initialization)* Calculate $j^* = \arg\max_{j \in J_2} \left\{ \|\boldsymbol{\xi}_{2,j}\| \right\}$ and set

$$\boldsymbol{\xi}_{2,\max} = \boldsymbol{\xi}_{2,j^*}, \quad \varepsilon_1 = \frac{\varepsilon}{2\max\{L_1, L_2, 1/2\}},$$
$$t_{\min} = r \cdot \frac{\varepsilon_1}{2\Big(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}\|\Big)},$$
$$t_{\max} = R\, t_{\min} \quad \text{and} \quad \theta = r\, t_{\min}\, \delta.$$

Select the value $t \in [t_{\min}, t_{\max}]$.

Step 2. *(Search direction)* Solve the problem

$$\min_{\boldsymbol{d} \in \mathbb{R}^n} \left\{ \Delta_1(\boldsymbol{d}) + \Delta_2(\boldsymbol{d}) + \frac{1}{2t}\|\boldsymbol{d}\|^2 \right\}$$

and from the solution $\boldsymbol{d}_t$ compute the predicted changes

$$\Delta_1(\boldsymbol{d}_t) = \max_{j \in J_1} \left\{ (\boldsymbol{\xi}_{1,j})^T \boldsymbol{d}_t - \alpha_{1,j} \right\} \quad \text{and}$$
$$\Delta_2(\boldsymbol{d}_t) = \min_{j \in J_2} \left\{ -(\boldsymbol{\xi}_{2,j})^T \boldsymbol{d}_t + \alpha_{2,j} \right\}.$$

If $\|\boldsymbol{d}_t\| < \theta$ then go to Step 3 else go to Step 4.

Step 3. *(Approximate stopping condition)* Set

$$J_1 = J_1 \setminus \{j \in J_1 \,|\, \alpha_{1,j} > \varepsilon\} \quad \text{and} \quad J_2 = J_2 \setminus \{j \in J_2 \,|\, \alpha_{2,j} > \varepsilon\}$$

and calculate values $\boldsymbol{\xi}_1^*$ and $\boldsymbol{\xi}_2^*$ such that

$$\|\boldsymbol{\xi}_1^* - \boldsymbol{\xi}_2^*\| = \begin{cases} \min & \|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\| \\ \text{s. t.} & \boldsymbol{\xi}_1 \in \operatorname{conv}\{\boldsymbol{\xi}_{1,j} \,|\, j \in J_1\} \\ & \boldsymbol{\xi}_2 \in \operatorname{conv}\{\boldsymbol{\xi}_{2,j} \,|\, j \in J_2\}. \end{cases} \tag{19}$$

If $\|\boldsymbol{\xi}_1^* - \boldsymbol{\xi}_2^*\| < \delta$ then STOP (ε-criticality achieved) else set

$$t_{\max} = t_{\max} - r\,(t_{\max} - t_{\min}), \tag{20}$$

select the value $t \in [t_{\min}, t_{\max}]$ and go back to Step 2.

Step 4. *(Descent test)* Set $\boldsymbol{y} = \boldsymbol{x}_k + \boldsymbol{d}_t$. If

$$f(\boldsymbol{y}) - f(\boldsymbol{x}_k) \le m \left(\Delta_1(\boldsymbol{d}_t) + \Delta_2(\boldsymbol{d}_t) \right) \tag{21}$$

then put $\boldsymbol{x}_{k+1} = \boldsymbol{y}$ and EXIT from the 'main iteration'.

Step 5. *(Bundle update)* Compute $\boldsymbol{\xi}_1 \in \partial f_1(\boldsymbol{y})$, $\boldsymbol{\xi}_2 \in \partial f_2(\boldsymbol{y})$ and set

$$\alpha_1 = f_1(\boldsymbol{x}_k) - f_1(\boldsymbol{y}) + \boldsymbol{\xi}_1^T \boldsymbol{d}_t \quad \text{and} \quad \alpha_2 = f_2(\boldsymbol{x}_k) - f_2(\boldsymbol{y}) + \boldsymbol{\xi}_2^T \boldsymbol{d}_t.$$

(a) If $f(\boldsymbol{y}) - f(\boldsymbol{x}_0) > 0$ and $\|\boldsymbol{d}_t\| > \varepsilon_1$ then set $t = t - r(t - t_{\min})$ and go back to Step 2.

(b) Otherwise insert the element:

$$(\boldsymbol{\xi}_1,\, \alpha_1) \text{ into } \mathcal{B}_1 \text{ for a suitable value of the index } \hat{\jmath} \in J_1$$

and, if $\Delta_2(\boldsymbol{d}_t) \geq 0$, then insert also the element:

$$(\boldsymbol{\xi}_2,\, \alpha_2) \text{ into } \mathcal{B}_2 \text{ for a suitable value of the index } \hat{\jmath} \in J_2.$$

Step 6. *(Parameter update)* If $\|\boldsymbol{\xi}_2\| > \|\boldsymbol{\xi}_{2,\max}\|$ then update

$$\boldsymbol{\xi}_{2,\max} = \boldsymbol{\xi}_2,\ t_{\min} = r \cdot \frac{\varepsilon_1}{2\Big(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}\|\Big)} \quad \text{and } \theta = r\, t_{\min}\, \delta.$$

Go back to Step 2.

The direction finding problem at Step 2 of the 'main iteration' is the problem (14) and it can be solved globally by using the approach introduced in the previous section. Therefore, during each round of the 'main iteration' we need to solve the subproblem (16) for each $i \in J_2$, since the global solution is obtained by selecting the 'best' solution from among all the subproblem minimizers. However, if the bundle update (Step 5(b)) is performed, then it is possible that we need not to resolve each of the subproblems during the next round. This follows from the fact that a new constraint inserted into the subproblem (17) (i.e. the one constructed from the newest bundle element of $\mathcal{B}_1$) may not be active. In such a case the previous solution of the subproblem does not change and we need not to resolve the corresponding subproblem.

In the 'main iteration', the initialization and update of the safeguard parameter $t_{\min}$ is based on [14]. However, since we are maintaining two separate bundles, we have to use two different norms in the denominator of $t_{\min}$. Moreover, the parameters $t_{\max}$ and θ are selected according to [15, 16] whereas the selection of the proximity parameter t is based on [14, 16].

In addition, the stopping conditions tested both at Step 0 and 3 are similar to those in [14, 15, 16], but now instead of substationarity we test ε-criticality. Due to this we can do the bundle deletion at Step 3 by using the linearization errors and we do not need to keep track of distances between auxiliary points and the current iteration point as is done in [14, 15, 16]. It is also worth noting that the descent condition (21) tested at Step 4 is

the one typically used in bundle methods (see, e.g., [14, 15, 16, 26, 29, 34]). Especially, if this condition does not hold, we have to improve our cutting plane model of f.

The reason for Step 3 is that, whenever the norm of the solution $\boldsymbol{d}_t$ is 'small', we have either achieved an ε-critical point or our cutting plane model is inconsistent [15]. To test ε-criticality we do the bundle deletion at Step 3 and calculate aggregated ε-subgradients $\boldsymbol{\xi}_1^*$ and $\boldsymbol{\xi}_2^*$ for DC components f_1 and f_2 (i.e. we solve the norm minimization problem (19)). If aggregated ε-subgradients are not close enough to each other, then our model is inconsistent and we have to decrease the value of the safeguard parameter $t_{\max}$. It is also worth noting that if nothing is removed from the bundles at Step 3, then the current solution $\boldsymbol{d}_t$ divided by t is a feasible solution to the norm minimization problem (19). Moreover, this feasible solution $\frac{1}{t}\boldsymbol{d}_t$ satisfies the approximate stopping condition, that is,

$$\frac{1}{t}\|\boldsymbol{d}_t\| < \frac{\theta}{t} = \frac{t_{\min}}{t}\, r\, \delta < \delta,$$

because $\|\boldsymbol{d}_t\| < \theta$ whenever we are executing Step 3. Therefore, we do not need to solve the norm minimization problem (19) and the PBDC algorithm can be stopped.

ALGORITHM 4.2. Proximal bundle method for DC functions (PBDC)

Data: Start with choosing the criticality tolerance $\delta > 0$ and the proximity measure $\varepsilon > 0$. Select also the descent parameter $m \in (0, 1)$, the decrease parameter $r \in (0, 1)$ and the increase parameter $R > 1$.

Step 0. *(Initialization)* Select a starting point $\boldsymbol{x}_0 \in \mathbb{R}^n$ and compute the value of DC components $f_1(\boldsymbol{x}_0)$ and $f_2(\boldsymbol{x}_0)$. Set $\boldsymbol{y}_1 = \boldsymbol{x}_0$ and initialize the iteration counter $k = 0$. Then calculate subgradients $\boldsymbol{\xi}_{1,1} \in \partial f_1(\boldsymbol{y}_1)$ and $\boldsymbol{\xi}_{2,1} \in \partial f_2(\boldsymbol{y}_1)$ and set $\alpha_{1,1}^k = \alpha_{2,1}^k = 0$. Initialize the bundles by setting

$$\mathcal{B}_1^k = \left\{(\boldsymbol{\xi}_{1,1}, \alpha_{1,1}^k)\right\} \quad \text{and} \quad \mathcal{B}_2^k = \left\{(\boldsymbol{\xi}_{2,1}, \alpha_{2,1}^k)\right\}$$

and $J_1^k = J_2^k = \{1\}$.

Step 1. *(Main iteration)* Execute 'main iteration' Algorithm 4.1. This either yields the new iteration point $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}_t^k$ or indicates that the current solution $\boldsymbol{x}_k$ is ε-critical. In the case of ε-criticality we STOP the algorithm with $\boldsymbol{x}_k$ as the final solution.

