J Glob Optim
DOI 10.1007/s10898-017-0541-x

Adaptive block coordinate DIRECT algorithm

Qinghua Tao[1] · **Xiaolin Huang**[2] · **Shuning Wang**[1] · **Li Li**[1]

Received: 20 December 2016 / Accepted: 23 June 2017

Abstract DIviding RECTangles (DIRECT) is an efficient and popular method in dealing with bound constrained optimization problems. However, DIRECT suffers from dimension curse, since its computational complexity soars when dimension increases. Besides, DIRECT also converges slowly when the objective function is flat. In this paper, we propose a coordinate DIRECT algorithm, which coincides with the spirits of other coordinate update algorithms. We transform the original problem into a series of sub-problems, where only one or several coordinates are selected to optimize and the rest keeps fixed. For each sub-problem, coordinately dividing the feasible domain enjoys low computational burden. Besides, we develop adaptive schemes to keep the efficiency and flexibility to tackle different functions. Specifically, we use block coordinate update, of which the size could be adaptively selected, and we also employ sequential quadratic programming to conduct the local search to efficiently accelerate the convergence even when the objective function is flat. With these techniques, the proposed algorithm achieves promising performance on both efficiency and accuracy in numerical experiments.

Keywords Global optimization · DIRECT · Coordinate update · SQP

This project is jointly supported by the National Natural Science Foundation of China (61473165, 61134012, 61603248).

✉ Li Li
li-li@mail.tsinghua.edu.cn

Qinghua Tao
taoqh14@mails.tsinghua.edu.cn

Xiaolin Huang
xiaolinhuang@sjtu.edu.cn

Shuning Wang
swang@mail.tsinghua.edu.cn

1 Department of Automation, Tsinghua University, Beijing 100084, People's Republic of China

2 Institute of Image Processing and Pattern Recognition, Shanghai Jiao Tong University, Shanghai 200240, People's Republic of China

1 Introduction

Bound constrained optimization problems have been applied in many applications, such as operation research, engineering, biology and biomathematics, chemistry, finance, etc [3,4,7,17,19,32]. Mathematically, this kind of problems can be formulated as the following,

$$\min_{x\in\Omega} f(\mathbf{x}), \tag{1}$$

where $\Omega = \{\mathbf{x} \in \mathbb{R}^n : l_i \leq x_i \leq u_i, i = 1, \ldots, n\}$, $\mathbf{l}, \mathbf{u} \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$. Notice that the convexity on $f(\mathbf{x})$ is not required, and thus global optimization techniques are needed.

There have been many efficient and important methods for global optimization, such as branch-and-bound methods, deterministic methods and some stochastic methods [14,19,23]. In recent years, DIviding RECTangles (DIRECT), based on the strategy of domain partition, has shown very good performance in dealing with problem (1), i.e., problems with bound constraints, since the basic idea of DIRECT is to do domain partitions [6,7,9,15,32]. In low dimensions, domain partition is quite effective and hence DIRECT has obtained success in some applications [4,7,17,19,32].

The existing DIRECT algorithms consider the whole domain and its computational complexity soars rapidly with dimension increasing. DIRECT loses its efficiency on both computational time and required storage space for high-dimensional problems. In fact, this phenomenon does not only appear in domain-partition-based methods, but also happens in gradient-based methods. For the latter, the combination with coordinate descent method (CDM) has become a promising alternative and has been widely applied in image reconstruction [5], dynamic programming [13], flow routing [27] and the dual of a linearly constrained [11,29], etc. A similar idea that coordinately partitions the domain is also applicable for domain-partition-based methods. Therefore, in this paper, we propose a coordinate DIRECT algorithm. To be specific, we transform the original problem into many sub-problems, where we only select one coordinate to conduct DIRECT and keep the rest fixed. Low computational burden can be expected with coordinate update, but we need to guarantee that there is efficient descent in each sub-problem for different objective functions. Hence, we introduce a switch to the combination with block coordinate descent method (BCDM) when necessary. We also employ SQP to locally find a good solution. Since SQP is efficient when the objective function performs smooth, the employment of SQP can accelerate the descent when the objective function is flat, where original DIRECT methods are reported to lose efficiency in convergence and descent [6,9,14,19]. Summarizing the above discussion, we establish the adaptive block coordinate DIRECT (ABCD) algorithm: its basic idea is coordinate update and it can adaptively choose single coordinate partition, block-coordinate partition, and local optimizer SQP. In numerical experiments, the proposed ABCD algorithm presents very promising performance in comparisons with DIRECT and other relevant algorithms.

The rest of this paper is organized as follows. Section 2 gives a review on the background and preliminaries. Section 3 establishes the proposed ABCD algorithm. A series of numerical experiments are conducted to demonstrate the overall performance of the proposed ABCD algorithm in Sect. 4. Section 5 ends this paper with conclusions.

2 Backgrounds

DIRECT was first introduced by Jones [6,15] to search the global optimum of a real valued objective function with bound constraints, i.e., problem (1). DIRECT emerges as a natural extension and generalization of the Lipschitz optimization (LO) methods proposed by Pijavskiy and Shubert [28]. For problem (1), it is customary to make assumptions on the objective function $f(\mathbf{x})$ for continuity, Lipschitz continuity, smoothness or differentiability [25]. LO requires Lipschitz continuity in searching domain, and it also demands the knowledge of the Lipschitz constant. These requirements are sometimes too demanding, which narrows the applications of LO. Different from LO, DIRECT needs no knowledge of Lipschitz constant, and DIRECT even does not require the objective function to be Lipschitz continuous [15]. It only requires the objective function to be continuous in the neighborhood of the global optimum, which enables it to deal with a wider range of problems.

Considering problem (1), DIRECT starts from unifying the searching domain Ω into a unit hyper-cube $\bar{\Omega}$. That is

$$\{\bar{\Omega} = \mathbf{x} \in \mathbb{R}^n : 0 \leq x_i \leq 1, i = 1, \ldots, n\}. \tag{2}$$

DIRECT optimizes the problem over the normalized hyper-rectangle, of which the center is denoted as $\mathbf{c}_1$.

In the initialization, DIRECT samples the center $f(\mathbf{c}_1)$. DIRECT evaluates the objective function $f(\mathbf{x})$ at points $\mathbf{c}_1 \pm \delta\mathbf{e}_i, i = 1, \ldots, n$, where δ is one third of the side length of the hyper-cube and $\mathbf{e}_i$ is the unit vector of ith dimension, i.e., a vector with a one in the ith position and zeros elsewhere.

We adopt the same strategy used in the original DIRECT to do the sampling and division, such that DIRECT always splits the selected hyper-rectangle along its longest sides to guarantee a shrinkage in each side, and then samples the centers of the newly obtained hyper-rectangles. Denote I as the set of dimensions with the longest side length in the hyper-rectangle with center $\mathbf{c}$. An easy way to divide the hyper-cube is arbitrarily selecting one dimension from set I and splitting it. However, arbitrariness is not efficient and DIRECT heuristically uses the following criterion:

$$w_i = \min(f(\mathbf{c} + \delta\mathbf{e}_i), f(\mathbf{c} - \delta\mathbf{e}_i)), i \in I, \tag{3}$$

and divides the hyper-rectangle starting with the smallest w_i into thirds. Then $\mathbf{c} \pm \delta_i$ is the center of new hyper-rectangle.

Next, DIRECT proceeds to the identification of potentially optimal hyper-rectangles (POHs), which have potentials to obtain better solutions, even the global optimum [15]. In DIRECT, a constant $\widetilde{K}$ is introduced to determine POHs. The basic idea of DIRECT is to explore better solutions among all the POHs. More precisely, DIRECT samples all "potentially optimal" hyper-rectangles as defined below, and this pattern is repeated in each iteration [15].

Definition 1 Let ε be a positive constant and $f_{\min}$ be the current minimum. Hyper-rectangle j is said to be potentially optimal if there exists a rate-of-change constant $\widetilde{K}$ such that

$$\begin{aligned} f(\mathbf{c}_j) - \widetilde{K}\mathbf{d}_j &\leq f(\mathbf{c}_i) - \widetilde{K}\mathbf{d}_i, \ \forall i \\ f(\mathbf{c}_j) - \widetilde{K}\mathbf{d}_j &\leq f_{\min} - \varepsilon|f_{\min}|, \end{aligned} \tag{4}$$

where $\mathbf{d}_j$ is a measure for hyper-rectangle j. Researchers commonly choose the distance from center $\mathbf{c}_j$ to its vertices as the measure [8,15]. The first condition in the definition forces the POHs to obtain the lowest objective values among the rectangles which share the same measure $\mathbf{d}_j$. The second condition insists that the POHs, based on the rate-of-change constant $\widetilde{K}$, exceed the current best solution by a nontrivial amount [15]. Parameter ε is usually chosen as 10^{-4} by researchers, since the experimental data show that ε has a negligible effect on the calculation when it is chosen within 10^{-2} to 10^{-7} [8]. Once a hyper-rectangle is identified to be the POH, then it needs to be divided into smaller ones. The dividing procedure is restrained to be conducted only along the dimensions with the longest side-length, which ensures a shrinkage in every dimension [8].

In DIRECT, the progress of the optimization is governed only by the evaluations of the objective function. It iteratively divides the longest side of the selected POHs and obtains several sub-rectangles. There are two key processes required in every iteration of optimization. The first key process is to identify the POHs, which are identified to potentially contain good, un-sampled points. The second one is to sample and divide POHs, which shrinks the location of the global optimum into smaller space. The repetition of the two key processes can approach the global optimum and confine the global optimum within a very small rectangle. Therefore, DIRECT iteratively approaches better solutions as iterations go on. Without the limit of iterations and function evaluations, DIRECT is guaranteed with global convergence and it can restrict the global optimum to a small rectangle with any given accuracy. In summary, the formal description of DIRECT algorithm is presented in Algorithms 1 and 2 [15].

Algorithm 1: Sampling and Dividing Algorithm

- Identify the set I of dimensions with the longest side length d_s of the selected hyper-rectangle, and set $\delta = d_s/3$.
- Sample $f(x)$ at $\mathbf{c} \pm \delta\mathbf{e}_i$, $i \in I$, where $\mathbf{c}$ is the center point of the hyper-rectangle.
- Divide the hyper-rectangle into thirds alongside the dimensions in I, starting with the dimension with the smallest w_i, where $w_i = \min f(\mathbf{c} \pm \delta\mathbf{e}_i)$ and $i \in I$.

