Kanzow, Christian; Neder, Tanja

A bundle-type method for nonsmooth DC programs

Journal of Global Optimization

Journal of Global Optimization (2024) 88:285–326
https://doi.org/10.1007/s10898-023-01325-5

A bundle-type method for nonsmooth DC programs

Christian Kanzow[1] · **Tanja Neder**[1]

Received: 16 February 2022 / Accepted: 5 September 2023 / Published online: 25 September 2023

Abstract
A bundle method for minimizing the difference of convex (DC) and possibly nonsmooth functions is developed. The method may be viewed as an inexact version of the DC algorithm, where each subproblem is solved only approximately by a bundle method. We always terminate the bundle method after the first serious step. This yields a descent direction for the original objective function, and it is shown that a stepsize of at least one is accepted in this way. Using a line search, even larger stepsizes are possible. The overall method is shown to be globally convergent to critical points of DC programs. The new algorithm is tested and compared to some other solution methods on several examples and realistic applications.

Keywords DC optimization · Bundle method · Global convergence · Critical points

1 Introduction

The problem under consideration, called a *DC program*, is the minimization problem

$$\min_{x \in \mathbb{R}^n} \; f(x) := g(x) - h(x), \tag{1.1}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is a difference of two convex and possibly nonsmooth functions $g, h : \mathbb{R}^n \to \mathbb{R}$. Therefore, f is referred to as a *DC function*, whereas g and h are the corresponding *DC components* of f (these DC components are, of course, not unique).

A versatile consideration of DC programs, both in terms of theory and implementation, can be found in [8]. These DC programs occur frequently in a variety of applications. Among them are the detection of edges in digital images [17], techniques utilized in data mining [2], like the minimum sum-of-squares clustering [25] or the multidimensional scaling problem [19], and the modeling of biochemical reaction networks [1], to name only a few.

The cornerstones for numerically tackling DC programs have already been placed some time ago (see [20] for an extensive treatment of the history of DC programs). The classical

✉ Christian Kanzow
kanzow@mathematik.uni-wuerzburg.de

Tanja Neder
tanja.neder@mathematik.uni-wuerzburg.de

[1] Institute of Mathematics, University of Würzburg, Emil-Fischer-Straße 30, 97074 Würzburg, Germany

approach for solving (1.1) is the DC Algorithm (DCA) (see e.g. [21]) which can be applied to DC programs with both DC components being nonsmooth. Provided that the first DC component g is continuously differentiable, the convergence of DCA can often be accelerated by applying a boosted version of the classical DC Algorithm, the so called Boosted DCA (BDCA). The crucial point thereby is that the iterates computed by DCA can be used to derive descent directions for the objective function, which allows to add a line search to the algorithm (see [2]).

The idea of using bundle methods to solve DC programs is also not new. In [16], gathering the subgradient information for each of the DC components in two separate bundles, leads to a nonconvex cutting plane model of the objective function which incorporates both the convex and the concave behavior of the DC function. The resulting proximal bundle method (PBDC) keeps in the bundles only information related to points close to the current iterate. Another algorithm presented in [10] is also based on a cutting plane approach. But this time just the bundle with respect to the first DC component is restricted to local information, whereas the one with respect to the second DC component keeps information related to distant points. Once again, a nonconvex DC piecewise-affine model is derived, which gives rise to the name DC Piecewise-Concave algorithm (DCPCA) of the resulting method. Yet another bundle method was derived in [15] as an improved variation of the previously mentioned PBDC. The major novelty of this proximal Double Bundle method for DC problems (DBDC) is the procedure to escape from critical points which are not approximate Clarke stationary. Within this procedure it is assured that the difference of subgradients of the DC components lies in the Clarke subdifferential of the objective function itself, which is usually not the case. In that way convergence to an approximate Clarke stationary point is achieved under the assumption that the subdifferentials of the DC components are polytopes. The notion of a Clarke stationary point is, in general, stronger than the frequently used notion of a critical point.

Moreover, bundle methods for the minimization of DC functions subject to some constraints have also been developed during the last few years. However, these algorithms usually are restricted to a certain structure of the constraints (see e.g. [7, 28]).

In contrast to these bundle methods, our approach utilizes the standard subproblem from the classical DC algorithm. This results in a convex subproblem which is then solved inexactly by a simple bundle method. We terminate this bundle method after its first serious step, meaning that we do not require to solve these subproblems exactly or almost exactly, not even close to a solution. Nevertheless, the resulting inexact solution is shown to yield a descent direction. Hence, a line search can be applied to globalize the overall method. This line search is shown to accept at least the full step, that is a stepsize of at least one, and it even allows to take larger stepsizes. However, it is known from the boosted DCA that it may not share the improved descent property of the boosted DCA in case both DC components are nonsmooth. On the other hand, the new bundle-type DC method is shown to have nice global convergence properties when both DC components g and h are nondifferentiable, whereas the boosted DCA requires g to be smooth.

The paper is organized as follows. In Sect. 2 we first recall some basic concepts and definitions as well as the elementary bundle method which forms the basis of the new algorithm. We then present the new bundle-type DC algorithm in Sect. 3, together with a convergence theory and an additional discussion of some descent properties. The results of an extensive numerical testing are provided in Sect. 4. We close with some final remarks in Sect. 5.

2 Preliminaries

This section first recalls some basic definitions and results from nonsmooth and convex analysis (see e.g. [13, 27]). We then provide some details concerning a basic bundle method like it is outlined for example in [11, 18, 22].

2.1 Tools from nonsmooth and convex analysis

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

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \ \forall \lambda \in (0,1).$$

It is called *uniformly convex* with modulus $\mu > 0$ if $f - \frac{\mu}{2}\|\cdot\|^2$ is convex, where $\|\cdot\|$ denotes the Euclidean norm on $\mathbb{R}^n$. Recall that uniformly convex functions always attain a unique global minimum. The (one-sided) *directional derivative* of a function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $\boldsymbol{x} \in \mathbb{R}^n$ in a direction $\boldsymbol{d} \in \mathbb{R}^n$ is defined as

$$f'(\boldsymbol{x}; \boldsymbol{d}) := \lim_{t \downarrow 0} \frac{f(\boldsymbol{x} + t\boldsymbol{d}) - f(\boldsymbol{x})}{t}$$

provided that the limit on the right-hand side exists. The latter holds, in particular, for convex functions f.

Given a convex function $f : \mathbb{R}^n \to \mathbb{R}$ and a parameter $\epsilon \geq 0$, the *ϵ-subdifferential* at a point $\boldsymbol{x} \in \mathbb{R}^n$ is the set

$$\partial_\epsilon f(\boldsymbol{x}) := \left\{ \boldsymbol{s} \in \mathbb{R}^n \mid f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{s}^T(\boldsymbol{y} - \boldsymbol{x}) - \epsilon \quad \forall \boldsymbol{y} \in \mathbb{R}^n \right\}.$$

The special case $\partial f(\boldsymbol{x}) := \partial_0 f(\boldsymbol{x})$ is known as the *(convex) subdifferential* of f at $\boldsymbol{x}$ and each element of this set is called a *subgradient* of f at $\boldsymbol{x}$. For arbitrary $\epsilon \geq 0$, the ϵ-subdifferential $\partial_\epsilon f(\boldsymbol{x})$ is a nonempty, convex and compact set for every $\boldsymbol{x} \in \mathbb{R}^n$. The directional derivative of a convex function f can be calculated using its subdifferential via the formula

$$f'(\boldsymbol{x}; \boldsymbol{d}) = \max_{\boldsymbol{s} \in \partial f(\boldsymbol{x})} \boldsymbol{s}^T \boldsymbol{d}. \tag{2.1}$$

For a locally Lipschitz continuous function $f : \mathbb{R}^n \to \mathbb{R}$ (recall that every convex function is locally Lipschitz), we denote the Clarke subdifferential of f at $\boldsymbol{x}$ by $\partial_C f(\boldsymbol{x})$. For a precise definition and some basic properties of the Clarke subdifferential, we refer to [5]. Note that for convex functions f the Clarke subdifferential coincides with the convex one. Furthermore, $\boldsymbol{0} \in \partial_C f(\boldsymbol{x}^*)$ is a necessary optimality condition for some point $\boldsymbol{x}^*$ to be a local minimum of f. Any point meeting this requirement is named a *Clarke stationary point*. Applying this optimality condition to the DC function $f := g - h$ and using some calculus rules of the Clarke subdifferential leads to the stationarity condition

$$\partial g(\boldsymbol{x}^*) \cap \partial h(\boldsymbol{x}^*) \neq \emptyset. \tag{2.2}$$

Each point $\boldsymbol{x}^* \in \mathbb{R}^n$ satisfying (2.2) is called a *critical point* of the DC function f (see also [12] for characterizations of a minimizer of DC functions). Due to the derivation each Clarke stationary point is a critical point. But the opposite needs not to be necessarily true. Hence, criticality is a weaker optimality condition than Clarke stationarity. For a convex function $f : \mathbb{R}^n \to \mathbb{R}$, the optimality condition $\boldsymbol{0} \in \partial f(\boldsymbol{x}^*)$ becomes even sufficient for having a (global) minimum at $\boldsymbol{x}^* \in \mathbb{R}^n$.

Given a function $f : \mathbb{R}^n \to \mathbb{R}$, we say that some vector $\boldsymbol{d} \in \mathbb{R}^n$ is a *descent direction* of f at $\boldsymbol{x}$ if there exists some $t^* > 0$ such that $f(\boldsymbol{x} + t\boldsymbol{d}) < f(\boldsymbol{x})$ for all $t \in (0, t^*]$. Note that in case f is directionally differentiable at $\boldsymbol{x}$ in a direction $\boldsymbol{d}$, the descent property $f'(\boldsymbol{x}; \boldsymbol{d}) < 0$ is a sufficient criterion for $\boldsymbol{d}$ being a descent direction.

Finally, we define the (Euclidean) distance between two sets $A,\ B \subseteq \mathbb{R}^n$ by

$$\operatorname{dist}(A, B) := \inf \left\{ \|\boldsymbol{a} - \boldsymbol{b}\| \;\middle|\; \boldsymbol{a} \in A,\ \boldsymbol{b} \in B \right\}.$$

In particular, the *projection* of a point $\boldsymbol{y} \in \mathbb{R}^n$ onto a nonempty, closed, and convex set $X \subseteq \mathbb{R}^n$ is determined as the (unique) point in X having the least distance towards $\boldsymbol{y}$. We write

$$P_X(\boldsymbol{y}) := \operatorname*{argmin}_{\boldsymbol{x} \in X} \|\boldsymbol{y} - \boldsymbol{x}\|$$

for this projection.

2.2 A bundle method for convex optimization

This section gives a short introduction to a (simple) bundle method for minimizing a convex function, mainly following the references [11, 18, 22]. Note that there exist more involved bundle schemes. However, the aim is to keep the presentation as simple as possible within this section. The bundle method and its convergence theory will later be used to solve the (convex, but nonsmooth) subproblems resulting in our algorithm for solving DC programs.

Therefore, consider the minimization problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) \tag{2.3}$$

for a convex function $f : \mathbb{R}^n \to \mathbb{R}$. Similar to the classical steepest descent method, the first idea is to compute a descent direction $\boldsymbol{d}^k \in \mathbb{R}^n$ of f at the current iterate $\boldsymbol{x}^k \in \mathbb{R}^n$ by solving the subproblem

$$\min_{\boldsymbol{d} \in \mathbb{R}^n} f'(\boldsymbol{x}^k; \boldsymbol{d}) \quad \text{s.t.} \quad \|\boldsymbol{d}\| \leq 1.$$

Using the relation (2.1), it is not difficult to see that

$$\boldsymbol{d}^k = -\frac{\boldsymbol{g}^k}{\|\boldsymbol{g}^k\|} \quad \text{with} \quad \boldsymbol{g}^k := P_{\partial f(\boldsymbol{x}^k)}(\boldsymbol{0})$$

is the unique solution of this subproblem. This suggests to choose $\boldsymbol{d}^k = -\boldsymbol{g}^k$ as a search direction. Although $\boldsymbol{d}^k$ can indeed be verified to be a descent direction of f at $\boldsymbol{x}^k$, simple examples show that the resulting method may not converge to a minimum, and that a successful version should include information of some neighboring subgradients. This idea leads to the search direction

$$\boldsymbol{d}^k = -\boldsymbol{g}^k \quad \text{with} \quad \boldsymbol{g}^k = P_{\partial_\epsilon f(\boldsymbol{x}^k)}(\boldsymbol{0}).$$

Unfortunately, the ϵ-subdifferential is difficult to compute and projections onto this set might not be easy to calculate. The idea is then to replace the ϵ-subdifferential by an inner approximation G_ϵ^k which has a simpler structure (like being polyhedral) and therefore allows the calculation of the projection

$$\boldsymbol{g}^k := P_{G_\epsilon^k}(\boldsymbol{0}) \tag{2.4}$$

with a significantly reduced effort. A suitable approximation G_ϵ^k can be obtained by using previously computed subgradients $s^j \in \partial f(x^j),\ j \in \{0, 1, \ldots, k\}$. More precisely, denoting the respective (nonnegative) *linearization errors* of f by

$$\alpha_j^k := f(x^k) - f(x^j) - (s^j)^T(x^k - x^j) \quad \forall j = 0, 1, \ldots, k, \tag{2.5}$$

the set G_ϵ^k is defined by

$$G_\epsilon^k := \left\{ \sum_{j=0}^k \lambda_j s^j \;\middle|\; \sum_{j=0}^k \lambda_j \alpha_j^k \le \epsilon,\ \sum_{j=0}^k \lambda_j = 1,\ \lambda_j \ge 0\ \forall j = 0, 1, \ldots, k \right\}.$$

One can verify that this set has indeed the property that $G_\epsilon^k \subseteq \partial_\epsilon f(x^k)$ holds, which justifies to call it an *inner approximation*. Since G_ϵ^k is a nonempty, convex, and compact set, the projections (2.4) do exist. Numerically, these projections require the solution of the quadratic program

$$\min\ \frac{1}{2}\left\| \sum_{j=0}^k \lambda_j s^j \right\|^2 \quad \text{s.t.} \quad \sum_{j=0}^k \lambda_j \alpha_j^k \le \epsilon, \quad \sum_{j=0}^k \lambda_j = 1, \quad \lambda_j \ge 0 \quad \forall j = 0, 1, \ldots, k. \tag{2.6}$$

If $\lambda^k := (\lambda_0^k, \lambda_1^k, \ldots, \lambda_k^k)$ denotes a solution of this quadratic program, then the projection $g^k := P_{G_\epsilon^k}(0)$ is given by

$$g^k = \sum_{j=0}^k \lambda_j^k s^j.$$

In practice, the coefficients gathered in λ^k are often computed by considering the closely related quadratic program

$$\min\ \frac{1}{2}\left\| \sum_{j=0}^k \lambda_j s^j \right\|^2 + \sum_{j=0}^k \lambda_j \alpha_j^k \quad \text{s.t.} \quad \sum_{j=0}^k \lambda_j = 1,\ \lambda_j \ge 0 \quad \forall j = 0, 1, \ldots, k. \tag{2.7}$$

Altogether, this (almost) motivates the following algorithm.

Algorithm 2.1 (Bundle method)

(S.0) *Choose* $x^0 \in \mathbb{R}^n$, $s^0 \in \partial f(x^0)$, $m \in (0, 1)$, *set* $k := 0$, $y^0 := x^0$, $\alpha_0^0 := 0$, $J_0 := \{0\}$.

(S.1) *Compute* λ_j^k, $j \in J_k$, *as a solution of the quadratic program*

$$\min\ \frac{1}{2}\left\| \sum_{j \in J_k} \lambda_j s^j \right\|^2 + \sum_{j \in J_k} \lambda_j \alpha_j^k \quad \text{s.t.} \quad \sum_{j \in J_k} \lambda_j = 1,\ \lambda_j \ge 0 \qquad \forall j \in J_k.$$

(S.2) *Set*

$$g^k := \sum_{j \in J_k} \lambda_j^k s^j, \quad \epsilon_k := \sum_{j \in J_k} \lambda_j^k \alpha_j^k, \quad d^k := -g^k, \quad \zeta_k := -\|g^k\|^2 - \epsilon_k.$$

(S.3) *If* $\zeta_k = 0$*: STOP.*

(S.4) *Set* $y^{k+1} = x^k + d^k$, *choose* $s^{k+1} \in \partial f(y^{k+1})$.
If

$$f(x^k + d^k) \leq f(x^k) + m\zeta_k,$$

set ("serious step")

$$t_k := 1, \quad x^{k+1} := x^k + d^k,$$

otherwise set ("null step")

$$t_k := 0, \quad x^{k+1} := x^k.$$

(S.5) *Set*

$$J_k^p := \{j \in J_k \mid \lambda_j^k > 0\}, \quad J_{k+1} := J_k^p \cup \{k+1\},$$
$$\alpha_j^{k+1} := f(x^{k+1}) - f(y^j) - (s^j)^T(x^{k+1} - y^j) \quad \forall j \in J_{k+1}.$$

(S.6) *Set* $k \leftarrow k+1$, *and go to (S.1).*

In order to restrict the number of constraints in (2.7) and to limit the amount of subgradients and linearization errors to be stored, the index set in the algorithm is reduced to a suitable subset $J_k \subseteq \{0, \ldots, k\}$. Moreover, the linearization errors in (S.5) are slightly modified in comparison to (2.5), since the intermediate points y^j, $j \in J_k$, are also taken into account. The underlying principle is very simple: If the search direction d^k provides a sufficient decrease in the function value, one proceeds in this direction (with stepsize $t_k = 1$). Otherwise one sticks with the current iterate, but adds some further information to the bundle in order to get a better search direction during the next iteration. Furthermore, the termination criterion in (S.3) gets motivated by the subsequent observation.

