Information Sciences 267 (2014) 140–157

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Linearized biogeography-based optimization with re-initialization and local search

Dan Simon [a,*], Mahamed G.H. Omran [b], Maurice Clerc [c]

[a] Department of Electrical and Computer Engineering, Cleveland State University, Cleveland, OH, United States
[b] Department of Computer Science, Gulf University for Science and Technology, Kuwait
[c] 204, route de la Nerulaz, 74570 Groisy, France

ARTICLE INFO

Article history:
Received 16 March 2013
Received in revised form 23 December 2013
Accepted 29 December 2013
Available online 16 January 2014

Keywords:
Biogeography-based optimization
Evolutionary algorithm
Memetic algorithm

ABSTRACT

Biogeography-based optimization (BBO) is an evolutionary optimization algorithm that uses migration to share information among candidate solutions. One limitation of BBO is that it changes only one independent variable at a time in each candidate solution. In this paper, a linearized version of BBO, called LBBO, is proposed to reduce rotational variance. The proposed method is combined with periodic re-initialization and local search operators to obtain an algorithm for global optimization in a continuous search space. Experiments have been conducted on 45 benchmarks from the 2005 and 2011 Congress on Evolutionary Computation, and LBBO performance is compared with the results published in those conferences. The results show that LBBO provides competitive performance with state-of-the-art evolutionary algorithms. In particular, LBBO performs particularly well for certain types of multimodal problems, including high-dimensional real-world problems. Also, LBBO is insensitive to whether or not the solution lies on the search domain boundary, in a wide or narrow basin, and within or outside the initialization domain.

1. Introduction

Evolutionary algorithms (EAs) have demonstrated their effectiveness over the last few decades as powerful optimizers for difficult, nonlinear, multimodal optimization problems [17]. EAs are generally, but not always, based on some natural process. Some popular EAs include particle swarm optimization (PSO) [27], differential evolution (DE) [65], ant colony optimization (ACO) [15], and genetic algorithms (GAs) [22]. EAs generally involve a collection of candidate solutions to some optimization problem. These candidate solutions are often called individuals, or simply solutions, and the collection of all the candidate solutions is called the population. Candidate solutions combine with each other and are also subject to random changes. The merging of candidate solutions is called recombination, and results in new candidate solutions. Random changes to candidate solutions are called mutations. Recombination and mutation create new solutions, and the EA thus progresses from one generation to the next in an attempt to find ever-improving solutions to a given problem.

One recently-developed EA is biogeography-based optimization (BBO), which, as its name indicates, is motivated by the mathematics of biogeography. Biogeography is the science and study of the migration of plant and animal life between habitats. Alfred Wallace and Charles Darwin were famous naturalists in the 19th century who conducted initial research in biogeography and introduced the subject to the scientific community [50]. Biogeography grew from a qualitative study to a

* Corresponding author. Tel.: +1 216 687 5407.
E-mail addresses: d.j.simon@csuohio.edu (D. Simon), omran.m@gust.edu.kw (M.G.H. Omran), maurice.clerc@writeme.com (M. Clerc).

0020-0255/$ - see front matter © 2014 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.12.048

quantitative study with the doctoral dissertation of Eugene Monroe in 1948 [11], and with a popular book authored by Robert MacArthur and Edward Wilson [40].

Biogeography motivated the development of biogeography-based optimization in 2008 [61]. Since then, BBO has been mathematically modeled as a Markov process [64] and as a dynamic system [62]. BBO has been successfully applied to many real-world problems, including robot control tuning [36], power system optimization [54], mechanical gear train design [57], satellite image classification [47], antenna design [59], image processing [51], prosthesis control optimization [69], and neuro-fuzzy system training for biomedical applications [46].

BBO has seen several improvements since it was first introduced. Ma and Simon [37] explored various migration curve shapes, which affect the selection pressure used for recombination. They also added blending to the BBO recombination logic, which resulted in the linear combination of independent variables during recombination [38]. Several researchers have hybridized BBO with other EAs, including DE [10], PSO [29], and oppositional learning [20].