Algorithm 2: DIRECT Algorithm

```
Input: $f, \varepsilon, N_1, N_2$
Output: $f_{\min}, \mathbf{x}_{\min}$
Normalize the search space to be the unit hypercube with center point $\mathbf{c}_1$.
Evaluate $f(\mathbf{c}_i)$, $f_{\min} = f(\mathbf{c}_1)$.
Set the number of iterations $t = 0$, and function evaluations $m = 1$.
while $t < N_1$ and $m < N_2$ do
    Identify the set $S$ of potentially optimal hyper-rectangles.
    while $S \neq \emptyset$ do
        Take $j \in S$.
        Sample new points, evaluate $f$ at the new points and divide the hyper-rectangles with
        Algorithm 1.
        Update $f_{\min}, x_{\min}$ and $m = m + \triangle m$, where $\triangle m$ is the number of new points sampled.
        Set $S = S - \{j\}$.
    end
    $t = t + 1$.
end
```

Unfortunately, the efficiency of DIRECT drops rapidly with dimension increasing, since working with all the coordinates of an optimization problem at each iteration may be inconvenient and the required computation is soaring with dimensions increasing. From [15], we know that, after r divisions, the rectangles have $p = \mathrm{mod}(r, n)$ sides of length $3^{-(k+1)}$ and $n - p$ sides of length 3^{-k}, where $k = (r - p)/n$. Thus, the center-to-vertex distance of the rectangle is given by

$$d = 0.5\sqrt{[3^{-2(k+1)}p + 3^{-2k}(n - p)]}. \tag{5}$$

From Eq. (5), it is easy to obtain the smallest required division r of a rectangle with the center-to-vertex distance d. In DIRECT, a rectangle with the center-to-vertex distance d is required to undergo at least $r = n\log_3(\sqrt{n-8/9p}/2d) + p$ divisions. Since new rectangles are formed by dividing existing ones into thirds on the longest side, every division is accompanied with two function evaluations. Thus, in DIRECT, any rectangle with the center-to-vertex distance d is accompanied with at least $N_0 = \lceil 2(n\log_3(\sqrt{n-8/9p}/2d) + p)\rceil$ function evaluations. When $n = 2$ and $\varepsilon = 10^{-4}$, $N_0 \approx 33$. When $n = 12$ and $\varepsilon = 10^{-4}$, $N_0 \approx 214$. This is computation only for one rectangle with a certain size of center-to-vertex distance d. In fact, numerous hyper-rectangles with various sizes of d are generated as iterations go on, since each function evaluation comes with a new hyper-rectangle. Therefore, the computational complexity of the DIRECT increases rapidly with dimension n increasing.

In high dimensions, DIRECT suffers from dimension curse, which is a common defect of algorithms based on strategies of domain partition and function evaluation. Besides the effect of dimension, DIRECT also converges slowly around smooth area, where the objective function is very flat. In such cases, even DIRECT gets close to the basin of global optimum, the smooth neighbor around the optimum may also hamper it to achieve higher accuracy. Along with the spirits of DIRECT, multilevel coordinate search (MSC) was proposed with the strategy of domain partition and function evaluation to tackle problem (1) [14]. MCS partitions the domain into small boxes with more irregular splitting. In contract to DIRECT, MCS simplifies the division procedure. To speed up the convergence, MCS introduces a local enhancement to start local searches. Although MCS improves computing speed compared with DIRECT, it lacks further exploration when the local searches are finished. Thus the accuracy of MCS is undesirable if the local search fails to bring the solution to the sufficiently small neighborhood of the global optimum with given accuracy. Recently, simultaneous optimistic optimization (SOO) was introduced by Preux et al. to solve problem (1) [25,30]. Similar to MCS, SOO is also closely related to DIRECT algorithm. It works by partitioning the domain into sub-parts, namely cells. SOO chooses to split the cell with the smallest value at its center. As its name suggests, the basic idea of SOO is to optimally choose the sub-domain whose objective value is the lowest in the corresponding depth. The basic idea of SOO is very simple, but SOO is a global optimization algorithm which has a finite-time performance under some weak assumptions on the objective function [22]. However, SOO still suffers from dimension curse. Moreover, SOO is hard to approach the global optimum when the surface of the objective function is complicated or the problem has many local optima. DIRECT, SOO and MCS all pose global convergence theoretically, and their global convergence comes when when it is allowed to have enough computational time and splitting depth [10,15,19].

3 Adaptive block coordinate DIRECT algorithm

DIRECT has shown to be efficient in low dimensions [1,10,15,16,19], but its computation complexity soars with dimension increasing, and working with all the coordinates at each iteration is inconvenient. Inspired by CDM, degenerating the original problems into many one-dimensional sub-problems is expected to bring fast speed. In this paper, we present the ABCD algorithm, whose basic idea is conducting the optimization coordinately to maintain the efficiency of DIRECT in low dimensions, so as to improve the speed. Instead of considering the optimization on all coordinates, we transform the original problem into a series of sub-problems, where we select only one coordinate to optimize with one-dimensional DIRECT. However, the coordinate update still possibly brings slow convergence, then we modify it to be adaptive. In such cases, we employ SQP to conduct a local search and then switch to the block coordinate update. The local optimizer helps to accelerate the convergence and explore further improvements with larger size of chosen coordinates, which brings more flexibilities.

3.1 DIviding RECTangles on one coordinate

DIRECT processes one-dimensional functions very fast. In order to maintain DIRECT's efficiency in low dimensions and tackle high-dimensional problems as well, we introduce a coordinate update DIRECT algorithm, whose main idea is optimizing one single coordinate each time with the rest fixed to constants. Coordinate update has been widely applied in the optimization with convex objective functions and proved to efficiently reduce computational complexity [31]. This basic pattern is to decrease problem dimension from n to 1, which formulates the original problem into many one-dimensional sub-problems. For high-dimensional problems, soaring computing costs lead to slow convergence. Usually, DIRECT exhausts its maximal iterations or maximal dividing depth before reaching the given accuracy. By contrast, the degeneration to low dimensions sharply relieves computational burden and also enables CPU memory to get released at the end of every sub-problem.

Instead of repeating sampling and dividing on the whole domain, we conduct DIRECT on just one coordinate, then the solution can be quickly restricted to sufficiently small space, which enables solve high-dimensional problems with fast speed. Even in the cases where DIRECT can reach the given accuracy within desirable time consumption, the coordinate update helps DIRECT improve its speed to a higher level. We first generate a starting point $\mathbf{x}_0$ in feasible domain Ω, and optimize the $i = 1$ coordinate with the rest fixed to $\mathbf{x}_0(j)$, $j = 1, \ldots, n$, $j \neq i$. When the one-dimensional DIRECT converges on coordinate $\mathbf{x}(i)$, we sequentially select the adjacent coordinate $i = i + 1, i < n$ or $i = 1, i = n$ to continue the next sub-problem. Algorithm stops when the objective function $f(\mathbf{x})$ no longer decreases among sub-problems or the optimum is restricted to a sufficiently small rectangle. This is the basic and simplest form of the proposed ABCD algorithm.

Take Levy function for example. Levy function belongs to Hedar test set which is regarded as a benchmark for global optimization [19]. DIRECT solves Levy function from 6 dimension to 18 dimension with given accuracy $\varepsilon = 10^{-4}$, which is defined as the difference between the objective value obtained by the test algorithm and the true global minimum. The results are shown in Fig. 1. DIRECT takes more than 6 s to reach the given accuracy $\varepsilon = 10^{-4}$ for Levy function from 6 dimensions to 18 dimensions. With coordinate update, the running time for Levy function decreases to less than 0.2 s. Thus, the coordinate DIRECT not only maintains to successfully solve the problem with given accuracy, but also greatly increases the efficiency.

Besides computational burden in high dimensions, DIRECT suffers slow convergence in a smooth area, which also affects the speed and accuracy. But when we update the problem coordinately, this problem can be overcome. To demonstrate this characteristic, we consider the function $f(x) = 0.1(x - 0.4)^6$, which is very flat in interval $[-0.6, 1.4]$. We apply one-dimensional DIRECT to function $f(x) = 0.1(x - 0.4)^6$ subjected to $x \in [-1, 1]$. The results are shown in Fig. 2.

It can be seen from Fig. 2 that the one-dimensional DIRECT only takes five iterations to achieve accuracy ε in the flat surface of function $f(x) = 0.1(x - 0.4)^6$. Even the degenerated sub-problem is very flat, the one-dimensional DIRECT shows to efficiently solve the sub-problems. Besides dimension degeneration, the fast convergence in flat objective function is another merit of coordinate update.

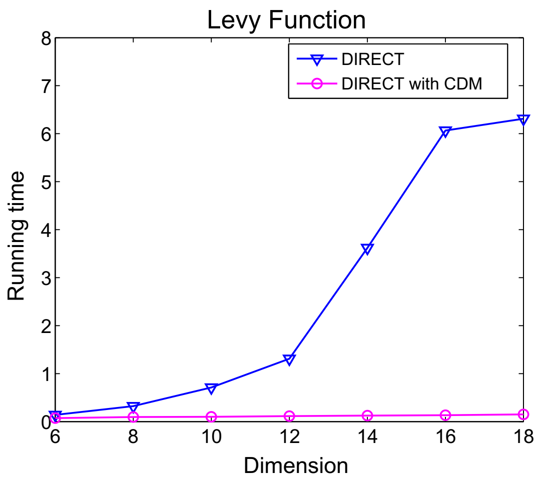

Fig. 1 Running time (s) of Levy function in dimension $n = 6, 8, \ldots, 18$ for reaching accuracy $\varepsilon = 1e - 4$. 'DIRECT' means the results of directly applying DIRECT to Levy function. 'DIRECT with CDM' represents the results of applying DIRECT with coordinate update to Levy function

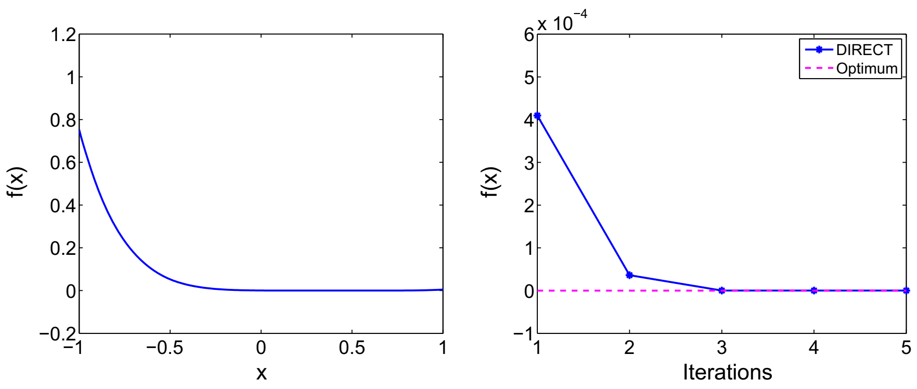

Fig. 2 Simulation results of function $f(x) = 0.1(x - 0.4)^6$ when applying DIRECT. **a** The curve of function $f(x) = 0.1(x - 0.4)^6$ in interval $[-1, 1]$. **b** Illustrates the results of DIRECT in every sub-problem. 'Optimum' represents the global minimum of function $f(x) = 0.1(x - 0.4)^6$

The original problem may possibly reach a saturation in descent with coordinate DIRECT. Thus the degeneration to one dimension possibly leads to a local optimum. Moreover, even if the algorithm runs very fast with the selected coordinate in each sub-problem, the descent of the objective in each sub-problem can still be quite small. In such cases, if we continue selecting only one coordinate to optimize, it is possible that the original objective function $f(\mathbf{x})$ shares very sparse contours and each sub-problem gets stuck in slow convergence. Although each sub-problem runs very fast in convergence, it requires tremendous numbers of sub-problems to reach the global optimum. To conquer these drawbacks, Sects. 3.2 and 3.3 are established.