Lemma 2.2 *If* $\zeta_k = 0$ *holds for some* $k \in \mathbb{N}_0$, *then the corresponding iterate* x^k *is already a minimizer of the convex objective function* f.

This last assertion comes from the elementary observation that

$$g^k \in \partial_{\epsilon_k} f(x^k) \quad \forall k \geq 0. \tag{2.8}$$

In case a solution of the convex optimization problem (2.3) exists, one gets the following global convergence result for Algorithm 2.1.

Theorem 2.3 *Assume that the solution set* $\mathcal{S} := \left\{x^* \in \mathbb{R}^n \;\middle|\; f(x^*) = \inf_{x \in \mathbb{R}^n} f(x)\right\}$ *is nonempty. Then every sequence* $\{x^k\}$ *generated by Algorithm 2.1 converges towards a minimizer* $x^* \in \mathcal{S}$ *of the convex objective function* f.

3 A bundle method for DC optimization

This section introduces the new algorithm for DC optimization using the approach of the classical DC Algorithm in combination with the previously presented bundle method. The precise statement together with a convergence theory is given in Sect. 3.1, whereas some additional descent properties are discussed in Sect. 3.2.

3.1 Algorithm and convergence properties

The aim of our approach is to develop an algorithm which, similar to the Boosted DCA (BDCA), computes descent directions of the objective function using the subproblems arising in the classical DC Algorithm (DCA) (see [2, 21]). This allows a line search to determine the subsequent iterate. In contrast to BDCA, however, the new algorithm is applicable to DC functions with both DC components being nonsmooth. To gain a suitable descent direction from the convex subproblems, we allow an inexact solution of these subproblems by the previous bundle technique.

Having a DC function f as defined in (1.1) to be minimized, the classical DCA approach replaces, in each step $l \in \mathbb{N}_0$, the second DC component h by some linear minorization

$$h_l(\boldsymbol{x}) := h(\boldsymbol{x}^l) + (\boldsymbol{s}^l)^T(\boldsymbol{x} - \boldsymbol{x}^l)$$

with some subgradient $\boldsymbol{s}^l \in \partial h(\boldsymbol{x}^l)$. That way, a convex majorization of the objective function f is obtained. Minimizing this model function is then equivalent to minimizing

$$\phi_l(\boldsymbol{x}) := g(\boldsymbol{x}) - (\boldsymbol{s}^l)^T\boldsymbol{x}. \tag{3.1}$$

To guarantee the existence of a minimizer one usually assumes g to be uniformly convex. This can be done without loss of generality by adding a uniformly convex term, for example $\frac{\rho}{2}\|\cdot\|^2$ with $\rho > 0$, to each convex DC component, if necessary. In contrast to the BDCA, the new algorithm does not require the exact minimization of the convex function ϕ_l in order to obtain a descent direction of the objective function. Instead, the bundle method from Sect. 2.2 is applied until a serious step is carried out. It turns out that the search direction of this step is a descent direction of f at the current iterate $\boldsymbol{x}^l$. A subsequent line search is then used to compute the next iterate. The convergence theory shows that this line search always accepts the full step, which means a stepsize of one. Even larger stepsizes are possible.

Algorithm 3.1 (DCBA–DC Bundle Algorithm)

(S.0) *Choose* $\boldsymbol{x}^0 \in \mathbb{R}^n$, $\beta \in (0,1)$, $m \in (0,1)$, $\mu \in (0,m]$, *set* $l := 0$.
(S.1) *Choose* $\boldsymbol{s}^l \in \partial h(\boldsymbol{x}^l)$, *and define* ϕ_l *as in* (3.1).
(S.2) *Apply the bundle method from Algorithm 2.1 to minimize* $\phi_l(\boldsymbol{x})$ *until a serious step is carried out or until it terminates. Retain* $(\boldsymbol{d}^l, \epsilon_l, \zeta_l)$ *from the corresponding quantities of the serious step or the termination step, respectively.*
In case of termination of the bundle method: STOP.
(S.3) *Choose* $\bar{\tau}_l \geq 1$, *compute* $\tau_l = \max\left(\left\{\bar{\tau}_l\beta^j \;\middle|\; j \in \mathbb{N}_0\right\} \cup \{1\}\right)$ *such that*

$$f(\boldsymbol{x}^l + \tau_l\boldsymbol{d}^l) \leq f(\boldsymbol{x}^l) + \mu\tau_l^2\zeta_l.$$

(S.4) *Set* $\boldsymbol{x}^{l+1} := \boldsymbol{x}^l + \tau_l\boldsymbol{d}^l$, $l \leftarrow l+1$, *and go to (S.1).*

To guarantee that (S.2) always terminates, we have to make sure that the function ϕ_l attains a minimum (cf. Theorem 2.3). Recall that this automatically holds if g is uniformly convex. Hence, we state this assumption explicitly in the following, which is implicitly supposed to hold throughout our convergence analysis. We stress once more, however, that this assumption is not at all restrictive since we can always add and subtract a uniformly convex function to the DC decomposition of f.

Assumption 3.2 The DC component g is a uniformly convex function.

Some comments are in order regarding Algorithm 3.1. First note that l denotes the iteration counter for the (outer) DC-type method, whereas we will use the letter k to denote the iterations of the inner (bundle) method. Hence, $J_{k,l}$ denotes the index set that occurs in iteration k of the bundle method, called in iteration l of Algorithm 3.1. The notation $\boldsymbol{d}^{k,l}$ is defined similarly.

Note that the bundle method executes null steps only except possibly in the last iteration. This, in particular, allows a simplified calculation of the linearization errors. Since the iterate $\boldsymbol{x}^l$ does not change in such a situation, only the computation of the linearization error corresponding to the new intermediate point is required, but not the computation for each index $j \in J_{k,l}$. Hence, in the kth sub-iteration with the search direction $\boldsymbol{d}^{k,l}$ and corresponding subgradient $\boldsymbol{v}^{k+1,l} \in \partial\phi_l(\boldsymbol{x}^l + \boldsymbol{d}^{k,l})$, the required linearization error can be obtained by

$$\alpha_l^{k+1} := \phi_l(\boldsymbol{x}^l) - \phi_l(\boldsymbol{x}^l + \boldsymbol{d}^{k,l}) + (\boldsymbol{v}^{k+1,l})^T \boldsymbol{d}^{k,l}. \tag{3.2}$$

Furthermore, a subgradient $\boldsymbol{v}^{k+1,l} \in \partial\phi_l(\boldsymbol{x}^l + \boldsymbol{d}^{k,l})$ can be computed by selecting some element $\boldsymbol{t}^{k+1,l} \in \partial g(\boldsymbol{x}^l + \boldsymbol{d}^{k,l})$ and setting $\boldsymbol{v}^{k+1,l} := \boldsymbol{t}^{k+1,l} - \boldsymbol{s}^l$.

The line search is an Armijo-type one but with two modifications. First, a quadratic stepsize is considered. This idea already occurs in [6] but with a more involved procedure for determining a suitable stepsize. Second, we look for a decrease in the function value of at least $\mu\tau_l^2(-\zeta_l)$. At first glance, one might expect $\|\boldsymbol{d}^l\|^2$ instead of the enlarged value $-\zeta_l = \|\boldsymbol{d}^l\|^2 + \epsilon_l$. This adjustment is motivated by Lemma 3.4 below. In addition, the initial stepsize $\bar{\tau}_l$ can be determined as a self-adaptive trial stepsize like the one suggested in [2]. One only needs to ensure $\bar{\tau}_l \geq 1$.

Before proving a global convergence result for Algorithm 3.1, we begin with some preliminary observations. To this end, we first justify the termination criterion in (S.2).

Lemma 3.3 *Suppose $\zeta_l = 0$ holds for some l. Then the current iterate $\boldsymbol{x}^l$ is a critical point of the objective function f.*

Proof As the iterate does not change during the bundle process, having $\zeta_l = 0$ for some l together with Lemma 2.2 implies that $\boldsymbol{x}^l$ minimizes ϕ_l. Hence, we have $\boldsymbol{0} \in \partial\phi_l(\boldsymbol{x}^l) = \partial g(\boldsymbol{x}^l) - \boldsymbol{s}^l$. On the other hand, $\boldsymbol{s}^l \in \partial h(\boldsymbol{x}^l)$ by our choice. Consequently, $\partial g(\boldsymbol{x}^l) \cap \partial h(\boldsymbol{x}^l) \neq \emptyset$ follows, showing that $\boldsymbol{x}^l$ is indeed a critical point of the DC function f. □

Motivated by Lemma 3.3, we assume, from now on, that $\zeta_l < 0$ holds for all l, which means Algorithm 3.1 does not stop after finitely many iterations. The following result then shows that the Armijo-type line search is always well-defined (together with Assumption 3.2 this implies that the entire Algorithm 3.1 is well-defined), and that the full step satisfies the line search criterion.

Lemma 3.4 *At each iteration l, there exists a stepsize τ_l, $\tau_l = 1$ or $\tau_l = \bar{\tau}_l\beta^j \geq 1$ with some $j \in \mathbb{N}_0$, such that*

$$f(\boldsymbol{x}^l + \tau_l \boldsymbol{d}^l) \leq f(\boldsymbol{x}^l) + \mu\tau_l^2\zeta_l \tag{3.3}$$

holds.

Proof Let l be arbitrarily chosen. Then

$$\begin{aligned} f(\boldsymbol{x}^l + \boldsymbol{d}^l) - f(\boldsymbol{x}^l) &= g(\boldsymbol{x}^l + \boldsymbol{d}^l) - g(\boldsymbol{x}^l) - h(\boldsymbol{x}^l + \boldsymbol{d}^l) + h(\boldsymbol{x}^l) \\ &\leq g(\boldsymbol{x}^l + \boldsymbol{d}^l) - g(\boldsymbol{x}^l) - (\boldsymbol{s}^l)^T \boldsymbol{d}^l \\ &= \phi_l(\boldsymbol{x}^l + \boldsymbol{d}^l) - \phi_l(\boldsymbol{x}^l) \\ &\leq m\zeta_l \leq \mu\zeta_l, \end{aligned}$$

where the first inequality exploits the fact that $s^l \in \partial h(x^l)$, the penultimate inequality comes from the serious step termination of the bundle method, and the last estimate uses $\mu \in (0, m]$ as well as $\zeta_l < 0$ (see the previous discussion). This shows that at least $\tau_l = 1$ has the desired property. Taking into account the construction of the stepsize yields the desired claim. □

Finally, we need the following auxiliary result for proving global convergence of Algorithm 3.1.

Lemma 3.5 *Assume that Algorithm 3.1 generates an infinite sequence $\{x^l\}$. Then the sequence $\{f(x^l)\}$ is monotonically decreasing. If, in addition, there exists a lower bound $f^* \in \mathbb{R}$ such that $f(x^l) \geq f^*$ holds for all l, then the estimate*

$$\sum_{l=0}^{\infty} \left(\|d^l\|^2 + \epsilon_l \right) \leq \frac{f(x^0) - f^*}{\mu}$$

holds. In particular, we then have $d^l \to 0$ and $\epsilon_l \to 0$ for $l \to \infty$.

Proof The monotonicity of the function values follows directly from ζ_l being negative and the line search in (S.3).

To establish the second assertion, note that (S.3) can be written as $-\mu \tau_l^2 \zeta_l \leq f(x^l) - f(x^{l+1})$ for all l. Taking the sum over $l = 0, \ldots, j-1$, we get

$$\mu \sum_{l=0}^{j-1} \tau_l^2 (-\zeta_l) \leq f(x^0) - f(x^j) \leq f(x^0) - f^* \qquad \forall j \in \mathbb{N}$$

by the boundedness assumption. Letting $j \to \infty$ therefore gives

$$\sum_{l=0}^{\infty} \tau_l^2 (-\zeta_l) \leq \frac{f(x^0) - f^*}{\mu}.$$

Using $\tau_l \geq 1$ and inserting the definition of ζ_l gives the desired inequality. □

The following is the main global convergence result for Algorithm 3.1.

Theorem 3.6 *Every accumulation point of a sequence $\{x^l\}$ generated by Algorithm 3.1 is a critical point of the objective function f.*

Proof Let x^* be an accumulation point of the sequence $\{x^l\}$ and $\{x^l\}_L$ be a corresponding subsequence converging to x^*. Since $s^l \in \partial h(x^l)$ for all l, the convergence of $\{x^l\}_L$ implies the boundedness of the sequence $\{s^l\}_L$. Hence, without loss of generality, we may assume that $\{s^l\}_L$ converges to some limit s^*. Due to the closedness property of the convex subdifferential, it follows that $s^* \in \partial h(x^*)$.

By continuity of f, we have $f(x^l) \to_L f(x^*)$. Hence, the monotonicity of the entire sequence $\{f(x^l)\}$ yields convergence of the entire sequence $\{f(x^l)\}$ to $f(x^*)$. The monotonicity also implies $f(x^l) \geq f(x^*)$ for all l. Thus, the previous lemma can be applied to obtain $d^l \to 0$ and $\epsilon_l \to 0$ for $l \to \infty$.

Furthermore, since we have $-d^l \in \partial_{\epsilon_l} \phi_l(x^l)$ in view of (2.8), we get

$$\phi_l(x) \geq \phi_l(x^l) - (d^l)^T (x - x^l) - \epsilon_l \qquad \forall x \in \mathbb{R}^n \ \forall l \in \mathbb{N}_0.$$

Using the definition of ϕ_l, this can be rewritten as

$$g(x) - (s^l)^T x \geq g(x^l) - (s^l)^T x^l - (d^l)^T (x - x^l) - \epsilon_l \qquad \forall x \in \mathbb{R}^n \ \forall l \in \mathbb{N}_0.$$

Taking $l \to_L \infty$ and exploiting the continuity of g therefore yields

$$g(\boldsymbol{x}) - (\boldsymbol{s}^*)^T \boldsymbol{x} \geq g(\boldsymbol{x}^*) - (\boldsymbol{s}^*)^T \boldsymbol{x}^* \qquad \forall \boldsymbol{x} \in \mathbb{R}^n$$

or, equivalently,

$$g(\boldsymbol{x}) \geq g(\boldsymbol{x}^*) + (\boldsymbol{s}^*)^T (\boldsymbol{x} - \boldsymbol{x}^*) \qquad \forall \boldsymbol{x} \in \mathbb{R}^n.$$

Consequently, we have $\boldsymbol{s}^* \in \partial g(\boldsymbol{x}^*)$. Together with $\boldsymbol{s}^* \in \partial h(\boldsymbol{x}^*)$, this shows that $\partial g(\boldsymbol{x}^*) \cap \partial h(\boldsymbol{x}^*) \neq \emptyset$. Hence, $\boldsymbol{x}^*$ is a critical point of the DC function f. □

Recall that we terminate our method(s), for our theoretical considerations, only if $\zeta_l = 0$. Numerically, one should replace this condition in (S.3) of the bundle method 2.1 by a more practical condition like

$$|\zeta_k| < \delta \quad \text{or, equivalently,} \quad \zeta_k > -\delta \tag{3.4}$$

with some given tolerance $\delta > 0$. The following result then shows that Algorithm 3.1 terminates after finitely many iterations in a point which approximately satisfies the condition of being a critical point of the DC function f.

Theorem 3.7 *Assume that f is bounded from below. Then Algorithm 3.1, with the modified termination criterion (3.4), terminates after finitely many iterations in a point $\boldsymbol{x}^L$ satisfying*

$$\operatorname{dist}\left(\partial_{\epsilon_L} g(\boldsymbol{x}^L), \partial h(\boldsymbol{x}^L)\right) < \sqrt{\delta} \quad \text{with} \quad \epsilon_L < \delta. \tag{3.5}$$

Proof First recall that, in each outer iteration l, due to Assumption 3.2, the bundle step (3.2) terminates after a finite number of inner iterations either meeting the termination criterion or detecting a descent direction $\boldsymbol{d}^l$. This is based on the fact that carrying out only null steps within the bundle iteration leads to $\{\zeta_k\}$ tending to zero (see e.g. [11]). Hence, the condition (3.4) eventually holds.