Step 2. *(Bundle update)* Compute the new DC component values and subgradients

$$f_i(\boldsymbol{x}_{k+1}) \quad \text{and} \quad \boldsymbol{\xi}_i(\boldsymbol{x}_{k+1}) \in \partial f_i(\boldsymbol{x}_{k+1}) \quad \text{for } i = 1, 2.$$

Choose the bundles $\mathcal{B}_1^{k+1} \subseteq \mathcal{B}_1^k$ and $\mathcal{B}_2^{k+1} \subseteq \mathcal{B}_2^k$ for the next round and update the linearization errors using the formula

$$\alpha_{i,j}^{k+1} = \alpha_{i,j}^k + f_i(\boldsymbol{x}_{k+1}) - f_i(\boldsymbol{x}_k) - (\boldsymbol{\xi}_{i,j})^T \boldsymbol{d}_t^k \text{ for all } i = 1, 2 \text{ and } j \in J_i^{k+1}.$$

Insert also the element

$(\boldsymbol{\xi}_1(\boldsymbol{x}_{k+1}), 0)$ into $\mathcal{B}_1^{k+1}$ for a suitable value of the index $\hat{\jmath} \in J_1^{k+1}$ and
$(\boldsymbol{\xi}_2(\boldsymbol{x}_{k+1}), 0)$ into $\mathcal{B}_2^{k+1}$ for a suitable value of the index $\hat{\jmath} \in J_2^{k+1}$.

Finally, update the iteration counter $k = k + 1$ and go back to Step 1.

It is worth noting that the choice of the new bundles $\mathcal{B}_1^{k+1}$ and $\mathcal{B}_2^{k+1}$ is not restricted at Step 2 of the PBDC algorithm. For this reason, it is also possible to set $\mathcal{B}_1^{k+1} = \emptyset$ and/or $\mathcal{B}_2^{k+1} = \emptyset$. However, the bundles $\mathcal{B}_1^{k+1}$ and $\mathcal{B}_2^{k+1}$ are never empty when we start the execution of the 'main iteration'. This is due to the fact that the bundle element corresponding to the new iteration point $\boldsymbol{x}_{k+1}$ is always inserted into both bundles $\mathcal{B}_1^{k+1}$ and $\mathcal{B}_2^{k+1}$ before going back to Step 1.

Another nice feature of the PBDC algorithm is that the size of the bundle $\mathcal{B}_2^k$ can also be bounded during 'main iteration' Algorithm 4.1. This means that we are allowed to select the maximum number of stored subgradients $J_{\max} \geq 2$ for the bundle $\mathcal{B}_2^k$ in the 'main iteration'. Therefore, we are also able to control the number of subproblems solved. The only restriction in this case is that the bundle element $(\boldsymbol{\xi}_2(\boldsymbol{x}_k), 0)$ of $\mathcal{B}_2^k$ (i.e. the bundle element corresponding to the current iteration point $\boldsymbol{x}_k$) cannot be replaced or deleted during the execution of the 'main iteration'.

5 Convergence

In this section we prove the convergence of the PBDC algorithm to an ε-critical point and show that the termination happens after a finite number of 'main iterations'. Throughout the section we again require the following assumptions:

- **A1** The set $\mathcal{F}_0 = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0)\}$ is compact.
- **A2** Overestimates of the Lipschitz constants of f_1 and f_2 are known on the set $\mathcal{F}_\varepsilon = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, d(\boldsymbol{x}, \mathcal{F}_0) \leq \varepsilon\}$, where $\varepsilon > 0$ is the proximity measure. These constants are denoted by $L_1 > 0$ and $L_2 > 0$, respectively.
- **A3** The objective function f is finite on $\mathbb{R}^n$.

We start with the following observation (the similar observation is given also in [14]).

REMARK 5.1. The bundle insertion rule at Step 5 of 'main iteration' Algorithm 4.1 guarantees that all the points corresponding to the bundle elements in the bundles $\mathcal{B}_1$ and $\mathcal{B}_2$ are on the set $\mathcal{F}_\varepsilon$. Together with the assumption **A2** this implies that we always have

$$\|\boldsymbol{\xi}_{1,j}\| \leq L_1 \text{ for points } \boldsymbol{y}_j \text{ on } \mathcal{B}_1 \text{ and } \|\boldsymbol{\xi}_{2,j}\| \leq L_2 \text{ for points } \boldsymbol{y}_j \text{ on } \mathcal{B}_2. \quad (22)$$

This in turn indicates that the parameters $t_{\min}$ and θ are bounded away from zero at all iterations of the algorithm, because now $t_{\min} \geq \bar{t} = r\varepsilon_1/(2L_1 + 2L_2) > 0$ and $\theta \geq \bar{\theta} = r\bar{t}\delta > 0$.

Next we prove that the solution $\boldsymbol{d}_t$ of the problem (14) is bounded in norm during the execution of the 'main iteration'.

LEMMA 5.2. *For any proximity parameter $t > 0$ it holds that*

$$\|\boldsymbol{d}_t\| \leq 2t \left(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}\|\right) \leq 2t\,(L_1 + L_2)\,,$$

where $\boldsymbol{x}_k$ is the current iteration point, $\boldsymbol{\xi}_1(\boldsymbol{x}_k) \in \partial f_1(\boldsymbol{x}_k)$ is the corresponding subgradient and $\|\boldsymbol{\xi}_{2,\max}\| = \max_{j\in J_2}\left\{\|\boldsymbol{\xi}_{2,j}\|\right\}$.

Proof. By the definition of $\Delta_2(\boldsymbol{d})$ taking into account that $\alpha_{2,j} \geq 0$ for all $j \in J_2$, we first observe that for all $\boldsymbol{d}$

$$\Delta_2(\boldsymbol{d}) = \min_{j\in J_2}\left\{-(\boldsymbol{\xi}_{2,j})^T\boldsymbol{d} + \alpha_{2,j}\right\} \geq \min_{j\in J_2}\left\{-(\boldsymbol{\xi}_{2,j})^T\boldsymbol{d}\right\} \geq -\|\boldsymbol{\xi}_{2,\max}\|\|\boldsymbol{d}\|.$$

The definition of $\Delta_1(\boldsymbol{d})$ yields

$$\Delta_1(\boldsymbol{d}) \geq (\boldsymbol{\xi}_{1,j})^T\boldsymbol{d} - \alpha_{1,j} \quad \text{for all } j \in J_1.$$

Now, combining the above inequalities, we see that

$$\Delta_1(\boldsymbol{d}) + \Delta_2(\boldsymbol{d}) \geq (\boldsymbol{\xi}_{1,j})^T\boldsymbol{d} - \alpha_{1,j} - \|\boldsymbol{\xi}_{2,\max}\|\|\boldsymbol{d}\| \quad \text{for all } \boldsymbol{d} \text{ and } j \in J_1. \quad (23)$$

Because at each iteration round in Algorithm 4.1 the current iteration point $\boldsymbol{x}_k$ belongs to the bundle $\mathcal{B}_1$ together with a subgradient $\boldsymbol{\xi}_1(\boldsymbol{x}_k) \in \partial f_1(\boldsymbol{x}_k)$, there exits an index $\bar{j} \in J_1$ which corresponds to this element and in particular, $\alpha_{1,j} = 0$ for $j = \bar{j}$. Now the inequality (23) holds for the index $\bar{j}$ and thus for all $\boldsymbol{d}$

$$\begin{aligned}\Delta_1(\boldsymbol{d}) + \Delta_2(\boldsymbol{d}) &\geq \left(\boldsymbol{\xi}_1(\boldsymbol{x}_k)\right)^T\boldsymbol{d} - \|\boldsymbol{\xi}_{2,\max}\|\|\boldsymbol{d}\| \\ &\geq -\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\|\|\boldsymbol{d}\| - \|\boldsymbol{\xi}_{2,\max}\|\|\boldsymbol{d}\| = -\left(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}\|\right)\|\boldsymbol{d}\|.\end{aligned}$$

In addition, when we take into account the property (iii) of Lemma 3.2, we can conclude that the solution $\boldsymbol{d}_t$ of the problem (14) satisfies the inequality

$$-\frac{1}{2t}\|\boldsymbol{d}_t\|^2 \geq \Delta_1(\boldsymbol{d}_t) + \Delta_2(\boldsymbol{d}_t) \geq -\Big(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}\|\Big)\|\boldsymbol{d}_t\|, \tag{24}$$

which gives the first part of the claim. The latter inequality follows directly from the property (22). □

REMARK 5.3. As in [15], the previous lemma can be used to guarantee that in 'main iteration' Algorithm 4.1 the condition $\|\boldsymbol{d}_t\| \leq \theta$ is never satisfied as a consequence of the choice of a too small proximity parameter t. To see this, we start with the observation that if the 'main iteration' does not stop at Step 0, then $\|\boldsymbol{\xi}_1(\boldsymbol{x}_k) - \boldsymbol{\xi}_2(\boldsymbol{x}_k)\| \geq \delta$. Now, the result follows by noticing that

$$\|\boldsymbol{d}_{t_{\min}}\| \leq 2t_{\min}\Big(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}\|\Big) = \frac{2\Big(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}\|\Big)}{r\delta}\,\theta,$$

where the right-hand side is strictly greater than θ, since

$$\begin{aligned}\frac{2\Big(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}\|\Big)}{r\delta}\,\theta &\geq \frac{2\left(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_2(\boldsymbol{x}_k)\|\right)}{r\delta}\,\theta \\ &\geq \frac{2\left(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k) - \boldsymbol{\xi}_2(\boldsymbol{x}_k)\|\right)}{r\delta}\,\theta \geq \frac{2}{r}\,\theta > \theta.\end{aligned}$$

The following lemma is needed when we prove the termination of the 'main iteration'.