3.2 DIviding RECTangles on block coordinates

The strategy of coordinate update can be extended. In every sub-problem, we can adaptivley select $1 < m \leq n$ coordinates to optimize and keep the rest fixed to constants, which resonates with the core idea of BCDM [12,21]. When $m = n$, the proposed algorithm degenerates to DIRECT. Thus, the proposed algorithm can be regarded as an extension and generalization of DIRECT algorithm from this view.

Different from Sect. 3.1, we partition the coordinates into several blocks. In each sub-problem, we focus on updating a single block only, and keep the remaining blocks fixed. Sequentially optimizing each coordinate with DIRECT algorithm is the basic and the simplest form of the proposed algorithm, where each coordinate gets the same chance to be optimized. However, as mentioned in the end of Sect. 3.1, when the objective function $f(\mathbf{x})$ fails to achieve further descent in sub-problems and even reaches a local optimum, adaptively selecting more coordinates to do the optimization may possibly leads further descent and more efficiency, since the objective function $f(\mathbf{x})$ may help bypass the local optimum and enjoy sharper descent in each sub-problem with larger number of chosen coordinates. Therefore, BCD update is incorporated, where we choose $1 < m \leq n$ coordinates to repeat the m-dimensional DIRECT until the stopping criterion is satisfied.

Take Rosenbrock function to illustrate this statement, where the way of choosing the block of coordinates and the size of the block can be varied. Rosenbrock is regarded as a benchmark function in global optimization, and it is shown to be difficult to tackle. The Rosenbrock function that we optimize comes from Hedar test set [19]. There are two popular variants for Rosenbrock function, and we refer it as variant A and variant B [18]. Variant A has only one stationary point for all dimensions, since the Hessian matrix is positive definite. Rosenbrock function in Hedar test set is the variant B. In variant B, it is found to have two minima with dimension $4 < n < 30$ and numerous stationary points. The selected Rosenbrock function is unimodal, and the global minimum lies in a narrow and parabolic valley. However, even though this valley is easy to find, convergence to the minimum is difficult [18,24]. We try to solve a 12-dimensional Rosenbrock function with both CDM update and BCD update. We first empirically choose only two coordinates in each sub-problem, i.e., $m = 2$. Similar to Sect. 3.1, We generate a feasible starting point $\mathbf{x}_0$, and optimize the $i_1 = 1, i_2 = 2$ coordinates with the rest fixed to $\mathbf{x}_0(j)$, $j = 1, \ldots, n$, $j \neq i_1, i_2$. When the two-dimensional DIRECT converges on coordinates $\mathbf{x}(i_1, i_2)$, we sequentially select the adjacent coordinate $i_1 = i_1 + 1, i_2 = i_2 + 1$, when $i_1, i_2 < n$, or we set $i_1, i_2 = 1$, when $i_1, i_2 = n$. We terminate the algorithm in different running time, which ranges from 2 to 6 s with the interval of 0.5 s. Figure 3 shows the CPU running time against the absolute error.

From Fig. 3, we can see that sequentially conducting DIRECT on one coordinate leads to a local minimum within 2 s. When algorithm runs longer, the coordinate update fails to get out of the current local minimum. However, sequentially conducting DIRECT on two coordinates bypasses the local minimum which the coordinate update gets stuck in. With more computing time, the block coordinate update gradually decreases the error. When the coordinate update fails to achieve higher accuracy and converges inefficiently, we can switch to block forms, and then we randomly select $m \geq 2$ coordinates to conduct DIRECT in each sub-problem. Therefore, the proposed algorithm incorporates BCDM, which is designed to check if there is any descent with different nmber of chosen coordinates, which possibly helps check further improvements to approach a better solution, even the global optimum.

3.3 Local optimizer

Although one-dimensional DIRECT works well in fast speed, it may show slow convergence and get stuck in local optimum in sub-problems. There are two possible explanations for this phenomenon. The first one is that one-dimensional DIRECT fails to bring further improvement. The second is that the original objective function $f(\mathbf{x})$ arrives in a smooth area, where

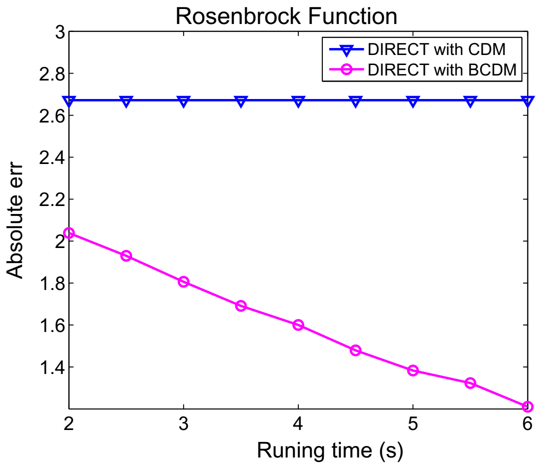

Fig. 3 Simulation results of optimizing Rosenbrock function. 'DIRECT with CDM' represents the absolute error when sequentially selecting one coordinate to conduct one-dimensional in each sub-problem. 'DIRECT with BCDM' denotes the absolute error when sequentially choosing two coordinate to optimize in each sub-problem

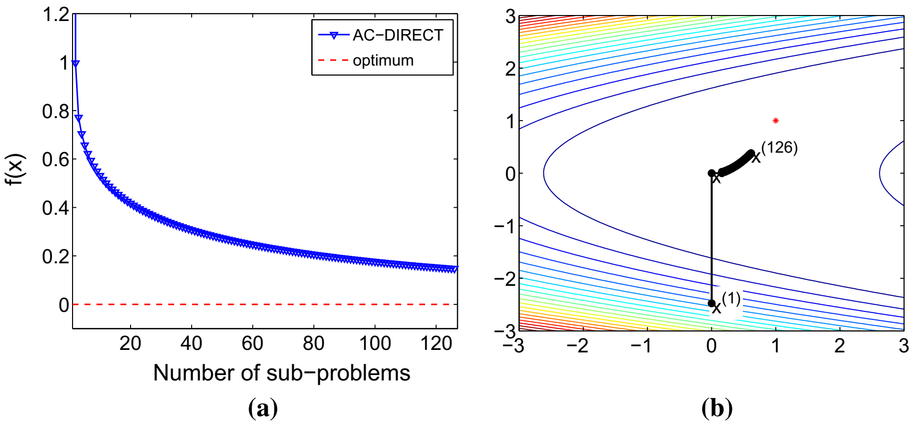

Fig. 4 Simulation results of Rosenbrock function when sequentially applying one-dimensional DIRECT in each sub-problem. **a** Illustrates descent of $f(x)$ in every sub-problem. **b** Depicts the searching trace on contour map, where the *red asterisk* is denoted as the global optimal. (Color figure online)

$f(\mathbf{x})$ is very flat. For the first condition, as mentioned in Sect. 3.2, the switch to block coordinate update provides possibilities to conquer it. However, if it is the latter, the switch to block coordinate update may still fail in achieving sharp descent in each sub-problem. Take Rosenbrock function for example. We tackle the two-dimensional Rosenbrock function with coordinate update, where the contour map can be drawn visibly. Below, Fig. 4 depicts the searching trace of the coordinate DIRECT.

Figure 4 shows that one-dimensional DIRECT brings smaller descent to the objective function $f(\mathbf{x})$ as sub-problems go on. The contours around the global optimum are very sparse, and our method only moves forward a small step to get closer to the global optimum in each sub-problem. Thus, tremendous numbers of sub-problems are required to approach the global optimum. Although each one-dimensional sub-problem runs efficiently with flat objective function, the huge number of sub-problems also brings inefficiency in solving the original problem. As mentioned in Sect. 3.2, in such cases, when one-dimensional DIRECT does not make any progress, we switch to block coordinate update. However, the results are

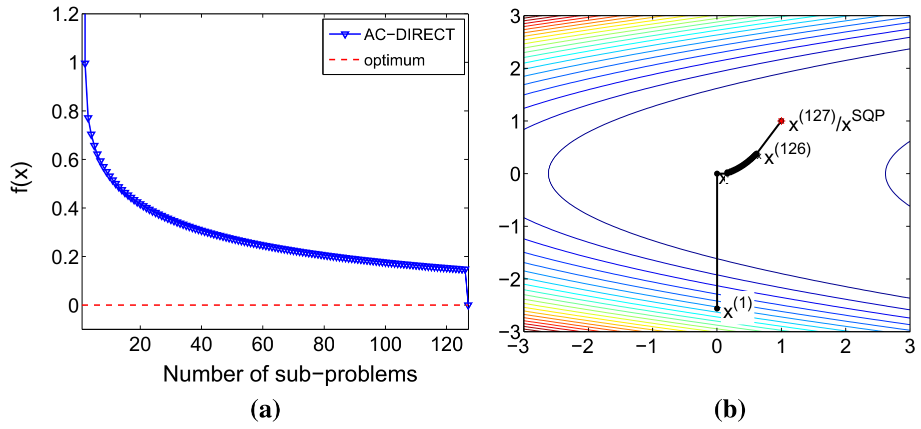

Fig. 5 Based on Fig. 4, simulation results of Rosenbrock function with SQP as the local optimizer

still unsatisfactory. Therefore, we turn to employing a local optimizer to tackle the problem when the contour is sparse. The employment of local optimizer can potentially help conquer slow convergence and escape from the local optimum around smooth area. In this paper, SQP is introduced to do the local search. SQP solves a sequence of optimization sub-problems, each of which optimizes a quadratic model of the objective function. If the problem is unconstrained, then the method reduces to Newton's method for finding a point where the gradient of the objective vanishes. SQP can be implemented with fast solving speed in Matlab, thus the incorporation with it as a local optimizer brings little expense to the speed of the proposed algorithm. For the experiment presented in Fig. 4, we apply SQP to Rosenbrock function in the 127th sub-problem. The results are shown in Fig. 5.

Figure 5 demonstrates that the employment of SQP makes the algorithm quickly approach the global optimum, which presents the efficacy of SQP around smooth area. Integrating the ideas in Sects. 3.1 and 3.2, we establish the ABCD algorithm, whose details are presented in the following subsection.

3.4 Algorithm implementation

In the basic form of the proposed ABCD algorithm, one-dimensional update comes first, then SQP is incorporated to conquer slow convergence and flatness of the objective, and finally two-dimensional update is introduced to explore further improvement. However, there are many adjustments which can be adaptively done to better tackle different problems. That is the reason for why we name it as an adaptive algorithm. In fact, the algorithm starts with applying DIRECT with blocks of size m_1, usually $m_1 = 1$, which is the basic form. While, in another round of DIRECT, the size of the blocks m_2 is usually selected as $m_2 = 2$ but not restricted to $m_2 = 2$. Still, instead of cyclical sequence, the coordinate choosing can be in a random manner. Moreover, the starting point $\mathbf{x}_0$ and the inserting of local optimizer SQP are also adjustable, where the index $flag$ is introduced. These potential modifications can be regarded as the flexibilities of the proposed algorithm. Due to these flexibilities, the proposed ABCD algorithm is capable of tackling different problems to obtain better solutions. In summary, the framework of the proposed ABCD algorithm is shown in Algorithm 3.