We next show that Algorithm 3.1 terminates after finitely many iterations. Assume, by contradiction, that an infinite sequence $\{\boldsymbol{x}^l\}$ is generated. Then each call of the bundle method ends with a serious step and hence the computation of a descent direction $\boldsymbol{d}^l$. The subsequent line search then yields

$$f(\boldsymbol{x}^{l+1}) \leq f(\boldsymbol{x}^l) + \mu \tau_l^2 \zeta_l \leq f(\boldsymbol{x}^l) + \mu \zeta_l \qquad \forall l \in \mathbb{N}_0,$$

where the final inequality results from $\tau_l \geq 1$ and ζ_l being negative. Summation over $l = 0, \ldots, j-1$ gives

$$f(\boldsymbol{x}^j) - f(\boldsymbol{x}^0) \leq \mu \sum_{l=0}^{j-1} \zeta_l \leq -\mu \delta j \qquad \forall j \in \mathbb{N},$$

since $\zeta_l \leq -\delta$ for all l by assumption (the inexact termination criterion never holds). Letting $j \to \infty$, the right-hand side tends to $-\infty$, whereas the left-hand side is bounded from below by assumption. This contradiction shows that Algorithm 3.1 terminates within a finite number of iterations.

Let $\boldsymbol{x}^L$ denote the point of termination. It remains to show that $\boldsymbol{x}^L$ satisfies the properties from (3.5). To this end, we first note that a simple calculation shows that $\partial_\epsilon \phi_l(\boldsymbol{x}) = \partial_\epsilon g(\boldsymbol{x}) - \boldsymbol{s}^l$ holds for all $l \in \mathbb{N}_0$, $\epsilon \geq 0$, and $\boldsymbol{x} \in \mathbb{R}^n$. Since $-\boldsymbol{d}^l \in \partial_{\epsilon_l} \phi_l(\boldsymbol{x}^l)$ by (2.8), we therefore obtain the existence of an element $\tilde{\boldsymbol{t}}^l \in \partial_{\epsilon_l} g(\boldsymbol{x}^l)$ such that $-\boldsymbol{d}^l = \tilde{\boldsymbol{t}}^l - \boldsymbol{s}^l$. Together with the fact that $\boldsymbol{s}^l \in \partial h(\boldsymbol{x}^l)$, we get

$$\operatorname{dist}\left(\partial_{\epsilon_l} g(\boldsymbol{x}^l), \partial h(\boldsymbol{x}^l)\right) = \inf\left\{\|\boldsymbol{t} - \boldsymbol{s}\| \;\middle|\; \boldsymbol{t} \in \partial_{\epsilon_l} g(\boldsymbol{x}^l),\ \boldsymbol{s} \in \partial h(\boldsymbol{x}^l)\right\}$$

$$\leq \left\|\tilde{\boldsymbol{t}}^l - \boldsymbol{s}^l\right\| = \|\boldsymbol{d}^l\| \leq \sqrt{|\zeta_l|},$$

where the last inequality just exploits the definition of ζ_l. This definition also implies $\epsilon_l \leq |\zeta_l|$. Since, upon termination, we have $|\zeta_L| < \delta$, the two estimates (3.5) follow. □

Note that (3.5) can be seen as an approximation of

$$\operatorname{dist}\left(\partial g(\boldsymbol{x}^*), \partial h(\boldsymbol{x}^*)\right) = 0. \tag{3.6}$$

Due to the closedness of the subdifferential, (3.6) is equivalent to $\partial g(\boldsymbol{x}^*) \cap \partial h(\boldsymbol{x}^*) \neq \emptyset$, which means that $\boldsymbol{x}^*$ is a critical point of the DC function f. We may therefore view the point of termination as an approximate critical point of the objective function.

3.2 Descent properties of search directions

The subject of this section is to discuss some additional properties of the search direction $\boldsymbol{d}^l$. We will see that $\boldsymbol{d}^l$ is indeed a descent direction of the objective function f at the current iterate $\boldsymbol{x}^l$, but, in general, not in the point $\boldsymbol{x}^l + \boldsymbol{d}^l$. Note that this latter property holds for the boosted DCA method from [2] if g is smooth. We also discuss a modified version for determining a suitable stepsize in (S.3).

Note that the convergence theory in Sect. 3.1 is completely independent of any descent properties of $\boldsymbol{d}^l$. In particular, the line search in (S.3) makes no explicit use of this feature (see also Lemma 3.4). Nevertheless, it is interesting to see that $\boldsymbol{d}^l$ is indeed a descent direction of f in $\boldsymbol{x}^l$.

Proposition 3.8 *The directional derivative of the objective function satisfies*

$$f'(\boldsymbol{x}^l; \boldsymbol{d}^l) \leq \phi_l'(\boldsymbol{x}^l; \boldsymbol{d}^l) < 0 \quad \forall l \in \mathbb{N}_0.$$

Hence, $\boldsymbol{d}^l$ is a descent direction of f in $\boldsymbol{x}^l$.

Proof Let l be fixed. We then obtain

$$\begin{aligned} f'(\boldsymbol{x}^l; \boldsymbol{d}^l) &= g'(\boldsymbol{x}^l; \boldsymbol{d}^l) - h'(\boldsymbol{x}^l; \boldsymbol{d}^l) \\ &= g'(\boldsymbol{x}^l; \boldsymbol{d}^l) - \max_{\boldsymbol{s} \in \partial h(\boldsymbol{x}^l)} \boldsymbol{s}^T \boldsymbol{d}^l \\ &\leq g'(\boldsymbol{x}^l; \boldsymbol{d}^l) - (\boldsymbol{s}^l)^T \boldsymbol{d}^l \\ &= \phi_l'(\boldsymbol{x}^l; \boldsymbol{d}^l), \end{aligned}$$

where the relation (2.1) together with $\boldsymbol{s}^l \in \partial h(\boldsymbol{x}^l)$ was exploited. By construction, $\boldsymbol{d}^l$ results from a serious step of the bundle method, hence $\phi_l(\boldsymbol{x}^l + \boldsymbol{d}^l) < \phi_l(\boldsymbol{x}^l)$ holds. A standard characterization of the directional derivative for convex functions therefore yields

$$\phi_l'(\boldsymbol{x}^l; \boldsymbol{d}^l) = \inf_{t>0} \frac{\phi_l(\boldsymbol{x}^l + t\boldsymbol{d}^l) - \phi_l(\boldsymbol{x}^l)}{t} \leq \frac{\phi_l(\boldsymbol{x}^l + \boldsymbol{d}^l) - \phi_l(\boldsymbol{x}^l)}{1} < 0.$$

Putting both estimates together completes the proof. □

Recall that Lemma 3.4 shows that a full step in the direction $\boldsymbol{d}^l$ is always accepted by the line search criterion (S.3). Using a minor modification in the bundle procedure ensures that DCBA even allows to take a stepsize τ_l strictly larger than one. The details are given in the subsequent proposition.

Proposition 3.9 *Assume that we replace the inequality in (S.4) of the Bundle Algorithm 2.1 by a strict one. Then there exists a stepsize $\tau_l > 1$ such that (3.3) holds.*

Proof For arbitrary l, one can follow the proof of Lemma 3.4 to see that the sharp estimate

$$f(\boldsymbol{x}^l + \boldsymbol{d}^l) < f(\boldsymbol{x}^l) + \mu\zeta_l$$

holds. Consequently, (3.3) is satisfied for $\tau_l = 1$, but with a strict inequality. Since the mapping $\tau \mapsto f(\boldsymbol{x}^l + \tau\boldsymbol{d}^l) - f(\boldsymbol{x}^l) - \mu\tau^2\zeta_l$ is continuous, (3.3) holds on an interval $[1, \tau_*)$ for some $\tau_* > 1$ (depending on l). This completes the proof. □

The previous result motivates to search for a suitable stepsize by using a strategy like

$$\tau_l = \max\left\{1 + \bar{\tau}_l\beta^j \;\middle|\; j \in \mathbb{N}_0\right\} \quad \text{such that} \quad f(\boldsymbol{x}^l + \tau_l\boldsymbol{d}^l) \le f(\boldsymbol{x}^l) + \mu\tau_l^2\zeta_l$$

for some $\beta \in (0, 1)$, where $\bar{\tau}_l$ can be determined by the self-adaptive trial stepsize strategy introduced in [2]. In this way, our approach shares some properties of the boosted version of DCA, but for general nonsmooth functions g and h.

At first glance, this modified stepsize seems to be rather promising, and is more likely to yield a stepsize larger than one than the original stepsize rule from (S.3). However, numerical tests indicate that the overall results are often better for the original stepsize rule, at least for most applications considered in this work. The computation of stepsizes larger than one by the modified rule sometimes leads to the situation where the method crosses a valley with a one-dimensional minimizer, ending up in a region of ascent again. Therefore, the progress in the function value of the objective is often less than accepting a stepsize of one. As this behavior accumulates, one keeps jumping from one side of the valley to the other, sometimes even on a straight line. We illustrate this behavior in Fig. 1 where Example 3.10 (see below) is used, with the standard choice of parameters given in Sect. 4.1 and initial point $(x^0, y^0)^T := (2.5, 1)^T$. It is remarkable that the vector provided by the bundle method often seems to be a pretty good choice for updating the iterate without any scaling.

Recall that Lemma 3.4 and Proposition 3.9 show that the full step in the direction $\boldsymbol{d}^l$ is always accepted by our line search rule(s). Similar to [2], one may therefore ask whether $\boldsymbol{d}^l$ is also a direction of descent at the point $\boldsymbol{x}^l + \boldsymbol{d}^l$. Note, however, that this cannot be expected for nonsmooth functions g since it was already shown in [2] that this property does not hold, in general, even if the convex subproblems are solved exactly. It is therefore not surprising to see that this descent property is also violated for our inexact solution $\boldsymbol{d}^l$ of the convex subproblem. This is shown by the following counterexample taken from [2].

Example 3.10 (**Failure of a boosted version of DCBA**) Consider the function

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x, y) := -\frac{5}{2}x + \frac{1}{2}(x^2 + y^2) + |x| + |y|$$

with uniformly convex DC components $g, h : \mathbb{R}^2 \to \mathbb{R}$ chosen as

$$g(x, y) := -\frac{5}{2}x + x^2 + y^2 + |x| + |y|, \quad h(x, y) := \frac{1}{2}(x^2 + y^2).$$

Taking $(x^0, y^0)^T := (0.5, 0.1)^T$ as a starting point, the bundle method applied to $\min_{(x,y)\in\mathbb{R}^2} \phi_0(x, y)$ with $m = 0.1$ stops after two iterations with the detection of the descent direction

$$\boldsymbol{d}^0 = \frac{1}{10,820}(6610, -3061)^T \approx (0.61091, -0.28290)^T.$$

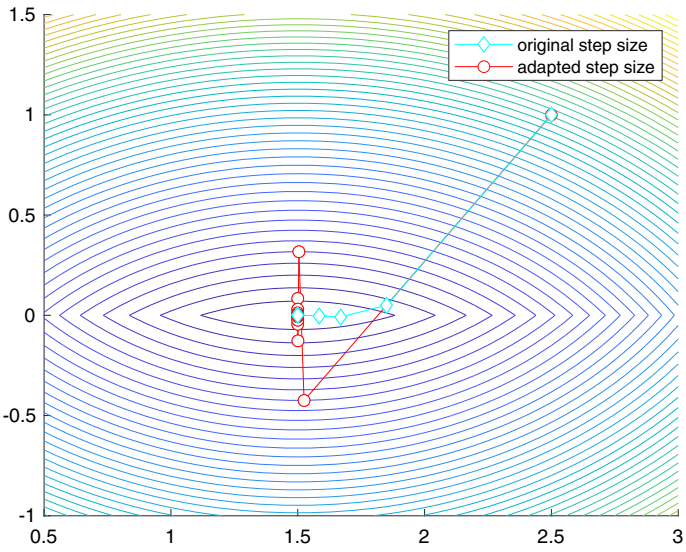


Fig. 1 Variants of stepsize strategies

Let us note, for the sake of completeness, that selecting m in a different way would not open the door for gaining a descent direction in the first iteration, as the first search direction $(d_1^{0,0}, d_2^{0,0})^T$ satisfies $\phi_0(x^0 + d_1^{0,0}, y^0 + d_2^{0,0}) - \phi_0(x^0, y^0) = 2 > 0$.

Before verifying analytically that $\boldsymbol{d}^0$ is indeed not a descent direction of f at the intermediate point

$$(x^0, y^0)^T + \boldsymbol{d}^0 = \frac{1}{10,820}(12,020, -1979)^T \approx (1.11091, -0.18290)^T,$$

let us have a look at Fig. 2, which contains a contour plot of the function f, revealing the basic situation. Starting at $(x^0, y^0)^T = (0.5, 0.1)^T$ and heading in the direction $\boldsymbol{d}^0$, one initially achieves a decrease in the function value. This was expectable as $\boldsymbol{d}^0$ is known to be a descent direction of f at $(x^0, y^0)^T$. But proceeding further (below the line $y = 0$ to be more precise), one leaves the region of descent, entering a region of ascent. Accepting a full step $\tau_0 = 1$, one touches the region of ascent. Therefore, moving further in the direction $\boldsymbol{d}^0$ from $(x^0, y^0)^T + \boldsymbol{d}^0$ would result in a continuing increase of the function value. Consequently, $\boldsymbol{d}^0$ is not a descent direction of f at the point $(x^0, y^0)^T + \boldsymbol{d}^0$. Analytically this claim is confirmed by the corresponding scalar product

$$\nabla f(x^0 + d_1^0, y^0 + d_2^0)^T \boldsymbol{d}^0 \approx 0.09695$$

being positive (note that one can indeed consider the gradient of the objective, as the considered point is located in a region where f is differentiable). In addition, this instance shows that the search direction $\boldsymbol{d}^0$ is not running tangential towards the contour line through $(x^0 + d_1^0, y^0 + d_2^0)^T$, which is marked pink in Fig. 2, although it might seem to be the case at first glance.

Recall that in [2] (making use of the notation therein) it is proven that for continuously differentiable functions g computing the exact solution $\boldsymbol{y}^l$ of the convex subproblem arising in DCA leads to $\bar{\boldsymbol{d}}^l := \boldsymbol{y}^l - \boldsymbol{x}^l$ being a descent direction of f at the optimal point $\boldsymbol{y}^l = \boldsymbol{x}^l + \bar{\boldsymbol{d}}^l$.

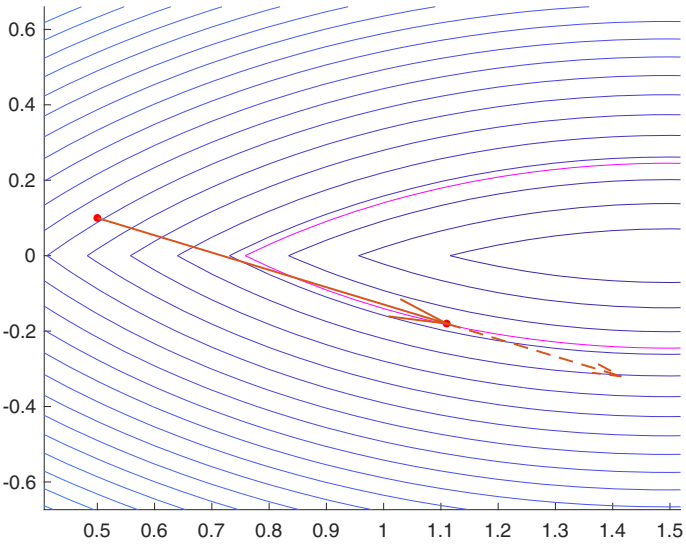


Fig. 2 Change of the function value in the direction $\boldsymbol{d}^0$ starting at $(0.5, 0.1)^T$

The boosted version of the DC Algorithm, BDCA, is based on this observation. But this descent property does not necessarily hold for our approach where $\boldsymbol{d}^l$ is only computed by means of an inexact solution of the convex subproblem. This is illustrated in the following smooth example which results from the previous one by replacing the absolute value function using a scaled and shifted version of Huber's loss (see [14]).