However, in spite of these and other improvements, BBO still changes only one independent variable at a time in its candidate solutions. This is explained in more detail in Section 2, but the important point to note here is that this single-feature-migration property of BBO can result in poor performance on nonseparable problems. A nonseparable problem is one whose fitness depends on combinations of variables, rather than on individual variables. Many real-world problems are nonseparable and so this shortcoming of BBO must be addressed to make it more applicable. In this paper we modify BBO to obtain an algorithm called linearized BBO (LBBO) that is intended to improve BBO's performance, especially on nonseparable problems.

Section 2 gives an overview of standard BBO. Section 3 extends the BBO algorithm to LBBO and augments with the algorithm with several additional features, including local search and re-initialization. Section 4 discusses our experimental setup for the evaluation of LBBO and compares it with other state-of-the-art EAs. Section 5 presents a sensitivity study of the contributions of the various components of LBBO, and especially shows the importance of gradient descent (local search). Section 6 provides some concluding comments and suggestions for further research.

2. Biogeography-Based Optimization (BBO)

BBO is a population-based optimization method where each candidate solution is called a *habitat*. Each habitat has a habitat suitability index (HSI), which corresponds to the fitness of a solution. A good solution is like a habitat with a high HSI (a habitat with large number of species) while a bad solution is like a habitat with a small HSI (a habitat with small number of species). Good solutions tend to share their features with other solutions, while bad solutions are more likely to accept features from other solutions. This principle is motivated by natural biogeography, where high-population islands are more likely to emigrate species, and low-population islands are more likely to immigrate species [40]. Each solution y_k in BBO has two parameters, the immigration rate λ_k and emigration rate μ_k, where λ_k is inversely proportional to the fitness of y_k while μ_k is proportional to the fitness of y_k. Both λ_k and μ_k are defined on the domain [0, 1]. Thus, good solutions have low λ and high μ, while bad solutions have high λ and low μ. BBO consists of two main steps: migration and mutation.

2.1. Migration, or information sharing

For each solution feature $y_{k,s}$, the immigration rate λ_k is used to probabilistically decide whether or not to immigrate to that solution feature. This is described in Algorithm 1.

Algorithm 1. BBO migration decision. y_k is the kth candidate solution. r is a random number taken from a uniform distribution on $(0,1)$. $\lambda_k \in [0,1]$ is the immigration rate and is described in Eq. (3). $y_{k,s}$ is the sth solution feature (that is, the sth independent variable) of $y_{k,s}$.

$r \sim U(0,1)$
If $r < \lambda_k$ then
  Immigrate to $y_{k,s}$
else
  Do not immigrate to $y_{k,s}$
End if

Note that Algorithm 1 is performed for each solution feature index $s \in [1,n]$, where n is the problem dimension. If a decision is made by Algorithm 1 to immigrate to $y_{k,s}$, then the emigrating solution y_j is chosen probabilistically (e.g., using roulette wheel selection) using the emigration rates of the entire population:

$$\text{Prob}(\text{emigration from } y_j) = \frac{\mu_j}{\sum_{m=1}^{N} \mu_m} \text{ for all } j \in [1,N] \tag{1}$$

where N is the population size. Migration is defined by

$$y_{k,s} \leftarrow y_{j,s} \tag{2}$$

where s is a solution feature (that is, one component of a solution). There are several migration models for λ and μ (e.g., linear, quadratic, sinusoidal and generalized sinusoidal). According to [37], the generalized sinusoidal model generally performs better than the other models on 23 benchmark functions. Hence, the generalized sinusoidal model is used in this study:

$$\begin{aligned} \lambda_k &= \frac{1}{2}(\cos(\pi f_k + \beta) + 1) \\ \mu_k &= \frac{1}{2}(-\cos(\pi f_k + \beta) + 1) \end{aligned} \tag{3}$$

where fitness of y_k, denoted as f_k, is normalized to $[0,1]$, and $\beta = -\pi/2$, as recommended in [37]. The normalization is done using rank-based fitness. Thus, for a population size of N, the fitness values are normalized to $1/(N+1), 2/(N+1), \ldots, N/(N+1)$.

2.2. Mutation and Elitism

The mutation operator randomly modifies a solution feature. Mutation adds diversity to the population. In this study, mutation is done for each feature of each solution as described in Algorithm 2.