LEMMA 5.4. *If the condition* (21) *at Step 4 of Algorithm 4.1 is not satisfied, then*

$$\boldsymbol{\xi}_1^T\boldsymbol{d}_t - \alpha_1 > m\Delta_1(\boldsymbol{d}_t) + (m-1)\Delta_2(\boldsymbol{d}_t),$$

where $\boldsymbol{\xi}_1 \in \partial f_1(\boldsymbol{y})$ *is a subgradient calculated at the new auxiliary point* $\boldsymbol{y} = \boldsymbol{x}_k + \boldsymbol{d}_t$ *and* $\alpha_1 = f_1(\boldsymbol{x}_k) - f_1(\boldsymbol{y}) + \boldsymbol{\xi}_1^T\boldsymbol{d}_t$ *is the corresponding linearization error.*

Proof. If the descent condition (21) is not satisfied at Step 4, then

$$f(\boldsymbol{y}) - f(\boldsymbol{x}_k) > m\Big(\Delta_1(\boldsymbol{d}_t) + \Delta_2(\boldsymbol{d}_t)\Big),$$

where $\boldsymbol{y} = \boldsymbol{x}_k + \boldsymbol{d}_t$ is the new auxiliary point computed in the main iteration. Using DC decomposition (2) of the objective function f the above inequality can be rewritten in the form

$$f_1(\boldsymbol{y}) - f_1(\boldsymbol{x}_k) > m\Big(\Delta_1(\boldsymbol{d}_t) + \Delta_2(\boldsymbol{d}_t)\Big) - \Big(f_2(\boldsymbol{x}_k) - f_2(\boldsymbol{y})\Big).$$

Furthermore, the property (ii) of Lemma 3.2 ensures that we always have

$$-f_2(\boldsymbol{y}) - (-f_2(\boldsymbol{x}_k)) = f_2(\boldsymbol{x}_k) - f_2(\boldsymbol{y}) \leq \Delta_2(\boldsymbol{d}_t),$$

because linearizations used in $\Delta_2(\boldsymbol{d})$ are above the concave function $-f_2$. Taking this into account, we obtain

$$f_1(\boldsymbol{y}) - f_1(\boldsymbol{x}_k) > m\Delta_1(\boldsymbol{d}_t) + (m-1)\Delta_2(\boldsymbol{d}_t)$$

and the result follows by noticing that

$$f_1(\boldsymbol{y}) - f_1(\boldsymbol{x}_k) = f_1(\boldsymbol{x}_k + \boldsymbol{d}_t) - f_1(\boldsymbol{x}_k) = \boldsymbol{\xi}_1^T \boldsymbol{d}_t - \alpha_1$$

when $\boldsymbol{\xi}_1 \in \partial f_1(\boldsymbol{y})$ and $\alpha_1 = f_1(\boldsymbol{x}_k) - f_1(\boldsymbol{y}) + \boldsymbol{\xi}_1^T \boldsymbol{d}_t$. □

Now we are ready to consider separately two possible cases which can occur in 'main iteration' Algorithm 4.1. The proofs of these properties are along guidelines of [14, 15, 16].

LEMMA 5.5. *Algorithm 4.1 cannot pass infinitely many times through Step 3.*

Proof. Suppose the assertion of the lemma is false. Then Step 3 is executed infinitely many times and let us index by $i \in \mathcal{I}$ all the quantities referred to ith passage of Step 3. Especially the conditions

$$\left\| \boldsymbol{d}_t^{(i)} \right\| < \theta_i \quad \text{and} \quad \left\| \boldsymbol{\xi}_1^{*\,(i)} - \boldsymbol{\xi}_2^{*\,(i)} \right\| > \delta$$

hold for each index $i \in \mathcal{I}$, because we are executing Step 3 and the approximate stopping condition cannot be satisfied.

First of all, we notice that the safeguard parameter t_{max} is reduced according to the formula (20) each time Step 3 is performed while $t_{\min}$ is both bounded and monotonically nonincreasing. Thus, the parameter $t_{\max}$ becomes arbitrarily close to the parameter $t_{\min}$ when the 'main iteration' is executed. From this it follows that also the proximity parameter t becomes arbitrarily close to the parameter $t_{\min}$, since $t \in [t_{\min}, t_{\max}]$. Together with Lemma 5.2 and the definition of the parameter $t_{\min}^{(i)}$, this information yields that asymptotically

$$\|\boldsymbol{d}_t^{(i)}\| \leq \varepsilon_1 = \frac{\varepsilon}{2\max\{L_1, L_2, 1/2\}}, \tag{25}$$

since we always have

$$\|\boldsymbol{d}_t^{(i)}\| \leq 2\, t_i \left(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}^{(i)}\| \right) \quad \text{and} \quad t_{\min}^{(i)} < \frac{\varepsilon_1}{2\Big(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}^{(i)}\| \Big)}.$$

This in turn indicates that if Step 5 is entered then the new bundle element is added to the bundle $\mathcal{B}_1$ and possibly also to the bundle $\mathcal{B}_2$. From Theorem 2.1 it also follows that in this case the corresponding linearization errors are always smaller than or equal to the crucial proximity measure $\varepsilon > 0$, which is used in the bundle deletion criteria at Step 3. Therefore, these bundle elements are never removed from the bundles. We also notice that whenever Step 3 is executed all bundle elements, for which the linearization error is larger than ε, are deleted from both bundles $\mathcal{B}_1$ and $\mathcal{B}_2$.

From the above considerations it follows that there exists an index $\bar{\imath} \in \mathcal{I}$ such that for all $i > \bar{\imath}$ nothing is removed from the bundles $\mathcal{B}_1$ and $\mathcal{B}_2$ when Step 3 is performed. Because according to Theorem 3.1 the solution to the problem (14) is always of the form

$$\boldsymbol{d}_t^{(i)} = -t_i \Bigg(\sum_{j \in J_1^{(i)}} \lambda_{t_i,j}^{(i)}(j^*) \boldsymbol{\xi}_{1,j} - \boldsymbol{\xi}_{2,j^*} \Bigg),$$

where $j^* \in J_2^{(i)}$ and $\sum_{j \in J_1^{(i)}} \lambda_{t_i,j}^{(i)}(j^*) = 1$, we obtain that for each index $i > \bar{\imath}$ and $i \in \mathcal{I}$ the direction $\boldsymbol{d}_t^{(i)}$ can be written as

$$\boldsymbol{d}_t^{(i)} = -t_i \left(\boldsymbol{\xi}_1^{(i)} - \boldsymbol{\xi}_{2,j^*} \right) \quad \text{with } \boldsymbol{\xi}_1^{(i)} \in \operatorname{conv} \left\{ \boldsymbol{\xi}_{1,j} \,|\, j \in J_1^{(i)} \right\}.$$

However, since $\left\| \boldsymbol{d}_t^{(i)} \right\| < \theta_i$ and $\left\| \boldsymbol{\xi}_1^{*\,(i)} - \boldsymbol{\xi}_2^{*\,(i)} \right\| > \delta$, we get

$$\theta_i > \left\| \boldsymbol{d}_t^{(i)} \right\| = t_i \left\| \boldsymbol{\xi}_1^{(i)} - \boldsymbol{\xi}_{2,j^*} \right\| \geq t_{min}^{(i)} \left\| \boldsymbol{\xi}_1^{*\,(i)} - \boldsymbol{\xi}_2^{*\,(i)} \right\| > t_{min}^{(i)} \delta = \frac{\theta_i}{r\delta}\, \delta = \frac{\theta_i}{r} > \theta_i.$$

This contradicts the assumption that the algorithm does not stop. □

LEMMA 5.6. *Algorithm 4.1 cannot pass infinitely many times through the sequence of steps from 4 to 6.*

Proof. We again suppose, contrary to our claim, that the sequence of steps from 4 to 6 is executed infinitely many times. We index by $i \in \mathcal{I}$ all the quantities referred to the ith passage. The previous lemma now ensures that Step 3 cannot be executed infinitely many times, so we can especially assume that Step 3 is not entered after some index $\bar{\imath} \in \mathcal{I}$. This means that

$$\|\boldsymbol{d}_t^{(i)}\| \geq \theta_i \quad \text{for all } i > \bar{\imath},$$

since otherwise we would end up in Step 3.

We begin by observing that Step 5(a) cannot occur infinitely many times. This is due to the fact that at Step 5(a) the proximity parameter t is reduced, while $t_{\min}$ is both bounded and monotonically nonincreasing. Thus, the proximity parameter t becomes arbitrarily close to the parameter $t_{\min}$. In

addition, the parameter $t_{\min}$ is always smaller than $\varepsilon_1/2(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\|+\|\boldsymbol{\xi}_{2,\max}^{(i)}\|)$ and therefore after a finite number of iterations also the proximity parameter t falls below the threshold $\varepsilon_1/2(\|\boldsymbol{\xi}_1(\boldsymbol{x}_k)\| + \|\boldsymbol{\xi}_{2,\max}^{(i)}\|)$. After this Step 5(a) cannot be executed, since $\|\boldsymbol{d}_t^{(i)}\| \leq \varepsilon_1$ according to Lemma 5.2. Thus, there exits an index $\hat{\imath} \in \mathcal{I}$ such that $\hat{\imath} > \bar{\imath}$ and for all $i \geq \hat{\imath}$ each bundle element corresponding to the new auxiliary point is inserted into $\mathcal{B}_1$ and possibly also into $\mathcal{B}_2$ while the parameter t is remaining unchanged.

Now, Lemma 5.2 together with the parameter selection rule $t \in [t_{\min}, t_{\max}]$ guarantees that the sequence $\{\boldsymbol{d}_t^{(i)}\}_{i\in\mathcal{I}}$ is bounded in norm and there exists a convergent subsequence $\{\boldsymbol{d}_t^{(i)}\}_{i\in\mathcal{I}'\subseteq\mathcal{I}}$ which converges to a limit $\hat{\boldsymbol{d}}$. In addition, the assumptions **A1** and **A2** together with the bundle insertion rule at Step 5 of the 'main iteration' guarantee that all the points $\boldsymbol{y}_j$ corresponding to the bundle elements inserted into the bundles $\mathcal{B}_1$ and $\mathcal{B}_2$ are on the compact set $\mathcal{F}_\varepsilon$. Also all the iteration points $\boldsymbol{x}_k$ are on the compact set $\mathcal{F}_0 \subset \mathcal{F}_\varepsilon$, since the algorithm is a descent one. Thus, there exists a constant $K > 0$ such that

$$\|\boldsymbol{x}_k - \boldsymbol{y}_j\| \leq K \quad \text{for all points } \boldsymbol{y}_j \text{ on } \mathcal{B}_1.$$

Combining this information with the assumption **A2** we see that

$$\begin{aligned} |\alpha_{1,j}| &= |f_1(\boldsymbol{x}_k) - f_1(\boldsymbol{y}_j) - (\boldsymbol{\xi}_{1,j})^T(\boldsymbol{x}_k - \boldsymbol{y}_j)| \\ &\leq |f_1(\boldsymbol{x}_k) - f_1(\boldsymbol{y}_j)| + \|\boldsymbol{\xi}_{1,j}\|\|(\boldsymbol{x}_k - \boldsymbol{y}_j)\| \\ &\leq L_1\|\boldsymbol{x}_k - \boldsymbol{y}_j\| + L_1 K \leq 2L_1 K \end{aligned}$$

for all points $\boldsymbol{y}_j$ on $\mathcal{B}_1$, since according to Remark 5.1 we always have $\|\boldsymbol{\xi}_{1,j}\| \leq L_1$. Similar considerations apply also for all points $\boldsymbol{y}_j$ on $\mathcal{B}_2$ and thus subgradients $\boldsymbol{\xi}_{i,j}$ and linearization errors $\alpha_{i,j}$ are bounded for $i = 1, 2$.

