Appl Intell
DOI 10.1007/s10489-017-0901-8

Adaptive pattern search for large-scale optimization

Vincent Gardeux[1] · **Mahamed G. H. Omran**[2] · **Rachid Chelouah**[1] · **Patrick Siarry**[3] · **Fred Glover**[4]

Abstract The emergence of high-dimensional data requires the design of new optimization methods. Indeed, conventional optimization methods require improvements, hybridization, or parameter tuning in order to operate in spaces of high dimensions. In this paper, we present a new adaptive variant of a pattern search algorithm to solve global optimization problems exhibiting such a character. The proposed method has no parameters visible to the user and the default settings, determined by almost no a priori experimentation, are highly robust on the tested datasets. The algorithm is evaluated and compared with 11 *state-of-the-art methods* on 20 benchmark functions of 1000 dimensions from the CEC'2010 competition. The results show that this approach obtains good performances compared to the other methods tested.

Electronic supplementary material The online version of this article (doi:10.1007/s10489-017-0901-8) contains supplementary material, which is available to authorized users.

✉ Vincent Gardeux
vincent.gardeux@eisti.eu

1 Department of Computer Science, EISTI Engineering School, Cergy, France

2 Department of Computer Science, Gulf University for Science & Technology, Kuwait City, Kuwait

3 LiSSi Laboratory, University of Paris-Est Creteil, Creteil, France

4 Leeds School of Business, University of Colorado, Boulder, CO, USA

Keywords Pattern search · Scatter search · Optimization · Continuous · High-dimension · Large-scale · Adaptive methods

1 Introduction

Many optimization algorithms have been proposed for solving the continuous function optimization problem:

$(\mathcal{P})$ Min $f(x) : x_{min} \leq x \leq x_{max}$

The vector $x = (x_1, \ldots, x_D)$ is composed of D real-valued variables, and the vectors x_{min} and x_{max} are assumed finite and to satisfy $x_{min} < x_{max}$. Here we address in particular the so-called "black-box problems" where the algebraic model of f and its derivatives are not available. Direct search methods are designed for this class of problems since they neither compute nor explicitly approximate derivatives of f. Unfortunately, most of the current optimization methods cannot solve $(\mathcal{P})$ in a decent time when it is scaled for a high number of dimensions (e.g. $D > 100$); their performance degrades as D increases. For example, traditional bio-inspired algorithms [1], such as Particle Swarm Optimization [2], or Genetic Algorithms [3] are only capable of handling low dimensional problems, mainly because of the curse of dimensionality [4]. However, large scale optimization problems (continuous and discrete) are widely studied in science and engineering, and this trend is now reinforced with the emergence of Biocomputing, Data Mining, and Big Data [5, 6]. For example, in bioinformatics it is common to consider gene expressions, protein binding affinities or genetic variants as continuous features across multiple samples. These features are then used to build prediction models [7, 8], perform feature selection [9, 10] (for e.g. using Estimation of Distribution Algorithms [11–13]), search for substructures of gene/protein networks [14], or perform clustering [15, 16]. Optimization techniques can be used to perform these tasks, but require to be scaled up since several species have tens of thousands of genes which translate into tens of thousands of decision variables [17, 18]. Such applications and many others demonstrate the growing need for optimization methods that are able to solve high-dimensional problems.

Some practical optimization problems are "*computationally expensive*" independent of their size. Some methods are specifically designed to tackle this category of problems [19, 20]. However, these problems are different from the ones at the focus of this study. Computationally expensive problems, in the sense discussed here, arise in settings where it is necessary to employ very specific time-consuming functions, causing each evaluation of the objective function to be time expensive. Methods for such problems are compelled to restrict the number of calls and rely on the hope of obtaining a good candidate solution in few evaluations. Consequently, such methods are usually not designed to achieve progressively better results when they are run for a high number of function evaluations. In the case of large scale global optimization (LSGO) at the focus of this paper, the evaluation of the objective function is not so time consuming that it limits the quest for solutions of especially high quality. Rather, the difficulty posed by LSGO problems resides in the high number of dimensions, since solution methods for such problems should be tuned for each variable. This leads to the curse of dimensionality mentioned earlier.

Several optimization methods have been proposed to solve LSGO problems (refer to Section 4 for a list of *state-of-the-art* methods). Most of them are hybrids or adaptations of existing low-dimensional optimization procedures [21–28]. Others automatically or manually tune the parameters of existing methods to work with high dimensional costs [29–31]. For example, Masegosa et al. have proposed a centralized cooperative strategy where a set of trajectory-based methods is controlled by a rule-driven coordinator [32]. Li et al. have proposed a cooperative coevolving particle swarm optimization algorithm where Cauchy and normal distribution are used to sample new points in the search space [33].

At the time of this writing, a special issue of the Information Science journal has appeared that is specifically dedicated to nature-inspired methods for solving LSGO problems [34]. A few of the articles in this issue are particularly notable, and are worth mentioing. One is a novel "dimension reduction" technique, which can be applied to the problem before performing the optimization [35]. Another is an Evolutionary Compact Embedding (ECE) method applied to large scale image classification that uses boosting for iteratively improving the results of a genetic programming scheme [36]. These dimension reduction methods can indeed be very important to consider in certain engineering and statistical applications, but we observe that they may not always be applicable to a particular problem. Other methods relying on the same principle have also been recently developed, such as the Large Scale optimization with Autoencoders (LASCA) which uses autoencoders as a reversible mapping between the original search space and a reduced space [37]. This approach can be applied more generally to many problems and can use any metaheuristic to perform the optimization on the reduced space, while the objective function is assessed in the original space. However, it raises an issue about the tradeoff between time spent on dimension reduction and extra time allowed to solve the problem, and is not concerned with this matter (or with the question of whether dimension reduction might lead to improved solutions). In addition, in the indicated special issue, 10 algorithms were compared on many different LSGO benchmarks, disclosing that the results may vary widely from one benchmark to another, thus raising the "no free lunch" question [38]. Indeed, even if some methods can be more general than others in solving LSGO problems, each method is expected to perform very well only for a certain class of problems in the recognition that a global best solving algorithm may not exist.

In this paper we propose an extended variant of the Enhanced Unidimensional Search (EUS) [39], which was directly inspired by Pattern Search methods and has been shown to be effective for operating in high-dimensional spaces. The new method, called adaptive EUS (aEUS) requires no tuning and has few parameters (ideally, none that are visible to the user). We have evaluated aEUS on a suite of 20 scalable functions and compared it to 11 state-of-the-art algorithms designed for high-dimensional problems. aEUS is freely available at http://gardeux-vincent.eu/aEUS.php (MATLAB and Java source code).