Algorithm 2. BBO mutation. $y_{k,s}$ is the sth independent variable in the kth candidate solution. $r \sim U(0,1)$ – that is, r is a random number taken from a uniform distribution on $(0,1)$. $p_m \in [0,1]$ is the user-specified mutation rate, and L_s and U_s are the minimum and maximum allowed values for feature s, respectively.

```
If r < p_m
  y_{k,s} ← U(L_s, U_s)
End if
```

One iteration of BBO is described in Algorithm 3 [37]. Migration and mutation of the population take place before any solution is replaced, which requires the use of temporary population Z in the algorithm. In addition, elitism is typically used so that the best two solutions are kept from one generation to the next. Note that elitism could also involve fewer than, or more than, two solutions each generation. A higher number of elites encourages exploitation, while fewer elites encourages exploration. We do not study the effect of the number of elites on BBO performance; we use two elites as a good balance between exploration and exploitation.

Algorithm 3. One iteration of the BBO algorithm. We use N to denote the population size. The population Y contains the candidate solutions y_k for $k \in [1,N]$.

```
For each solution y_k ∈ Population Y, define λ_k and μ_k using Eq. (3)
Z ← Y
For each solution z_k ∈ Z
  For each solution feature s
    Use λ_k to probabilistically decide whether to immigrate to z_k (Algorithm 1)
    If immigrating then
      Use {μ_i} (i = 1,...,N) to probabilistically select the emigrating solution y_j
      z_{k,s} ← y_{j,s}
    End if
    Probabilistically decide whether to mutate z_{k,s} (Algorithm 2)
  Next solution feature
Next solution
Y ← Z
```

3. Linearized BBO (LBBO) with local search and re-initialization

This section introduces several new components to BBO. One drawback of BBO is that it treats each solution feature independently – that is, it is not rotationally invariant. An algorithm is rotationally invariant if its performance on an objective

function is independent of the rotation of the objective function [72]. BBO's rotational variance means that it generally performs poorly when applied to nonseparable functions. To address this drawback, BBO migration is linearized in Section 3.1 to make it more rotationally invariant. Note that perfect rotational invariance is not possible unless the search space is a hypersphere [12].

Another weakness of BBO is its local search ability, and so we describe the addition of gradient descent to BBO in Section 3.2. Next, since many real-world optimization solutions lie on constraint boundaries, we add boundary search in Section 3.3. Next, in order to systematically cover the search space, we add a global grid search strategy in Section 3.4. Next, in order to systematically cover the search space in a region near the current best individual, we add a Latin hypercube search strategy in Section 3.5. We include re-initialization and restart strategies in Sections 3.6 and 3.7. Finally, Section 3.8 summarizes the entire LBBO algorithm and its tuning parameters.

3.1. LBBO migration

For each solution, z_k, the immigration rate λ_k is used to probabilistically decide whether to immigrate or not. If we decide to immigrate, ψ emigrating solutions are probabilistically chosen using their emigration rates, where $\psi \in [1,n]$ is a uniformly distributed random parameter. The solution z_k is linearly combined with the ψ emigrating solutions such that z_k moves towards each emigrating solution y_j in an amount that is proportional to its emigration rate μ_j:

$$z_k \leftarrow z_k + \mu_j(y_j - z_k) \tag{4}$$

Thus, an immigrating solution moves toward ψ emigrating solutions with an amount of change that is proportional to the fitness rank of the emigrating solutions (as determined by μ). The linearized migration method is described in Algorithm 4.

Algorithm 4. The LBBO migration step. This migration step replaces the standard BBO migration loop in Algorithm 3. See Section 3.1 for a discussion.

```
For each solution z_k
  Use λ_k to probabilistically decide whether to immigrate to z_k
  If immigrating then
    ψ ← uniformly distributed random integer ∈ [1,n]
    For i = 1 to ψ
      Use {μ_i} (i = 1,...,N) to probabilistically select the emigrating solution y_j
      z_k ← z_k + μ_j (y_j − z_k)
    Next i
  End if
  Probabilistically decide whether to mutate z_{k,s}
Next solution
```

3.2. Gradient descent

LBBO is augmented in this paper with several local search operators to improve its performance as it nears the global optimum. LBBO, like many EAs, is primarily intended for global search. It is therefore effective at finding the neighborhood of the global optimum, but has difficulty in homing in on the exact optimum. We implement gradient descent as shown in Algorithm 5.