The above considerations imply that the corresponding subsequences $\{\Delta_1(\boldsymbol{d}_t^{(i)})\}_{i\in\mathcal{I}'\subseteq\mathcal{I}}$ and $\{\Delta_2(\boldsymbol{d}_t^{(i)})\}_{i\in\mathcal{I}'\subseteq\mathcal{I}}$ are also bounded. Hence, they admit a convergent subsequence for $i \in \mathcal{I}'' \subseteq \mathcal{I}'$ and we denote the limits by $\hat{\Delta}_1$ and $\hat{\Delta}_2$, respectively. Moreover, since $\|\boldsymbol{d}_t^{(i)}\| \geq \theta_i$, we obtain as a consequence of the property (iii) of Lemma 3.2 that

$$\Delta_1\big(\boldsymbol{d}_t^{(i)}\big) + \Delta_2\big(\boldsymbol{d}_t^{(i)}\big) \leq -\frac{1}{2t_i}\big\|\boldsymbol{d}_t^{(i)}\big\|^2 \leq -\frac{\theta_i^2}{2t_i} < 0 \quad \text{for all } i \in \mathcal{I} \tag{26}$$

and thus also

$$\hat{\Delta}_1 + \hat{\Delta}_2 \leq -\frac{\hat{\theta}^2}{2\hat{t}} < 0,$$

where $\hat{t} = \lim_{i\to\infty} t_i > 0$ and $\hat{\theta} = \lim_{i\to\infty} \theta_i > 0$. Now the strictly positive limit $\hat{t}$ exists for the parameter t_i, because t_i is not changed after the round $\hat{\imath} \in \mathcal{I}$ and the safeguard parameter $t_{\min}^{(i)}$ is always strictly positive according to Remark 5.1. Furthermore, from Remark 5.1 we obtain that the local proximity measure θ_i is always strictly positive. Since θ_i is bounded below and also nonincreasing, it has a strictly positive limit $\hat{\theta}$.

Next, let r and s be two successive indices in $\mathcal{I}''$ and $\alpha_{1,r} = f_1(\boldsymbol{x}_k) - f_1(\boldsymbol{x}_k + \boldsymbol{d}_t^{(r)}) + (\boldsymbol{\xi}_{1,r})^T \boldsymbol{d}_t^{(r)}$ with $\boldsymbol{\xi}_{1,r} \in \partial f_1(\boldsymbol{x}_k + \boldsymbol{d}_t^{(r)})$. We get

$$(\boldsymbol{\xi}_{1,r})^T \boldsymbol{d}_t^{(r)} - \alpha_{1,r} > m\Delta_1\Big(\boldsymbol{d}_t^{(r)}\Big) + (m-1)\Delta_2\Big(\boldsymbol{d}_t^{(r)}\Big) \tag{27}$$

and

$$\Delta_1\Big(\boldsymbol{d}_t^{(s)}\Big) \geq (\boldsymbol{\xi}_{1,r})^T \boldsymbol{d}_t^{(s)} - \alpha_{1,r}\,, \tag{28}$$

where the first inequality follows from Lemma 5.4 and the latter one is an immediate consequence of the definition of $\Delta_1(\boldsymbol{d})$. Finally, combining of (27) and (28) gives

$$\Delta_1\Big(\boldsymbol{d}_t^{(s)}\Big) - m\Delta_1\Big(\boldsymbol{d}_t^{(r)}\Big) + (1-m)\Delta_2\Big(\boldsymbol{d}_t^{(r)}\Big) > (\boldsymbol{\xi}_{1,r})^T\Big(\boldsymbol{d}_t^{(s)} - \boldsymbol{d}_t^{(r)}\Big)$$

and passing to the limit yields

$$(1-m)\Big(\hat{\Delta}_1 + \hat{\Delta}_2\Big) \geq 0,$$

which contradicts the fact that $\hat{\Delta}_1 + \hat{\Delta}_2 < 0$, since $m \in (0,1)$. □

From Lemmas 5.5 and 5.6 we immediately get the next result.

THEOREM 5.7. *The 'main iteration' terminates after a finite number of steps.*

Finally, we show the finite termination of the overall bundle algorithm. The proof is again obtained in the same way as in [14, 15, 16]. Moreover, we can show that the convergence of the PBDC algorithm does not depend on the starting point $\boldsymbol{x}_0$ if the assumption **A1** holds for all $\boldsymbol{x}_0 \in \mathbb{R}^n$. Thus, PBDC can be called globally convergent in a sense that starting from any arbitrary point $\boldsymbol{x}_0 \in \mathbb{R}^n$ it generates a sequence $\{\boldsymbol{x}_k\}$ that converges to an ε-critical point $\boldsymbol{x}^*$. Note that the solution obtained is actually a critical one if the stopping condition tested at Step 0 of the 'main iteration' is the cause of the termination of the algorithm.

THEOREM 5.8. *For any parameter $\delta > 0$ and $\varepsilon > 0$, the execution of the PBDC algorithm 4.2 stops after a finite number of 'main iterations' at a point $\boldsymbol{x}^*$ satisfying the approximate ε-criticality condition*

$$\|\boldsymbol{\xi}_1^* - \boldsymbol{\xi}_2^*\| \leq \delta \quad \textit{with } \boldsymbol{\xi}_1^* \in \partial_\varepsilon f_1(\boldsymbol{x}^*) \textit{ and } \boldsymbol{\xi}_2^* \in \partial_\varepsilon f_2(\boldsymbol{x}^*). \tag{29}$$

Proof. First of all, the stopping condition tested both at Step 0 and 3 of the 'main iteration' is exactly the approximate ε-criticality condition (29). Next, we prove that one of these stopping conditions has to be verified after

a finite number of 'main iterations', because otherwise the objective function f is not bounded below.

Suppose by contradiction that the claim is false. Then the 'main iteration' is entered infinitely many times and let us index by $k \in \mathcal{K}$ all the quantities obtained from the kth passage. From Theorem 5.7 it follows that each of these 'main iterations' produces a new iteration point and especially the descent condition (21) is satisfied. Thus

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1}) \leq m\Big(\Delta_1\big(\boldsymbol{d}_t^{(k)}\big) + \Delta_2\big(\boldsymbol{d}_t^{(k)}\big)\Big) \quad \text{for all } k \in \mathcal{K}$$

and summing up the first k of the above inequalities yields

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_0) \leq \sum_{i=1}^{k} m\Big(\Delta_1\big(\boldsymbol{d}_t^{(i)}\big) + \Delta_2\big(\boldsymbol{d}_t^{(i)}\big)\Big).$$

Letting $k \to \infty$ and taking into account (26), together with the facts that θ_k is bounded away from zero and $t_k \leq t_{\max}^{(k)} < \infty$, we conclude that

$$\lim_{k\to\infty} f(\boldsymbol{x}_k) - f(\boldsymbol{x}_0) \leq -\infty.$$

This is a contradiction, since the function f is assumed to be finite. □

6 Computational results

In this section we present some numerical results to verify the practical efficiency of the proposed PBDC algorithm. We especially consider some academic test problems with nonsmooth DC objectives and those test problems are introduced in the next subsection. After that we give a short description of the implementation of the PBDC algorithm and also present three other existing methods for nonsmooth optimization which are used for comparisons. Finally, we report our numerical experiments and analyze the results.

6.1 Test problems

All test problems are unconstrained DC optimization problems, where objective functions are presented as DC functions:

$$f(\boldsymbol{x}) = f_1(\boldsymbol{x}) - f_2(\boldsymbol{x}).$$

Therefore in the description of all test problems we present only functions f_1 and f_2. The following notations are used to describe test problems:

$\boldsymbol{x}_0 \in \mathbb{R}^n$ – starting point;
$\boldsymbol{x}^* \in \mathbb{R}^n$ – known best solution;
f^* – known best value of the objective function.

PROBLEM 1. *[4]*
Dimension: $n = 2$,
Component functions:
$f_1(\boldsymbol{x}) = \max\{f_1^1(\boldsymbol{x}), f_1^2(\boldsymbol{x}), f_1^3(\boldsymbol{x})\} + f_2^1(\boldsymbol{x}) + f_2^2(\boldsymbol{x}) + f_2^3(\boldsymbol{x})$,
$f_2(\boldsymbol{x}) = \max\{f_2^1(\boldsymbol{x}) + f_2^2(\boldsymbol{x}), f_2^2(\boldsymbol{x}) + f_2^3(\boldsymbol{x}), f_2^1(\boldsymbol{x}) + f_2^3(\boldsymbol{x})\}$,
$f_1^1(\boldsymbol{x}) = x_1^4 + x_2^2$, $f_1^2(\boldsymbol{x}) = (2 - x_1)^2 + (2 - x_2)^2$, $f_1^3(\boldsymbol{x}) = 2e^{-x_1+x_2}$,
$f_2^1(\boldsymbol{x}) = x_1^2 - 2x_1 + x_2^2 - 4x_2 + 4$, $f_2^2(\boldsymbol{x}) = 2x_1^2 - 5x_1 + x_2^2 - 2x_2 + 4$,
$f_2^3(\boldsymbol{x}) = x_1^2 + 2x_2^2 - 4x_2 + 1$,
Starting point: $\boldsymbol{x}_0 = (2, 2)^T$,
Optimum point: $\boldsymbol{x}^* = (1, 1)^T$,
Optimum value: $f^* = 2$.