Example 3.11 (**Failure of a boosted version of DCBA in case of smooth DC components**) Consider the smoothed function

$$\tilde{f} : \mathbb{R}^2 \to \mathbb{R}, \quad \tilde{f}(x, y) := -\frac{5}{2}x + \frac{1}{2}(x^2 + y^2) + \psi_\epsilon(x) + \psi_\epsilon(y)$$

with

$$\psi_\epsilon : \mathbb{R} \to \mathbb{R}, \quad \psi_\epsilon(x) := \begin{cases} |x| & \text{if } |x| \geq \epsilon, \\ \frac{1}{2\epsilon}x^2 + \frac{\epsilon}{2} & \text{if } |x| < \epsilon \end{cases}$$

being the described smooth adaption of the absolute value function for sufficiently small $\epsilon > 0$, for example $\epsilon = 10^{-3}$. Similarly to the previous example the uniformly convex DC components $\tilde{g}, \tilde{h} : \mathbb{R}^2 \to \mathbb{R}$ are chosen as

$$\tilde{g}(x, y) := -\frac{5}{2}x + x^2 + y^2 + \psi_\epsilon(x) + \psi_\epsilon(y), \quad \tilde{h}(x, y) := \frac{1}{2}(x^2 + y^2).$$

Note that the smooth modification $\tilde{f}$ coincides with the function f from the previous example on the set $Q_\epsilon := \left\{(x, y) \in \mathbb{R}^2 \,\middle|\, |x| \geq \epsilon,\ |y| \geq \epsilon\right\}$. Therefore, taking $\epsilon > 0$ sufficiently small, and starting again in $(x^0, y^0)^T = (0.5, 0.1)^T$, the approximation function $\tilde{\phi}_0$ matches the corresponding one from Example 3.10 at least on the relevant part of the domain of definition, namely Q_ϵ. Thus, all calculations from Example 3.10 remain valid, as we stay inside Q_ϵ during the whole computation. Consequently, also in case of a smooth first DC component the accepted search direction turns out not to be a descent direction of the smooth objective $\tilde{f}$ at the intermediate point $(x^0 + d_1^0, y^0 + d_2^0)^T$.

The previous example, of course, shows that our approach has a drawback compared to the boosted DCA. Recall, however, that our primary aim was to develop a method for solving DC programs where both DC components g and h are nonsmooth. In this situation, the descent property of $\boldsymbol{d}^l$ cannot be expected at the point $\boldsymbol{x}^l + \boldsymbol{d}^l$ even for the exact solution of the convex subproblem.

4 Numerical experiments and applications

This section presents some numerical experiments of the new algorithm DCBA and gives a comparison with some existing solvers for DC programs. In particular, our method is compared with the solvers DCA [21], BDCA [2], PBDC [16], and DCPCA [10] which are briefly reviewed in Sect. 4.1, together with some details of our implementations. The numerical experiments are then carried out using a broad class of academic test problems [2, 16] as well as some examples arising from applications, namely minimum sum-of-squares clustering [2, 25], multidimensional scaling [2, 19], and edge detection by means of a clustering technique [17].

4.1 Methods and implementation

This section first provides some details of the DC solvers that are used in our numerical studies. The standard method for solving DC problems is the DCA [21, 26], which can be accelerated to a boosted version, namely the BDCA [2], in suitable cases. In addition, we use two bundle methods, PBDC [16] and DCPCA [10]. A brief overview of these algorithms is given in Table 1.

As already noted, DCA derives, in each iteration, a convex majorization of the objective function by approximating the second DC component by some linear minorization. The minimizer of this model function yields the next iterate, which means DCA solves the subproblems exactly and uses no line search globalization (see [21]).

BDCA is introduced in [2]. It is an accelerated version of DCA being motivated by the observation that, in case of a smooth first DC component, the solution of the convex subproblem occurring in DCA gives rise to a descent direction at the point that is accepted by DCA as the next iterate. The latter detection allows to add a line search right after solving the very same convex subproblem as in DCA. This often speeds up the convergence.

PBDC is described in [16]. This bundle method constructs two separate cutting plane models, one for each DC component. Combining both leads to a piecewise linear, nonconvex model of the objective function which incorporates the convex behavior of the DC function as well as its concave one. The computation of the search direction uses a stabilizing term which includes a proximity parameter. Thus, a line search is superfluous. The termination criterion directly refers to the definition of a critical point of the DC function (see (2.2)) and estimates the distance between the respective two subdifferentials.

DCPCA originates from [10]. Similar to PBDC, it develops two separate cutting plane models, one for each DC component. This initially leads to a nonconvex piecewise linear approximation. The two DC components, however, are not treated equally. The bundle related to the first DC component is restricted to local information only, whereas the bundle concerning the second DC component is not. The resulting model, a pointwise maximum of concave functions, is then approximated by a local quadratic program which, in turn, is used to compute a (candidate) search direction. In case no satisfactory solution can be found,

Table 1 Summary of the methods used in our numerical experiments

Abbreviation	Denomination	References
DCA	Difference of Convex Algorithm	[21]
BDCA	Boosted DCA	[2]
DCBA	DC Bundle Algorithm	Algorithm 3.1
PBDC	Proximal Bundle method for DC optimization	[16]
DCPCA	DC Piecewise-Concave Algorithm	[10]

the method switches to an auxiliary (also quadratic) program. Having found an appropriate search direction, a line search follows.

In order to achieve a better comparability of the numerical results, the termination criteria of the different algorithms are adapted to some extent. While BDCA stops whenever the computed descent direction $\boldsymbol{d}^l$ is (close to) zero, for DCBA the sum of the squared norm of the search direction and the ϵ-tolerance of the current approximation of the subdifferential is checked to be (close to) zero. This motivates to split the termination criterion for DCBA into two parts, namely

$$\|\boldsymbol{d}^l\| < \bar{\varepsilon}_1 \quad \text{and} \quad \epsilon_l < \bar{\varepsilon}_2$$

with some given tolerances $\bar{\varepsilon}_1,\ \bar{\varepsilon}_2 > 0$. Note that suppressing the square of the norm in the first condition is not essential. Accordingly, the termination criterion for BDCA is inherited as $\|\boldsymbol{d}^l\| < \bar{\varepsilon}_1$. As the stopping condition for DCA and BDCA coincides, it is clear how to choose the respective one for DCA.

Comparing the convergence theorems for PBDC (see Theorem 6 in [16]) with the corresponding Theorem 3.7 for DCBA, one realizes an immediate similarity. The first result ensures that at the point $\boldsymbol{x}^L$ of termination the approximate criticality condition $\operatorname{dist}\left(\partial_{\bar{\varepsilon}_2} g(\boldsymbol{x}^L), \partial_{\bar{\varepsilon}_2} h(\boldsymbol{x}^L)\right) \leq \bar{\varepsilon}_1$ is satisfied whenever the termination criterion

$$\left\| \boldsymbol{\xi}_1^* - \boldsymbol{\xi}_2^* \right\| < \bar{\varepsilon}_1$$

is met (the notation is taken from [16]). The second one proves the final estimate $\operatorname{dist}\left(\partial_{\epsilon_L} g(\boldsymbol{x}^L), \partial h(\boldsymbol{x}^L)\right) < \bar{\varepsilon}_1$ with $\epsilon_L < \bar{\varepsilon}_2$. This suggests to stuck with the termination criterion for PBDC as it is stated in the cited work. In addition, it indicates in which range the second termination tolerance $\bar{\varepsilon}_2$ should be taken.

Having in mind Remark 2 of [10], one gets a direct connection of the termination criterion for DCPCA to the one for DCBA, which gives rise to adapt the stopping condition of DCPCA in the same manner as the one for DCBA towards

$$\|\bar{\boldsymbol{d}}\| < \bar{\varepsilon}_1 \quad \text{and} \quad -\|\bar{\boldsymbol{d}}\|^2 - \bar{v} = \sum_{i \in I} \bar{\lambda}_i \alpha_i^{(1)} < \bar{\varepsilon}_2$$

(the notation is taken from [10]). Note that both, $\alpha_i^{(1)}$ in terms of DCPCA and α_l^{k+1} in terms of DCBA, denote linearization errors corresponding to the first DC component. Indeed the latter one was defined as a linearization error of the approximation ϕ_l in (3.2), but, as this function differs from the first DC component g only by a linear function, α_l^{k+1} in fact yields the linearization error with respect to g. This last modification ensures that for DCPCA in the point of termination $\boldsymbol{x}^L$, the estimate $\operatorname{dist}\left(\partial_{\bar{\varepsilon}_2} g(\boldsymbol{x}^L), h(\boldsymbol{x}^L)\right) \leq \bar{\varepsilon}_1$ holds, similar to the previous ones for PBDC and DCBA.

Note that even though choosing $\bar{\varepsilon}_1 \ll \bar{\varepsilon}_2$ in the presented examples, in case of DCBA as well as DCPCA, the second termination quantity related to the linearization errors undershoots even the lower critical tolerance $\bar{\varepsilon}_1$ in the point of termination with only two exceptions. The first one is the reproduction of the Bavarian map by means of multidimensional scaling in Sect. 4.4. For this special instance, DCBA ended with a precision of 0.089 and DCPCA with 0.062, while having a comparatively large $\bar{\varepsilon}_1 = 10^{-2}$. The second exception is Problem 7 of the academic test collection in Sect. 4.6. Here DCBA terminates with an accuracy of 0.009 for the (combined) linearization errors and DCPCA with 0.012 while having $\bar{\varepsilon}_1 = 10^{-3}$.

To be able to consider the running time for comparison at least in most applications all algorithms are implemented from scratch using the same computer language. The codes used in Sects. 4.2–4.6 are implemented in Matlab version 2022b and executed on a 8xIntel®Core™i7-7700 CPU @ 3.60 GHz computer with 31.1 GiB RAM under an open SUSE Leap 15.4 (64-bit) system. The only exceptions are the test runs of DCBA for the multidimensional scaling problem in Sect. 4.4 which are run in GNU Octave 5.1.0 on a Radeon Vega Mobile Gfx 2.00 GHz computer with AMD Ryzen 5 2500U CPU and 8.00 GB RAM under Windows 10 (64-bit). The simple reason for this exception is that the quadratic programming solver of Matlab claims for quite some instances to converge to the solution although it does verifiably not.

In all numerical experiments, the initial stepsize of every line search procedure contained in BDCA as well as DCBA gets computed by means of a self-adaptive trial stepsize strategy introduced in [2]. For the remaining part of the line search within DCBA a standard backtracking approach as described in [4] is used. With our choice of parameters γ, the enlargement factor within the self-adaptive trial stepsize, and β, the reduction factor within the actual line search, we ensure that, unless the line search does not terminate beforehand, at least $\tau = 1$ gets tested and hence approved.

Furthermore, the proximity parameter t for PBDC is chosen as $t = 0.8\,(t_{min} + t_{max})$, as suggested in [16], though it is not consistent with the request $t \in [t_{min}, t_{max}]$. But numerical experiments confirm that this choice is to be preferred. In addition, we adopt the proposed modifications from [16] while implementing PBDC.[1] On the one hand, we add a subgradient aggregation technique for the first bundle which is based on [18]. On the other hand, we restrict the size of the first bundle to $\min\{n + 5, 1000\}$, where n is the number of variables.

Moreover, the quadratic programs occurring in the bundle methods DCBA, PBDC and DCPCA are solved using the `quadprog`[2] command. Thereby, in case of the academic testproblems we switch from the default option `interior-point-convex` for the algorithm to `active-set` in case of the dimension n being at most 10 as the latter method performs better for problems with a small number of variables. The solution methods applied to the convex subproblems of DCA differ depending on the smoothness property of the first DC component. Whenever it is nonsmooth the bundle method from Algorithm 2.1 following [11] gets executed. Whenever the first DC component is smooth and hence, BDCA can be applied as well, we use a Limited-Memory BFGS method (see [23, 24]).

For most of the numerical tests, a similar parameter setting is used. Unless said otherwise, the termination tolerances are set to $\bar{\varepsilon}_1 = 10^{-3}$ and $\bar{\varepsilon}_2 = 10^{-1}$. The remaining parameters are chosen as follows, referring once again to the notation in the respective papers from Table 1: In terms of BDCA we take $\alpha = 0.1,\ \beta = 0.5, \gamma = 4$, and $\bar{\lambda}_1 = 4$. The last two parameters

[1] With the named modifications we follow the publicly available Fortran source of PBDC which can be found on http://napsu.karmitsa.fi/pbdc/.

[2] See https://de.mathworks.com/help/optim/ug/quadprog.html?searchHighlight=quadprog&s_tid=srchtitle_quadprog_1 for the Matlab and https://octave.sourceforge.io/optim/function/quadprog.html as well as https://github.com/AsmaAfzal/octave_workspace/blob/master/quadprog.m for the implementation in Octave.

Table 2 Absolute frequency of sequences converging to the respective critical point by the different DC algorithms

	$(-1,-1)$	$(-1,0)$	$(0,-1)$	$(0,0)$
DCA	2438	2573	2438	2551
BDCA	10,000	0	0	0
DCBA	10,000	0	0	0
PBDC	9994	2	3	1
DCPCA	9774	116	110	0

concerning the self-adaptive trial stepsize strategy are also used for DCBA. In addition, the remaining parameters for this algorithm are set to $\beta = 0.5$, $m = 0.5$ and $\mu = 0.1$. The missing parameters for PBDC are chosen as $m = 0.5$, $R = 10^7$, $L_1 = L_2 = 1000$ as well as the maximum size of the second bundle as 3 and $r = 0.75$ whenever the spatial dimension n is less than 10, $r = 0.99$ in case of $n \geq 300$, and $r = \lfloor \frac{n}{n+5} \cdot 100 \rfloor / 100$ else. This last bunch of selections corresponds to the default values from [16]. Moreover, for DCPCA the still missing parameters are also taken as the default values from [10], namely $\eta = 0.7$, $m = 0.5$, $\sigma = 0.5$ and $\rho = 0.95$.

4.2 An academic test problem

The essential aim of examining the subsequent academic test problem is to investigate how often the algorithms under consideration converge to the known global minimum of the objective function and not just to a critical point. This test setting is inspired by Example 3.1 in [2].

For the objective function $f : \mathbb{R}^2 \to \mathbb{R}$ with

$$f(x, y) := x^2 + y^2 + x + y - |x| - |y| \qquad x, y \in \mathbb{R},$$

the DC composition $f := g - h$ to be examined is chosen as

$$g(x, y) := \frac{3}{2}(x^2 + y^2) + x + y, \qquad h(x, y) := |x| + |y| + \frac{1}{2}(x^2 + y^2) \qquad x, y \in \mathbb{R},$$

so that the DC components g, h are uniformly convex. The global minimum is at $(-1, -1)$. But there exist three additional critical and non-optimal points $(-1, 0)$, $(0, -1)$ and $(0, 0)$. In fact, these three points are not even local minimizers.

To investigate the ability of the different solvers to find the optimal point, 10,000 test runs for minimizing f are considered. Thereby, all five algorithms start at the same initial points which are chosen quasi-randomly from the rectangle $[-1.5, 1.5]^2$. In the end, the sequences converging to each of the critical points are counted.

The result is shown in Table 2. BDCA and DCBA are both able to find the minimizer in every single instance. After all, PBDC succeeds in 99.9% of the test cases and DCPCA in 97.7%. However, DCA converges to each of the four critical points more or less the same number of times and determines the optimum in only 24.4% of instances.

4.3 The minimum sum-of-squares clustering problem

We first provide a short introduction of the minimum sum-of-squares clustering problem. Then we present two test settings that are examined afterwards, one related to randomly

generated data and another one referring to real data in the form of geographic coordinates of Bavarian cities.

Clustering describes the separation of a data set into disjoint subsets, so called clusters, by gathering points of similarity. The method is used in data mining for the analysis of huge data sets to get a better (or condensed) overview of the information actually contained in the given data. Thereby, the measure of similarity may differ depending on the application. In the following, each cluster gets characterized by its centroid and the classification gets done by considering the (minimal) squared Euclidean distance of each data point towards these centroids. Hence, denoting with $A := \{\boldsymbol{a}^1, \ldots, \boldsymbol{a}^k\}$ the set of points $\boldsymbol{a}^i \in \mathbb{R}^n,\ i = 1, \ldots, k$, to be partitioned, and letting p be the desired number of clusters, the aim is to determine p centroids $\boldsymbol{x}^j \in \mathbb{R}^n,\ j = 1, \ldots, p$, such that the (averaged) sum over the squared distance of each data point towards the corresponding centroid gets minimal. Thus, using the notation $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^p) \in \mathbb{R}^{n \times p}$, the problem under consideration is

$$\min_{\boldsymbol{X} \in \mathbb{R}^{n \times p}} f(\boldsymbol{X}) := \frac{1}{k} \sum_{i=1}^{k} \min_{j=1,\ldots,p} \left\| \boldsymbol{a}^i - \boldsymbol{x}^j \right\|^2.$$

In [25], a DC composition of f is derived. Adding a quadratic term to each DC component $g, h : \mathbb{R}^{n \times p} \to \mathbb{R}$ like suggested in [2], one obtains

$$g(\boldsymbol{X}) := \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{p} \left\| \boldsymbol{a}^i - \boldsymbol{x}^j \right\|^2 + \frac{\rho}{2} \|\boldsymbol{X}\|^2,$$

$$h(\boldsymbol{X}) := \frac{1}{k} \sum_{i=1}^{k} \max_{j=1,\ldots,p} \sum_{\substack{t=1,\ldots,p \\ t \neq j}} \left\| \boldsymbol{a}^i - \boldsymbol{x}^t \right\|^2 + \frac{\rho}{2} \|\boldsymbol{X}\|^2.$$

Thereby, a modulus $\rho > 0$ ensures uniform convexity and $\|\boldsymbol{X}\|$ denotes the Frobenius norm of the matrix $\boldsymbol{X}$. The following tests are carried out using $\rho = 0.1$.