Algorithm 5. Local search using gradient descent. See Section 3.2 for a discussion.

```
If FE > α FE_max or (f_min(g) − f_min(g + 1))/f_min(g) < ε_1 then
  Perform gradient descent on the N_g best individuals
End if
```

Algorithm 5 shows that gradient descent is activated under two conditions.

1. FE is the current number of function evaluations that have been performed so far, and FE_{max} is the maximum function evaluation limit. $\alpha \in [0,1]$ is a factor that determines when gradient descent is activated. We typically use $\alpha = 1/2$ so that gradient descent is activated whenever we have used up 50% or more of our allotted function evaluations.

2. $f_{\min}(g)$ is the minimum function value obtained by LBBO during the gth generation. The quantity $(f_{\min}(g) - f_{\min}(g+1))/f_{\min}(g)$ indicates the relative improvement in the best function value found by LBBO from the gth generation to the $(g+1)$st generation. ε_1 is a threshold that determines when gradient descent is activated. This condition assumes that the cost value $f(\cdot)$ is positive for all candidate solutions. We typically use $\varepsilon_1 = 0.1$ so that gradient descent is activated whenever the best individual in the population improves by less than 10% from one generation to the next.

Algorithm 4 shows that we implement gradient descent on the N_g best individuals. We typically use $N_g = 2$. We implement gradient descent with the interior point algorithm, in particular the MATLAB® Optimization Toolbox™ implementation **fmincon**. The tuning parameters are set to default values, or are based on common-sense rules of thumb.

- Termination tolerance on the function value: **TolFun** = $e/100$, where e is the admissible function error that defines optimization success (see Section 4).
- Maximum allowable function evaluations: **MaxIter** = 1000.
- Termination tolerance on the independent variable: **TolX** = 10^{-8}.
- Typical values for the independent variable: **TypicalX** = $(U - L)/100$, where U and L are the upper and lower limits of the search domain, respectively.
- Finite difference method: **FinDiffType** = 'central' – that is central differences are used for gradient estimation.
- Gradient descent algorithm: **Algorithm** = 'interior-point'.

3.3. Boundary search

Many real-world optimization problems have their solution on the boundary of the search space. This is not surprising because, for example, we normally expect to obtain the best engineering design, allocation of resources, or other optimization goal, by using all of the available energy, or force, or some other resource [9]. This idea has given rise to the approach of searching the constraint boundary for the solution to constrained optimization problems [32].

We implement boundary search in LBBO as follows. If any of the dimensions of the best individual in the population are within a certain threshold of the search space boundary, then we move that dimension to the search space boundary and perform local search (gradient descent) on the other dimensions. This idea is shown in Algorithm 6.

Algorithm 6. Boundary search combined with gradient descent. See Section 3.3 for a discussion of the algorithm and a description of the parameters.

```
If (f_min(g) − f_min(g + 1))/f_min(g) < ε_2 then
  For k = 1 to N_s
    For each solution feature s
      If (U_s − z_{k,s}) < α_s(U_s − L_s) then
        z_{k,s} ← U_s
      End if
      If (z_{k,s} − L_s) < α_s(U_s − L_s) then
        z_{k,s} ← L_s
      End if
    Next solution feature
    Perform gradient descent on unbounded dimensions of z_k
  Next k
End if
```

Algorithm 6 shows that boundary search is implemented under similar conditions as gradient descent in Algorithm 5. That is, boundary search is implemented whenever the best individual in the population improves by a factor of less than ε_2 from one generation to the next. We typically set ε_2 equal to the computer precision so that boundary search is implemented only if the best individual does not improve at all from one generation to the next. We implement boundary search for the best N_s individuals, and we typically use $N_s = 2$. If any independent variable of the 2 best individuals is within a factor of α_s of the upper boundary of the search domain, we set that independent variable equal to the upper boundary. Similarly, if any independent variable of the 2 best individuals is within a factor of α_s of the lower boundary of the search domain, we set that independent variable equal to the lower boundary. Then we perform gradient descent on those individuals. However, we only perform gradient descent on dimensions that are not equal to a search space boundary. The default value of α_s is 0.01.

3.4. Global grid search

The next type of search that we implement is a global grid search. This search systematically covers the search space, and is shown in Algorithm 7.

Algorithm 7. Global grid search. See Section 3.4 for a discussion.