PROBLEM 2. *[4]*
Dimension: $n = 2$,
Component functions: $f_1(\boldsymbol{x}) = |x_1 - 1| + 200 \max\{0, |x_1| - x_2\}$,
$f_2(\boldsymbol{x}) = 100(|x_1| - x_2)$,
Starting point: $\boldsymbol{x}_0 = (-1.2, 1)^T$,
Optimum point: $\boldsymbol{x}^* = (1, 1)^T$,
Optimum value: $f^* = 0$.

PROBLEM 3. *[4]*
Dimension: $n = 4$,
Component functions: $f_1(\boldsymbol{x}) = |x_1 - 1| + 200 \max\{0, |x_1| - x_2\}$
$+180 \max\{0, |x_3| - x_4\} + |x_3 - 1| + 10.1(|x_2 - 1| + |x_4 - 1|) + 4.95|x_2 + x_4 - 2|$,
$f_2(\boldsymbol{x}) = 100(|x_1| - x_2) + 90(|x_3| - x_4) + 4.95|x_2 - x_4|$,
Starting point: $\boldsymbol{x}_0 = (1, 3, 3, 1)^T$,
Optimum point: $\boldsymbol{x}^* = (1, 1, 1, 1)^T$,
Optimum value: $f^* = 0$.

PROBLEM 4. *[4]*
Dimension: $n = 2, 5, 10, 50, 100, 150, 200, 250, 350, 500, 750$,
Component functions: $f_1(\boldsymbol{x}) = n \max\{|x_i| : \ i = 1, \ldots, n\}$, $\quad f_2(\boldsymbol{x}) = \sum_{i=1}^n |x_i|$,
Starting point: $\boldsymbol{x}_0 = (i, \ i = 1, \ldots, \lfloor n/2 \rfloor, \ -i, \ i = \lfloor n/2 \rfloor + 1, \ldots, n)^T$,
Optimum point: $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)^T$, $x_i^* = \alpha$ *or* $x_i^* = -\alpha$, $\alpha \in \mathbb{R}$, $i = 1, \ldots, n$,
Optimum value: $f^* = 0$.

PROBLEM 5. *[4]*
Dimension: $n = 2, 5, 10, 50, 100, 150, 200, 250, 300, 350, 400, 500, 1000, 1500, 3000, 10\,000, 15\,000, 20\,000, 50\,000$,
Component functions: $f_1(\boldsymbol{x}) = 20 \max\left\{\left|\sum_{i=1}^n (x_i - x_i^*) t_j^{i-1}\right| : j = 1, \ldots, 20\right\}$,
$f_2(\boldsymbol{x}) = \sum_{j=1}^{20} \left|\sum_{i=1}^n (x_i - x_i^*) t_j^{i-1}\right|$, $t_j = 0.05j$, $j = 1, \ldots, 20$,
Starting point: $\boldsymbol{x}_0 = (0, \ldots, 0)^T$,
Optimum point: $\boldsymbol{x}^* = (1/n, \ldots, 1/n)^T$,
Optimum value: $f^* = 0$.

PROBLEM 6.
Dimension: $n = 2$
Component functions: $f_1(\boldsymbol{x}) = x_2 + 0.1(x_1^2 + x_2^2) + 10\max\{0, -x_2\}$,
$f_2(\boldsymbol{x}) = |x_1| + |x_2|$,
Starting point: $\boldsymbol{x}_0 = (10, 1)^T$,
Optimum point: $\boldsymbol{x}^* = (5, 0)^T$,
Optimum value: $f^* = -2.5$.

PROBLEM 7.
Dimension: $n = 2$
Component functions: $f_1(\boldsymbol{x}) = |x_1 - 1| + 200\max\{0, |x_1| - x_2\}$
$+ 10\max\{x_1^2 + x_2^2 + |x_2|, x_1 + x_1^2 + x_2^2 + |x_2| - 0.5, |x_1 - x_2| + |x_2| - 1, x_1 + x_1^2 + x_2^2\}$,
$f_2(\boldsymbol{x}) = 100(|x_1| - x_2) + 10(x_1^2 + x_2^2 + |x_2|)$,
Starting point: $\boldsymbol{x}_0 = (-2, 1)^T$,
Optimum point: $\boldsymbol{x}^* = (0.5, 0.5)^T$,
Optimum value: $f^* = 0.5$.

PROBLEM 8.
Dimension: $n = 3$
Component functions: $f_1(\boldsymbol{x}) = 9 - 8x_1 - 6x_2 - 4x_3 + 2|x_1| + 2|x_2| + 2|x_3|$
$+ 4x_1^2 + 2x_2^2 + 2x_3^2 + 10\max\{0, x_1 + x_2 + 2x_3 - 3, -x_1, -x_2, -x_3\}$,
$f_2(\boldsymbol{x}) = |x_1 - x_2| + |x_1 - x_3|$,
Starting point: $\boldsymbol{x}_0 = (0.5, 0.5, 0.5)^T$,
Optimum point: $\boldsymbol{x}^* = (0.75, 1.25, 0.25)^T$,
Optimum value: $f^* = 3.5$.

PROBLEM 9.
Dimension: $n = 4$
Component functions: $f_1(\boldsymbol{x}) = x_1^2 + (x_1 - 1)^2 + 2(x_1 - 2)^2 + (x_1 - 3)^2 + 2x_2^2$
$+ (x_2 - 1)^2 + 2(x_2 - 2)^2 + x_3^2 + (x_3 - 1)^2 + 2(x_3 - 2)^2 + (x_3 - 3)^2$
$+ 2x_4^2 + (x_4 - 1)^2 + 2(x_4 - 2)^2$
$f_2(\boldsymbol{x}) = \max\{(x_1 - 2)^2 + x_2^2, (x_3 - 2)^2 + x_4^2\}$
$+ \max\{(x_1 - 2)^2 + (x_2 - 1)^2, (x_3 - 2)^2 + (x_4 - 1)^2\}$
$+ \max\{(x_1 - 3)^2 + x_2^2, (x_3 - 3)^2 + x_4^2\} + \max\{x_1^2 + (x_2 - 2)^2, x_3^2 + (x_4 - 2)^2\}$
$+ \max\{(x_1 - 1)^2 + (x_2 - 2)^2, (x_3 - 1)^2 + (x_4 - 2)^2\}$,
Starting point: $\boldsymbol{x}_0 = (4, 2, 4, 2)^T$,
Optimum point: $\boldsymbol{x}^* = (7/3, 1/3, 0.5, 2)$,
Optimum value: $f^* = 11/6$.

PROBLEM 10.
Dimension: $n = 2, 4, 5, 10, 20, 50, 100, 150, 200$
Component functions: $f_1(\boldsymbol{x}) = \sum_{i=1}^n x_i^2, \quad f_2(\boldsymbol{x}) = \sum_{i=2}^n |x_i - x_{i-1}|$,
Starting point: $\boldsymbol{x}_0 = (x_{0,1}, \ldots, x_{0,n})^T$, $x_{0,i} = 0.1i$,
Optimum point: For even n: $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)^T$: $x_1^* = -0.5$, $x_n^* = 0.5$,
$x_{2i}^* = 1$, $i = 1, \ldots, n/2 - 1$, $x_{2i+1}^* = 1$, $i = 1, \ldots, n/2 - 1$,
For odd $n \geq 3$: $x^* = (x_1^*, \ldots, x_n^*)^T$: $x_1^* = -0.5$, $x_n^* = 0.5$, $x_j^* = 0$,
$j = \lfloor n/2 \rfloor + 1$, $x_{2i}^* = 1$, $x_{2i+1}^* = -1$, *for* $2i \leq \lfloor n/2 \rfloor$, $x_{2i}^* = -1$,
$x_{2i+1}^* = 1$, *for* $2i > \lfloor n/2 \rfloor + 1$,
Optimum value: For even n: $f^* = 1.5 - n$,
For odd $n \geq 3$: $f^* = 2.5 - n$.

6.2 Implementation and parameters

In order to compare the results obtained with the PBDC algorithm, we have used three other algorithms for nonsmooth optimization. The tested implementations of all the algorithms are presented in Table 1 together with the references to more detailed description of the methods. In what follows, we give a short description of each algorithm and its implementation.

Table 1: Tested pieces of software

Software	Author	Algorithm	Reference
`PBDC`	Joki	Bundle method for DC function	
`MPBNCG`	Mäkelä	Proximal bundle method	[30, 31]
`NonsmoothDCA`	Bagirov	DCA (DC algorithm)	[2, 3]
`TCM`	Bagirov	Truncated codifferential method	[6]

`PBDC` is an implementation of the bundle method proposed in this article. However, this implementation slightly differs from the PBDC algorithm presented in Section 4, since in the 'main iteration' algorithm it utilizes a subgradient aggregation technique of the type introduced by Kiwiel [24] into the bundle $\mathcal{B}_1$ (a similar type of aggregation is also used in [14]). This way we are able to store some information from the previous iterations even thought the size of the bundle $\mathcal{B}_1$ has to be kept bounded in the implementation.

Next we look closer how the aggregation scheme is performed in 'main iteration' Algorithm 4.1. First of all, let $i^* \in J_2$ be the index of the subproblem (16) yielding the global solution $\boldsymbol{d}_t$ at Step 2 of the 'main iteration'. Now according to Theorem 3.1

$$\boldsymbol{d}_t = \boldsymbol{d}_t(i^*) = -t\left(\sum_{j\in J_1} \lambda_{t,j}(i^*)\boldsymbol{\xi}_{1,j} - \boldsymbol{\xi}_{2,i^*}\right)$$
$$v_t(i^*) = -\frac{1}{t}\|\boldsymbol{d}_t(i^*)\|^2 - \sum_{j\in J_1} \lambda_{t,j}(i^*)\alpha_{1,j} + \alpha_{2,i^*}.$$

Thus, we are able to compute the following aggregated quantities

$$\boldsymbol{\xi}_{1,a} = \sum_{j\in J_1} \lambda_{t,j}(i^*)\boldsymbol{\xi}_{1,j} = -\frac{1}{t}\boldsymbol{d}_t(i^*) + \boldsymbol{\xi}_{2,i^*}$$
$$\alpha_{1,a} = \sum_{j\in J_1} \lambda_{t,j}(i^*)\alpha_{1,j} = -\frac{1}{t}\|\boldsymbol{d}_t(i^*)\|^2 - v_t(i^*) + \alpha_{2,i^*},$$

which can be inserted into the bundle $\mathcal{B}_1$ at Step 2 of the 'main iteration' algorithm. It is also worth noting that we store only one aggregated element at a time and for this reason the old aggregated element is overwritten each

time when we obtain a new one. We can also easily verify that for $i^* \in J_2$ the solution $\boldsymbol{d}_t(i^*)$ of the subproblem

$$\begin{cases} \text{minimize} & (\boldsymbol{\xi}_{1,a} - \boldsymbol{\xi}_{2,i^*})^T \boldsymbol{d} - \alpha_{1,a} + \alpha_{2,i^*}^k + \frac{1}{2t}\|\boldsymbol{d}\|^2 \\ \text{subject to} & \boldsymbol{d} \in \mathbb{R}^n, \end{cases}$$

where the bundle $\mathcal{B}_1$ contains only the aggregated element, is also $\boldsymbol{d}_t$. Thus, we are never going to overwrite the bundle element of $\mathcal{B}_2$ which yields the most recent global solution.