In the first numerical experiment for a varying combination of parameters, k quasi-randomly generated vectors in $\mathbb{R}^n$ with normally distributed entries having a mean of zero and a standard deviation of ten get to be clustered in p groups. For that matter, $k \in \{500, 750, 1000, 2500, 5000\}$, $n \in \{2, 5, 10, 15\}$, and $p \in \{5, 10, 20, 50, 75\}$ are considered. For each triple of parameters, ten test runs with all algorithms listed at the beginning of this section are passed. We start with p centroid candidates in $\mathbb{R}^n$ which get generated according to the same rules as the points to be clustered. During the ten test runs the set of points to be partitioned remains identical, whereas the initial centroid candidates vary with each run. However, all algorithms start from the same initial situation.

In the process, methods are rated successful whenever they reach the smallest function value in comparison. We call such a value an optimum although it may not connect to a global solution of the underlying optimization problem. To cope with the nature of numerics, we further call a method successful whenever it yields a function value deviating from the declared optimal function value less than 0.1 in case of $n = 2$ and 1 in case of $n \in \{5, 10, 15\}$. Throughout this experiment, the number of iterations, the running time as well as the number of evaluations of the DC components and their (sub-)gradients are reported and evaluated by means of the performance profiles introduced in [9]. Thereby, for the bundle methods an iteration gets identified with a change of the iterate, which means null steps are not counted as such. Though the iterations of the respective methods, of course, are not directly comparable in view of computational effort, the related information is added to the resulting diagrams

shown in Fig. 3. Note that DCA does not require evaluation of the second DC component h and thus does not appear in the corresponding profile.

While DCA, DCPCA and PBDC are able to solve the problem with nearly the same probability, BDCA and DCBA are both slightly ahead. DCBA even has a minor lead over BDCA. But in terms of efficiency, that is requiring the least running time and number of diverse evaluations, BDCA clearly outperforms all other algorithms. This might not be surprising as BDCA is specifically designed for DC problems with smooth first DC component. Comparing the bundle methods, PBDC and DCBA show on average similar profiles, whereas DCPCA drops behind. In particular, DCPCA usually has the longest running time.

With regard to DCBA additional analysis reveals that during all test runs the part of the number of function evaluations of the first DC component g in context with the bundle procedure remains pretty stable around 66%. The remaining 34% of the function evaluations of g are allotted to the line search. Note that the latter percentage gives also the ratio of the number of function evaluations of the second DC component h and the first DC component g.

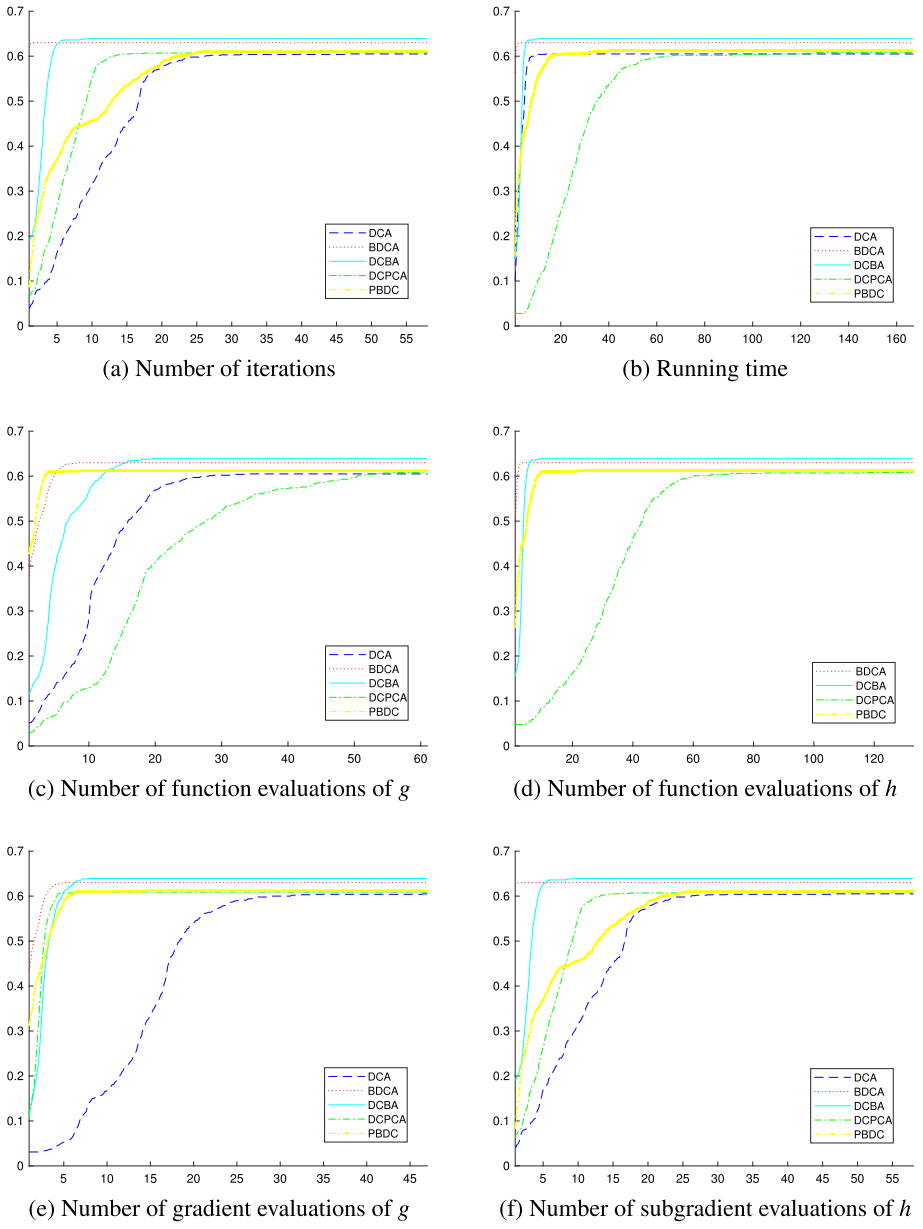
Moreover, further plots show some minor differences in the performance profiles with respect to varying values of the spatial dimension n or number of clusters p. Considering distinction with n first, the ability of solving a problem sinks with growing n, starting with $80-90\%$, ending up with slightly less than 45% in terms of BDCA and DCBA and concerning DCA, PBDC and DCPCA $30-40\%$. While for $n = 2$ PBDC is, with some minor advance, the most likely algorithm to solve the problem, for $n = 5$ hardly any and for $n = 10$ only small differences between the methods are visible. Surprisingly, for $n = 15$ the picture changes noticeably. BDCA and DCBA are now clearly ahead of DCPCA which still outgoes PBDC and DCA. The latter two perform nearly identical in this matter.

The distinctions with varying number of clusters p are more diffuse. Increasing p initially results in a diminished ability of solving a problem. But in the end, further rising leads to the methods being more likely to find the optimum again. The value of p for which the algorithm performs worst varies from method to method. All algorithms start at $p = 5$ with a probability to solve the problem of about 70% with BDCA being slightly on top. For DCA, DCPCA and PBDC, $p = 20$ is the value for which they perform worst with having a probability of solving the problem around 50% in terms of DCA as well as DCPCA and 55% with regard to PBDC. The latter percentage corresponds also to the worst performance of BDCA, which is, however, met at $p = 50$. DCBA remains on a level about 60% for $p = 10$ to $p = 50$. In the end, for $p = 75$ the ability of finding the optimum lies for each algorithm in the range of $60 - 70\%$ with DCPCA falling slightly behind.

In the second illustrative example, we investigate how the administrative districts of Bavaria would look like if their cities got grouped by the minimum sum-of-squares clustering method. To this end, a data set of geographic coordinates of $k = 2073$ Bavarian cities and towns is considered,[3] which consequently get to be separated in $p = 7$ clusters. As initial guess, seven vectors with quasi-random entries in the range $[9.06, 13.80] \times [47.41, 50.52]$ of the geographic coordinates of the considered data set are selected.

All five algorithms determine (approximately) the same seven centroids, but differ in convergence speed. In Fig. 4, the resulting clusters are shown with their centroids marked as pentagrams. Additionally, the first ten iterations of each method beside the ones of DCBA are drawn. Moreover, in Fig. 5 the evolution of the function value with proceeding iterations is plotted for each algorithm. Essentially, the differences in convergence speed are similar to the experiment with random data, with the exception that, in this special instance, PBDC

[3] Source: http://www.fa-technik.adfc.de/code/opengeodb/?C=D;O=A, called on April 15, 2021; determination of membership towards Bavaria by means of postal code.



(a) Number of iterations

(b) Running time

(c) Number of function evaluations of g

(d) Number of function evaluations of h

(e) Number of gradient evaluations of g

(f) Number of subgradient evaluations of h

Fig. 3 Performance profiles of the minimum sum-of-squares clustering problem with random data

is the most promising solution method. BDCA is still ahead of DCBA which, in turn, beats DCPCA as well as DCA.

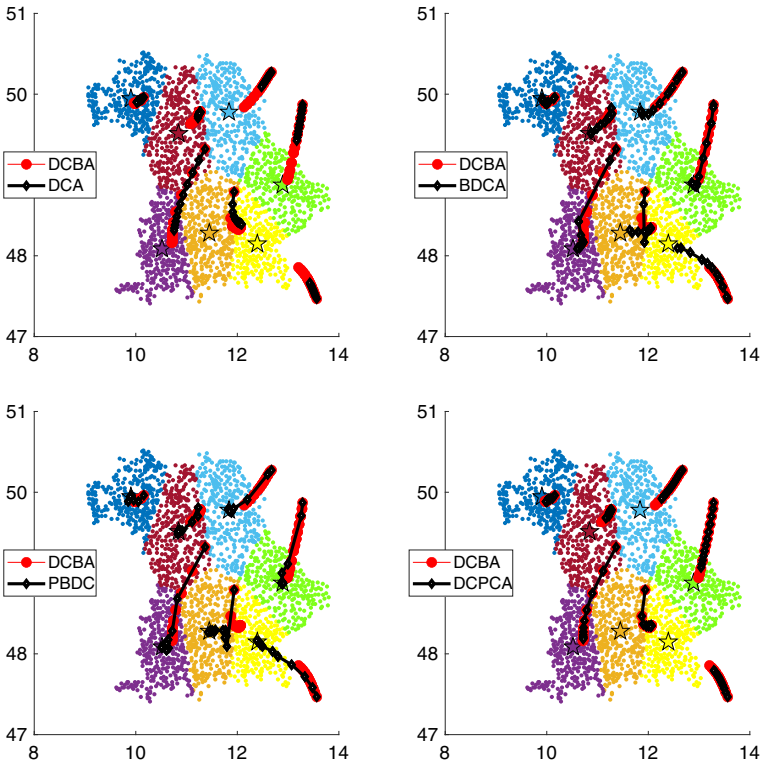


Fig. 4 Division of Bavarian cities into administrative districts by means of minimum sum-of-squares clustering

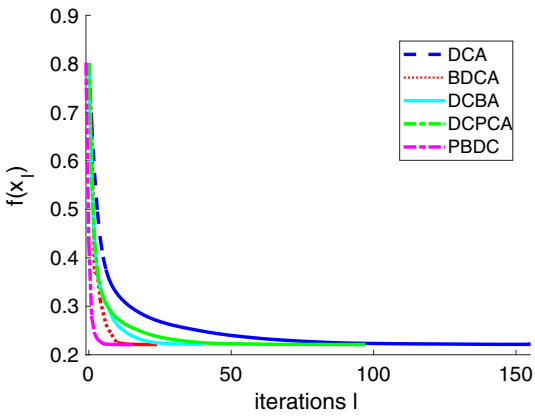


Fig. 5 Evolution of the function value with proceeding iterations while applying minimum sum-of-squares clustering towards Bavarian cities

4.4 The multidimensional scaling problem

Similar to the organization of the previous section, we first give a brief introduction of the problem under consideration (see [2, 19]) and then use two different test settings, one based

on random data and the other one with real data based on the geographic problem from the previous section.

Multidimensional scaling is also a method from data mining. This time, the preprocessing of large data sets for further analysis happens by summarizing data through reduction. To be more precise, having a data set consisting of k points, each of dimension n, the aim is to replace the data set by the same number of points with dimension $p \leq n$. Of course, one tries to keep the relevant information in the best possible way. To this end, the differences within the data set are considered by means of the dissimilarity matrix $\boldsymbol{\delta} \in \mathbb{R}^{k\times k}$ with entries

$$\delta_{ij} := d_{ij}(\tilde{\boldsymbol{X}}) := \left\| \tilde{\boldsymbol{x}}^i - \tilde{\boldsymbol{x}}^j \right\| \qquad i, j = 1, \ldots, k,$$

where $\tilde{\boldsymbol{X}} := (\tilde{\boldsymbol{x}}^1, \tilde{\boldsymbol{x}}^2, \ldots, \tilde{\boldsymbol{x}}^k) \in \mathbb{R}^{n\times k}$ contains the data points to be analyzed. Now, the goal is to find a matrix $\hat{\boldsymbol{X}} \in \mathbb{R}^{p\times k}$ whose dissimilarity matrix approximates the one of the original data (i.e. $\boldsymbol{\delta}$) in an optimal way, and hence reflects the existing differences in the data. Therefore, the underlying optimization problem is

$$\min_{\boldsymbol{X}\in\mathbb{R}^{p\times k}} \tilde{f}(\boldsymbol{X}) := \frac{1}{2} \sum_{i<j} \omega_{ij} \left(d_{ij}(\boldsymbol{X}) - \delta_{ij} \right)^2,$$

where $\boldsymbol{\omega} \in \mathbb{R}^{k\times k}$ is a symmetric matrix consisting of nonnegative weights with zeros on its diagonal. In the following experiments, it is taken as

$$\omega_{ij} = \begin{cases} 1 & \text{if } i \neq j, \\ 0 & \text{if } i = j. \end{cases}$$

Obviously, the case $p = n$ does not yield a reduction in the data set, but leads towards the question whether the original data set can be reproduced by its dissimilarity matrix. Consequently, the optimal function value for this special instance is known to be zero.

Neglecting constant terms, the primary optimization problem can be rewritten as a DC problem with the adapted objective function $f : \mathbb{R}^{p\times k} \to \mathbb{R}$ given by its DC components $g,\ h : \mathbb{R}^{p\times k} \to \mathbb{R}$,

$$g(\boldsymbol{X}) := \frac{1}{2} \sum_{i<j} \omega_{ij} d_{ij}^2(\boldsymbol{X}) + \frac{\rho}{2} \|\boldsymbol{X}\|^2, \quad h(\boldsymbol{X}) := \sum_{i<j} \omega_{ij} \delta_{ij} d_{ij}(\boldsymbol{X}) + \frac{\rho}{2} \|\boldsymbol{X}\|^2.$$

Thereby, a modulus $\rho > 0$ ensures the uniform convexity of g and h and $\|\boldsymbol{X}\|$ denotes the Frobenius norm of the matrix $\boldsymbol{X}$. In the subsequent experiments, $\rho = \frac{1}{kp}$ is chosen depending on the size of the data set and the dimension of the destination space.

The first numerical experiment uses quasi-randomly generated data $\tilde{\boldsymbol{X}} \in \mathbb{R}^{n\times k}$ consisting of normally distributed entries, having a mean of zero and a standard deviation of ten. Reproducing the data as well as reducing each data point to half its size is tested. For that matter, the parameters are taken as $k \in \{25, 50, 75, 100, 125, 150\}$ and $p \in \{2, 3\}$ (and, consequently, $n \in \{p, 2p\}$). For each parameter combination, ten test runs with all algorithms from the beginning of this section are executed.