If $(f_{\min}(g) - f_{\min}(g+1))/f_{\min}(g) < \varepsilon_3$ then
  $\Delta = \alpha_o(U_s - L_s)$
  For $k = 1$ to N_o
    $w_k = z_k$
    For $s = 1$ to D
      While $w_{k,s} \geqslant L_s$
        $z_k \leftarrow \text{argmin}\ \{f(z_k), f(w_k)\}$
        $w_{k,s} \leftarrow w_{k,s} - \Delta$
      Loop
    Next s
    $w_k = z_k$
    For $s = 1$ to D
      While $w_{k,s} \leqslant U_s$
        $z_k \leftarrow \text{argmin}\ \{f(z_k), f(w_k)\}$
        $w_{k,s} \leftarrow w_{k,s} + \Delta$
      Loop
    Next s
  Next k
End if

Algorithm 7 shows that global grid search is implemented under similar conditions as gradient descent in Algorithm 5 and boundary search in Algorithm 6. That is, global grid search is implemented whenever the best individual in the population improves by a factor of less than ε_3 from one generation to the next. We typically set ε_3 equal to the computer precision so that global grid search is implemented only if the best individual does not improve at all from one generation to the next. We implement global grid search for the best N_o individuals, and we typically use $N_o = 2$.

Algorithm 7 shows that for the best N_o individuals, we increment or decrement each independent variable by a specific fraction α_o of the search space size. We typically use $\alpha_o = 0.1$. With this setting, global grid search decreases the value of a given dimension of z_k by an increment equal to 10% of the search space size, one increment at a time, until the dimension reaches the lower boundary of the search space. Global grid search then increases the value of a given dimension until it reaches the upper bound of the search space. Global grid search performs this process for each dimension, and replaces the individual with the best value that it finds.

3.5. Latin hypercube search

Latin hypercube sampling divides a domain into intervals in each dimension, and then places sample points in such a way that each interval in each dimension contains only one sample point [63], Example 21.2. This idea is illustrated in Fig. 1.

Latin hypercube sampling can sometimes capture the unpredictable, unknown nature of a function better than uniform sampling. Also, Latin hypercube sampling is more efficient than uniform sampling. With uniform sampling of a D-dimensional search space where each dimension is divided into n intervals, we need n^D sample points; but with Latin hypercube sampling, we need only n sample points.

We perform Latin hypercube sampling every G_L generations around the current best individual in the population if we are getting close to the optimum. We perform this search each generation under the following conditions. Both conditions must be satisfied before we perform Latin hypercube sampling.

1. The best individual in the population has a cost that is less than βe, where e is the function value required for success (see Section 4), and β is a scale factor. Note that this requires that we know *a priori* what function value is required for success. Although this is not always the case, in real-world problems we often know the target value of our optimization problem ahead of time. We typically set $G_L = 10$ and $\beta = 1000$.
2. The best individual in the population is not improving sufficiently fast. That is,

$$(f_{\min}(g - G_L) - f_{\min}(g))/f_{\min}(g - GL) < \varepsilon_4 \tag{5}$$

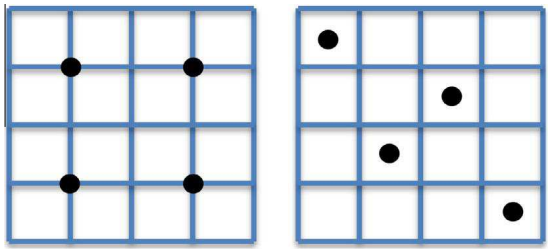

Fig. 1. The figure on the left illustrates uniform sampling of four points in a two-dimensional search space. The figure on the right illustrates Latin hypercube sampling of four points in a two-dimensional search space.

where $f_{\min}(g)$ is the cost function value of the best individual in the population, and ε_4 is a relative tolerance. We typically set $\varepsilon_4 = e$, where e is the function value required for success (see Section 4).

We perform a Latin hypercube search within a domain of size $\alpha_L(U - L)$ that is centered at the best individual in the population, where U and L are the upper and lower search space bounds, and α_L defines the relative size of the hypercube within which we search. We typically set $\alpha_L = 1/50$. We divide each dimension of the search space into n evenly-spaced points within the search range, and then find n search points within the search range. We typically set $n = 1000$. We then perform gradient descent on the best n_L of those individuals, where we typically set $n_L = 10$. We combine these individuals with the N-member population to obtain a temporary population size of $N + n_L$. We then use the best N of these individuals as the population of the next generation.