2 Pattern Search

Pattern Search methods constitute a category of local search algorithms supported by solid convergence proofs [40–42]. They operate by the principle of evaluating the objective function iteratively in a "stencil-based" manner. The size of the stencil is modified as iterations proceed, which leads to convergence as the stencil tends towards 0. Several methods have issued from this basic principle including the Coordinate Descent (or Compass Search) [43] and the original Hooke-Jeeves Direct Search algorithm [44]. The Enhanced

Unidimensional Search (EUS) [39] is directly inspired by this principle and constitutes an adaptive line search algorithm which examines one dimension at a time. EUS does not thoroughly minimize the objective function at each iteration, and is adapted for global search. The method is easy to implement, designed to quickly locate the local optimum and is particularly efficient in handling high-dimensional problems. It was initially developed and compared to other state-of-the-art methods during the SOCO challenge [45].

EUS starts from a randomly-generated initial solution x. For initializing the size of the "stencil", a difference ("delta") vector h is created and initialized by setting $h = x_{max} - x_{min}$. The method successively focuses on a single variable x_i, $i \in \{1, 2, \ldots, D\}$, and selects the best neighbor from the three alternatives x, $x^u = \mathrm{x} + h_i \hat{e}_i$ and $x^d = \mathrm{x} - h_i \hat{e}_i$, where $\hat{e}_i$ is the unit vector with a value 1 in position i and 0's elsewhere. Since the search space is bounded, the computed values are always constrained within the search space limits $[x_{min}, x_{max}]$. The search then updates x to be the best of the three alternatives (x, x^u, and x^d), hence setting $x = \arg\min(f(x), f(x + h_i \hat{e}_i), f(x - h_i \hat{e}_i))$. Then, successive dimensions i are treated in the same manner. At the conclusion of an iteration, if the solution has not been improved along at least one dimension, then h is multiplied by a ratio R fixed to 0.5, therefore reducing the size of the components h_i of h. The choice of $R = 0.5$ is very typical in Pattern Search techniques as it corresponds to decreasing the step size by half. The h_i values continue to shrink in this fashion until $h < h_{min}$, where the latter is a vector of constants all fixed to 1.00E-15. Therefore, an iteration of the algorithm consists of scanning every dimension i successively from 1 to D and search for the best of the three alternatives on each dimension. Not all of the 2^D possible solutions are investigated, as EUS does not apply a thorough search on each dimension but improves the solution by small steps at a time. Consequently, each iteration consists of $2D$ evaluations of the objective function, since for each dimension, both solutions $f(x - h_i \hat{e}_i)$ and $f(x + h_i \hat{e}_i)$ are investigated. This model is close to the "opportunistic polling" scheme for Pattern Search algorithms, which is designed to alleviate the curse of dimensionality characterizing high-dimensional problems [39]. The main difference between opportunistic polling and the corresponding portion of our approach is that EUS always evaluates both of the solutions $f(x - h_i \hat{e}_i)$ and $f(x + h_i \hat{e}_i)$ while opportunistic polling skips the second solution if the first is already better than $f(x)$. Also, since pattern search methods are originally used to perform local search, EUS goes a step farther to improve its operation by a guided restart mechanism that keeps the best solutions found so far and re-initializes h to a new starting value after reaching the termination point given by $h < h_{min}$. Therefore, increasing the vector h_{min} may increase the potential number of restarts of the algorithm, but tends to decrease the error accuracy. Every solution found after reaching this termination point is called a local optimum and added to a reference set following the design of Scatter Search (see, e.g., [46]). In order to better explore the search space, when the restart procedure is activated, a new solution is generated that lies far from this reference set. This technique uses a diversification generation method based on the Scatter Search algorithm that generates a collection of diverse trial solutions and selects the one farthest from a reference set of previously visited local optima—i.e. the one that maximizes the minimum distance from the solutions within this set.

3 Adaptive pattern search

The adaptive EUS (aEUS) algorithm relies on the same principles as EUS, i.e. optimizing the solution by small steps on each dimension successively, in order to alleviate the curse of dimensionality. Each step focuses on a single variable x_i, $i \in \{1, 2, \ldots, D\}$, and selects the best neighbor from the three alternatives x, x^u and x^d as described in Section 2. In contrast to the original EUS method, aEUS does not scan all dimensions at each iteration. Instead it uses a *dimension winnowing* approach [47, 48] where each iteration scans a set of dimensions N_0 starting with $N_0 = N = \{1, 2, \ldots, D\}$ and then removes these dimensions over which the line search yields no improvement during the current iteration. Each successive iteration inherits the reduced N_0 bequeathed by its predecessor until N_0 becomes empty. The dimension winnowing process is a candidate list strategy motivated by the goal of accelerating computation - using a *drop-the-losers* strategy (which discards variables as soon as they are found to yield no improvement). We define a *pass* to be a series of iterations that starts from the full set of dimensions $N_0 = N = \{1, 2, \ldots, D\}$ and proceeds until $N_0 = \emptyset$. At the conclusion of each pass, h is updated by setting $h = h * R$ and the next pass commences by once again setting $N_0 = N$.

In EUS, as previously noted, the ratio R was a parameter fixed to 0.5 which served to refine the coarse grain of the search by reducing the value of h when no more improvements of the result could be made Additional experiments with EUS showed that if this coefficient was selected randomly after each iteration, the algorithm could occasionally become trapped in a local optimum. Further analysis showed that in these "bad" case scenarios the solution was continuing to improve, but too slowly to be able to reach the optimum in decent time. Even the simplest function

Sphere could fail to reach its optimum, which implies that a guided and smart decrease of the h vector was required. However, still later experiments with EUS have shown that in some particular cases, this average value could be dynamically changed for improved performance. Consequently, in aEUS, we made this parameter adaptive, by using a parameter decay technique for updating R inspired by Simulated Annealing [49], though we use the decay process in a different fashion and for a different purpose. We set a temperature variable T to D at the beginning of the simulation, and decreased T after each unsuccessful pass. The ratio R is then dynamically calculated depending on T by a factor of $e^{-\frac{T}{D}}$. This allows the R value to decrease and stabilize smoothly, in order to accelerate the descent to the local optimum at each unsuccessful pass, but decreases R by a progressively smaller amount at each iteration

Moreover, in aEUS the restart procedure has been simplified, as well as the triggering criterion. The method does not jump to a newly generated solution but keeps the current solution and resets the h, T and R values. Despite the fact that this behavior tends to constrain the solution to a local optimum, this behavior was preferred in order to better refine existing solutions. Moreover, we found experimentally that in some cases the re-initialization of the parameters was enough to jump out of a local optimum, since the initial values were big enough to allow an extensive scanning of the search space.

The restart procedure is triggered if two successive passes give no improvement, as follows:

$$h = (x_{max} - x_{min}) * rand$$
$$T = D$$
$$R = rand$$

where $rand$ is a randomly generated number from the interval [0,1]. Of note, the random generator is called twice for resetting h and R, hence the two $rand$ values should be different. Moreover, since R is a ratio used for decreasing the h convergence vector, it should be taken from the interval]0,1[. A value of 1 would make the search stationary. The bigger the value, the slower will be the convergence We specifically chose 0.9 as initial value in order to have at least one pass with a slow convergence rate. For the same reasons, we fixed the initial h value to the maximum allowed one $(x_{max} - x_{min})$. The use of the random generator for the restart procedure helps generating diversity, and avoids falling in the same local optimum. Of note, we observed that the elements of N_0 could be processed in sequential or random order without any impact on the results.