`PBDC` is implemented in double precision Fortran 95. The subroutine `PLQDF1` [27] is used to solve the quadratic subproblems (17) and the norm minimization problem (19) is solved by the subroutine `PVMM` [28], which is a variable metric algorithm for unconstrained and linearly constrained optimization. The Fortran source code of `PBDC` can be downloaded from http://napsu.karmitsa.fi/pbdc/.

`MPBNGC` is an implementation of the proximal bundle method [31], which is designed for a general nonsmooth objective. Therefore, we cannot utilize the DC decomposition of the objective function f. However, the function f is assumed to be locally Lipschitz continuous. In the method we approximate the subdifferential of the original objective function f with a bundle, which consists of subgradients of the objective function from the previous iterations. Similarly to the PBDC method this subgradient information is used to construct a specific cutting plane model for the objective function f and a search direction is then obtained as a solution to a quadratic direction finding problem using this piecewise linear approximation. If the direction obtained yields a sufficient descent, then we are able to calculate a new iteration point. Otherwise we add more information into the bundle to improve the cutting plane model and solve a new quadratic direction finding problem. This method utilizes also the subgradient aggregation technique [24]. Moreover, linearization errors are substituted with subgradient locality measures [25]. It is proved that under the upper semi-smoothness assumption [9] the proximal bundle method converges to a substationarity point of the function f [26, 31, 34].

The code of `MPBNGC` includes constraint handling together with a possibility to solve multiobjective programming problems. This software also uses the subroutine `PLQDF1` [27] to solve the quadratic direction finding problem usually encountered in bundle methods. The Fortran 77 source code of `MPBNGC` can be downloaded from http://napsu.karmitsa.fi/proxbundle/.

`NonsmoothDCA` is an implementation of the well-known DCA (DC algorithm) [2, 3]. Next we give a very brief description of the algorithm. Detailed description of DCA and its deeper insight analysis can be found in [2, 3].

The main idea behind DCA is to replace in the DC programming problem (1), at the current point $\boldsymbol{x}_k$, the second DC component f_2 with its affine minorization defined by

$$\tilde{f}_2^k(\boldsymbol{x}) = f_2(\boldsymbol{x}_k) + \boldsymbol{\xi}_{2,k}^T(\boldsymbol{x} - \boldsymbol{x}_k),$$

where $\boldsymbol{\xi}_{2,k} \in \partial f_2(\boldsymbol{x}_k)$ is a subgradient calculated at $\boldsymbol{x}_k$. Then the next iteration point $\boldsymbol{x}_{k+1}$ is obtained as a solution to the convex program

$$\begin{cases} \text{minimize} & f_1(\boldsymbol{x}) - \tilde{f}_2^k(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathbb{R}^n. \end{cases} \tag{30}$$

The algorithm repeats iterations until the convergence of the sequence $\{\boldsymbol{x}_k\}$. In general, the DCA converges to a critical point of the function f.

`NonsmoothDCA` is implemented in Fortran 77. Since in most test problems the function f_1 is nonsmooth the problem (30) is solved using the implementation of the quasisecant method from [5]. The Fortran 77 source code can be downloaded from http://napsu.karmitsa.fi/dca/.

`TCM` is an implementation of the truncated codifferential method, which was introduced in [6]. The truncated codifferential method is based on the concept of codifferential [11] and it uses the explicit DC decomposition of the function f. At the current iteration point $\boldsymbol{x}_k$ this method approximates the hypodifferential [11] of the DC component f_1 using subgradients at points from the ball with a given radius and the subdifferential of the DC component f_2 at the current iteration point $\boldsymbol{x}_k$. A search direction is then found by using these approximations. If this search direction gives a sufficient descent then the new iteration point $\boldsymbol{x}_{k+1}$ is computed, otherwise the algorithm improves the approximation of the codifferential. It is proved that a procedure for finding search directions is finite convergent: after finite number of steps either the algorithm finds the descent direction or some necessary condition for a minimum is satisfied. In the latter case the algorithm decreases the radius of the ball. It is also showed that under some conditions the truncated codifferential method converges to an inf-stationary point [11] of the objective function f.

In this paper we have used the same implementation for the truncated codifferential method as in [6]. The Fortran 77 source code of TCM can be downloaded form http://napsu.karmitsa.fi/tcm/.

All the algorithms were implemented in Fortran with double precision arithmetic. The codes were compiled by using `f95`, the Fortran 95 compiler, and tests were performed on an Intel® Core™ i5-2400 CPU (3.10GHz, 3.10GHz) running on Windows 7.

The preliminary testing of PBDC has showed that the tuning of the parameters is particularly relevant to the performance of the new method. Therefore, we have chosen the criticality tolerance

$$\delta = \begin{cases} 0.005n, & \text{if } n < 150 \\ 0.015n, & \text{if } 150 \leq n \leq 200 \\ 0.05n, & \text{if } n > 200, \end{cases}$$

the proximity measure $\varepsilon = 0.1$, the decrease parameter

$$r = \begin{cases} 0.75, & \text{if } n < 10 \\ \text{the first two decimals of } n/(n+5), & \text{if } 10 \leq n < 300 \\ 0.99, & \text{if } n \geq 300, \end{cases}$$

the increase parameter $R = 10^7$ and the descent parameter $m = 0.2$. Moreover, the proximity parameter t is set to $0.8(t_{\min}+t_{\max})$, whenever it has to be selected from the interval $[t_{\min}, t_{\max}]$. In order to implement the algorithm, we have limited the number of stored subgradients and the maximum size of the bundle $\mathcal{B}_1$ has been set to $\min\{n+5, 1000\}$ (the aggregated element is not taken into account in this value). Furthermore, with $n = 50\,000$, we had to limit the size of the bundle $\mathcal{B}_1$ to 20, because otherwise we could not compile the code PBDC. In all cases, the size of the bundle $\mathcal{B}_2$ has been set to 3. The PBDC algorithm needs also approximations of Lipschitz constants of DC components and, since these values can be overestimated, we have selected $L_1 = L_2 = 1000$.

In MPBNGC, the maximum size of the bundle has been set to $\min\{n+3, 1000\}$, when $n < 20\,000$. For larger dimensions, we have used the bundle size 20. Moreover, the final objective function accuracy parameter ε has been set to 10^{-10}. For all the other parameters we have used the default settings of the code MPBNGC [30]. Furthermore, in the parameter selections of NonsmoothDCA and TCM we have used the defaults values [5, 6].

6.3 Numerical results

We have summarized the results of our numerical experiments in Table 2, where we have used the following notations:

- *Prob.* is the number of the problem
- n is the number of variables
- n_f is the number of function evaluations for the objective function f
- n_{f_i} is the number of function evaluations for the DC component f_i
- $n_{\boldsymbol{\xi}}$ is the number of subgradient evaluations for the objective function f
- $n_{\boldsymbol{\xi}_i}$ is the number of subgradient evaluations for the DC component f_i

- *time* is the CPU time in seconds
- f is the obtained value of the objective function when the algorithm stops.

In addition, MPBNGC uses the same amount of function and subgradient evaluations, that is $n_f = n_{\boldsymbol{\xi}}$. It is also worth noting that for some solvers we report the function and subgradient evaluations for the DC components. Therefore, to obtain somewhat comparable results we have summed up the DC component-wise values and compared $n_{f_1} + n_{f_2}$ and $n_{\boldsymbol{\xi}_1} + n_{\boldsymbol{\xi}_2}$ to n_f and $n_{\boldsymbol{\xi}}$, respectively. However, this comparison does not take into account the fact that the evaluations for the DC components are less costly than the ones for the original objective function. Due to this the values $n_{f_1} + n_{f_2}$ and $n_{\boldsymbol{\xi}_1} + n_{\boldsymbol{\xi}_2}$ overestimate computational effort when compared to n_f and $n_{\boldsymbol{\xi}}$.

The results presented in Table 2 show that the new solver PBDC is more reliable to find global minimizers than the other solvers. PBDC only fails to find the global minimizer in two cases out of 46 (Problem 10 ($n = 150$, 200)) and both of these problems are quite difficult, since they are not solved globally by any of the solvers. Moreover, the solver PBDC seems to be especially efficient to solve globally Problems 7–10, whereas the other methods usually do not succeed in that, except for TCM. Despite of being more unreliable than PBDC, the solvers MPBNGC and TCM have also quite a good performance, since they both find the best solution approximately in 75 % of the cases. However, the solver NonsmoothDCA is the most unreliable one, since it fails to find the global minimizer in 21 cases out of 46. It is also worth noting that in Problem 5, when $n \geq 3000$, neither of the solvers NonsmoothDCA and TCM can be compiled or run. The same thing happens also in MPBNGC, when $n \geq 20\,000$, and therefore the new solver PBDC is the only one yielding a solution.