(1) Starting point

Algorithm 3: Adaptive Block Coordinate DIRECT Algorithm

```
Input: Objective function $f(\mathbf{x})$, and pre-settings $t_{\max}, Iter_{\max}, \varepsilon, \varepsilon_0, \varepsilon_1, \varepsilon_2, T_0, T_1, T_2, m_1, m_2, flag$.
Output: The achieved optimal point $\mathbf{x}_{\min}$ and optimal object value $f_{\min}$.
• Select a feasible starting point $\mathbf{x}_0$. Denote $\hat{\mathbf{x}}^* = \mathbf{x}_0$ as the current optimal point and $\hat{f}^* = f(\hat{\mathbf{x}}^*)$ as
the current optimal value.
• Set the current computer running time $t = 0(s)$ and the current number of function evaluations
$Iter = 0$.
if $flag = 0$ & $\hat{f}^*$ does not get within $\varepsilon$ of the known optimum $f^*$ & $t \le t_{\max}$ & $Iter \le Iter_{\max}$ then
    Conduct local optimizer SQP starting from point $\hat{\mathbf{x}}^*$ with default settings in Matlab toolbox
    "fmincon".
    Update $\hat{\mathbf{x}}^*$, $\hat{f}^*$, $t$ and $Iter$.
end
while $\hat{f}^*$ does not get within $\varepsilon$ of the known optimum $f^*$ & $\hat{f}^*$ does not fail to decrease $\varepsilon_1$ for $T_1$
times in succession & $t \le t_{\max}$ & $Iter \le Iter_{\max}$ do
    Select a subset $I_1$ from $\{1, \dots, n\}$, where $I_1$ contains $m_1 \le n$ elements.
    Choose coordinates $\mathbf{x}(I_1)$ to optimize with DIRECT, and keep the rest fixed to $\hat{\mathbf{x}}^*(i), i \notin I_1$.
    When objective value fails to decrease $\varepsilon_0$ for $T_0$ times in a row, terminate the $m_1$-dimensional
    DIRECT and record the optimal point $\hat{\mathbf{y}} \in \mathbb{R}^{m_1}$.
    Update $\hat{\mathbf{x}}^*(I_1) = \hat{\mathbf{y}}$, $\hat{f}^* = f(\hat{\mathbf{x}}^*)$, $t$ and $Iter$.
end
if $flag = 1$ & $\hat{f}^*$ does not get within $\varepsilon$ of the known optimum $f^*$ & $t \le t_{\max}$ & $Iter \le Iter_{\max}$ then
    Conduct local optimizer SQP starting from point $\hat{\mathbf{x}}^*$ with default settings in Matlab toolbox
    "fmincon".
    Update $\hat{\mathbf{x}}^*$, $\hat{f}^*$, $t$ and $Iter$.
end
while $\hat{f}^*$ does not get within $\varepsilon$ of the known optimum $f^*$ & $\hat{f}^*$ does not fail to decrease $\varepsilon_2$ for $T_2$ times
in succession & $t \le t_{\max}$ & $Iter \le Iter_{\max}$ do
    Select a subset $I_2$ from $\{1, \dots, n\}$, where $I_2$ contains $m_2 \le n$ elements.
    Choose coordinates $\mathbf{x}(I_2)$ to optimize with DIRECT, and keep the rest fixed to $\hat{\mathbf{x}}^*(i), i \notin I_2$.
    When objective value fails to decrease $\varepsilon_0$ for $T_0$ times in a row, terminate the $m_2$-dimensional
    DIRECT and record the optimal point $\hat{\mathbf{y}} \in \mathbb{R}^{m_2}$.
    Update $\hat{\mathbf{x}}^*(I_2) = \hat{\mathbf{y}}$, $\hat{f}^* = f(\hat{\mathbf{x}}^*)$, $t$ and $Iter$.
end
• $\mathbf{x}_{\min} = \hat{\mathbf{x}}^*$, $f_{\min} = \hat{f}^*$
```

The proposed algorithm initially starts from a point $\mathbf{x}_0$. A good starting point may help accelerate the convergence of $f(\mathbf{x})$ and approach a better solution within less iterations. The starting point $\mathbf{x}_0$ can be generated anywhere in the feasible domain. Inspired by SOO, we can introduce the optimistic choosing mechanism to select the starting point. SOO positively chooses to divide the sub-domain with the lowest objective value. Similarly, we generate a set of points and select the one with the lowest objective value.

In implementation, we randomly generate q points $\hat{\mathbf{x}}_i$, $i = 1, \dots, q$ in the feasible domain. Next, we choose point $\hat{\mathbf{x}}_j$ with the lowest objective value as the starting point, i.e., $j = \{i \mid \min\{f(\hat{\mathbf{x}}_i), i = 1, \dots, q\}\}$. Since $\hat{\mathbf{x}}_j$ is the best point among the sampled points, we positively believe that $\hat{\mathbf{x}}_j$ possibly lies in the area which is closer to the global optimum. The selection of starting points only requires q function evaluations, whose time expenses are trivial. In fact, the q initial points can be generated in a varied ways. Other methods of selecting the starting point are also acceptable if they take little cost in computing and potentially bring improvements to the next optimization procedures.

(2) Number of chosen coordinates

In each sub-problem, the proposed ABCD algorithm focuses on updating a single coordinate or a block of coordinates, while keeping the rest fixed. First, we select a single coordinate

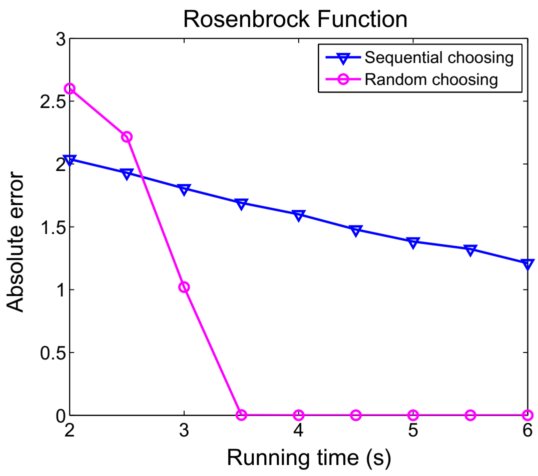

Fig. 6 Simulation results of Rosenbrock function with ABCD algorithm. 'Sequential choosing' and 'Random choosing' represent the results of sequentially selecting two coordinates and randomly choosing two coordinates to conduct DIRECT

to optimize. One-dimensional DIRECT algorithm runs very fast in each sub-problem, but it requires more sub-problems. Meanwhile, for some problems, one-dimensional optimization may ignore the potential of further improvement among blocks of coordinates. Therefore, in Algorithm 3, the number of chosen coordinates can be adjusted, and it is unnecessarily kept the unchanged as optimization goes on, thus we introduce a switch. This switch is designed to check further descent of the objective function $f(\mathbf{x})$ with different number of chosen coordinates. Meanwhile the switch condition ε_1, T_1 is also adjustable. In this paper, we firstly set $m_1 = 1$ to choose only one coordinate to optimize. If the speed appears undesirable, we switch to the block coordinate update with $m_2 = 2$. In fact, m_1 and m_2 are not restricted to 1 and 2, and it can be selected as any integer $m_1, m_2 \leq n$ if necessary.

(3) Coordinates choosing

Once we decide the number of chosen coordinates in each sub-problem, the way of choosing these coordinates also affects the performance. Sequentially optimizing the coordinates is considered as a simple way, where each coordinate has the same chance to get explored and optimized. This mode presets a certain sequence of coordinates to get optimized, and it equals repeating the optimization on the same coordinates after a round of optimizing all the coordinates. For some problems, sequential optimization may bring slow speed, since the objective function $f(\mathbf{x})$ can happen to enjoy slow convergence under the current mode of coordinates choosing. Thus, randomly choosing coordinates to optimize may help get out of the current mode to explore further improvements. The efficacy of randomized BCDM for convex problems are illustrated in [2,26]. For optimization problems with convex objective function, reference [26] proves that the existence of the minimal number of iteration for any given mathematical expectation of the objective function. That is to say, for convex problems, desirable results can be expected with enough iterations. Although the minimal number of iterations can not be specifically deduced for problem (1) when the convexity is unguaranteed, the spirits of randomized BCDM are also applicable in the proposed algorithm. Based on the test in Fig. 3, instead of sequentially selecting two coordinates to optimize, we turn to randomly choosing two coordinates to get optimized in each sub-problem.

Figure 6 illustrates that randomly choosing two coordinates to optimize brings sharper descent compared with sequential choosing. In this paper, we firstly sequentially select one coordinate to optimize, which enables every coordinates share the same chance to update.

When we begin the switch, we turn to the mode of random choosing to check further descent of objective function $f(\mathbf{x})$.

(4) Local optimizer

In Algorithm 3, the local optimizer SQP can be inserted either after the coordinate update or at the beginning of the algorithm. For the former, when the coordinate update fails to achieve further descent in each sub-problem, we introduce SQP to accelerate the speed around smooth area. In fact, the switch condition m_1 and T_1 can also be adjusted. Larger values of m_1 allow SQP to start the local search at a earlier stage in the proposed algorithm, which means less tolerance for cost in CDM. While smaller values of m_1 allow more running time spent on CDM. A fixed setting of m_1, T_1 may not be able to achieve fairly satisfactory solutions for all test functions, thus the inserting of the local optimizer is not necessary after the CDM and before the BCDM. Sometimes, SQP performs very well when it gets applied to the test function directly. In such case, SQP can be placed SQP at the beginning of the algorithm, which can be also regarded a degeneration of the proposed ABCD algorithm. Hence, the index $flag$ is introduced in Algorithm 3 to determine when to conduct SQP.

4 Numerical experiments

In this section, we conduct numerical experiments to evaluate the proposed algorithm, especially for problems in high dimensions. A series of comparisons among the proposed ABCD algorithm, DIRECT, SOO and MCS are presented. The toolbox of DIRECT, given by Jones, lacks efficiency in the implementation with Matlab, while Björkman discussed a more efficient implementation of DIRECT in [1]. To distinguish it from the toolbox of Jones, the implementation of DIRECT in [1] is called as glbSolve. In glbSolve, the core ideas of identifying the POHs and domain partition still resemble that of DIRECT. For storing the information of each rectangle, instead of using the tree structure in Jones' toolbox, glbSolve uses a straightforward matrix/index-vector technique. It is proved in [1] that glbSolve outperforms Jones' DIRECT toolbox in implementation. Thus, in order not to put DIRECT at a disadvantage, we adopt glbSolve toolbox to implement DIRECT algorithm in numerical experiments.

Previous researches show that DIRECT can efficiently solve low-dimensional problems, hence we first apply the proposed algorithm to Jones test set, which is regarded as benchmarks for comparing global searching methods in low dimensions. In Sect. 4.1, we tentatively show the efficacy of coordinate update in DIRECT, and demonstrate that our algorithm maintains superiority in low dimensions. Next, in Sect. 4.2, we extend Jones test set to Hedar test set to evaluate the performance on more test functions, whose dimensions are adjustable to higher dimensions. Section 4.2 is designed to compare the proposed algorithm with more relevant algorithms, when tackling different functions in different dimensions. To avoid randomness, the experiments are repeated 10 times for each test function, thus figures in this section are the average values. All the experiments are run on the Windows 7 platform of Matlab R2013a in Core(TM) i7-4790 3.60 GHz, 16 GB RAM.