3.6. Re-initialization

Every N_r function evaluations, we perform a partial re-initialization. Given that the population size is N, we generate N new random individuals, along with two individuals at each extreme of the search domain. This gives us a temporary population size of $2N + 2$. We then select the best N individuals out of these $2N + 2$ individuals for the next generation. We typically set $N_r = 1000$. With a population size of 50, this typically works out to once every 20 generations.

3.7. Restart, and LBBO tuning parameters

If the population fails to improve after all of the search strategies, we start over. That is, we discard the entire population and restart with a randomly-generated population.

The LBBO algorithm, including all of the new components discussed in the preceding sections, is summarized in Algorithm 8. As with standard BBO, elitism is typically used where the best two solutions are kept from one generation to the next (see Section 2.2). Each method in Algorithm 8 executes regardless of the success or failure of the previous methods in the algorithm. Note that the performance of LBBO might be affected if the methods in Algorithm 8 are implemented in a different order; however, we did not conduct any experiments to confirm this. We suggest this as an area for future research.

Algorithm 8. One iteration of the LBBO algorithm.

For each solution y_k, define λ_k and μ_k based on fitness rank $f(y_k)$, where $f(y_k) \in [0,1]$
$Z \leftarrow Y$
Use LBBO migration to update Z (Section 3.1)
Apply gradient descent if needed (Section 3.2)
Apply boundary search if needed (Section 3.3)
Apply global grid search if needed (Section 3.4)
Apply Latin hypercube search if needed (Section 3.5)
Re-initialize if needed (Section 3.6)
Restart if needed (Section 3.7)
$Y \leftarrow Z$

3.8. Summary of LBBO parameters

We see from the preceding sections that LBBO includes many tuning parameters. We did not attempt to optimize the tuning parameters, but instead we mostly used default parameters. We do not study any tradeoffs on these parameters in this

paper. However, we do note that BBO tuning parameters have been studied in other papers [39], and that self-adaptive versions of BBO have also been proposed [23]. Future LBBO research could involve self-adaptation. The LBBO tuning parameters are summarized as follows.

- Migration (Section 3.1) – no tuning parameters.
- Gradient descent (Section 3.2).
  - Function evaluation threshold α.
  - Cost function improvement threshold ε_1.
  - Number of individuals N_g.
  - Termination tolerance on the function value.
  - Maximum allowable function evaluations.
  - Termination tolerance on the independent variable.
  - Typical values for the independent variable.
  - Finite difference method.
  - Gradient descent algorithm.
- Boundary search (Section 3.3).
  - Cost function improvement threshold ε_2.
  - Number of individuals N_s.
  - Boundary threshold α_s.
- Global grid search (Section 3.4).
  - Cost function improvement threshold ε_3.
  - Number of individuals N_o.
  - Search space fraction α_o.
- Latin hypercube search (Section 3.5).
  - Scale factor threshold β.
  - Generation increment G_L.
  - Cost function improvement threshold ε_4.
  - Relative size of hypercube α_L.
  - Number of search points n.
  - Number of gradient descent points n_L.
- Re-initialization (Section 3.6).
  - Function evaluation increment N_r.
- Restart (Section 3.7) – no tuning parameters.

4. Simulation results

This section presents LBBO simulation results. Section 4.1 discusses the simulation setup and the benchmark problems from the 2005 Congress on Evolutionary Computation (CEC). Section 4.2 presents results for 10-dimensional benchmarks, and Section 4.3 presents results for 30-dimensional benchmarks. Section 4.4 discusses and summarizes the 2005 CEC results. Section 4.5 presents results for real-world benchmark problems from the 2011 CEC.

4.1. Simulation Setup for the 2005 CEC Benchmarks

We test the performance of the proposed LBBO method on the 23 non-noisy benchmark functions from the 2005 CEC ($F_1 - F_{25}$, excluding noisy benchmarks F_4 and F_{17}). We do not test on the noisy benchmark functions because we have not included any noise-handling capabilities in LBBO. For details about these functions, the reader is referred to [66].