The pseudo-code of the aEUS algorithm is shown in Algorithm 1. The stopping criterion is defined as reaching a chosen maximum number of function evaluations.

Algorithm 1 Pseudo-code of the aEUS algorithm

```
Procedure aEUS
$h = x_{max} - x_{min}$
$R = 0.9$
$T = D$
$N_0 = N$
begin
    while stopping criterion is not reached do
        (begin current pass)
        while $N_0 \neq \emptyset$ do
            (begin current iteration by scanning $N_0$
             treat $N_0$ as a list whose elements are
             accessed successively in numerical
             order)
            for $i \in N_0$
                $x^u = x + h_i \hat{e}_i$
                $x^d = x - h_i \hat{e}_i$
                (update $x$ to be the best of the 3
                solutions)
                $x = \arg\min(f(x), f(x^u), f(x^d))$
                if $x$ has not been improved then
                remove $i$ from $N_0$ end
            end
        end
        if the pass gives no improvement then
            (refine the precision grain of the search)
            $N_0 = N$
            $R = R * e^{-\frac{T}{D}}$
            $h = h * R$
            (cooling schedule/reducing temperature)
            $T = 0.1 * T$
        end
        if two successive passes give no
        improvement then
            (restart procedure)
            $N_0 = N$
            $H = (X_{max} - X_{min}) * rand$
            $T = D$
            $R = rand$
        end
    end
end
```

4 Experimental setup

To test the performance of the proposed method, we chose a benchmark (see Omidvar et al. for best benchmark practice [50]) of 20 scalable functions provided for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization [51]. It consists of separable, partially-separable and non-separable functions of dimension $D =$

1000. All functions are derived from the sphere, elliptic, Schwefel's problem 1.2, Rosenbrock, Rastrigin and Ackley's functions. Table 1 summarizes the main properties of the benchmark functions.

A control parameter $m = 50$ is used to define the degree of "*separability*" of a given function. For all the benchmark functions we have used three different "time points" of function evaluations (FEs) to evaluate the performances of our algorithm: 1.2E5, 6.0E5 and 3.0E6. The error measure is defined as the absolute difference between the best-of-run $f(x^*)$ value and the actual optimum $f(x')$ of a given function: err. $= \left|f(x^*) - f(x')\right|$. Optima for all the functions are known by construction. The fact that the optimum value of every function is 0^D explains why all the functions are shifted. These optima are provided by the benchmark to compute error values, but they are not used during the optimization process. The optimization always adopts the objective of minimizing the objective value. Moreover, the optimization is considered to be a black-box operation and thus neither the algebraic model of f nor its derivatives are available during the computation.

The proposed method was compared with EUS, as well as 10 *state-of-the-art* methods published along with the CEC'2010 conference. These methods are listed below:

1) DECC-G: Group size s = 100 [52]
2) DECC-G*: Same as DECC-G, except that the grouping structure was used as prior knowledge (group size s = 50) [52]
3) Multilevel Cooperative Coevolution (MLCC) [53]
4) Differential Ant-Stigmergy Algorithm (DASA, C-7136) [54]
5) Sequential DE Enhanced by Neighborhood Search (SDENS, C-7273) [55]
6) Two-stage based Ensemble Optimization (EOEA, C-7306) [56]
7) Memetic Algorithm Based on Local Search Chains (MA-SW-Chains, C-7330) [57]
8) Self-adaptive Differential Evolution Algorithm (jDElsgo, C-7392) [58]
9) Cooperative Co-evolution with Delta Grouping (DECC-DML, C-7597) [50]
10) Dynamic Multi-Swarm Particle Swarm Optimizer with Subregional Harmony Search (DMS-PSO-SHS, C-7938) [59]
11) Enhanced Unidimensional Search (EUS) [39]

Methods 1, 2, 5, and 8 are adaptations of the Differential Evolution algorithm, which is known to perform well on large-scale problems [60]. Algorithms 1, 2, 3, and 9 are more specifically based on the Cooperative Coevolution scheme [61], which partitions the problem into multiple evolutionary sub-problems and then optimizes them separately. Algorithms 4 and 10 are based on Swarm algorithms, respectively consisting of Ant Colony Optimization [62] and Particle Swarm Optimization [2, 63]. They were specifically adapted for high-dimension optimization. Additional details about each method can be found in the papers cited.

Table 1 Properties of the benchmark functions

Function	Name	Separable
F1	Shifted elliptic	Yes
F2	Shifted Rastrigin	Yes
F3	Shifted Ackley	Yes
F4	Single-group shifted and m-rotated elliptic function	Partially
F5	Single-group shifted and m-rotated Rastrigins function	Partially
F6	Single-group shifted and m-rotated Ackleys function	Partially
F7	Single-group shifted m-dimensional Schwefel's problem 1.2	Partially
F8	Single-group shifted m-dimensional Rosenbrocks function	Partially
F9	D/(2m) group shifted and m-rotated elliptic function	Partially
F10	D/(2m) group shifted and m-rotated Rastrigins function	Partially
F11	D/(2m) group shifted and m-rotated Ackleys function	Partially
F12	D/(2m) group shifted m-dimensional Schwefel's problem 1.2	Partially
F13	D/(2m) group shifted m-dimensional Rosenbrocks function	Partially
F14	D/(m) group shifted and m-rotated elliptic function	Partially
F15	D/(m) group shifted and m-rotated Rastrigins function	Partially
F16	D/(m) group shifted and m-rotated Ackleys function	Partially
F17	D/(m) group shifted m-dimensional Schwefel's problem 1.2	Partially
F18	D/(m) group shifted m-dimensional Rosenbrocks function	Partially
F19	Shifted Schwefel's problem 1.2	No
F20	Shifted Rosenbrock	No

The above 11 methods were applied on the 20 benchmark functions using the experimental settings described in CEC'2010 [51]. We gathered the published average errors from the 10 first methods for comparison with aEUS without recomputing them.

We also performed a comparison with the Pattern Search method that is closest in conception to EUS: the coordinate descent (also called compass search). This method is designed for local search only, and thus we used two versions in order to provide a fair comparison: (1) using the original algorithm (the C algorithm provided by [43]), and (2) adding the same restart procedure used by EUS. The results including the restart procedure are reported in Supplement Table S1. The results of the original algorithm are not shown because they were either similar or worse on every function, compared to the second variant (that uses our restart procedure). Supplement Table S1 shows that F1 was not solved when $D = 1000$, although it is one of the easiest functions (F1 is separable and unimodal). Of note, the coordinate descent algorithm managed to solve F1 with a good precision when $D \leq 500$ (data not shown) but seems to converge too slowly for higher dimensions, thus exhibiting a scalability problem. Since the algorithm was not capable of coping with the increase in dimensionality, we did not include it in the remaining analyses.