4.1 Test on the Jones test set

The Jones test set is regarded as a benchmark for comparing global searching methods in low dimensions [6,7,9,14,15,19]. In Jones test set, there are nine test functions whose details

Table 1 Functions of Jones test set

Function	Abbreviation	Dimension	Local optimum	Global optimum
Shekel 5	S5	4	5	1
Shekel 7	S7	4	7	1
Shekel 10	S10	4	10	1
Hartman 3	H3	3	4	1
Hartman 6	H6	6	4	1
Branin RCOS	BR	2	3	3
Goldstein and Price	GP	2	4	1
Six-hump camel	C6	2	6	2
2D Shubert	SHU	2	760	18

Table 2 Running time (s) for reaching accuracy ε_1 and ε_2, where $\varepsilon_1 = 10^{-4}$ and $\varepsilon_2 = 10^{-6}$

	S5	S7	S10	H3	H6	BR	GP	C6	SHU
DIRECT: ε_1	0.04	0.04	0.04	0.04	0.06	0.03	0.03	0.03	Fail(t)
ABCD(1): ε_1	0.06	0.05	0.05	0.04	0.07	0.03	0.05	0.03	0.03
DIRECT: ε_2	0.07	0.06	0.06	0.08	0.10	0.03	0.04	0.03	Fail(t)
ABCD(1): ε_2	0.05	0.06	0.06	0.04	0.08	0.06	0.04	0.03	0.04

'ABCD(1)' means $m_1 = 1$ in Algorithm 3, where the procedures of the local optimizer SQP and the switch to other size of chosen coordinates are ignored

are shown in Table 1. These test functions have multiple local optima, and some even have multiple global optima.

We compare with the original DIRECT to show the efficacy of the coordinate update in the proposed algorithm even for low-dimensional functions. Similarly, we perform the test as done in [15]. Each algorithm is terminated when

$$\hat{f}^* - f^* \leq \varepsilon. \tag{6}$$

where ε is the given accuracy. Since the test functions are low-dimensional ($n \in [2, 6]$), sequentially optimizing the coordinates one by one is capable of obtaining desirable results. Thus, in Algorithm 3, we set $flag = 0$ and $m_1 = 1$, where the local optimizer as well as the block coordinate update are ignored. Specifically, the number q of the potential starting points is determined by the bounds and the dimensions, i.e., $q = \lceil \max\{l_{\max}, u_{\max}\} \rceil * n$, where $l_{\max}$ and $u_{\max}$ are the elements with the biggest absolute value in bounds $\mathbf{L} \in \mathbb{R}^n$ and $\mathbf{U} \in \mathbb{R}^n$ respectively. The results are summarized in Table 2. If algorithm fails to stop within 10 s CPU running time, we say that it fails in solving the problem within acceptable running time and mark 'Fail(t)' in Table 2.

Table 2 illustrates that the proposed algorithm is comparable to DIRECT, when the accuracy is selected as $\varepsilon_1 = 10^{-4}$. The proposed ABCD algorithm iteratively selects one coordinate to optimize, and the test function self is low-dimensional. Thus, it makes sense that the proposed algorithm may not be distinctively superior to DIRECT in low dimensions. However, when accuracy is restricted to a higher level, say $\varepsilon_2 = 10^{-6}$, DIRECT converges slower and slower in the fairly close vicinity of the global optimum. While, the proposed algorithm converges faster that DIRECT for $\varepsilon_2 = 10^{-6}$.

It is worth mentioning that, for Shubert function, DIRECT fails both in ε_1 and ε_2. Since there are 760 local optima and 18 global optima, DIRECT gets trapped in a local optimum and unable to get out of it, thus it fails to reach the given accuracy either for ε_1 or ε_2. While, the proposed algorithm successfully bypasses the local optima and quickly converges to the global optimum. The proposed ABCD algorithm may cannot guarantee the global optima for each problem within limited budgets. But the results in Table 2 illustrates that it can speed up the convergence, and provide possibilities to bypass the local optima to approach a better solution or even the global optima. Thus, the efficacy of coordinate update in ABCD algorithm is tentatively proven when compared with DIRECT in low dimensions.

4.2 Test on the Hedar test set

To further solidify the experiments, we compare the performance of the proposed ABCD algorithm with DIRECT, SOO and MCS on more test functions. The Jones test set focuses on low-dimensional problems, on which DIRECT has been reported as a very good algorithm. Our method is comparable and slightly better than DIRECT, showing the adaptiveness of ABCD. In the following, we turn to Hedar test set, which is largely extended from the Jones test set and contains many high-dimensional test functions. The Hedar test set is regarded as a benchmark for global search methods [19,20].

In low dimensions, DIRECT, SOO and MCS are all capable of obtaining a satisfactory solution within desirable costs. In Sect. 4.1, we have already proved the efficacy of the proposed algorithm for low-dimensional functions. Thus, further experiments are established to present the divergent performance of the tested algorithms for high-dimensional functions. In Hedar test set, there are many test functions which are adjustable to high dimensions. We skip the test functions included in the Jones test set already, and also ignore the test functions in $\mathbb{R}^2$ space. We only consider the test functions with relatively larger scales, say dimension $n > 4$. To investigate the influence of dimension n, the selected test functions are required to be changeable in dimensions n. Therefore, we obtain 13 test functions from Hedar test set to do the following experiments.

In this subsection, we evaluate the performance via CPU running time t, number of function evaluations z and the absolute error ε, which is defined in Eq. (5). We choose $\varepsilon = 10^{-4}$ as the target accuracy. Considering the case where algorithm converges in a local optimum or algorithm fails to reach the given accuracy ε within the budget of CPU running time t and maximum number of function evaluation z, we terminate the algorithm when any of the following criteria gets satisfied.

- $|\hat{f}^* - f^*| < \varepsilon$.
- $\hat{f}^*$ fails to decrease $\varepsilon_2 = 10^{-5}$ for T_2 sub-problems in succession, where $T_2 = \min\{n, 6\}$.
- $t > 20\,\mathrm{s}$.
- $t > n \times 10^4$.

In many functions, the optimal point is right on the axis origin and the searching domain is axis symmetrical at the same time. However, DIRECT first samples the center point of the initial domain, and then it reaches the global optimum at the first iteration. In such cases, the lower bounds and upper bounds are set asymmetrical to guarantee the fairness of the comparisons. Referring to [19], the lower bound $\mathbf{L}$ is adjusted to $0.8\mathbf{L}$ and the upper bound $\mathbf{U}$ is changed to $1.2\mathbf{U}$ when the mentioned symmetry happens.

In this subsection, we exert the proposed algorithm with its flexibilities. We set $flag = 1$ and $m_1 = 1$, which means sequentially conducting one-dimensional DIRECT on every coordinate at first. The switch condition is set as $T_1 = 3$ and $\varepsilon_0 = 10^{-2}$ in this paper. That

Table 3 Winning ratios of tested algorithms for reaching accuracy $\varepsilon = 10^{-4}$

Algorithm	SOO	MCS	DIRECT	ABCD
Winning ratio	0	2/13	0	11/13

'ABCD' represents the results of the proposed Algorithm 3 with computer running time

is to say, when the coordinate update fails to decrease $\varepsilon_0 = 10^{-2}$ for $T_1 = 3$ sub-problems in succession, then we switch to SQP. Then we turn to the block coordinate update, where we iteratively select 2 coordinates to optimize in randomness instead of in sequence. As for the m_1-dimensional DIRECT and m_2-dimensional DIRECT algorithms, the converging condition is set as $\varepsilon_0 = 10^{-5}$ and $T_0 = 5$, which also plays an important role in obtaining a good result. Restrictive converging condition means more computing costs on coordinate update, while relaxing converging condition may be unable to bring a good starting point for local optimizer SQP. The detailed analysis is followed in Sect. 4.2.2.

In Algorithm 3, we empirically place SQP after coordinate update, since SQP greatly depends the starting point. However, in some cases, SQP is capable of solving the problem without other strategies. In fact, the inserting of local optimizer SQP can be adaptively adjusted, thus we add the experiment where we set $flag = 0$ and replace SQP before coordinate update, which also shows the flexibility of the proposed ABCD algorithm. In the experiment, we use $ABCD^0$ to denote its corresponding numerical results. Moreover, to solidify the comparisons and further explore the contribution of ABCD algorithm, we add the experiments of ABCD with purely one-dimensional update, purely two-dimensional update and SQP. When an algorithm performs best in s test functions, we say that the winning ratio of this algorithm is $s/13$.

Computer running time directly reveals the efficiency of the algorithm in practical programming, and it is a general and basic measurement to evaluate the performance of algorithms. However, the implementation can be different for the same algorithm. SOO, MCS and the proposed ABCD algorithm use different strategies in domain partition to approach optimal solution, but function evaluation is the critical procedure that they all share. Thus, the number of function evaluations can be adopted as another measurement to compare the performance of the tested algorithms in efficiency. Even though SOO, MCS and the proposed ABCD algorithm are all based on the strategy of iterative domain partition, the cost of each partition are varied since they go through different operations accompanying with each function evaluation. Therefore, we use both computer running time t and number of function evaluations z to evaluate the performance, so as to present sufficient comparisons among the algorithms from different perspectives.

4.2.1 Comparisons based on computer running time

We first use computer running time t as the measurement, and compare SOO, MCS, DIRECT and ABCD on Hedar test set. Below, Table 3 summarizes the winning ratios of the algorithms.