We limit each simulation to 10,000n function evaluations (FEs), where n is the problem dimension (either 10 or 30). We use a population size of 50, we run 25 Monte Carlo simulations (i.e. independent runs) for each benchmark, we use a mutation probability $p_m = 0.01$, and the LBBO parameter ψ is an integer that is randomly distributed between 1 and n for each migration (see Algorithm 4). We implement elitism by retaining the top two solutions for the next generation. We implement the LBBO code in MATLAB, and we generate random numbers with the default random number generator, which is the Mersenne Twister algorithm [42].

Benchmark functions $F_1 - F_6$ are unimodal problems,[1] and $F_7 - F_{25}$ are multimodal problems. $F_1 - F_7$, $F_9 - F_{12}$, and F_{15} have known solutions. F_8, $F_{13} - F_{14}$, and $F_{16} - F_{25}$ are unsolved. Problems $F_1 - F_5$ are considered solved if we come within 10^{-6} of the global minimum. Problems F_6, F_7, $F_9 - F_{12}$, and F_{15} are considered solved if we come within 10^{-2} of the global minimum.

We compared LBBO to the algorithms that were accepted for the 2005 CEC competition, in addition to other recently published algorithms.

[1] Note that F_6, which is a shifted Rosenbrock function, is multimodal for dimensions greater than 3. However, in order to retain the same groupings as in previous comparisons of the 2005 CEC benchmarks [25], we include F_6 with the unimodal functions.

1. BLX-GL50, which is a two-sex GA with unique crossover operators [21].
2. BLX-MA, which is an adaptive memetic algorithm [43].
3. CoEvo, which is a co-evolutionary algorithm [48].
4. DE, which is differential evolution [53].
5. DMS-L-PSO, which is a multi-swarm particle swarm method [34].
6. EDA, which is an estimation of distribution algorithm [73].
7. G-CMA-ES, which is a covariance matrix adaptation evolution strategy [4].
8. K-PCX, which is an amalgamation of various EA strategies [60].
9. L-CMA-ES, which is another covariance matrix adaptation evolution strategy [5].
10. L-SaDE, which is an adaptive differential evolution algorithm [49].
11. SPC-PNX, which is a continuous genetic algorithm [6].
12. PS-CMA-ES, which is a combination of particle swarm optimization, covariance matrix adaptation, and evolution strategy [44].
13. EvLib, which is a self-adaptive algorithm that combines a variety of EAs [8].
14. FEA, which is a flexible EA that self-adapts its crossover and mutation operators [1].
15. PLES, which is a parameter-less evolution strategy [13].
16. STS, which is a combination of scatter search and tabu search [16].
17. RMA, which is a region-based memetic algorithm [30].
18. CMA-GA-PSO, which is a hybrid of covariance matrix adaptation, genetic algorithm, and particle swarm optimization [70].
19. IPOP-CMA-ES, which is a variant of the CMA-ES algorithm that uses a varying population size [35].
20. ADE, which is an adaptive differential evolution algorithm [26].

We collected benchmark performance data for these EAs from the above references and from [25]. Some of the above references do not include data for every benchmark. In the following sections, we only report results that are reported in the references. As in the 2005 CEC competition, we rank the algorithms based on the number of problems that they solve at least once out of 25 Monte Carlo simulations. In case of a tie, the algorithm that is successful more often is better.

4.2. 10-Dimensional results for the 2005 CEC benchmarks

Table 1 shows the results of the EAs on the 10-dimensional unimodal problems. The "# of solved functions" column shows how many of the benchmark problems the algorithm solved at least once out of 25 Monte Carlo simulations. The "success rate" column shows the percentage of simulations that successfully solved a problem, out of a total of (25×5) simulations.

Each cell in Table 1 corresponds to a given algorithm and a given benchmark function, and contains two numbers whose meaning depends on whether or not the algorithm successfully solved the benchmark.

- If the algorithm was successful in solving the problem at least once out of 25 simulations, then the number of successes is in round parentheses. The number outside of the parentheses is the average number of function evaluations required to achieve success divided by the success rate for that benchmark, normalized to the best CEC 2005 algorithm. For example,

Table 1
Comparison between LBBO and other EAs on five 10-dimensional unimodal problems. The algorithms are listed in order from best to worst. See the text for a more detailed description of this table. LBBO performance is shown in bold font.