We also compared aEUS results with a more extended Pattern Search method called PSwarm [64]. This algorithm combines Particle Swarm Optimization (PSO) [2] for exploring the search space with the Pattern Search technique for performing local search. PSwarm was not specifically designed for large-scale optimization but, similarly to EUS and aEUS, it uses a variant of opportunistic polling. PSwarm was downloaded in Matlab from the authors' website. Contrary to the other methods, we ran PSwarm for only 5 runs (instead of 25) on function F1, but we kept the other settings. We applied this modified protocol for two reasons: (1) PSwarm requires ∼28 h for running 3E6 FEs of function F1 (for only 1 run), compared to aEUS which requires only 15 s under the same conditions, and (2) the results of PSwarm on F1 were not good enough to qualify it for consideration in the rest of this study (average error = 3.61E08). For these two reasons, we did not include further results from PSwarm.

Taken together, the results of PSwarm and the coordinate descent show that conventional Pattern Search techniques must be adapted in order to provide a method capable of effectively handling large-scale problems. We structure aEUS to overcome the limitation of these conventional techniques by using a very stringent form of the opportunistic polling scheme which is reinforced with the "dimension winnowing" approach of [46, 47]. The results obtained by the two Pattern Search algorithms we have used for comparison should be interpreted by recognizing that both methods are not initially designed for large-scale problems.

5 Results and discussion

5.1 aEUS results

The best, median, worst, mean errors and standard deviations obtained by the aEUS algorithm on each function are shown in Table 2. The mean error values are computed for 25 runs and are reported for each of the 20 functions.

Table 2 shows that aEUS solved the three first separable functions easily. This behavior is expected from a method that optimizes each dimension successively, and thus the results are concordant with prior experience. Non-separable functions F19 and F20, in the contrary, are less well solved by our approach. However, the other algorithms also encounter difficulties on those functions (see the complete result table provided as Supp. File 1). In particular, it is worth noting that aEUS obtains the best results over all methods for function F19 and second best (after DMS-PSO-SHS) for F20.

Another interesting feature of aEUS is its very quick convergence. On many functions, the best value is found early ($D = 1.2E5$). Despite the fact that aEUS is ranked in the top-3 methods for 11 functions through 20, it has visible shortcomings for functions F6 (rank 10th), F11 (rank 7th) and F16 (rank 8th) at $D = 1000$. Interestingly, these functions all correspond to variants of Ackley's function, which implies that this function is particularly resistant to solution by aEUS, as was also the case for EUS. We hypothesize that the cause stems from the nature of Ackley's function, which consists of a very flat landscape with many local optima. aEUS and its local scheme is probably trapped too many times in local optima before having a chance to reach the global optimum. It is also worth noting that aEUS performs globally equally or better than EUS over each function. This confirms the enhancement brought by the adaptive method we present in this paper.

5.2 aEUS vs. other state-of-the-art methods

The aEUS method was compared with the 11 methods listed in Section 4. Table 3 shows the average rank of aEUS compared to the other 11 methods for each function and for each FEs cutoff. It was computed by first averaging the error values over the 25 runs for every function. Then every method was ranked based on this average error and the average rank was computed across all functions. Table 3 reports this average rank for every FEs time point. MA-SW-Chains and aEUS are the two best methods for early convergence

Table 2 Results of aEUS on CEC'2010 functions for 25 independent runs with 1000 dimensions

D = 1000		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
1.2E5	Best	1.95E-11	0.00E+00	8.09E-09	9.46E+12	6.11E+07	1.97E+07	7.91E+09	2.77E+07	2.98E+08	6.25E+03
	Median	2.30E-11	0.00E+00	8.32E-09	1.90E+13	7.11E+07	1.99E+07	2.33E+10	3.97E+08	3.68E+08	7.20E+03
	Worst	2.85E-10	0.00E+00	2.84E-08	3.96E+13	9.48E+07	2.01E+07	7.43E+10	8.75E+09	5.36E+08	7.87E+03
	Mean	6.31E-11	0.00E+00	1.54E-08	2.09E+13	7.18E+07	1.99E+07	2.72E+10	1.26E+09	3.73E+08	7.15E+03
	StDev	2.16E-19	0.00E+00	2.28E-15	1.25E+27	1.26E+17	1.58E+11	5.55E+21	8.31E+19	6.07E+16	3.32E+06
6.0E5	Best	3.56E-24	0.00E+00	1.80E-12	1.21E+12	6.11E+07	1.97E+07	2.34E+07	1.38E+07	3.17E+07	6.25E+03
	Median	8.05E-24	0.00E+00	1.89E-12	2.38E+12	7.11E+07	1.99E+07	4.55E+08	9.46E+07	4.22E+07	7.20E+03
	Worst	1.44E-23	0.00E+00	1.99E-12	5.09E+12	9.48E+07	2.01E+07	3.16E+09	1.42E+09	5.13E+07	7.87E+03
	Mean	8.32E-24	0.00E+00	1.90E-12	2.67E+12	7.18E+07	1.99E+07	8.23E+08	2.42E+08	4.18E+07	7.15E+03
	StDev	1.60E-46	0.00E+00	5.08E-26	3.83E+25	1.26E+17	1.58E+11	1.67E+19	2.94E+18	4.89E+14	3.33E+06
3.0E6	Best	3.56E-24	0.00E+00	1.80E-12	1.32E+11	6.11E+07	1.97E+07	1.92E+01	2.20E+03	6.28E+06	6.25E+03
	Median	8.05E-24	0.00E+00	1.89E-12	2.41E+11	7.11E+07	1.99E+07	1.02E+03	4.42E+07	7.63E+06	7.20E+03
	Worst	1.44E-23	0.00E+00	1.99E-12	7.08E+11	9.48E+07	2.01E+07	1.61E+04	1.04E+09	8.51E+06	7.87E+03
	Mean	8.32E-24	0.00E+00	1.90E-12	2.84E+11	7.18E+07	1.99E+07	2.73E+03	1.29E+08	7.57E+06	7.15E+03
	StDev	1.60E-46	0.00E+00	5.08E-26	4.26E+23	1.26E+17	1.58E+11	4.63E+08	1.21E+18	6.48E+12	3.33E+06