Table 3 shows that the proposed ABCD algorithm holds the highest winning ratio for Hedar test set. The proposed ABCD algorithm succeeds in reaching the given accuracy $\varepsilon = 10^{-4}$ within shorter CPU running time for most of the test functions. To further investigate the performance of the tested algorithms. The detailed results for each test function are demonstrated in Table 4. t_{SOO}, t_{MCS} and t_{DIR} represent the CPU running time of SOO, MCS and DIRECT. t_{ABCD} is denoted for the most general form of ABCD algorithm where

Table 4 Running time for reaching given accuracy $\varepsilon = 10^{-4}$

Function	Dim	t_{SOO}	t_{MCS}	t_{DIR}	t_{ABCD}	t_{ABCD^0}	$t_{ABCD(1)}$	$t_{ABCD(2)}$	t_{SQP}
Ackley	6	1.17	0.47	Fail(z)	**0.08**	0.35	0.08	0.23	Fail(l)
	12	8.38	0.70	Fail(z)	**0.13**	1.06	0.12	0.56	Fail(l)
	18	Fail(t)	Fail(t)	Fail(z)	**0.17**	2.59	0.16	0.92	Fail(l)
Dixon–Price	6	Fail(l)	Fail(l)	5.36	**0.49**	Fail(l)	0.43	0.31	Fail(l)
	12	Fail(l)	Fail(l)	Fail(l)	**0.62**	Fail(l)	0.95	0.65	Fail(l)
	18	Fail(l)	Fail(l)	Fail(l)	**0.87**	Fail(l)	1.10	1.05	Fail(l)
Griewank	6	0.18	Fail(l)	Fail(l)	**0.10**	0.32	0.11	Fail(t)	Fail(l)
	12	0.80	0.62	Fail(l)	**0.42**	0.36	0.13	Fail(t)	Fail(l)
	18	2.24	Fail(l)	Fail(l)	**0.46**	0.39	0.17	Fail(t)	0.35
Levy	6	0.21	0.39	0.08	**0.06**	0.38	0.06	0.07	Fail(l)
	12	1.29	0.39	1.35	**0.09**	0.41	0.09	0.12	Fail(l)
	18	4.19	0.41	6.27	**0.12**	0.50	0.12	0.18	Fail(l)
Michalewicz	5	Fail(l)	Fail(l)	Fail(l)	**0.05**	0.39	0.05	Fail(l)	Fail(l)
	10	Fail(l)	Fail(l)	Fail(l)	**0.07**	0.41	0.07	Fail(l)	Fail(l)
Powell	6	0.51	**0.35**	Fail(t)	0.44	0.37	1.45	0.21	0.37
	12	Fail(l)	**0.44**	Fail(t)	0.58	0.41	4.16	0.38	0.41
	18	Fail(l)	**0.51**	Fail(t)	0.69	0.43	Fail(z)	Fail(t)	0.43
Rastrigin	6	0.64	Fail(l)	0.18	**0.06**	0.44	0.06	0.09	Fail(l)
	12	4.19	0.90	Fail(z)	**0.09**	0.59	0.10	0.14	Fail(l)
	18	14.25	Fail(l)	Fail(z)	**0.12**	0.92	0.12	0.19	Fail(l)
Rosenbrock	6	0.50	1.00	Fail(t)	**0.45**	0.36	Fail(l)	Fail(l)	0.42
	12	3.40	Fail(l)	Fail(t)	**0.68**	0.41	Fail(l)	Fail(l)	Fail(l)
	18	10.84	Fail(l)	Fail(t)	**0.85**	Fail(l)	Fail(l)	Fail(l)	Fail(l)
Schwefel	6	Fail(l)	Fail(l)	Fail(t)	**0.07**	0.41	0.07	0.09	Fail(l)
	12	Fail(l)	Fail(l)	Fail(t)	**Fail(l)**	Fail(l)	Fail(l)	Fail(l)	Fail(l)
	18	Fail(t)	Fail(l)	Fail(t)	**Fail(l)**	Fail(l)	Fail(l)	Fail(l)	Fail(l)
Sphere	6	0.25	0.25	0.06	**0.05**	0.32	0.05	0.07	0.32
	12	1.48	0.29	1.13	**0.07**	0.32	0.07	0.19	0.32
	18	4.43	0.37	Fail(z)	**0.09**	0.32	0.09	0.13	0.32
Sum square	6	0.23	0.25	0.04	**0.05**	0.36	0.05	0.07	0.36
	12	2.43	0.28	0.11	**0.07**	0.39	0.08	0.08	0.39
	18	7.78	0.33	6.03	**0.10**	0.45	0.09	0.11	0.45
Trid	6	Fail(t)	**0.37**	1.14	0.75	0.35	0.90	0.82	0.35
	12	Fail(t)	**0.46**	Fail(t)	3.01	0.36	Fail(z)	Fail(z)	0.36
	18	Fail(t)	**0.52**	Fail(t)	4.24	0.37	Fail(l)	Fail(l)	0.37
Zakharov	6	0.43	0.35	Fail(t)	**0.44**	0.33	0.67	1.14	0.33
	12	3.82	0.64	Fail(t)	**0.61**	0.34	Fail(z)	Fail(z)	0.34
	18	Fail(l)	Fail(l)	Fail(t)	**0.45**	0.34	Fail(z)	Fail(z)	0.42

the local optimizer comes after the coordinate update, say $flag = 1$. While t_{ABCD^0} is for ABCD algorithm when putting SQP at the beginning of the Algorithm 3, say $flag = 0$. To distinguish the efficacy of coordinate update and local optimizer SQP, we apply ABCD with purely one-dimensional update and purely two-dimensional update, where the running time is denoted as $t_{ABCD(1)}$ and $t_{ABCD(2)}$ respectively. We also independently apply SQP to the test function, where the running time is written as t_{SQP}. Thus, four columns on the right side of Table 4 are added. 'Fail(t)" and "Fail(z)" mean that algorithm keeps achieving descent as time goes on but fails to reach the given accuracy $\varepsilon = 10^{-4}$ with $t_{\max}$ and $Iter_{\max}$, which are set as 20 s and $n \times 10^4$ respectively. While 'Fail(l)' represents that algorithm gets stuck into a local optimum and fails to get out of the current local optimum within the budgets of CPU running time t and number of function evaluations z. In Table 4, for each test function in each selected dimension, the best performance among the tested algorithms is printed in boldface and marked with an underline.

Table 4 illustrates that the proposed ABCD algorithm has predominant advantages among most of the test functions. DIRECT fails to achieve the given accuracy $\varepsilon = 10^{-4}$ in test test functions, and SOO and MCS also fail in more than five test functions. For function Schwefel, ABCD runs efficiently to reach the given accuracy $\varepsilon = 10^{-4}$ in dimension 6, and then it gets trapped in local optimum with $Dim = 12, 18$. However, the final absolute error obtained by ABCD is only 1.5×10^{-4} and 2.3×10^{-4} with $Dim = 12, 18$, which are quite close to the targeted accuracy. While the absolute errors of SOO and MCS for function Schwefel range from 500 to more than 5000, which are much larger than ABCD. It is reasonable that the proposed algorithm fails to achieve 100% winning ratio for all the test functions under the current settings $flag = 1$, ε_1, T_1, ε_0 and T_0 in Algorithm 3, since we adopt the same settings for all the test functions, however the current settings are unnecessarily the best for all the test functions under any circumstance. It is worth mentioning that the proposed ABCD algorithm succeeds in every test function for accuracy $\varepsilon = 10^{-4}$, except function Schwefel in dimension 12 and 18. Especially, for function Dixon–Price and Michalewicz, MCS, SOO and DIRECT all fail to reach the given accuracy $\varepsilon = 10^{-4}$, while ABCD algorithm successfully solves it with high speed. Moreover, for function Ackley, Griewank and Zakharov, SOO and MCS all fail with dimension increasing, but ABCD algorithm remains capable of reaching the given accuracy in higher dimensions. Besides, for the cases Fail(t) in Table 4, the results remains either the same or reach a local optimum, when we increase the budget of computer running time from 20 to $t = 60$ s.

From the four columns on right side of Table 4, it is obvious that coordinate update is the fundamental mechanism of the proposed ABCD algorithm and the employment of SQP assists the coordinate update to better handle more test functions. It can also be seen that, for function Ackely, Griewank, Levy, Michalewicz, Rastrigin, sphere and sum square, the results are mainly contributed by the one-dimensional coordinate update. For function Dixon–Price, Powell, Rosenbrock, Trid and Zakharov, the employment of SQP accelerates the convergence and the two-dimensioanl update further helps improve the accuracy. If SQP is directly applied to test functions, it fails for most of the test functions. SQP only enjoys its advantages with good start points, which can be obtained by coordinate update. For some functions, such as Powell, Trid and Zakharov, ABCD algorithm performs slightly better than MCS when SQP is conducted before coordinate update, since SQP can successfully and efficiently solve these problems on it own. In such cases, the procedures of coordinate update and block coordinate update are skipped. However, for most of the functions, SQP fails to achieve a satisfactory solution without a good starting point, thus coordinate update is considered in priority. In summary, the introduction of coordinate update and the local optimizer SQP are shown to be useful in combination, since they mutually enables the proposed algorithm better solve more different problems.

Table 4 shows that SOO, MCS and ABCD all successfully reach the given accuracy $\varepsilon = 10^{-4}$ for function Levy, Rastringin, sphere and sum square in dimensions $n = 6, 12, 18$. However, the running time t and the changing tendency of t are different with dimension n

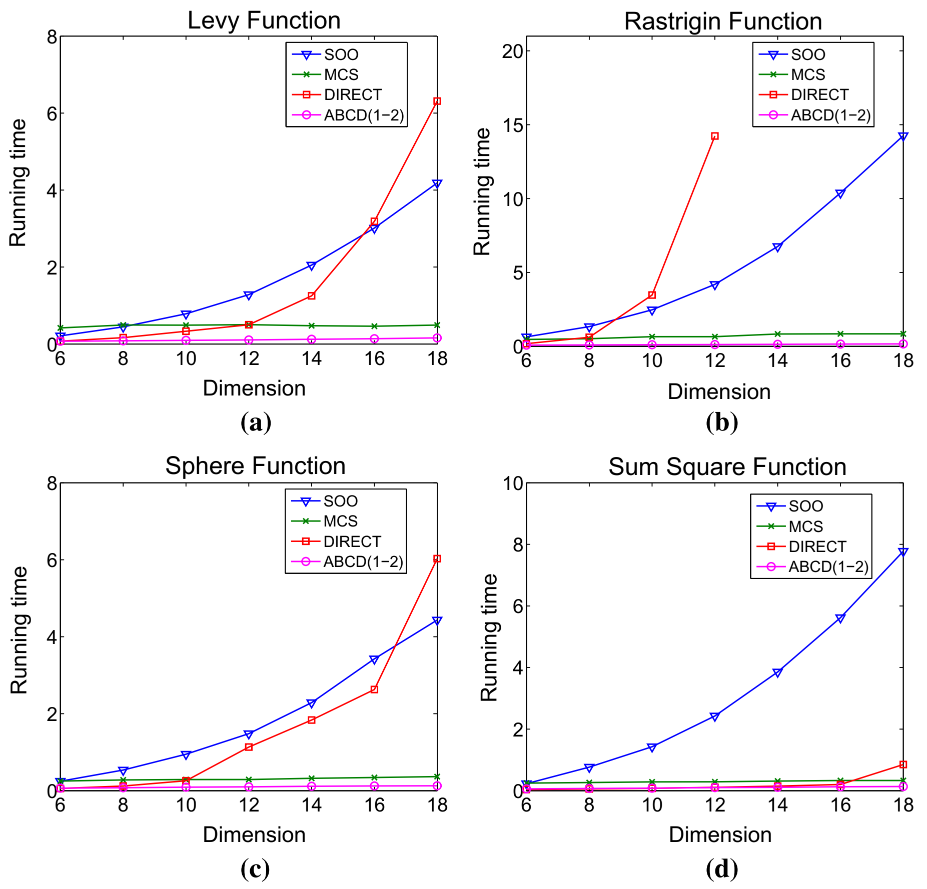

Fig. 7 Sub-test for Levy, Rastrigin, sphere and sum square. Running time (s) for reaching given accuracy $\varepsilon = 10^{-4}$. DIRECT fails to achieve accuracy ε within the budget of running time for Rastrigin when dimension $n \geq 14$. Thus, the running time of DIRECT for Rastrigin in dimension $n = 14, 16, 18$ is omitted

increasing. Figure 7 illustrates the running time t of SOO, MCS and ABCD against dimension n.