To construct a suitable initial guess $\boldsymbol{X}^0 \in \mathbb{R}^{p\times k}$, we follow the suggestion in [2]. Thus, we first create a matrix $\tilde{\boldsymbol{X}}^0$ of the same size as $\boldsymbol{X}^0$ with quasi-random entries drawn from the same normal distribution as the data set itself. Afterwards,

$$\boldsymbol{X}^0 := \tilde{\boldsymbol{X}}^0 \left(\mathbf{I}_{k\times k} - \frac{1}{k} \mathbf{E}_{k\times k} \right) \tag{4.1}$$

is set, with $\mathbf{I}_{k\times k}$ denoting the identity matrix and $\mathbf{E}_{k\times k}$ the matrix consisting of ones only. Both, this initial guesses as well as the data to be approximated, change with each test run. Nevertheless, all algorithms have to deal with the very same data.

This time, some parameters are adopted, namely in case of BDCA $\alpha = 0.05$ and $\beta = 0.1$. In addition, the quantities related to the self-adaptive trial stepsize strategy are set to $\gamma = 10$ and $\bar{\lambda}_1 = 10$. The latter two are also used for DCBA, for which $\beta = 0.1$, $m = 0.2$ and $\mu = 0.05$ are selected. Comparably, $m = 0.2$ is also set for the bundle methods PBDC and DCPCA together with $\sigma = 0.1$ for the latter method.

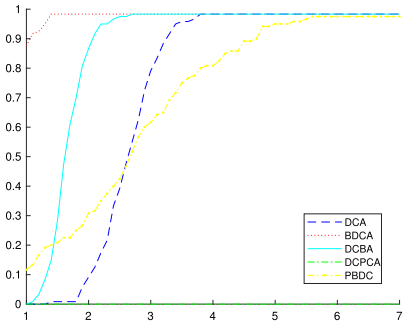
During the overall process, in the case of $p = n$, a method gets rated successful whenever approaching the (a priori known) optimal function value. In the case of $p < n$ for each test run we identify the algorithm among the five considered ones which yields the lowest objective function value. (Note that, in general, there will be more than one method yielding the lowest objective value.) It seems likely to classify this method as successful. Once again, also methods obtaining function values which deviate less than a given tolerance, namely 1, from the optimal or the lowest function value are graded successful.

Reporting for each test run the number of iterations and the running time as well as the number of function and (sub-)gradient evaluations for each DC component leads to the performance profiles shown (in extracts) in the Figs. 6 and 7. The evaluation is done separately for the cases of data reproduction (the case $p = n$) and data reduction (the case $p < n$). Whenever necessary to make the differences between the most efficient algorithms visible, performance profiles are only drawn partially. To get still an impression of the extent of the respective performance profiles, at least the factor allowing even the most inefficient method to reach its maximum probability for solving the problem should be reported. For Figs. 6a and 6f the factor is 63, for 6b as well as 6c 290, for 6d 354, for 7b 2900, for 7c 171 and for 7d 199. In any case, DCPCA is the method with the highest values of the measured quantities. Besides, for Fig. 6b the factor at which the curve of DCBA flattens is 234. Once again, note that DCA never evaluates the second DC component h during iterations as only linearizations of this function are considered. In specific, no line search is performed. Note further that due to using different computers (see Sect. 4.1) the running time is not comparable for this problem, at least not with respect to DCBA.

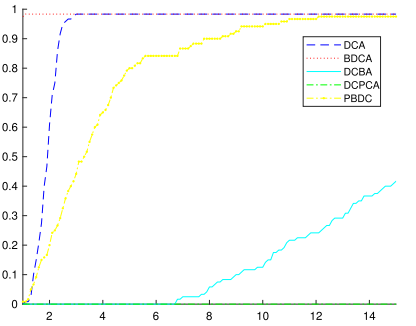
While considering the performance profiles, the first thing to be mentioned is that for each algorithm the probability of being able to solve a problem is lower when reducing the data in comparison to reconstructing the data. But in both cases the probabilities are nearly identical for all algorithms. Further, it is noticeable that PBDC usually requires less function and gradient evaluations of g than BDCA and DCBA, but having a look at the corresponding figures for the function h the relationship reverses nearly completely.

Let us note, in addition, that for DCBA $74 - 81\%$ of the function evaluations of the first DC component fall upon the bundle procedure and the remaining ones upon the line search. There is a clear tendency that the higher the number k of data points to be considered, the higher the percentage of function evaluations in conjunction with the bundle method. Keep in mind that function evaluations of the second DC component only occur during the line search procedure.

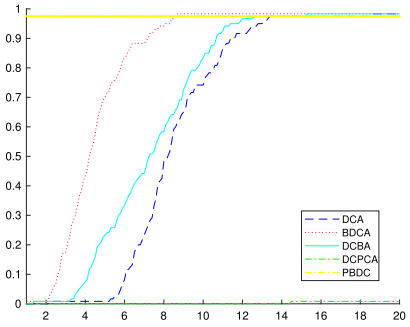
The second example deals, once again, with geographic coordinates of Bavarian cities. Taking the identical set of data as in Sect. 4.3, the task is to reproduce the geographic coordinates on the basis of the dissimilarity matrix related to the raw data. Thereby, the initial guess is constructed similarly as in the previous example. For this purpose, the columns of the auxiliary matrix $\tilde{\boldsymbol{X}}^0 \in \mathbb{R}^{2\times 2073}$ take quasi-random values ranging in $[0, 4.74] \times [0, 3.12]$, motivated by the east–west as well as the north–south extension of Bavaria, determined on



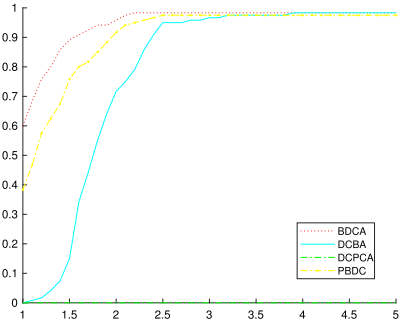
(a) Number of iterations (extract)



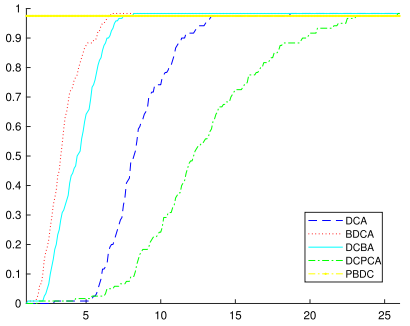
(b) Running time (extract)



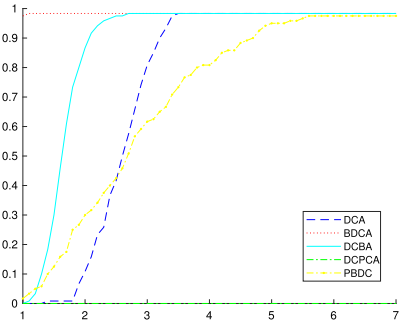
(c) Number of function evaluations of g (extract)



(d) Number of function evaluations of h (extract)



(e) Number of gradient evaluations of g

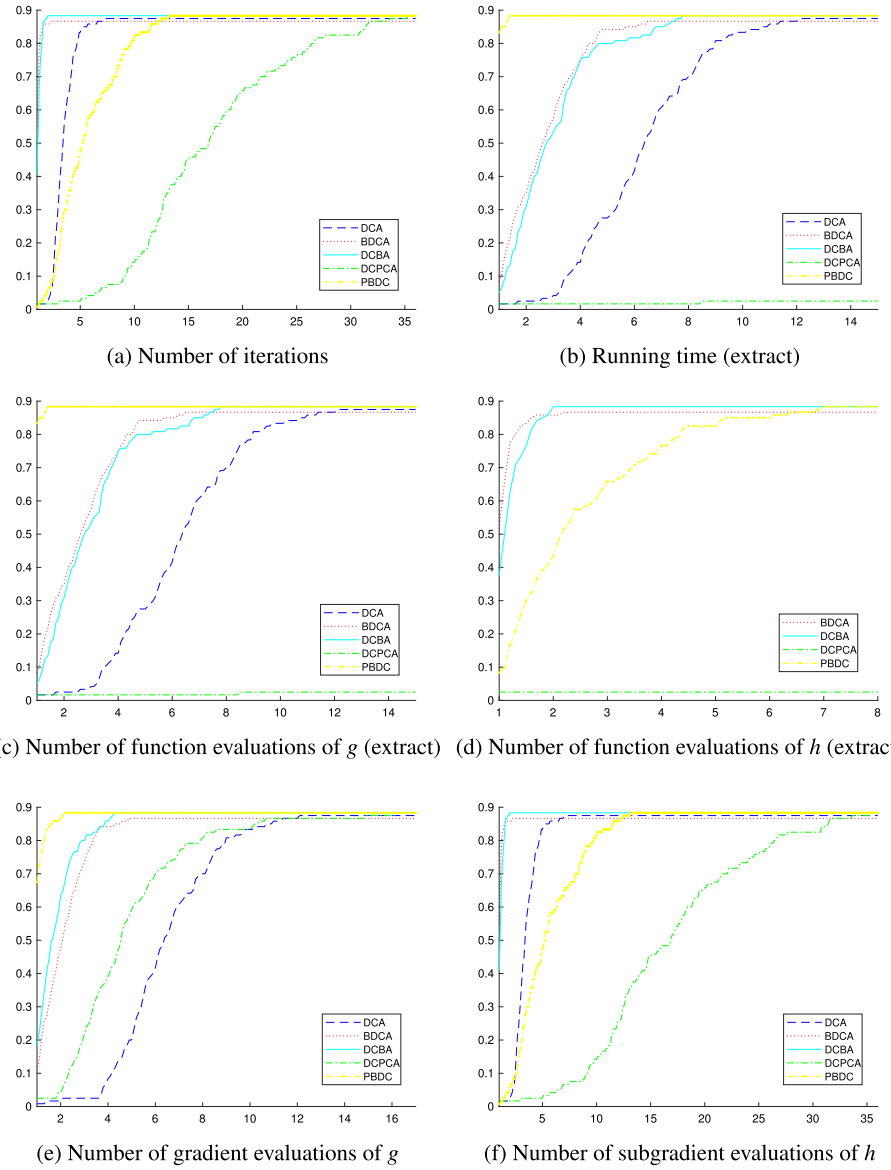


(f) Number of subgradient evaluations of h (extract)

Fig. 6 Performance profiles (partly extracts) of reconstructing (case $p = n$) random data by means of multi-dimensional scaling

the basis of the underlying data set. To each column of the resulting matrix $\boldsymbol{X}^0$ from (4.1), the geographic mean of Bavaria $(11.43, 48.93)^T$ (once again calculated with respect to the cities under consideration) is added. We use the standard parameter settings from the beginning of this subsection, with the only exception that $\bar{\varepsilon}_1 = 10^{-2}$.

All algorithms manage to restore the map of Bavarian cities, but with differences in the speed of convergence. This distinction gets already foreshadowed in Fig. 8, in which next to



(a) Number of iterations

(b) Running time (extract)

(c) Number of function evaluations of g (extract)

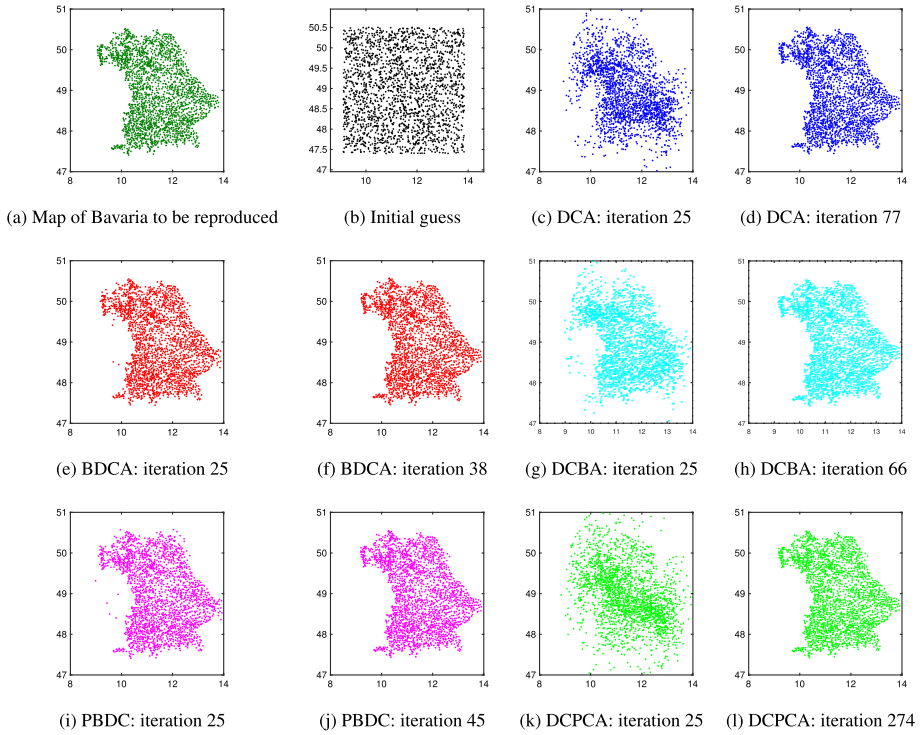
(d) Number of function evaluations of h (extract)

(e) Number of gradient evaluations of g

(f) Number of subgradient evaluations of h

Fig. 7 Performance profiles (partly extracts) of data reduction (case $p < n$) of random data by means of multidimensional scaling

the map to be reproduced and the initial guess also, for each method, the current standing at iteration 25 as well as the final result is pictured (note that the slight rotation is due to the formulation of the problem, estimating only the distances between cities). Further plots reveal that the major progress happens during the first 60% of iterations, whereas the changes following afterwards can hardly be seen in corresponding images. All methods yield satisfying results although BDCA and PBDC come out on top in terms of iterations.



(a) Map of Bavaria to be reproduced (b) Initial guess (c) DCA: iteration 25 (d) DCA: iteration 77

(e) BDCA: iteration 25 (f) BDCA: iteration 38 (g) DCBA: iteration 25 (h) DCBA: iteration 66

(i) PBDC: iteration 25 (j) PBDC: iteration 45 (k) DCPCA: iteration 25 (l) DCPCA: iteration 274

Fig. 8 Reproducing the map of Bavaria by means of multidimensional scaling

4.5 Edge detection by means of a DC optimization based clustering technique

Edge detection is a well known method in image segmentation for carving out certain objects in an image. It is based on the idea of determining contours of objects marked by discontinuities and erratic changes in the brightness values of the grey scale image. The technique presented in the following, using a DC optimization based clustering approach, was developed in [17]. To each pixel, a vector representing the differences in the grey scale values with respect to nearby pixels gets assigned. Subsequently, the norms of these vectors are split into two groups yielding a differentiation into pixels belonging to an edge and the ones which do not.

Consider a grey scale image consisting of $(n+2) \times (m+2)$ pixels with coordinates $(k,l) \in \{1,\ldots,n+2\} \times \{1,\ldots,m+2\}$. To each pixel of the interior of the image with coordinates $(k,l) \in \{2,\ldots,n+1\} \times \{2,\ldots,m+1\}$ a vector $\boldsymbol{v}_i \in \mathbb{R}^4$, $i = 1,\ldots,N$ with $N = nm$, containing the differences in the brightness values of the pixel under consideration and the four vertical and horizontal immediately adjacent pixels gets assigned. For obvious reasons, marginal pixels are neglected and also will not get classified in the progress. Taking the norm $a_i := \|\boldsymbol{v}_i\|$, $i = 1,\ldots,N$, of each such vector, one attains a measure of change in the grey scale values in relation to the neighborhood of the central pixel. A high value indicates an affiliation towards an edge, whereas a low value does not. This motivates to separate the set $\{a_i\}_{i=1,\ldots,N}$ into two groups. To this end, the well known K-means clustering method gets applied with $K = 2$. Denoting with $z_1, z_2 \in \mathbb{R}$ the variables for determining the

centroids, the resulting optimization problem is given by

$$\min_{z_1,z_2\in\mathbb{R}} f(z_1,z_2) := \sum_{i=1}^{N} \min\{|a_i - z_1|, |a_i - z_2|\}.$$

Similar to the clustering method introduced in Sect. 4.3, the objective function f can be written as a DC function with (uniformly) convex DC components $g, h : \mathbb{R}^2 \to \mathbb{R}$,

$$g(z_1,z_2) := \sum_{i=1}^{N} (|a_i - z_1| + |a_i - z_2|) + \frac{\rho}{2} \left\| (z_1,z_2)^T \right\|^2,$$

$$h(z_1,z_2) := \sum_{i=1}^{N} \max\{|a_i - z_1|, |a_i - z_2|\} + \frac{\rho}{2} \left\| (z_1,z_2)^T \right\|^2$$

with modulus $\rho \geq 0$. In the following, ρ is chosen to be 0.1.

On the basis of [17], the starting point is selected as

$$\begin{pmatrix} z_1^0 \\ z_2^0 \end{pmatrix} := \kappa\,(a_{max} - a_{min}) \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

with $a_{max} = \max_{i=1,\ldots,N} a_i$ and $a_{min} = \min_{i=1,\ldots,N} a_i$. However, κ varies in our experiments with the image under consideration, as not only but especially DCBA turns out to be pretty sensitive regarding the initial choice.