From Table 2 we can observe, that the new solver PBDC is the one using the least evaluations in Problems 2–6. In other problems PBDC usually loses to MPBNGC, since the function and subgradient evaluations of PBDC are a little bit greater than those of MPBNGC. However, it is worth noting that in some of the problems (Problems 7–10) MPBNGC does not find the global minimizer unlike PBDC, making it hard to say if MPBNGC really is more efficient than PBDC in those cases. The solvers PBDC and MPBNGC are also the fastest ones and in general the CPU times of those solvers are approximately the same. Only in some cases of Problem 4 ($n = 150$, 200, 250) and Problem 5 ($n = 15\,000$) PBDC loses to MPBNGC. However, when the dimension of Problem 4 grows (i.e. $n = 350, 500, 750$), the new solver PBDC is much faster than MPBNGC. Furthermore, NonsmoothDCA uses significantly more computational efforts than the other solvers and therefore, it has the worst performance among all the methods. TCM requires also a lot more function and subgradient evaluations than PBDC and MPBNGC. Despite of that, the CPU times of TCM are approximately the same as in PBDC and MPBNG, whenever the global minimizer is found.

Table 2: Summary of the numerical results

		PBDC					MPBNGC			NonsmoothDCA						TCM			
Prob.	n	n_f	$n_{\boldsymbol{\xi}_1}$	$n_{\boldsymbol{\xi}_2}$	*time*	f	$n_f, n_{\boldsymbol{\xi}}$	*time*	f	n_{f_1}	n_{f_2}	$n_{\boldsymbol{\xi}_1}$	$n_{\boldsymbol{\xi}_2}$	*time*	f	n_f	$n_{\boldsymbol{\xi}}$	*time*	f
1	2	22	17	16	0.00	2.0000000198	17	0.00	2.0000000000637	116	4	80	2	0.00	3.1638363108 13*	333	135	0.01	2.00000000016
2	2	21	15	15	0.00	$1.10977 \cdot 10^{-12}$	39	0.00	$2.30926 \cdot 10^{-14}$	139	4	105	2	0.00	1.000000006012*	526	129	0.01	$8.44528 \cdot 10^{-9}$
3	4	25	15	11	0.00	$2.26888 \cdot 10^{-12}$	112	0.00	$2.22091 \cdot 10^{-11}$	439	5	358	3	0.00	$3.87083 \cdot 10^{-9}$	517	218	0.01	$3.71039 \cdot 10^{-9}$
4	2	6	3	3	0.00	$8.43769 \cdot 10^{-15}$	7	0.00	$4.440892 \cdot 10^{-16}$	100	4	53	2	0.01	$2.43370 \cdot 10^{-10}$	125	55	0.00	$4.05524 \cdot 10^{-11}$
4	5	13	6	5	0.00	0.000000000000	30	0.00	$3.55271 \cdot 10^{-15}$	325	4	246	2	0.00	$2.85320 \cdot 10^{-10}$	399	180	0.00	$3.66982 \cdot 10^{-10}$
4	10	16	11	9	0.00	$5.68434 \cdot 10^{-14}$	61	0.00	0.000000000000	717	4	616	2	0.01	$9.15171 \cdot 10^{-10}$	844	388	0.01	$1.06388 \cdot 10^{-9}$
4	50	52	51	12	0.13	$9.09494 \cdot 10^{-13}$	561	0.29	$1.12621 \cdot 10^{-12}$	5823	6	5609	4	5.32	$1.26078 \cdot 10^{-9}$	6891	3368	3.56	$2.80851 \cdot 10^{-9}$
4	100	102	102	30	1.99	$9.09494 \cdot 10^{-13}$	1489	3.67	$1.17310 \cdot 10^{-11}$	12717	7	12459	5	150.2	$2.71484 \cdot 10^{-9}$	23299	11479	114.3	$1.66346 \cdot 10^{-9}$
4	150	198	198	70	15.00	$1.67092 \cdot 10^{-5}$	1592	10.17	$-1.00897 \cdot 10^{-11}$	974363	647	972524	645	15919	38.55251*	21474	10389	75.11	1156.55819*
4	200	341	341	132	64.15	$3.63797 \cdot 10^{-12}$	3528	36.08	$9.66338 \cdot 10^{-13}$	956640	634	954261	632	12676	570.20237*	26645	12735	51.23	2515.04507*
4	250	502	502	187	210.8	$7.27595 \cdot 10^{-12}$	3619	80.28	$1.04108 \cdot 10^{-10}$	938663	620	935851	618	15295	704.72555*	55145	26833	167.4	4608.73643*
4	350	705	705	331	743.7	$7.27595 \cdot 10^{-12}$	9594	1191.3	$2.66339 \cdot 10^{-10}$	827404	535	821439	533	13052	2091.28680*	72241	34944	207.9	14168.40099*
4	500	1357	1357	677	4120.3	$5.38420 \cdot 10^{-10}$	100000	16367	$-3.54219 \cdot 10^{-10}$	795446	511	788690	509	12504	5472.92134*	61177	29345	157.2	40663.80100*
4	750	2258	2258	1489	22919	$3.55066 \cdot 10^{-9}$	49043	57396	$7.42923 \cdot 10^{-10}$	369108	212	349011	210	4249	20872.37875*	138040	66836	421.8	104683.25419*
5	2	10	4	4	0.00	0.000000000000	5	0.00	$8.88178 \cdot 10^{-16}$	70	4	45	2	0.00	$1.67526 \cdot 10^{-9}$	149	66	0.00	$1.99221 \cdot 10^{-10}$
5	5	15	7	6	0.00	$6.12843 \cdot 10^{-14}$	25	0.00	$3.60822 \cdot 10^{-15}$	943	5	290	3	0.01	$1.53050 \cdot 10^{-10}$	1176	411	0.01	$7.12619 \cdot 10^{-10}$
5	10	21	14	11	0.00	$3.54161 \cdot 10^{-13}$	131	0.01	$8.08532 \cdot 10^{-11}$	104114	573	102099	571	2.01	$1.52056 \cdot 10^{-11}$	1316	479	0.01	$2.56584 \cdot 10^{-5}$
5	50	103	94	69	0.27	$8.10106 \cdot 10^{-13}$	51	0.02	$8.19851 \cdot 10^{-11}$	494993	614	491865	612	55.89	$8.45877 \cdot 10^{-4}$	1683	707	0.04	$1.73820 \cdot 10^{-5}$
5	100	47	28	23	0.36	$8.56592 \cdot 10^{-13}$	45	0.03	$1.00426 \cdot 10^{-10}$	941771	624	939068	622	153.2	$3.05781 \cdot 10^{-3}$	1785	754	0.08	$1.58279 \cdot 10^{-5}$
5	150	80	53	43	0.32	$2.58052 \cdot 10^{-12}$	43	0.05	$7.72262 \cdot 10^{-11}$	923022	611	919835	609	248.7	$7.47693 \cdot 10^{-4}$	1800	778	0.13	$3.56810 \cdot 10^{-5}$
5	200	107	56	47	0.40	$4.63365 \cdot 10^{-12}$	100000	209.8	$3.85558 \cdot 10^{-2}$	925026	614	921923	612	178.6	$3.11222 \cdot 10^{-4}$	1633	685	0.12	$2.95805 \cdot 10^{-6}$
5	250	55	49	40	0.43	$3.05058 \cdot 10^{-11}$	58	0.11	$7.69885 \cdot 10^{-12}$	958656	637	956191	635	184.6	$4.12652 \cdot 10^{-3}$	1744	745	0.19	$4.17524 \cdot 10^{-5}$
5	300	36	27	24	0.48	$1.84993 \cdot 10^{-11}$	105	0.29	$6.85744 \cdot 10^{-11}$	933889	619	931010	617	216.5	$1.93779 \cdot 10^{-3}$	1827	779	0.21	$7.26738 \cdot 10^{-5}$
5	350	43	34	27	0.39	$1.27391 \cdot 10^{-10}$	117	0.55	$9.87137 \cdot 10^{-11}$	948611	629	946077	627	248.6	$1.76198 \cdot 10^{-3}$	1599	678	0.21	$3.30601 \cdot 10^{-4}$
5	400	44	36	24	0.91	$7.88742 \cdot 10^{-11}$	95	0.36	$8.43775 \cdot 10^{-11}$	938748	623	935999	621	277.3	$2.09469 \cdot 10^{-4}$	2274	954	0.40	$8.77218 \cdot 10^{-5}$
5	500	29	22	18	1.46	$2.10487 \cdot 10^{-10}$	52	0.25	$1.84398 \cdot 10^{-12}$	935634	621	932878	619	371.5	$1.62630 \cdot 10^{-4}$	1493	637	0.30	$3.66659 \cdot 10^{-5}$
5	1000	32	26	22	0.62	$4.74784 \cdot 10^{-10}$	42	0.42	$8.81669 \cdot 10^{-11}$	959271	639	957081	637	715.2	$3.34356 \cdot 10^{-2}$	968	414	0.43	$5.39214 \cdot 10^{-4}$
5	1500	25	20	16	0.73	$6.26813 \cdot 10^{-10}$	41	0.74	$4.89645 \cdot 10^{-11}$	950142	631	947652	629	1065.0	$8.81968 \cdot 10^{-3}$	1380	562	0.93	$1.33563 \cdot 10^{-3}$
5	3000	29	24	16	2.36	$1.94150 \cdot 10^{-9}$	33	1.42	$3.54419 \cdot 10^{-11}$					fail					fail
5	10000	22	19	11	4.92	$5.93449 \cdot 10^{-9}$	33	5.53	$5.29412 \cdot 10^{-11}$					fail					fail
5	15000	178	54	48	51.83	$2.65698 \cdot 10^{-8}$	29	7.43	$2.93902 \cdot 10^{-11}$					fail					fail
5	20000	152	43	39	50.16	$1.41162 \cdot 10^{-3}$			fail					fail					fail
5	50000	99	26	26	74.97	$5.71421 \cdot 10^{-4}$			fail					fail					fail
6	2	27	19	15	0.00	−2.49999973	53	0.00	−2.4999999999998	78	4	64	2	0.00	−2.4999999999999973	164	71	0.00	−2.4999999999999942
7	2	72	63	40	0.00	0.5000000035	27	0.00	1.0000000000129*	6671	64	4879	62	0.02	1.0000000013337100*	482	166	0.01	0.50000000572692649
8	3	84	64	40	0.00	3.5000001579	23	0.00	3.7727272727785*	114	4	82	2	0.00	3.7500000000000004*	337	148	0.00	3.5000000000000000
9	4	90	78	41	0.00	1.8333334317	4	0.00	9.199999999999995*	149	6	147	4	0.00	9.19999999999999957*	307	135	0.01	9.1999999999999886*
10	2	21	13	7	0.00	−0.499999982	18	0.00	−0.499999999977	120	4	74	2	0.00	−0.50000000000000000	270	116	0.00	−0.49999999999999967
10	4	23	16	9	0.00	−2.499999788	11	0.00	−2.499999999991	201	5	136	3	0.00	−2.4999999999999991	342	148	0.00	−2.4999999999999996
10	5	22	13	10	0.00	−2.499999876	18	0.00	−2.499999999985	108	4	72	2	0.00	−0.49999999999999800*	341	156	0.01	−2.4999999999999978
10	10	59	45	23	0.01	−8.499999618	27	0.00	−6.499999999994*	504	8	355	6	0.00	−4.49999999999999982*	408	185	0.01	−8.5000000000000000
10	20	60	44	20	0.01	−18.49998801	88	0.00	−16.499999999998*	112	4	78	2	0.00	−0.50000003220404700*	524	237	0.01	−18.500000000000004
10	50	140	121	54	0.12	−48.49989423	5	0.00	−0.500000000000*	114717	661	121080	659	1.65	−4.5000000000000009*	801	350	0.01	−48.500000000000028
10	100	352	343	92	1.50	−98.49997663	141	0.02	−90.49999999996*	123	4	75	2	0.00	−0.50000002475810812*	1166	520	0.62	−98.499999999999986
10	150	447	446	104	4.47	−126.489324*	154	0.04	−134.499999999994*	141	4	88	2	0.01	−0.49999999999999656*	810	332	0.01	−145.54188264339533*
10	200	321	320	86	4.34	−90.4958107*	8	0.00	−82.500000000000*	342628	661	347655	659	28.85	−4.4999999999999991*	1388	587	0.13	−192.50000000000011*