Figure 7 tells that the running time t_{SOO} increases rapidly with dimension increasing for Levy, Rastrigin, sphere and sum square. DIRECT holds the highest slope for most of the cases in Fig. 7. Although the running time t_{MCS} shows a low increasing rate when dimension n increases, ABCD remains outperforming MCS in speed in every tested dimension. In summary, SOO, MCS and DIRECT all succeed in solving Levy, Rastrigin, sphere and sum square, but the proposed algorithm maintains its advantages in fast speed and a lower rate in running time with dimension increasing.

For function Powell and Trid, MCS outperforms the proposed algorithm when SQP is placed after coordinate update. It can be seen from Table 4 that our local optimizer SQP performs a bit faster than MCS when it is directly applied to Powell and Trid, i.e., t_{ABCD^0} and t_{SQP}. To generalize the experiments on Hedar test set, the proposed ABCD algorithm uses the same settings for all the test functions, while the settings of ABCD can be adjusted separately. It makes sense that the proposed ABCD algorithm can be slower than purely

Table 5 Winning ratios of tested algorithms for reaching accuracy $\varepsilon = 10^{-4}$ with number of function evaluations

Algorithm	SOO	MCS	DIRECT	ABCD
Winning ratio	1/13	4/13	0	8/13

using one-dimensional update or purely using SQP in some cases. If we set $flag = 0$, the proposed algorithm still outperforms MCS for Powell and Trid, since SQP directly reaches the given accuracy ε. In such cases, SQP can be regarded as a degeneration of the proposed algorithm to some extent. This characteristic supports the fact that our algorithm is adaptive and holds more flexibilities.

4.2.2 Comparisons based on number of function evaluations

Although computer running time is a critical measurement to evaluate the efficiency of algorithms and it can directly reveal the practical efficiency in computer, the implementation can be different for the same algorithm. SOO, MCS and ABCD use different strategies to partition the domain in order to iteratively approach the optimal solution, but function evaluation is the key procedure shared by the three algorithms. Thus, we additionally take the number of function evaluations as the measurement to do the experiments, which enables the comparison more sufficient, along with the measurement of computer running time. Below, Table 5 summarizes the winning ratios.

It can be seen from Table 5 that the results are different from Table 3. However, the proposed ABCD algorithm still holds the highest winning ratio with the number of function evaluations. The details of the tested algorithms in this experiment are listed in Table 6, where z is number of function evaluations of the corresponding algorithm.

Table 6 shows a different result from Table 4. In Table 6, ABCD performs best in 8 functions out of the total 13 test functions in Hedar test set, while ABCD wins in 11 functions in Table 4. Nevertheless, ABCD still maintains the best performance compared with SOO and MCS. Since the partition mechanisms in SOO, MCS, DIRECT and ABCD are different, the cost of each function evaluation can also be varied. Take function Levy for example, the numbers of function evaluations in SOO, MCS, DIRECT and ABCD are 533, 140, 17,075 and 670, while the corresponding computer running time is 0.21, 0.39, 0.08 and 0.06 s. For DIRECT algorithm, it has the simplest procedure in domain partition, where there is little cost and few auxiliary operations accompanying each function evaluation. Thus, even it goes through 17,075 function evaluations, the computer running is quite short. SOO goes through less computer running time than MCS, but SOO takes more function evaluations. Even MCS requires the least function evaluations, its computer running time is not necessarily shorter, since MCS conducts more procedures and operations when doing each function evaluation. Thus, we use both measurements of computer running time and the number of function evaluations to do the comparisons.

For function Griewank and Rosenbrock, ABCD algorithm requires more function evaluations than SOO with dimension 6 and 12. However, ABCD requires less function evaluations with dimension increasing, such as dimension 18. We continue increasing the dimension in function Griewank and Rosenbrock, and the tendency is similar. We also notice that MCS takes the least function evaluations for function Zakharov with dimension 6 and 12, but it gets trapped into local optimum when the dimension increases to 18. While the proposed

Table 6 Function evaluations for reaching given accuracy $\varepsilon = 10^{-4}$ with number of function evaluations

Function	Dim	z_{SOO}	z_{MCS}	z_{DIR}	z_{ABCD}	z_{ABCD^0}	$z_{ABCD(1)}$	$z_{ABCD(2)}$	z_{SQP}
Ackley	6	2733	1801	Fail(z)	**1190**	824	1158	6977	Fail(l)
	12	11,151	7200	Fail(z)	**2416**	28,765	2516	18,873	Fail(l)
	18	Fail(t)	Fail(t)	Fail(z)	**3826**	91,674	3884	32,859	Fail(l)
Dixon–Price	6	Fail(l)	Fail(l)	Fail(l)	**2853**	Fail(l)	11,568	9398	Fail(l)
	12	Fail(l)	Fail(l)	Fail(l)	**6532**	Fail(l)	26,756	22,270	Fail(l)
	18	Fail(l)	Fail(l)	Fail(l)	**12,353**	Fail(l)	32,567	34,962	Fail(l)
Griewank	6	**447**	Fail(t)	Fail(l)	1372	126	1456	Fail(t)	Fail(l)
	12	1887	Fail(t)	7200	**1835**	377	2342	Fail(t)	Fail(l)
	18	4371	Fail(t)	Fail(l)	**2209**	703	3352	Fail(t)	367
Levy	6	533	**140**	17,075	670	925	670	1037	Fail(l)
	12	2733	**337**	14,979	1372	1468	1372	2534	Fail(l)
	18	6885	**601**	158,797	2074	4058	2074	4163	Fail(l)
Michalewicz	5	Fail(t)	Fail(t)	Fail(t)	**453**	558	453	Fail(l)	Fail(l)
	10	Fail(t)	Fail(t)	Fail(t)	**807**	1050	804	Fail(l)	Fail(l)
Powell	6	1215	**257**	Fail(t)	2516	125	31,505	6146	125
	12	Fail(t)	**1191**	Fail(t)	3861	954	102,889	9452	954
	18	Fail(t)	**1580**	Fail(t)	5023	1402	Fail(z)	Fail(t)	1402

Table 6 continued

Function	Dim	z_{SOO}	z_{MCS}	z_{DIR}	z_{ABCD}	z_{ABCD^0}	$z_{ABCD(1)}$	$z_{ABCD(2)}$	z_{SQP}
Rastrigin	6	1563	Fail(l)	7751	**822**	1020	822	1757	Fail(l)
	12	7043	7269	Fail(z)	**1644**	6790	1644	3562	Fail(l)
	18	17,529	Fail(l)	Fail(z)	**2466**	8760	2466	5357	Fail(l)
Rosenbrock	6	**1247**	1675	Fail(t)	2286	365	Fail(l)	Fail(l)	551
	12	**6025**	Fail(l)	Fail(t)	7653	1132	Fail(l)	Fail(l)	Fail(l)
	18	14,219	Fail(l)	Fail(t)	**11,562**	Fail(l)	Fail(l)	Fail(l)	Fail(l)
Schwefel	6	Fail(l)	Fail(l)	Fail(t)	**726**	1720	726	1617	Fail(l)
	12	Fail(l)	Fail(l)	Fail(t)	**Fail(l)**	Fail(l)	Fail(l)	Fail(l)	Fail(l)
	18	Fail(t)	Fail(l)	Fail(t)	**Fail(l)**	Fail(l)	Fail(l)	Fail(l)	Fail(l)
Sphere	6	677	**76**	1585	594	35	594	1155	35
	12	3157	**184**	61,175	1188	53	1188	2402	53
	18	7547	**331**	Fail(z)	1782	95	1782	3515	95
Sum square	6	783	**75**	569	594	103	594	731	103
	12	4637	**183**	3143	1188	327	1188	1476	327
	18	10,955	**330**	7745	1782	615	1782	2225	615
Trid	6	Fail(l)	**120**	21,887	12,960	71	26,166	27,018	71
	12	Fail(l)	**633**	Fail(t)	81,565	263	Fail(z)	Fail(z)	263
	18	Fail(l)	**1339**	Fail(t)	125,111	517	Fail(l)	Fail(l)	517
Zakharov	6	1137	**662**	Fail(t)	3367	97	19,721	37,382	97
	12	**6553**	8315	Fail(t)	22,603	275	Fail(z)	Fail(z)	275
	18	19,109	Fail(l)	Fail(t)	**2917**	495	Fail(z)	Fail(z)	495

ABCD algorithm not only remains solving the problem with dimension increasing, but also becomes more efficient. We can see that ABCD algorithm takes less function evaluations in dimension 18 than dimension 6 and 12 for function Zakharov. It is reasonable, since the one-dimensional coordinate update in ABCD may become less effective to bring a sharp descent in higher dimensions, and then the ABCD turns to local optimizer SQP earlier, where SQP is capable of effectively solving the problems. Thus, ABCD shows more efficient in higher dimensions for function Zakharov. Furthermore, for function Powell, Sphere, Trid and Zakharov, MCS outperforms our algorithm especially in dimension 6 and 12, but it should be mentioned that our local optimizer SQP performs better than MCS when we conduct SQP before coordinate update in ABCD algorithm or we directly apply pure SQP to the functions, i.e., z_{ABCD^0} and z_{SQP}. To generalize the experiments on Hedar test set, the settings are fixed the same for all the test functions, while ABCD can adjust its mode. If we relax the switch condition to replace local optimizer SQP at the beginning, ABCD still outperforms MCS among these four test functions when taking the number of function evaluations as the measurement. In fact, the proposed algorithm is capable of wining almost all the test functions in Hedar test, when it is allowed to adjust its settings for different test functions.

To illustrate the effect of converging condition in ABCD, we set different ε_0 in ABCD, whose results are shown in Table 7. In conditions 1, 2, and 3, we set $\varepsilon_0 = 10^{-4}$, $T_0 = 5$, $\varepsilon_0 = 10^{-5}$, $T_0 = 5$ and $\varepsilon_0 = 10^{-6}$, $T_0 = 5$, where the settings in condition 2 are identical to Tables 4 and 6. The smaller ε_0 is, the more iterations are required in each one-dimensional update. In condition 4, we conduct the local optimizer SQP before the one-dimensional update. Table 7 shows the results, which are demonstrated by computer running time $t(s)$, along with the number of function evaluations z, i.e. $t-(z)$.