Moreover, in contrast to many other applications, the minimization of this clustering problem does not need to be carried out with a high precision in order to yield a satisfactory classification of edges. Therefore, for our subsequent numerical experiments with some classical test images for edge detection, the termination tolerance $\bar{\varepsilon}_1$ is reduced to 0.1. In addition, the maximum number of iterations is limited to five. It turns out that these adaptions still allow acceptably good approximations of the centroids leading to satisfactory classification of edges.

The parameter m for all three bundle methods is adapted to 0.1. Note that, this time, BDCA cannot be applied to solve the optimization problem, as the corresponding first DC component g is nonsmooth. Three classical test images for edge detection are considered. Thereby, both, the cameraman as well as the house, contain 256×256 pixels whereas the moon spans 537×358 pixels. For determining the initial point we choose $\kappa = \frac{1}{3}$ in terms of the cameraman, $\kappa = 0.3$ with regard to the house and $\kappa = 0.25$ for the moon.

The results are shown in Fig. 9, in which also the input images are displayed in the first column. While for the cameraman the output seems apparently identical, differences can be seen for the moon. Here, DCPCA and PBDC detect the fading edge of the moon more clearly than DCA and DCBA. For the house, differences only get visible while considering DCBA. This method recognizes the top end of the chimney slightly better than the remaining algorithms, and also identifies, in a better way, some less pronounced edges as dotted lines.

Let us note that, in connection with the house, the sensitivity of DCBA regarding the initial value gets particularly clear, especially when adapting the parameter m at the same time. Only slight modifications lead from recognizing hardly any edges over a result comparable to the ones from the other methods to detection of every brick stone (see Fig. 10 for which $m = 0.5$ and $\kappa = 0.25$ is chosen). In contrast, the output of the remaining algorithms stays pretty stable.

Although the number of iterations is not reported here in detail, it is worth mentioning that, in many instances, the desired accuracy of $\bar{\varepsilon}_1 = 0.1$ is often reached within less than

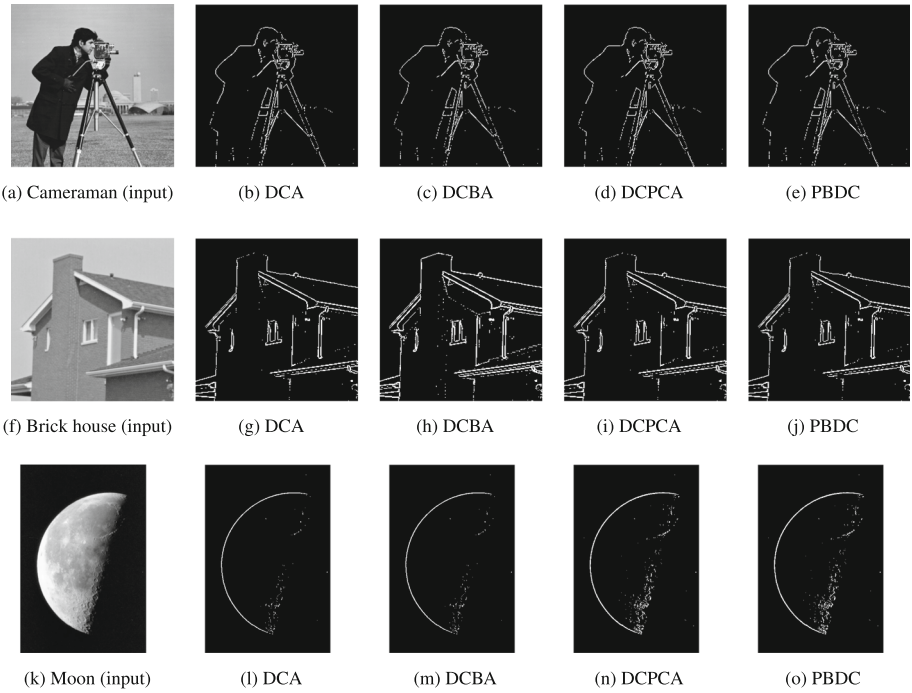


Fig. 9 Edge detection by means of a DC optimization based clustering technique

Fig. 10 Detecting brick stones with DCBA

the maximum of five iterations. Only DCPCA exploits the full number of iterations for each image. Altogether all remaining methods yield pretty similar results for this application, detecting sharp edges quite reliably.

Table 3 Instance of termination for each method applied to the collection of 46 academic test problems

	Known best function value f^*	Function value larger than f^*	Timeout
DCA	42	4	0
DCBA	42	4	0
PBDC	37	5	4
DCPCA	31	10	5
BDCA	8	2	0

4.6 A collection of academic benchmark problems

In this section, a set of ten academic DC test problems from [16], which in parts originate from [3], is considered to further assess the performance of the newly introduced algorithm in comparison to the methods mentioned at the beginning of this overall section. Some of the problems have a variable dimension n, so that a range of $n = 2$ to $n = 50{,}000$ is covered by a total of 46 test instances. All functions are nonsmooth, which is why BDCA, requiring the first DC component to be smooth, can only be applied to the Problems 9 and 10. Besides, the initial guesses proposed in the cited paper are used as starting points.

Above all, we add to the respective termination criteria of the algorithms a further escape procedure which is independent of the used method. Since in the paper [16] for each test instance a known best value is reported, we also terminate a method whenever it approaches the corresponding function value within a tolerance of $\min\{10^{-3}n, 0.1\}$ where n denotes the number of variables. Moreover, some parameters have to be adapted. For DCBA, we set $m = 0.2, \mu = 0.05$ as well as $\beta = 0.05$ whenever $n < 10$, and $\beta = 0.6$ whenever $n \geq 10$. In addition, for the self-adaptive trial stepsize strategy used for DCBA and BDCA, the parameters are chosen as $\bar{\lambda}_1 = \gamma = 20$ whenever $n < 10$, and $\bar{\lambda}_1 = \gamma = \left(\frac{10}{6}\right)^3$ whenever $n \geq 10$. Besides, for BDCA β is selected in the same way as for DCBA and α is set to 0.05. For PBDC as well as DCPCA m is changed to 0.2 and in terms of DCPCA, σ is chosen in the same way as β for DCBA and BDCA, respectively.

For each instance of the academic DC test problems, we report the value of the objective function at the point of termination, the running time, the number of iterations, as well as the number of evaluations of the first and second DC component itself and their (sub)gradients. The detailed listing of the results can be found in the "Appendix". For the sake of completeness, we tabulate the respective outcomes from the academic test problem (see Sect. 4.2), clustering of the Bavarian cities by the minimum sum-of-squares approach (see Sect. 4.3), reproducing the Bavarian map (see Sect. 4.4), and edge detection for the cameraman, the brick house and the moon (see Sect. 4.5).

For each instance of the academic DC test problems, an algorithm is considered as failed if the running time exceeds a limit of one hour which only happens with the two bundle methods PBDC and DCPCA. Moreover, in a few cases, some of the algorithms terminate in a point for which the corresponding function value is worse than the best known one. A quick overview of how often these instances occur with each algorithm within the 46 problem instances, is given in Table 3. Note that BDCA can merely be applied to ten of the test problem instances. It is worth mentioning that DCBA together with DCA is the method with the (relatively) least instances of not detecting the known best function value f^*.

Let us note some distinctive features in the detailed results. Having a look at the problems with variable dimensions that finally cover large scales, clearly DCA, DCBA and, for

Problem 10, also BDCA come out on top. The two bundle methods PBDC and DCPCA suffer from rapidly increasing running times with growing dimensions in Problem 4, finally exceeding the time limit. For Problem 10 in case of higher number of variables both methods end up with function values deviating heavily from the known best ones. The results obtained by DCA for Problem 4 are standing out. For every single test instance it requires only one single iteration which favors quite low numbers of evaluations compared to DCBA. Though also Problem 5 covers large dimensions, the situation here is a bit different. Following [16], the starting point is selected in such a way that, with increasing dimension, one already has a pretty good estimate for the optimum. This may explain, to some extent, the unexpected and heterogeneous behavior of the algorithms. While for some methods a few measured quantities hardly change with growing dimension, other measures show a rather continuing but quite low increase and yet others reveal a fairly erratic behavior. At this point, we want to thank an anonymous referee for drawing our attention to the special structure of Problem 5.

Furthermore, there are also some visible differences in the small-scale problems whose number of variables are at most four. First of all, PBDC is the only algorithm to solve Problem 9. Though the both bundle methods, PBDC as well as DCPCA, are able to find the optimum for Problem 2 DCPCA requires much less evaluations than PBDC. Similarly, for Problem 7 next to PBDC it is DCBA which succeeds and, once again, PBDC is the one with the noticeably higher number of evaluations. Nevertheless, in case of convergence PBDC fails to reach the known best function value only for some instances of one single problem class, namely Problem 10.

5 Concluding remarks

In this article, a bundle method to solve unconstrained DC optimization problems (DCBA) was introduced. In contrast to various existing bundle methods designed for this topic, the bundles are not directly constructed with respect to the DC components of the objective function. Instead, a convex approximation of the function to be minimized, which is already known from the classical DC Algorithm (DCA, see [21]), gets constructed first. Applying the bundle method towards the model function yields a descent direction for the original objective function which allows to add a line search afterwards. Hence, the concept of DCBA is pretty similar to the one of the Boosted DCA (BDCA) which is introduced in [2]. The latter one considers the same convex approximation known from the DCA and constructs a descent direction by minimizing this model function. Contrary to BDCA, the bundle method in DCBA is not carried out until a minimizer of the approximation is found, but only until a serious step is executed the very first time. An advantage of DCBA against BDCA is that the new method can be applied to DC problems with both DC components being nonsmooth, whereas the convergence theory of BDCA requires the first DC component to be smooth.

The algorithm was shown to be well-defined for functions being bounded from below. In addition, the method was proven to be globally convergent in the sense that every accumulation point is a critical point of the objective function. Moreover, termination of the algorithm (with numerically implementable termination criterion) occurs within a finite number of iterations in a point satisfying an approximate criticality measure.

The performance of the algorithm was tested by means of diversified nonsmooth DC problems and compared to four other DC methods, DCA, BDCA and two bundle methods, namely the Proximal Bundle method for DC optimization (PBDC, see [16]) and the DC Piecewise-Concave Algorithm (DCPCA, see [10]). In specific, DCBA together with DCA

are the algorithms succeeding most frequently while considering a bunch of various academic test problems. Clearly, DCA falls behind by far performing even worse than most of the other methods while considering the applications of minimum sum-of-squares clustering and multidimensional scaling. With these problems DCBA visibly gets outperformed by BDCA which still has the disadvantage of only being applicable to a restricted class of DC problems. But none of the two remaining bundle methods has proven to be apparent advantageous over DCBA. In addition, the results for detecting edges with the new algorithm are rather encouraging.

So far, this method is applicable to unconstrained DC programs only. Our future work will concentrate on suitable extensions to constrained problems, first to DC programs with convex constraints, and then also to DC programs with general DC-type constraints.

Acknowledgements The authors would like to thank two anonymous referees for their suggestions and comments which helped a lot to improve the numerical tests.

Funding Open Access funding enabled and organized by Projekt DEAL. No funds, grants or other support was received for conducting this study.

Data availability The geographic data that support the findings of this study are available from the ADFC, http://www.fa-technik.adfc.de/code/opengeodb/?C=D;O=A.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose

Appendix: Detailed results for the collection of academic test problems

A quick overview over the test instances taken from [16] is given in Table 4, which is inspired by [10]. Thereby, ID is an instance identifier, n gives the spatial dimension and f^* the known best value of the objective function. Note that in [10] for Problem 10.03 f^* has been adapted to a newly detected better function value. In addition, the problems AP, C_BY, R_BY, ED_C, ED_H and ED_M refer to the academic test problem from Sect. 4.2, the clustering of the Bavarian cities, the reproduction of the Bavarian map and edge detection for the cameraman, the house and the moon.

The detailed results are then shown in Tables 5, 6, 7, 8 and 9. Here the value of the objective function at the point of termination f, the running time, the number of iterations it, as well as the number of evaluations of the first and second DC component itself and their (sub)gradients, in sequence denoted by n_{f_1}, n_{f_2}, n_{ξ_1}, n_{ξ_2}, gets reported. Note that in case of the problem AP the average quantities over all 10,000 test runs are tabulated. Besides, for the problems related to edge detection the function value f at the point of termination is not listed, as no minimization of the objective function is aspired in the first place. Instead,

the algorithms may terminate after a preset number of iterations. Furthermore, DCA does not require an evaluation of the second DC component h during the iterations. Moreover, for DCA, BDCA, DCBA and DCPCA one has $it = n_{\xi_2}$ and for PBDC $n_{f_1} = n_{f_2}$. See the references in Table 1 for clarification. Correspondingly, the respective columns are combined or neglected. Besides, in case a method fails within a test class not only once but for all subsequent instances, only the first fail gets reported in Tables 5, 6, 7, 8 and 9. Furthermore, an instance is marked with † if for an algorithm the function value at the point of termination differs significantly from the known best one f^*.

Table 4 An overview of the benchmark problems

ID	n	f^*
1.01	2	2
2.01	2	0
3.01	4	0
4.01	2	0
4.02	5	0
4.03	10	0
4.04	50	0
4.05	100	0
4.06	150	0
4.07	200	0
4.08	250	0
4.09	350	0
4.10	500	0
4.11	750	0
5.01	2	0
5.02	5	0
5.03	10	0
5.04	50	0
5.05	100	0
5.06	150	0
5.07	200	0
5.08	250	0
5.09	300	0
5.10	350	0
5.11	400	0
5.12	500	0
5.13	1000	0
5.14	1500	0
5.15	3000	0
5.16	10,000	0
5.17	15,000	0
5.18	20,000	0
5.19	50,000	0
6.01	2	−2.5
7.01	2	0.5
8.01	3	3.5
9.01	4	1.83
10.01	2	−0.5
10.02	4	−2.5
10.03	5	−3.5
10.04	10	−8.5

Table 4 continued

ID	n	f^*
10.05	20	−18.5
10.06	50	−48.5
10.07	100	−98.5
10.08	150	−148.5
10.09	200	−198.5
AP	2	−2
C_BY	14	Unknown
R_BY	4146	0
ED_C	2	Unknown
ED_H	2	Unknown
ED_M	2	Unknown

Table 5 Computational results for DCA being applied to the benchmark problems

ID	f	Time	n_{f_1}	n_{ξ_1}	$it,\ n_{\xi_2}$
1.01	2.0014560657	1.01	178	178	7
2.01	$1^{\dagger}$	0.01	11	11	2
3.01	6.72795e−13	0.01	10	10	1
4.01	0	0.01	3	3	1
4.02	1.77636e−15	0.01	6	6	1
4.03	0	0.00	11	11	1
4.04	1.17234e−09	0.04	60	60	1
4.05	2.83126e−09	1.30	125	125	1
4.06	4.28008e−09	0.26	189	189	1
4.07	5.44605e−09	0.40	251	251	1
4.08	6.77755e−09	0.67	315	315	1
4.09	2.23299e−08	1.52	439	439	1
4.10	3.04426e−08	3.92	629	629	1
4.11	4.69154e−08	14.20	958	958	1
5.01	0	0.01	4	4	1
5.02	8.17124e−14	0.00	7	7	1
5.03	1.86073e−06	0.01	18	18	1
5.04	3.15173e−06	0.01	16	16	1
5.05	1.08981e−05	0.01	19	19	1
5.06	1.16688e−05	0.04	19	19	1
5.07	1.10469e−05	0.04	19	19	1
5.08	1.13287e−05	0.01	19	19	1
5.09	1.14836e−05	0.01	19	19	1
5.10	1.15818e−05	0.01	19	19	1
5.11	1.16505e−05	0.01	19	19	1
5.12	1.17393e−05	0.01	19	19	1
5.13	1.18952e−05	0.02	19	19	1

Table 5 continued

ID	f	Time	n_{f_1}	n_{ξ_1}	$it,\ n_{\xi_2}$
5.14	1.19418e−05	0.02	19	19	1
5.15	1.19871e−05	0.02	19	19	1
5.16	1.20178e−05	0.03	19	19	1
5.17	1.20221e−05	0.03	19	19	1
5.18	1.20243e−05	0.04	19	19	1
5.19	1.20281e−05	0.08	19	19	1
6.01	−2.4999762506	0.01	27	27	1
7.01	1.0000037703 †	0.51	1445	1445	61
8.01	3.5000001213	0.02	40	40	2
9.01	9.2†	0.01	7	7	2
10.01	−0.5	0.01	3	3	1
10.02	−2.5	0.00	9	9	3
10.03	−2.5†	0.00	10	10	4
10.04	−8.5	0.00	18	18	6
10.05	−18.5	0.32	33	33	11
10.06	−48.5	0.00	78	78	26
10.07	−98.5	0.00	153	153	51
10.08	−148.5	0.00	228	228	76
10.09	−198.5	0.01	303	303	101
AP	−0.9887†	1.8693e−04	28.90	28.90	7.23
C_BY	0.2213836038	1.53	620	620	155
R_BY	0.3395275599	29.35	1078	1078	77
ED_C		0.11	54	54	3
ED_H		0.09	45	45	2
ED_M		0.03	5	5	1