* the obtained value of the objective function f is not optimal

All in all, we can conclude that the new solver `PBDC` is really efficient to solve the DC programming problems presented in this article, since it requires often the least computational effort and also finds the best solution almost in each case. Our results also confirm that `PBDC` has a significantly better ability to find global minimizers of DC functions than the other three methods tested.

7 Conclusions

In this paper, we have developed a new proximal bundle method (PBDC) for unconstrained nonsmooth DC optimization. The overall structure of the algorithm is a quite typical one for proximal bundle methods. However, in the model construction it utilizes explicitly the DC decomposition of the objective function, since the cutting plane model of the objective function is obtained by combining the separate approximations of the DC components. Therefore, unlike the other bundle methods, PBDC maintains two different bundles which approximate the subdifferentials of the convex DC components. The global convergence of the new method has been proved to an ε-critical point under mild assumptions.

The presented results of numerical experiments confirm that the new PBDC algorithm is efficient for solving nonsmooth DC programming problems. The most interesting fact is that, even though PBDC is only a local solution method, it nearly always succeeds in finding the global minimizer of a problem. Therefore, the new method seems to be a good alternative for minimization of nonsmooth DC functions, if only a DC representation of the objective function is available.

Acknowledgments. This work has been financially supported by the Jenny and Antti Wihuri Foundation, the Turku University Foundation and the University of Turku. The research by Dr. A.M. Bagirov was supported under Australian Research Council's Discovery Projects funding scheme (project number: DP140103213).

References

[1] L.T.H. An and P.D. Tao: Solving a class of linearly constrained indefinite quadratic problems by D.C. algorithms. *Journal of Global Optimization,* 11(3):253–285, 1997.

[2] L.T.H. An and P.D. Tao: The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. *Annals of Operations Research,* 133:23–46, 2005.

[3] L.T.H. An, H.V. Ngai and P.D. Tao: Exact penalty and error bounds in DC programming. *Journal of Global Optimization*, 52(3):509–535, 2012.

[4] A.M. Bagirov: A method for minimizing of quasidifferentiable functions. *Optimization Methods and Software*, 17(1):31–60, 2002.

[5] A.M. Bagirov and A.N. Ganjehlou: A quasisecant method for minimizing nonsmooth functions. *Optimization Methods and Software*, 25(1):3–18, 2010.

[6] A.M. Bagirov and J. Ugon: Codifferential method for minimizing nonsmooth DC functions. *Journal of Global Optimization*, 50(1):3–22, 2011.

[7] A.M. Bagirov and J. Yearwood: A new nonsmooth optimisation algorithm for minimum sum-of-squares clustering problems. *European Journal of Operational Research*, 170(2):578–596, 2006.

[8] A.M. Bagirov, N. Karmitsa and M.M. Mäkelä: *Introduction to Nonsmooth Optimization: Theory, Practice and Software.* Springer, Cham, Heidelberg, 2014.

[9] A. Bihain: Optimization of upper semidifferentiable functions. *Journal of Optimization Theory and Applications*, 44(4):545–568, 1984.

[10] F.H. Clarke: *Optimization and Nonsmooth Analysis.* John Wiley & Sons, New York, 1983.

[11] V.F. Demyanov and A.M. Rubinov: *Constructive Nonsmooth Analysis (Approximation and Optimization).* Vol. 7, Peter Lang, Frankfurt am Main, Germany, 1995.

[12] V.F. Demyanov, A.M. Bagirov and A.M. Rubinov: A method of truncated codifferential with application to some problems of cluster analysis. *Journal of Global Optimization*, 23(1):63–80, 2002.

[13] A. Ferrer: Representation of a polynomial function as a difference of convex polynomials with an application. *Lectures Notes in Economics and Mathematical Systems*, 502:189–207, 2001.

[14] A. Fuduli, M. Gaudioso and G. Giallombardo: A DC piecewise affine model and a bundling technique in nonconvex nonsmooth minimization. *Optimization Methods and Software*, Vol. 19(1):89–102, 2004.

[15] A. Fuduli, M. Gaudioso and G. Giallombardo: Minimizing nonconvex nonsmooth functions via cutting planes and proximity control. *SIAM Journal on Optimization*, 14(3):743–756, 2004.

[16] A. Fuduli, M. Gaudioso and E.A. Nurminski: A splitting bundle approach for non-smooth non-convex minimization. *Optimization*, 2013, 1–21. doi: 10.1080/02331934.2013.840625

[17] P. Hartman: On functions representable as a difference of convex functions. *Pacific Journal of Mathematics*, 9(3):707–713, 1959.

[18] J.B. Hiriart-Urruty: Generalized differentiability, duality and optimization for problems dealing with differences of convex functions. *Lecture Note in Economics and Mathematical Systems*, 256:37–70, 1985.

[19] J.B. Hiriart-Urruty: *From convex optimization to nonconvex optimization. Part I: Necessary and sufficient conditions for global optimality.* Nonsmooth Optimization and Related Topics, Ettore Majorana International Sciences Series, Volume 43, Plenum Press, 1988.

[20] K. Holmberg and H. Tuy: A production-transportation problem with stochastic demand and concave production costs. *Mathematical Programming*, 85(1):157–179, 1999.

[21] R. Horst and N.V. Thoai: DC programming: Overview. *Journal of Optimization Theory and Applications*, 103(1):1–43, 1999.

[22] R. Horst and H. Tuy: *Global Optimization: Deterministic Approaches.* Springer-Verlag, Heilderberg, first edition, 1990.

[23] N. Karmitsa, A. Bagirov and M.M. Mäkelä: Comparing different nonsmooth minimization methods and software. *Optimization Methods and Software*, 27(1):131–153, 2012.

[24] K.C. Kiwiel: An aggregate subgradient method for nonsmooth convex minimization. *Mathematical Programming*, 27(3):320–341, 1983.

[25] K.C. Kiwiel: *Methods of Descent for Nondifferentiable Optimization.* Lecture Notes in Mathematics, Vol. 1133, Springer-Verlag, Berlin, 1985.

[26] K.C. Kiwiel: Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, 46:105–122, 1990.

[27] L. Lukšan: Dual method for solving a special problem of quadratic programming as a subproblem at linearly constrained nonlinear minimax approximation. *Kybernetika*, 20(6):445–457, 1984.

[28] L. Lukšan and E. Spedicato: Variable metric methods for unconstrained optimization and nonlinear least squares. *Journal of Computational and Applied Mathematics*, 124:61–95, 2000.

[29] M.M. Mäkelä: Survey of bundle methods for nonsmooth optimization. *Optimization Methods and Software*, 17(1):1–29, 2002.

[30] M.M. Mäkelä: Multiobjective proximal bundle method for nonconvex nonsmooth optimization: Fortran subroutine MPBNGC 2.0. *Reports of the Department of Mathematical Information Technology*, Series B. Scientific Computing, No. B 13/2003 University of Jyväskylä, Jyväskylä, 2003.

[31] M.M. Mäkelä and P. Neittaanmäki: *Nonsmooth Optimization.* World Scientific Publishing Co. Inc., River Edge, NJ, 1992.

[32] C. Pey-Chun, P. Hansen, B. Jaumard and H. Tuy: Solution of the multisource Weber and conditional Weber problems by d.c. programming. *Operations Research*, 46(4):548–562, 1998.

[33] R.T. Rockafellar: *Convex Analysis.* Princeton University Press, Princeton, New Jersey, 1970.

[34] H. Schramm and J. Zowe: A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results. *SIAM Journal on Optimization*, 2(1):121–152, 1992.

[35] W.Y. Sun, R.J.B. Sampaio and M.A.B. Candido: Proximal point algorithm for minimization of DC functions. *Journal of Computational Mathematics*, 21(4):451–462, 2003.

[36] P.D. Tao and L.T.H. An: Convex analysis approach to DC programming: Theory, algorithms and applications. *Acta Mathematica Vietnamica,* 22(1):289–355, 1997.

[37] J.F. Toland: Duality in nonconvex optimization. *Journal of Mathematical Analysis and Applications*, 66(2):399–415, 1978.

[38] J.F. Toland: On subdifferential calculus and duality in nonconvex optimization. *Bulletin de la Société Mathématique de France*, Mémoire, 60:173–180, 1979.

[39] H. Tuy: *Convex Analysis and Global Optimization.* Kluwer Academic Publishers, Dordrescht, first edition, 1998.

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 A, 20520 TURKU, Finland | www.tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics

Turku School of Economics
- Institute of Information Systems Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

ISBN 978-952-12-3177-3
ISSN 1239-1891