Table 7 shows that different converging conditions result in varied computer running time t and number of function evaluations z, and it even affects the final solution. When ε_0 is set smaller, it means that algorithm imposes higher standard on convergence in one-dimensional update, thus the cost in condition 3 is higher than conditions 1 and 2 when they all succeeds in reaching the given accuracy ε. However, for function Ackley, Dixon–Price, Michalewicz, Powell, Schewefel and Trid, ABCD fails to reach the given accuracy ε under condition 1. Since ε_1 is not small enough in condition 1, and the one-dimensional update is unable to reach a sufficient convergence among all the coordinates, which makes it switch to local optimizer SQP too early before obtaining a good enough starting point for SQP. As the given accuracy ε is 10^{-4}, ε_0 can be set a bit smaller than ε, which ensures a fine convergence and a sufficient descent in the procedure of one-dimensional update. Meanwhile ε_0 cannot be chosen too high, which not only brings more redundant cost but also may make the algorithm unable to satisfy the convergence and fail to switch to the local optimizer. $\varepsilon_0 = 10^{-5}$ is suitable, while $\varepsilon_0 = 10^{-4}$ and $\varepsilon_0 = 10^{-6}$ is either too relaxed or over restricted. Since the computing costs of each function evaluation are different in DIRECT and SQP, the index $t-(z)$ is also different. For instance, function sphere is required shorter running time in condition 3 than condition 4, while the number of function evaluations is much larger than condition 4. Therefore, the measurements of computer running time t and number of function evaluations z are both effective and complement each other to evaluate the efficiency of algorithms. The best setting for each test function can be varied, thus the performance of ABCD algorithm can be improved if we adopt different strategies for different test functions. In practical application, when the solution is unsatisfactory, we can adjust the settings in ABCD to obtain a better solution. Therefore, ABCD algorithm holds flexibilities when dealing with wider ranges of problems.

Table 7 Computer running time and number of function evaluations for reaching given accuracy $\varepsilon = 10^{-4}$

Function	Dim	Condition 1	Condition 2	Condition 3	Condition 4
Ackley	6	0.08 – (1184)	0.08 – (1190)	0.10 – (1636)	0.35 – (824)
	12	0.41 – (2383)	0.13 – (2416)	0.16 – (3368)	1.06 – (28,765)
	18	0.43 – (3229)	0.17 – (3826)	0.22 – (5190)	0.59 – (91,674)
Dixon–Price	6	0.41 – (2168)	0.49 – (2853)	0.42 – (2419)	Fail(l)
	12	0.51 – (4080)	0.62 – (6532)	0.53 – (5305)	Fail(l)
	18	0.61 – (7377)	0.87 – (12,353)	0.81 – (11,432)	Fail(l)
Griewank	6	Fail(t)	0.10 – (1372)	0.11 – (1732)	0.32 – (126)
	12	Fail(t)	0.42 – (1835)	0.18 – (2493)	0.36 – (377)
	18	Fail(t)	0.46 – (2209)	0.32 – (2786)	0.39 – (703)
Levy	6	0.05 – (442)	0.06 – (670)	0.07 – (932)	0.38 – (925)
	12	0.07 – (832)	0.09 – (1372)	0.11 – (1186)	0.41 – (1468)
	18	0.08 – (1222)	0.12 – (2074)	0.16 – (2840)	0.50 – (4058)
Michalewicz	5	0.05 – (369)	0.05 – (453)	0.07 – (553)	0.39 – (558)
	10	Fail(l)	0.07 – (807)	0.08 – (946)	0.41 – (1050)
Powell	6	0.41 – (2597)	0.44 – (2516)	0.43 – (2211)	0.37 – (125)
	12	0.52 – (3233)	0.58 – (3861)	0.58 – (4924)	0.41 – (954)
	18	0.58 – (4352)	0.69 – (5032)	0.69 – (6559)	0.43 – (1402)
Rastrigin	6	0.06 – (702)	0.06 – (822)	0.08 – (1098)	0.44 – (1020)
	12	0.09 – (1644)	0.09 – (1644)	0.12 – (2196)	0.59 – (6790)
	18	0.12 – (2466)	0.12 – (2466)	0.16 – (3294)	0.92 – (8760)
Rosenbrock	6	0.48 - 4313)	0.45 – (2286)	0.51 – (4864)	0.36 – (365)
	12	0.58 – (5231)	0.68 – (7653)	0.72 – (8737)	0.41 – (1132)
	18	Fail(l)	0.85 – (11,562)	0.91 – (16,999)	Fail(l)
Schwefel	6	0.08 – (726)	0.07 – (618)	0.08 – (1098)	0.41 – (1720)
	12	Fail(l)	Fail(l)	Fail(l)	Fail(l)
	18	Fail(l)	Fail(l)	Fail(l)	Fail(l)
Sphere	6	0.05 – (594)	0.05 – (594)	0.07 – (954)	0.32 – (35)
	12	0.07 – (1188)	0.07 – (1188)	0.10 – (1908)	0.32 – (53)
	18	0.10 – (1782)	0.10 – (1782)	0.13 – (2862)	0.32 – (95)
Sum square	6	0.83 – (11,535)	0.05 – (594)	0.06 – (702)	0.36 – (103)
	12	2.51 – (70,661)	0.07 – (1188)	0.10 – (1772)	0.39 – (327)
	18	Fail(z)	0.10 – (1782)	0.14 – (3026)	0.45 – (615)
Trid	6	0.43 – (3276)	0.75 – (12,960)	0.82 – (17,287)	0.35 – (71)
	12	0.52 – (5232)	3.01 – (81,565)	3.32 – (107,213)	0.36 – (263)
	18	0.49 – (4524)	4.24 – (125,111)	Fail(z)	0.37 – (517)
Zakharov	6	0.43 – (3676)	0.44 – (3367)	0.42 – (3179)	0.33 – (97)
	12	0.52 – (5232)	0.61 – (8315)	0.49 – (3512)	0.34 – (275)
	18	0.49 – (4524)	0.45 – (2917)	0.64 – (4652)	0.42 – (495)

5 Conclusion

Adaptive block coordinate DIRECT algorithm, presented in this paper, incorporates the strategy of coordinate update to achieve high speed in high dimensions, while DIRECT is shown to lose efficiency and accuracy. Instead of constantly repeating the procedures of sampling and dividing on the whole domain, we iteratively select only one coordinate to do the optimization and keep the rest fixed. Coordinate update maintains the efficiency of DIRECT in low dimensions, but it lacks checking further improvements with other size of chosen coordinates. Thus, we adaptively switch to block coordinate update to explore further improvement with more chosen coordinates. Besides, coordinate update may possibly bring slow convergence and get stuck in local optimum in each sub-problem when the objective function is very flat, hence local optimizer SQP is employed to accelerate the convergence around smooth area. Furthermore, in the proposed algorithm, the starting point, the number of chosen coordinates, the way of selecting coordinates as well as the inserting of SQP can be adaptively adjusted. These alternative adaptations make the proposed ABCD algorithm more flexible to better solve a wider range of problems.

References

1. Björkman, M., Holmstrom, K.: Global optimization using the DIRECT algorithm in Matlab. Matlab Adv. Model. Optim. **1**(2), 1–8 (2002)
2. Bubeck, S.: Convex optimization: algorithms and complexity. Found. Trends Mach. Learn. **8**(3–4), 231–357 (2015)
3. Burkardt, J., Gunzburger, M., Peterson, J.: Insensitive functionals, inconsistent gradients, spurious minima, and regularized functionals in flow optimization problems. Int. J. Comput. Fluid Dyn. **16**(3), 171–185 (2002)
4. Carter, R.G., Gablonsky, J.M., Patrick, A., Kelley, C.T., Eslinger, O.J.: Algorithms for noisy problems in gas transmission pipeline optimization. Optim. Eng. **2**(2), 139–157 (2001)
5. Censor, Y., Zenios, S.A.: Parallel Optimization: Theory, Algorithms, and Applications. Oxford University Press, Oxford (1997)
6. Finkel, D.E., Kelley, C.T.: Additive scaling and the DIRECT algorithm. J. Global Optim. **36**(4), 597–608 (2006)
7. Finkel, D.E.: Global optimization with the DIRECT algorithm. Ph.D. thesis, North Carolina State University, Raleigh, North Carolina (2005)
8. Finkel, D.E.: DIRECT optimization algorithm user guide. Center for Research in Scientific Computation (2003)
9. Gablonsky, J.M.: Modifications of the DIRECT algorithm. Ph.D. thesis, North Carolina State University, Raleigh, North Carolina (2001)
10. Grbić, R., Nyarko, E.K., Scitovski, R.: A modification of the DIRECT method for Lipschitz global optimization for a symmetric function. J. Global Optim. **57**(4), 1193–1212 (2013)
11. Han, S.P.: A succesive projection method. Math. Program. **40**(1), 1–14 (1987)
12. Hildreth, C.: A quadratic programming procedure. Nav. Res. Logist. Q. **4**(1), 79–85 (1957)
13. Howson, H.R., Sancho, N.G.F.: A new algorithm for the solution of multistate dynamic programming problems. Math. Program. **8**(1), 104–116 (1975)
14. Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search. J. Global Optim. **14**(4), 331–355 (1999)
15. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. J. Optim. Theory Appl. **79**(1), 157–181 (1993)
16. Jones, D.R.: DIRECT Global Optimization Algorithm, pp. 431–440. Springer, New York (2001)
17. Kearsley, A.J.: The use of optimization techniques in the solution of partial differential equations from science and engineering. Ph.D. thesis, Department of Computational and Applied Mathematics, Rice University, Houston, TX (1996)
18. Kok, S., Sandrock, C.: Locating and characterizing the stationary points of the extended Rosenbrock function. Evol. Comput. **17**(3), 437–453 (2009)

19. Liu, Q., Cheng, W.: A modified DIRECT algorithm with bilevel partition. J. Global Optim. **60**(3), 483–499 (2014)
20. Liu, Q., Zeng, J.: Global optimization by multilevel partition. J. Global Optim. **61**(1), 47–69 (2015)
21. Li, L., Huang, X., Suykens, J.A.K.: Signal recovery for jointly sparse vectors with different sensing matrices. Signal Process. **108**(C), 451–458 (2015)
22. Munos, R.: Optimistic optimization of a deterministic function without the knowledge of its smoothness. In: NIPS, pp. 783–791 (2011)
23. Pardalos, P.M., Schoen, F.: Recent advances and trends in global optimization: deterministic and stochastic methods. In: Proceedings of the Sixth International Conference on Foundations of Computer-Aided Process Design, pp. 119–131 (2004)
24. Picheny, V., Wagner, T., Ginsbourger, D.: A benchmark of kriging-based infill criteria for noisy optimization. Struct. Multidiscip. Optim. **48**(3), 607–626 (2013)
25. Preux, P., Munos, R., Valko, M.: Bandits attack function optimization. In: Proceedings of IEEE Congress on Evolutionary Computation, pp. 2245–2252 (2014)
26. Richtárik, P., Takáč, M.: Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. Math. Program. **44**(1), 1–38 (2014)
27. Stern, T.A.: Class of decentralized routing algorithms using relaxation. IEEE Trans. Commun. **25**(10), 1092–1102 (1977)
28. Shubert, B.O.: A sequential method seeking the global maximum of a function. SIAM J. Numer. Anal. **9**(3), 379–388 (1972)
29. Tseng, P.: Dual ascent methods for problems with strictly convex costs and linear constraints: a unified approach. SIAM J. Control Optim. **28**(1), 214–242 (1988)
30. Valko, M., Carpentier, A., Munos, R.: Stochastic simultaneous optimistic optimization. In: International Conference on Machine Learning, pp. 19–27 (2013)
31. Wright, S.J.: Coordinate descent algorithms. Math. Program. **151**(1), 3–34 (2015)
32. Zhu, H., Bogy, D.B.: DIRECT algorithm and its application to slider air-bearing surface optimization. IEEE Trans. Magn. **38**(5), 2168–2170 (2002)