D=1000		F11	F12	F13	F14	F15	F16	F17	F18	F19	F20
1.2E5	Best	1.98E+02	2.27E+05	4.53E+03	7.47E+08	1.29E+04	3.82E+02	7.59E+05	1.24E+04	1.59E+07	1.74E+03
	Median	1.99E+02	3.73E+05	9.54E+03	9.27E+08	1.43E+04	3.98E+02	1.15E+06	3.35E+04	3.47E+07	1.97E+03
	Worst	2.00E+02	6.14E+05	2.48E+04	1.37E+09	1.50E+04	3.99E+02	1.28E+06	6.10E+04	5.11E+07	5.50E+03
	Mean	1.99E+02	3.97E+05	1.06E+04	9.52E+08	1.42E+04	3.98E+02	1.10E+06	3.51E+04	3.48E+07	2.54E+03
	StDev	1.75E+00	2.45E+11	5.53E+08	4.54E+17	7.39E+06	2.71E+02	5.48E+11	3.59E+09	2.33E+15	3.63E+07
6.0E5	Best	1.98E+02	6.67E+03	1.36E+03	9.97E+07	1.29E+04	3.82E+02	2.87E+04	4.36E+03	1.67E+06	1.21E+03
	Median	1.99E+02	1.44E+04	2.43E+03	1.22E+08	1.43E+04	3.98E+02	4.09E+04	1.62E+04	2.14E+06	1.40E+03
	Worst	2.00E+02	1.91E+04	1.63E+04	1.56E+08	1.50E+04	3.99E+02	7.35E+04	4.77E+04	5.09E+06	1.63E+03
	Mean	1.99E+02	1.45E+04	3.42E+03	1.24E+08	1.42E+04	3.98E+02	4.35E+04	1.81E+04	2.50E+06	1.43E+03
	StDev	1.75E+00	2.06E+08	2.75E+08	5.25E+15	7.28E+06	2.71E+02	3.56E+09	2.87E+09	1.82E+13	4.10E+05
3.0E6	Best	1.98E+02	8.21E-02	1.38E+02	1.41E+07	1.29E+04	3.82E+02	1.67E+00	1.40E+03	7.66E+03	5.41E+00
	Median	1.99E+02	2.54E-01	8.95E+02	1.71E+07	1.43E+04	3.98E+02	3.66E+00	2.68E+03	9.14E+03	3.98E+02
	Worst	2.00E+02	9.44E-01	3.45E+03	1.96E+07	1.50E+04	3.99E+02	1.12E+01	5.25E+03	1.22E+04	9.69E+02
	Mean	1.99E+02	3.28E-01	1.09E+03	1.71E+07	1.42E+04	3.98E+02	3.98E+00	2.97E+03	9.44E+03	4.51E+02
	StDev	1.75E+00	1.46E+00	1.50E+07	4.08E+13	7.28E+06	2.71E+02	1.19E+02	3.60E+07	3.83E+07	1.21E+06

(FEs $\leq$ 6.0E5), and they both remain well ranked (respectively third and fourth) at the end of the experiment (FEs = 3.0E6).

We also performed a more global comparison between aEUS, EUS and all the other methods presented at the CEC'2010 conference. We used the same ranking metric used for summarizing the results of the conference. It is a Formula 1 ranking system whose description is available on slide 6 of nical.ustc.edu.cn/wcci2010/CompetitionSummary.pdf. In brief, it ranks each method for each function result (mean error, variance, etc.), creates a score for each rank (the better rank, the higher score) in each condition, and sums them all. The results are available in Supp. File 1, and are summarized in Table 4.

This table shows that overall aEUS is ranked second, after MA-SW-Chains. This implies that our methodology is able to solve not only separable problems, but also non-separable ones, with competitive accuracy. The separable problems alone are very well solved (see Section 5.1), in a very few number of function evaluations.

Finally, we examined the global performances of the 11 algorithms using their respective *performance profiles* [65] relative to the error posted by each method after 1.2E5, 6.0E5, and 3.0E6 evaluations of the objective function. That is, for each method, we plot the fraction $\boldsymbol{p}$ of problems for which the method is within a factor $\boldsymbol{\tau}$ of the best-achieved precision. Figure 1 shows the performance profiles of the 11 methods compared to aEUS and EUS. We have specifically

Table 3 Average ranks of aEUS on CEC’2010 functions for 25 independent runs with 1000 dimensions

Method	FEs = 1.2E5	FEs = 6.0E5	FEs = 3.0E6
	Average rank	Average rank	Average rank
DECC-G	No data	No data	9.60
DECC-G*	No data	No data	5.30
MLCC	No data	No data	7.75
DASA	4.15	4.60	6.90
SDENS	7.10	7.45	9.10
EOEA	**4.10**	**3.95**	5.45
MA-SW-Chains	**2.20**	**3.10**	**4.45**
jDElsgo	7.60	6.55	**4.35**
DECC-DML	6.35	5.75	7.25
DMS-PSO-SHS	5.55	4.40	**4.30**
EUS	4.25	5.45	8.35
aEUS	**3.65**	**3.70**	5.05

Top 3 results for every FEs timepoint are bolded

highlighted the comparison with MA-SW-Chains, and more results are plotted in Supp. Figure S1.

The Figure shows that all methods exhibit similar performances on every function. In particular, for FEs = 1.2E5 and FEs = 6.0E5, it is clear that aEUS is the method that yields the best precision on most problems. These outcomes demonstrate that aEUS converges very quickly toward an acceptable solution. This perception is further verified in Supp. Table S2, where the Area Under the Curves (AUCs) show that aEUS performs the best under these two conditions. The curves also show that the aEUS curve dominates the EUS curve. For FEs = 3.0E6, it is difficult to differentiate between the methods visually. However, AUCs show that aEUS performs comparably to MA-SW-Chains and jDElsgo and is therefore one of the best methods.

We also performed a quick comparison with a more recent method that improves the artificial bee algorithm using cooperative coevolution, named CCOABC [66]. This method was also specifically designed for high-dimension optimization, and was applied on the CEC’2010 benchmark under the same protocol. Results are available in Supp. File 1 and show that aEUS obtains better results than CCOABC for half of the functions (10), while CCOABC was better for the other half (10) for D = 1000. A statistical comparison (Wilcoxon signed rank test) of the mean errors gives p = 0.2575, which implies that we cannot conclude that one method is better than the other. These results confirm that aEUS is very competitive and obtains results at least comparable to those of recent state-of-the-art methodologies.

Table 4 Formula 1 scores of aEUS and the algorithms presented at the CEC’2010 conference

Method	Formula 1 score	Rank
DASA	3455.00	5
SDENS	1768.00	9
EOEA	**3844.00**	**3**
MA-SW-Chains	**4821.00**	**1**
jDElsgo	2561.00	7
DECC-DML	2537.00	8
DMS-PSO-SHS	3602.00	4
EUS	2803.00	6
aEUS	**3900.00**	**2**

Top 3 results are bolded

5.3 Statistical analysis

In order to further assess the significance of our results, we performed a statistical analysis of the results applying the non-parametric testing presented in [67]. We performed a Wilcoxon signed-rank test between our method (aEUS) and all the other methods presented at the CEC’2010 conference two-by-two (paired), for each FEs cutoff. The calculations were made on the average error values, as suggested in [67]. Table 5 recapitulates the p-values obtained for each comparison.

The table show several comparisons where the p-value is below 5 %, implying a statistically significant difference of performance between aEUS and the related methods. In order to know which of two statistically different methods performs the best, we checked the W^+ and W^- scores for each of these cases. We found that for every significant comparison the signed ranks corresponding to aEUS were higher, implying that aEUS performed better. These results reinforce the results discussed before, aEUS seems to obtain outcomes whose quality is similar to that obtained by MA-SW-Chains and EOEA and these three methods are not significantly statistically different. DMS-PSO-SHS also seems to perform better at FEs = 3.0E6. Finally, we can say that aEUS outperforms EUS, and this is particularly visible when the number of FEs grows.