Table 6 Computational results for DCBA being applied to the benchmark problems

ID	f	Time	n_{f_1}	n_{f_2}	n_{ξ_1}	$it,\ n_{\xi_2}$
1.01	2.0003825803	0.05	96	13	73	5
2.01	1†	0.01	34	11	13	5
3.01	2†	0.01	38	8	22	4
4.01	0	0.01	8	4	2	1
4.02	1.77636e−15	0.01	16	6	6	2
4.03	7.10543e−15	0.02	87	26	45	8
4.04	5.45469e−10	0.19	423	60	325	19
4.05	5.54792e−11	1.40	2264	122	2060	41
4.06	4.72937e−10	4.02	5500	187	5187	63
4.07	8.17818e−09	8.99	10,446	259	10,013	87
4.08	6.54836e−11	18.25	17,253	323	16,710	110
4.09	3.56522e−10	53.21	33,341	451	32,584	153

Table 6 continued

ID	f	Time	n_{f_1}	n_{f_2}	n_{ξ_1}	$it,\ n_{\xi_2}$
4.10	3.81260e−09	188.12	66,809	636	65,739	217
4.11	2.50293e−09	899.81	154,149	974	152,511	332
5.01	0	0.01	9	4	3	1
5.02	7.60605e−04	0.01	18	6	8	2
5.03	0.0078666857	0.02	108	29	59	10
5.04	0.0313034366	0.05	112	32	60	10
5.05	0.0529657260	0.02	67	19	36	6
5.06	0.0966214273	0.05	105	28	59	9
5.07	0.0660497863	0.04	126	35	69	11
5.08	0.0326968000	0.03	77	21	42	7
5.09	0.0713988745	0.03	79	21	44	7
5.10	0.0596119960	0.03	69	19	38	6
5.11	0.0598831112	0.03	69	19	38	6
5.12	0.0602438913	0.02	69	19	38	6
5.13	0.0609126393	0.03	69	19	38	6
5.14	0.0730848593	0.05	116	33	63	10
5.15	0.0732227854	0.05	116	33	63	10
5.16	0.0733163697	0.07	116	33	63	10
5.17	0.0733295474	0.09	116	33	63	10
5.18	0.0729041586	0.13	123	33	70	10
5.19	0.0729143051	0.22	123	33	70	10
6.01	−2.4986816434	0.01	92	41	17	17
7.01	0.5055140618	0.09	378	53	279	23
8.01	3.5000014977	0.01	38	11	19	4
9.01	$9.2000020146^{\dagger}$	0.02	57	15	28	7
10.01	−0.5	0.01	8	4	2	1
10.02	−2.5	0.01	20	8	6	3
10.03	$-2^{\dagger}$	0.01	17	6	5	3
10.04	−8.4949669717	0.01	84	36	24	12
10.05	−18.4878486549	0.75	114	50	32	16
10.06	−48.4722549152	0.04	210	90	60	30
10.07	−98.4375704937	0.08	378	162	108	54
10.08	−148.4375704936	0.09	555	239	158	79
10.09	−198.4375704936	0.14	729	313	208	104
AP	-2	0.0040	20.14	7.02	7.07	3.02
C_BY	0.2213811086	0.81	311	106	123	41
R_BY	0.2480235999	112.53	1394	156	1104	66
ED_C		0.13	103	56	35	5
ED_H		0.17	134	54	68	5
ED_M		0.04	8	1	5	1

Table 7 Computational results for PBDC being applied to the benchmark problems

ID	f	Time	it	n_{f_1}, n_{f_2}	n_{ξ_1}	n_{ξ_2}
1.01	2.0002067031	0.04	3	32	16	9
2.01	0.0019842936	17.22	7428	57,543	15,082	9708
3.01	3.80085e−14	0.10	27	213	112	55
4.01	0	0.01	1	12	3	3
4.02	1.77636e−15	0.02	3	30	17	12
4.03	0	0.05	5	54	36	18
4.04	2.70126e−06	60.50	38	1173	1133	384
4.05	1.43283e−05	245.54	286	19,042	18,754	4356
4.06	0.0896309132	2072.82	785	78,623	77,836	9770
4.07	Fail					
4.08	5.49378e−04	1225.90	84	12,181	12,180	624
4.09	Fail					
5.01	3.55271e−15	0.01	0	6	2	2
5.02	9.56679e−04	0.03	7	57	25	16
5.03	0.0086002902	0.01	2	24	8	7
5.04	0.0412995891	0.11	6	44	20	15
5.05	0.0749914399	0.04	4	23	7	7
5.06	0.0327626849	0.13	4	23	7	7
5.07	0.0999494987	0.05	7	33	8	8
5.08	0.0760572740	0.18	23	99	26	26
5.09	0.0433670333	0.07	3	17	8	6
5.10	0.0892753859	0.03	1	8	3	3
5.11	0.0853199242	0.03	1	8	3	3
5.12	0.0786194939	0.03	1	8	3	3
5.13	0.0324731826	0.16	9	40	10	10
5.14	0.0282671300	0.57	24	100	25	25
5.15	0.0177153034	0.13	0	5	2	2
5.16	0.0028749154	8.68	25	106	27	27
5.17	0.0018997433	10.66	20	86	22	22
5.18	0.0014116230	14.19	17	74	19	19
5.19	5.96260e−04	42.04	11	50	13	13
6.01	−2.4999982503	0.01	1	10	4	3
7.01	0.5005553918	6.76	1599	13,351	6507	5045
8.01	3.5005070834	0.01	3	26	8	6
9.01	1.8353351846	0.03	5	34	21	18
10.01	−0.4998947160	0.02	2	17	6	4
10.02	−2.4997482719	0.01	2	16	6	3
10.03	$-2.4999998830^{\dagger}$	0.04	11	67	53	34
10.04	−8.4929465556	0.01	4	26	8	5
10.05	−18.4996188085	0.12	17	83	50	22
10.06	$-42.4999999582^{\dagger}$	0.14	23	112	98	26
10.07	$-88.4999999781^{\dagger}$	4.80	28	333	315	100

Table 7 continued

ID	f	Time	it	n_{f_1}, n_{f_2}	n_{ξ_1}	n_{ξ_2}
10.08	$-10.4999999564^{\dagger}$	0.14	14	63	54	17
10.09	$-10.4999999775^{\dagger}$	0.17	23	86	70	25
AP	−1.9993	0.03	5.15	33.02	23.69	8.20
C_BY	0.2208007332	0.80	15	74	73	70
R_BY	6.98492e−08	170.48	45	105	93	68
ED_C		0.06	5	19	17	5
ED_H		0.04	3	14	11	4
ED_M		0.07	2	12	9	4

Table 8 Computational results for DCPCA being applied to the benchmark problems

ID	f	Time	n_{f_1}	n_{f_2}	n_{ξ_1}	it, n_{ξ_2}
1.01	2.0017910150	0.07	180	168	33	21
2.01	3.90799e−14	0.01	26	25	9	8
3.01	2.23210e−14	0.05	350	308	68	26
4.01	8.88178e−16	0.01	39	39	10	10
4.02	3.55271e−15	0.04	163	162	41	40
4.03	0	0.09	776	710	97	31
4.04	4.42917e−07	111.37	46,117	42,670	3756	309
4.05	4.68210e−07	261.97	323,695	299,560	24,955	820
4.06	1.90172e−06	1610.43	971,588	901,459	71,656	1527
4.07	Fail					
5.01	0	0.01	12	11	3	2
5.02	8.46999e−04	0.01	42	39	10	7
5.03	0.0013435944	0.02	223	211	19	7
5.04	0.0313034366	0.05	112	32	60	10
5.05	0.0120487789	0.04	131	120	15	4
5.06	0.0172676489	0.13	316	298	24	6
5.07	0.0621247512	0.12	321	302	25	6
5.08	0.0704560942	0.14	197	178	25	6
5.09	0.0423469850	0.28	570	541	37	8
5.10	0.0475144774	0.18	272	257	20	5
5.11	0.0670609172	0.27	352	332	26	6
5.12	0.0678427869	0.38	355	335	26	6
5.13	0.0692553445	1.01	374	354	26	6
5.14	0.0696868377	1.64	375	355	26	6
5.15	0.0646056422	2.58	307	292	20	5
5.16	0.0654089843	16.43	324	309	20	5
5.17	0.0655219769	28.22	341	326	20	5
5.18	0.0655783301	46.06	342	327	20	5

Table 8 continued

ID	f	Time	n_{f_1}	n_{f_2}	n_{ξ_1}	$it,\ n_{\xi_2}$
5.19	0.0656792602	227.08	378	363	20	5
6.01	−2.4985065896	0.03	112	112	32	32
7.01	1.0115349152†	0.02	80	73	14	7
8.01	3.7500006789†	0.03	118	104	24	10
9.01	9.2000020146†	0.03	56	50	16	10
10.01	−0.5	0.02	31	30	9	8
10.02	−2.5	0.02	115	114	24	23
10.03	−2.5†	0.02	122	121	26	25
10.04	−6.5†	0.01	48	47	8	7
10.05	−6.5†	0.13	48	47	8	7
10.06	−6.5†	0.03	52	51	8	7
10.07	−6.5†	0.02	56	55	9	8
10.08	−4.5†	0.04	61	60	10	9
10.09	−4.5†	0.03	62	61	10	9
AP	−1.9774	0.01	24.72	24.06	6.33	5.66
C_BY	0.2210706670	3.97	662	662	98	98
R_BY	0.0358464699	656.53	3921	3903	292	274
ED_C		0.17	146	141	12	5
ED_H		0.16	190	184	13	5
ED_M		0.30	120	119	8	5

Table 9 Computational results for BDCA being applied to the benchmark problems

ID	f	Time	n_{f_1}	n_{f_2}	n_{ξ_1}	$it,\ n_{\xi_2}$
9.01	9.2†	0.00	7	0	7	2
10.01	−0.5	0.00	4	0	4	2
10.02	−2.5	0.00	9	3	6	2
10.03	−2.5†	0.01	12	5	7	3
10.04	−8.5	0.01	68	50	18	6
10.05	−18.5	0.01	95	62	33	11
10.06	−48.5	0.00	185	107	78	26
10.07	−98.5	0.00	338	185	153	51
10.08	−148.5	0.01	483	255	228	76
10.09	−198.5	0.01	634	331	303	101
AP	−2	7.5831e−05	22.16	10.13	12.03	3.76
C_BY	0.2209037511	0.53	155	59	96	24
R_BY	0.4782017898	18.43	620	88	532	38

References

1. Aragón Artacho, F.J., Fleming, R.M.T., Vuong, P.T.: Accelerating the DC algorithm for smooth functions. Math. Program. **169**(1), 95–118 (2018). https://doi.org/10.1007/s10107-017-1180-1
2. Aragón Artacho, F.J., Vuong, P.T.: The boosted difference of convex functions algorithm for nonsmooth functions. SIAM J. Optim. **30**(1), 980–1006 (2020). https://doi.org/10.1137/18M123339X
3. Bagirov, A.M.: A method for minimization of quasidifferentiable functions. Optim. Methods Softw. **17**(1), 31–60 (2002). https://doi.org/10.1080/10556780290027837
4. Beck, A.: Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with MATLAB. Society for Industrial and Applied Mathematics, MOS-SIAM Series on Optimization (2014)
5. Clarke, F.H.: Optimization and Nonsmooth Analysis. Canadian Mathematical Society Series of Monographs and Advanced Texts. Wiley, New York (1983)
6. de Leone, R., Gaudioso, M., Grippo, L.: Stopping criteria for linesearch methods without derivatives. Math. Program. **30**(3), 285–300 (1984). https://doi.org/10.1007/BF02591934
7. de Oliveira, W.: Proximal bundle methods for nonsmooth DC programming. J. Glob. Optim. **75**(2), 523–563 (2019). https://doi.org/10.1007/s10898-019-00755-4
8. de Oliveira, W.: The ABC of DC programming. Set-Valued Var. Anal. **28**(4), 679–706 (2020). https://doi.org/10.1007/s11228-020-00566-w
9. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002). https://doi.org/10.1007/s101070100263
10. Gaudioso, M., Giallombardo, G., Miglionico, G., Bagirov, A.M.: Minimizing nonsmooth DC functions via successive DC piecewise-affine approximations. J. Glob. Optim. **71**(1), 37–55 (2018). https://doi.org/10.1007/s10898-017-0568-z
11. Geiger, C., Kanzow, C.: Theorie und Numerik restringierter Optimierungsaufgaben. Springer-Lehrbuch Masterclass. Springer, Berlin (2002)
12. Hiriart-Urruty, J.B.: Generalized differentiability/duality and optimization for problems dealing with differences of convex functions. In: Ponstein, J. (ed.) Convexity and Duality in Optimization, pp. 37–70. Springer, Berlin (1985)
13. Hiriart-Urruty, J.B., Lemaréchal, C.: Fundamentals of Convex Analysis. Grundlehren Text Editions. Springer, Berlin (2001)
14. Huber, P.J.: Robust estimation of a location parameter. Ann. Math. Stat. **35**(1), 73–101 (1964). https://doi.org/10.1214/aoms/1177703732
15. Joki, K., Bagirov, A.M., Karmitsa, N., Mäkelä, M.M., Taheri, S.: Double bundle method for finding Clarke stationary points in nonsmooth DC programming. SIAM J. Optim. **28**(2), 1892–1919 (2018). https://doi.org/10.1137/16M1115733
16. Joki, K., Bagirov, A.M., Karmitsa, N., Mäkelä, M.M.: A proximal bundle method for nonsmooth DC optimization utilizing nonconvex cutting planes. J. Glob. Optim. **68**, 501–535 (2017). https://doi.org/10.1007/s10898-016-0488-3
17. Khalaf, W., Astorino, A., D'Alessandro, P., Gaudioso, M.: A DC optimization-based clustering technique for edge detection. Optim. Lett. **11**(3), 627–640 (2017). https://doi.org/10.1007/s11590-016-1031-7
18. Kiwiel, K.C.: An aggregate subgradient method for nonsmooth convex minimization. Math. Program. **27**, 320–341 (1983). https://doi.org/10.1007/BF02591907
19. Le Thi, H.A., Pham Dinh, T.: D.C. programming approach to the multidimensional scaling problem, pp. 231–276. Springer US, Boston, MA (2001). https://doi.org/10.1007/978-1-4757-5284-7_11
20. Le Thi, H.A., Pham Dinh, T.: DC programming and DCA: thirty years of developments. Math. Program. **169**(1), 5–68 (2018). https://doi.org/10.1007/s10107-018-1235-y
21. Le Thi, H.A., Pham Dinh, T., Le Dung, M.: Numerical solution for optimization over the efficient set by D.C. optimization algorithms. Oper. Res. Lett. **19**(3), 117–128 (1996). https://doi.org/10.1016/0167-6377(96)00022-3
22. Lemaréchal, C., Mifflin, R.: Nonsmooth Optimization: Proceedings of a IIASA Workshop, March 28–April 8, 1977. Elsevier Science (2014)
23. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Math. Program. **45**, 503–528 (1989). https://doi.org/10.1007/BF01589116
24. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer Series in Operations Research and Financial Engineering, 2nd edn. Springer, New York (2006)
25. Ordin, B., Bagirov, A.M.: A heuristic algorithm for solving the minimum sum-of-squares clustering problems. J. Glob. Optim. **61**, 341–361 (2015). https://doi.org/10.1007/s10898-014-0171-5
26. Pham Dinh, T., El Bernoussi, S.: Algorithms for solving a class of nonconvex optimization problems. methods of subgradients. In: Hiriart-Urruty, J.B. (ed.) Fermat Days 85: Mathematics for Optimization,

North-Holland Mathematics Studies, vol. 129, pp. 249–271. North-Holland, New York (1986). https://doi.org/10.1016/S0304-0208(08)72402-2
27. Rockafellar, R.T.: Convex Analysis, Princeton Mathematical Series, vol. 28. Princeton University Press, Princeton (1970)
28. van Ackooij, W., Demassey, S., Javal, P., Morais, H., de Oliveira, W., Swaminathan, B.: A bundle method for nonsmooth DC programming with application to chance-constrained problems. Comput. Optim. Appl. **78**(2), 451–490 (2021). https://doi.org/10.1007/s10589-020-00241-8