5.4 aEUS computational running time

As shown previously, EUS runs very fast as a consequence of its local search design and algorithmic simplicity. We therefore assumed that this should also be the case for aEUS. To test this assumption, we investigated the efficiency of aEUS by computing its average running time on every function of the CEC’2010 benchmark on an Intel core i7@3.60GHz with 16Gb RAM. Given the preceding results, we also investigated the running time of EUS and MA-SW-Chains. Supp. Table S3 shows that aEUS runs slightly faster than EUS, executing 3.0E6 function evaluations of

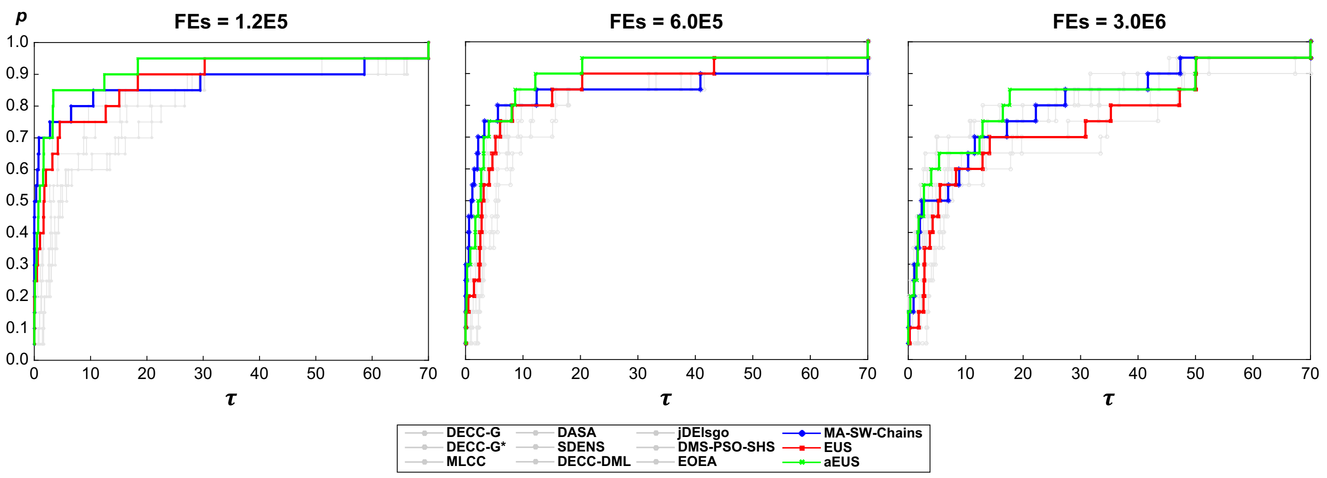

Fig. 1 Performance profiles of the 12 studied algorithms. This Figure shows the performance profiles, of each method across the 20 functions. Specifically, we show the performance profiles at FEs (number of function evaluations) = {1.2E5, 6.0E5, 3.0E6}. In order to avoid unreadable Figure, we merely colored EUS, aEUS and MA-SW-Chains results. Further results at FEs = 3.0E6 are available in Supp. Figure S1. The complete list of Area Under the Curve (AUC) for each function is available in Supp. Table S2

non-expensive 1000D objective functions in ∼7 s. More expensive functions require up to ∼300 s to be processed. By contrast, MA-SW-Chains require much more computing time. As a comparison basis, the less expensive 1000D functions require ∼160 s to be processed using MA-SW-Chains under the same conditions, which is ∼25-fold more than aEUS. Therefore, if aEUS were allotted the same computing time as MA-SW-Chains, it might achieve better results.

Next, we used these outcomes to evaluate the time complexity of aEUS. We ran aEUS on $D = \{50, 100, 200, 500, 1000\}$ and computed the time required for achieving 3.0E6 function evaluations. For each dimension D, we repeated the procedure 25 times and calculated the average computation time. Figure 2a shows the graph $t = f(D)$ obtained for function F1, where t represents the running time in milliseconds and D the dimension. As may be seen, this graph does not show a linear relationship. To clarify the relationship produced, we drew two other graphs with the same values, one with a logarithmic scale for t and a decimal scale for D (Fig. 2b) and another with both logarithmic scales (Fig. 2c). The latter shows that a linear relationship exists between $\log(t)$ and $\log(D)$ (Pearson's correlation coefficient $\alpha = 0.999$). This indicates that the variation of running time in relation to dimension is a power function. Using a simple linear regression, we found that $t = c * D^{1.88}$, with c a constant value. We applied the same test to several other functions and obtained similar results, with a linear coefficient ranging from

Table 5 p-values from Wilcoxon signed-rank test between aEUS and the algorithms presented at the CEC'2010 conference

	aEUS					
	FEs = 1.2E5		FEs = 6.0E5		FEs = 3.0E6	
	p	FDR	p	FDR	p	FDR
DECC-G	No data	No data	No data	No data	**0.01718**	0.17180
DECC-G*	No data	No data	No data	No data	0.49801	1.00000
MLCC	No data	No data	No data	No data	0.1054	0.63240
DASA	0.28623	1.00000	0.23517	0.80980	**0.04187**	0.31992
SDENS	**0.00422**	**0.02954**	**0.00422**	**0.02954**	**0.02148**	0.19332
EOEA	0.52167	1.00000	0.75617	0.80980	0.84082	1.00000
MA-SW-Chains	0.230513	1.00000	0.20245	0.80980	0.98544	1.00000
jDElsgo	**0.00059**	**0.00472**	**0.01362**	0.08172	0.985435	1.00000
DECC-DML	**0.01923**	0.11538	0.13273	0.66365	**0.03999**	0.31992
DMS-PSO-SHS	0.2611	1.00000	0.24549	0.80980	0.647655	1.00000
EUS	0.95187	1.00000	**0.00266**	**0.02128**	**0.00036**	**0.00396**

p-values were adjusted using False Discovery Rate (FDR) correction [68] for each FEs time point

Results passing 5% significance threshold are bolded

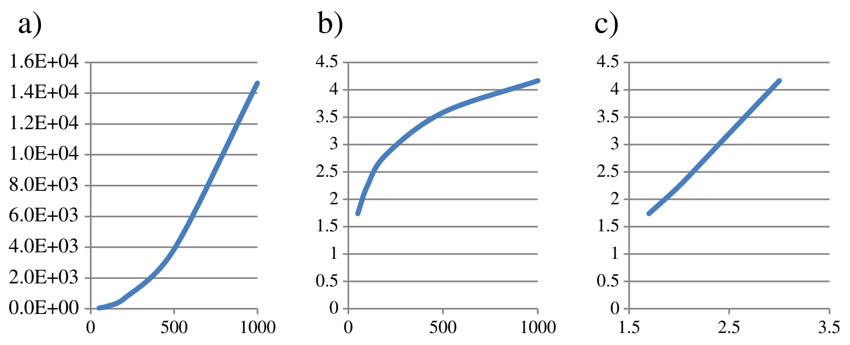

Fig. 2 aEUS computational running time. The figure shows aEUS computing times (in ms) for achieving 3.0E6 function evaluations on different dimension scales (D = {50,100,200,500,1000}). The times are averaged across 25 runs. **Panel (a)** shows a linear plot t = f(D), **panel (b)** shows the same plot with the time t in logarithmic scale log(t) = f(D), and **panel (c)** shows the same values in log-log scale log(t) = f(log(D))

1.88 (F1) to 1.99 (F9) (Supp. File 2). This discloses that the computational complexity of the algorithm is approximately $C = \mathcal{O}\left(D^2\right)$.

6 Conclusions and future work

We have demonstrated how an adaptive pattern search algorithm can be effective for high-dimensional problems. Like EUS, which it extends, aEUS improves the solution vector dimension by dimension. However, instead of scanning all dimensions at each iteration, it employs a dimension winnowing approach to drop those dimensions over which the line search yields no improvement. Furthermore, aEUS uses a parameter decay process to adapt the ratio value of EUS and significantly simplifies the restart procedure of EUS.

The aEUS approach was tested on 20 scalable functions and was compared with 11 state-of-the-art methods. The results showed that aEUS performed very well on all functions, and more specifically on separable ones. Overall, in addition of being very simple to use and implement aEUS ranked 4th (out of 11 methods) at the end of the 3.0E6 FEs, and ranked 2nd in earlier FEs cutoffs Moreover, aEUS ranked second according to the Formula 1 equation used to rank the algorithms that participated in the CEC'2010 competition Finally, aEUS required an order of magnitude less computation time to obtain solutions comparable to those obtained by its best competitors.

Future works are planned to improve aEUS. They can be pursued in many different ways given the modular nature of the framework in which aEUS is designed. First, possible hybridization of aEUS can be studied in the scope of more global research algorithms, as already successfully investigated by external research teams, in particular with the Artificial Bee Colony algorithm [69, 70]. Other variants of the dimension winnowing approach could also be investigated, since this component of our algorithm led to significant improvements both in convergence speed and quality of results. Typically, subsets of dimensions could be analyzed in parallel, and results could be merged in the end, for purposes such as using Path Relinking algorithms. We believe that the parallelization of our algorithm could lead to even faster convergence, while keeping the robustness of the results. Another possible improvement would be to use a more elaborate line search procedure, in order to make the unidimensional search more thorough, as by using the 3-2-3 line search procedure proposed in [48]. In sum, we believe that even though aEUS already exhibits very satisfactory results, it could be further improved in many different ways. This derives mainly from the fact that the aEUS algorithm stands on simple concepts that were assembled to make a core algorithm specifically designed for high dimensional optimization.

References

1. Olariu S, Zomaya AY (2005) Handbook of bioinspired algorithms and applications. Chapman & Hall/CRC, London
2. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks. Perth
3. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley Longman Publishing Co., Inc., p 372
4. Bellman R (1957) Dynamic programming. Princeton University Press, Princeton
5. Lee EK (2007) Large-scale optimization-based classification models in medicine and biology. Ann Biomed Eng 35(6):1095–1 1109
6. Nasiri JA et al (2009) High dimensional problem optimization using distributed multi-agent PSO. In: Third UKSim European symposium on computer modeling and simulation, 2009. EMS '09
7. Larranaga P et al (2006) Machine learning in bioinformatics. Brief Bioinform 7(1):86–112
8. Levitsky V et al (2007) Effective transcription factor binding site prediction using a combination of optimization, a genetic algorithm and discriminant analysis to capture distant interactions. BMC Bioinform 8(481):1–20

9. Saeys Y, Inza I, Larrañaga P (2007) A review of feature selection techniques in bioinformatics. Bioinformatics 23(19):2507–2517
10. Ghalwash MF et al (2016) Structured feature selection using coordinate descent optimization. BMC Bioinform 17:158
11. Blanco R, Larrañaga P (2001) Selection of highly accurate genes for cancer classification by estimation of distribution algorithms. in: Workshop of Bayesian models in medicine. AIME 2001. 1–4 July. Cascais
12. Saeys Y et al (2004) Feature selection for splice site prediction: a new method using EDA-based feature ranking. BMC Bioinform 5(64):1–11
13. Armananzas R et al (2008) A review of estimation of distribution algorithms in bioinformatics. BioData Mining 1(6):1–12
14. Dittrich M et al (2008) Identifying functional modules in protein-protein interaction networks: an integrated exact approach. Bioinformatics 24(13):I223–I231
15. Xiao X et al (2003) Gene clustering using self-organizing maps and particle swarm optimization. In: Parallel and distributed processing symposium, 22–26 April. IEEE Computer Society
16. Maulik U, Bandyopadhyay S, Mukhopadhyay A (2011) Multiobjective genetic algorithms for clustering: applications in data mining and bioinformatics. Springer Science & Business Media
17. Gardeux V et al (2013) Optimization for feature selection in DNA microarrays. In: Heuristics: theory and applications. Nova Publishers
18. Handl J, Kell D, Knowles J (2007) Multiobjective optimization in bioinformatics and computational biology. IEEE/ACM Trans Comput Biol Bioinform 4(2):279–292
19. Shan S, Wang GG (2010) Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. Struct Multidiscip Optim 41(2):219–241
20. Regis R (2013) An initialization strategy for high-dimensional surrogate-based expensive black-box optimization. In: Zuluaga LF, Terlaky T (eds) Modeling and optimization: theory and applications. Springer, New York, pp 51–85
21. Hvattum LM, Glover F (2009) Finding local optima of high-dimensional functions using direct search methods. Eur J Oper Res 195(1):31–45
22. LaTorre A, Muelas S, Pena JM (2011) A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test. Soft Comput 15(11):2187–2199
23. Wang H, Wu ZJ, Rahnamayan S (2011) Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems. Soft Comput 15(11):2127–2140
24. Yang ZY, Tang K, Yao X (2011) Scalability of generalized adaptive differential evolution for large-scale continuous optimization. Soft Comput 15(11):2141–2155
25. Zhao S-Z, Suganthan PN, Das S (2010) Self-adaptive differential evolution with modified multi-trajectory search for CEC'2010 large scale optimization. In: Swarm, evolutionary, and memetic computing. Springer, Berlin, pp 1–10
26. Hedar A-R, Ali A (2012) Tabu search with multi-level neighborhood structures for high dimensional problems. Appl Intell 37(2):189–206
27. Stanarevic N (2012) Hybridizing artificial bee colony (ABC) algorithm with differential evolution for large scale optimization problems. Int J Math Comput Simul 6(1):194–202
28. You X (2010) Differential evolution with a new mutation operator for solving high dimensional continuous optimization problems. J Comput Inf Syst 6(9):3033–3039
29. Ros R, Hansen N (2008) A simple modification in CMA-ES achieving linear time and space complexity. In: Rudolph G et al (eds) Parallel problem solving from nature—PPSN X. Springer, Berlin, pp 296–305
30. Liao T, Montes de Oca MA (2011) Tuning parameters across mixed dimensional instances: a performance scalability study of Sep-G-CMA-ES. In: Proceedings of the 13th annual conference companion on genetic and evolutionary computation. ACM, Dublin, pp 703–706
31. Montes de Oca MA, Aydın D, Stützle T (2011) An incremental particle swarm for large-scale continuous optimization problems: an example of tuning-in-the-loop (re)design of optimization algorithms. Soft Comput 15(11):2233–2255
32. Masegosa AD, Pelta DA, Verdegay JL (2013) A centralised cooperative strategy for continuous optimisation: the influence of cooperation in performance and behaviour. Inf Sci 219(0):73–92
33. Li X, Yao X (2012) Cooperatively coevolving particle swarms for large scale optimization. IEEE Trans Evol Comput 16(2):210–224
34. Li X et al (2015) Editorial for the special issue of Information Sciences Journal (ISJ) on "Nature-inspired algorithms for large scale global optimization". Inf Sci 316:437–439
35. Tsurkov V (2001) Large-scale optimization. Applied optimization. Springer US
36. Liu L, Shao L, Li X (2015) Evolutionary compact embedding for large-scale image classification. Inf Sci 316:567–581
37. Miranda V, Martins J, Palma V (2014) Optimizing large scale problems with metaheuristics in a reduced space mapped by autoencoders-application to the wind-hydro coordination. IEEE Trans Power Syst 29(6):3078–3085
38. LaTorre A, Muelas S, Pena J (2015) A comprehensive comparison of large scale global optimizers. Inf Sci 316:517–549
39. Gardeux V et al (2009) Unidimensional search for solving continuous high-dimensional optimization problems. In: Ninth international conference on intelligent systems design and applications. ISDA '09. November 30–December 2, 2009. IEEE Computer Society, Pisa
40. Yang X-S, Koziel S (2011) Computational optimization and applications in engineering and industry, vol 359. Springer Science & Business Media
41. Conn AR, Scheinberg K, Vicente LN (2009) Introduction to derivative-free optimization, vol 8. SIAM, Philadelphia
42. Torczon V (1997) On the convergence of pattern search algorithms. SIAM J Optim 7(1):1–25
43. Kolda TG, Lewis RM, Torczon V (2003) Optimization by direct search: new perspectives on some classical and modern methods. SIAM Rev 45(3):385–482
44. Hooke R, Jeeves TA (1961) "Direct search" solution of numerical and statistical problems. J ACM 8(2):212–229
45. Lozano M, Molina D, Herrera F (2011) Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. Soft Comput 15(11):2085–2087
46. Glover F et al (1998) A template for scatter search and path relinking. In: Hao J-K (ed) Artificial evolution. Springer, Berlin, pp 1–51
47. Glover F (1995) Tabu thresholding: improved search by nonmonotonic trajectories. INFORMS J Comput 7(4):426–442
48. Gardeux V et al (2011) EM323: a line search based algorithm for solving high-dimensional continuous non-linear optimization problems. Soft Comput 15(11):2275–2285
49. Kirkpatrick S, Gelatt CD Jr, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
50. Omidvar MN, Li X, Yao X (2010) Cooperative co-evolution with delta grouping for large scale non-separable function optimization. In: IEEE congress on evolutionary computation (CEC 2010). 18–23 July. IEEE Computer Society, Barcelona
51. Tang K et al (2010) Benchmark functions for the CEC'2010 special session and competition on large scale global optimization. In: Nature inspired computation and applications laboratory, USTC, China: http://nical.ustc.edu.cn/cec10ss.php

52. Yang Z, Tang K, Yao X (2008) Large scale evolutionary optimization using cooperative coevolution. Inf Sci 178(15):2985–2999
53. Yang Z, Tang K, Yao X (2008) Multilevel cooperative coevolution for large scale optimization. In: IEEE congress on evolutionary computation (CEC 2008). June 1–6. IEEE Computer Society, Hong Kong
54. Korosec P, Tashkova K, Silc J (2010) The differential Ant-Stigmergy Algorithm for large-scale global optimization. In: IEEE congress on evolutionary computation (CEC 2010). 18–23 July. IEEE Computer Society, Barcelona
55. Wang H et al (2010) Sequential DE enhanced by neighborhood search for large scale global optimization. In: IEEE congress on evolutionary computation (CEC 2010). 18–23 July. IEEE Computer Society, Barcelona
56. Wang Y, Li B (2010) Two-stage based ensemble optimization for large-scale global optimization. In: IEEE congress on evolutionary computation (CEC 2010). 18–23 July. IEEE Computer Society, Barcelona
57. Molina D, Lozano M, Herrera F (2010) MA-SW-Chains: memetic algorithm based on local search chains for large scale continuous global optimization. In: IEEE congress on evolutionary computation (CEC 2010). 18–23 July. IEEE Computer Society, Barcelona
58. Brest J et al (2010) Large scale global optimization using self-adaptive differential evolution algorithm. In: IEEE congress on evolutionary computation (CEC 2010). 18–23 July. IEEE Computer Society, Barcelona
59. Zhao S-Z, Suganthan PN, Das S (2010) Dynamic multi-swarm particle swarm optimizer with sub-regional harmony search. In: IEEE congress on evolutionary computation (CEC 2010). 18–23 July. IEEE Computer Society, Barcelona
60. Brest J et al (2008) High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction. In: IEEE congress on evolutionary computation (CEC 2008). June 1–6. IEEE Computer Society, Hong Kong
61. Potter MA, Jong KAD (1994) A cooperative coevolutionary approach to function optimization. In: Proceedings of the international conference on evolutionary computation. The third conference on parallel problem solving from nature: parallel problem solving from nature. Springer, pp 249–257
62. Dorigo M, Birattari M (2010) Ant colony optimization. In: Encyclopedia of machine learning. Springer, pp 36–39
63. Kennedy J (2010) Particle swarm optimization. In: Encyclopedia of machine learning. Springer, pp 760–766
64. Vaz AIF, Vicente LN (2007) A particle swarm pattern search method for bound constrained global optimization. J Glob Optim 39(2):197–219
65. Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. Math Program 91(2):201–213
66. Ren Y, Wu Y (2013) An efficient algorithm for high-dimensional function optimization. Soft Comput 17(6):995–1004
67. García S et al (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. J Heuristics 15(6):617–644
68. Benjamini Y, Hochberg Y (1995) Controlling the false discovery rate—a practical and powerful approach to multiple testing. J R Stat Soc Ser B Methodol 57(1):289–300
69. Dass P et al (2015) Hybridisation of classical unidimensional search with ABC to improve exploitation capability. Int J Artif Intell Soft Comput 5(2):151–164
70. Jadon S, Bansal J, Tiwari R (2016) Escalated convergent artificial bee colony. J Exp Theor Artif Intell 28(1–2):181–200