	# of solved functions	Success rate (%)	F_1	F_2	F_3	F_5	F_6
G-CMA-ES	5	100	1.6 (25)	1.0 (25)	1.0 (25)	1.0 (25)	1.5 (25)
L-CMA-ES	5	100	1.7 (25)	1.7 (25)	1.1 (25)	1.0 (25)	1.3 (25)
CMA-GA-PSO	5	100	1.7 (25)	1.1 (25)	1.2 (25)	1.2 (25)	1.5 (25)
EDA	5	96	10 (25)	4.6 (25)	2.5 (23)	4.2 (25)	9.6 (22)
DMS-L-PSO	5	96	12.0 (25)	5.0 (25)	1.8 (25)	18.6 (20)	7.7 (25)
DE	5	95	29 (25)	19.2 (25)	18.5 (20)	6.9 (25)	6.6 (24)
LBBO	**5**	**90**	**1.7 (25)**	**0.9 (25)**	**0.4 (25)**	**18.1 (13)**	**1.3 (25)**
BLX-GL50	4	80	19.0 (25)	17.1 (25)	[12] 571	4.7 (25)	7.3 (25)
SPC-PNX	3	60	6.7 (25)	12.5 (25)	[15] 108060	6.8 (25)	[13] 18.9
CoEVO	3	60	23.0 (25)	11.3 (25)	6.8 (25)	[13] 2.13	[12] 12.5
PS-CMA-ES	3	60	21.4 (25)	13.9 (25)	[11] 1.59	[16] 68.0	11.4 (25)
L-SaDE	5	59	10.0 (25)	4.2 (25)	8.0 (16)	[11] 0.01	6.8 (25)
K-PCX	3	58	1.0 (25)	1.0 (25)	[10] 0.42	[15] 48.5	1.0 (22)
EvLib	3	53	6.7 (25)	1.9 (25)	[13] 6090	[14] 10.8	11.5 (16)
PLES	3	41	6.0 (25)	16.0 (25)	[16] 118060	166 (1)	[14] 30.6
BLX-MA	2	40	12.0 (25)	15.4 (25)	[14] 47,710	[12] 0.02	[11] 1.49
FEA	2	38	18.5 (25)	144 (23)	[17] 646000	[17] 351	[15] 332

consider F_6. K-PCX was the best performing algorithm for F_6. It solved F_6 22 out of 25 times, with an average of 7053 function evaluations in those 22 successful runs. On the other hand, LBBO solved F_6 25 times, with an average of 10,499 function evaluations. Therefore, the number outside of the parentheses in the LBBO F_6 cell in Table 1 is

$$\frac{10499/(25/25)}{7053/(22/25)} = 1.3$$

Note that these values are less than 1 for LBBO functions F_2 and F_3 because LBBO performed better than the best 2005 CEC algorithm for those functions.

- If the algorithm was *not* successful even once in solving the benchmark, then the number in square brackets indicates the performance rank of the algorithm for that particular problem, out of all the algorithms listed in the table. In this case, the number outside of the square brackets shows the normalized function value that was achieved by the algorithm, averaged over 25 Monte Carlo simulations.

Table 1 shows that LBBO is able to solve all five of the unimodal benchmarks; however, six other algorithms are also able to solve all of the unimodal benchmarks, and all six of them perform better, on average, than LBBO. On the other hand, LBBO is the best algorithm for the F_2 and F_3 benchmarks.

Table 2 shows the results of the EAs on the solved multimodal problems. LBBO ranks 7th best out of the 17 algorithms. LBBO is the best algorithm for the F_9 benchmark.

Table 3 shows the results of the EAs on the most difficult 2005 CEC benchmarks: unsolved multimodal problems. LBBO ranks 5th best out of the 19 algorithms. LBBO is the best algorithm for the F_8 benchmark (tied with six other algorithms), and it is also the best algorithm for the F_{25} benchmark.

4.2.1. Summary of 10-dimensional benchmark results

Tables 1–3 show that for the 10-dimensional benchmarks, LBBO ranks 7th best on the unimodal problems, 7th best on the solved multimodal problems, and 5th best on the unsolved problems. Combining the ranks of the algorithms that appear in all three of these tables, we see that overall, LBBO ranks 4th best out of the 16 algorithms.

4.3. 30-Dimensional results for the 2005 CEC benchmarks

Table 4 shows results for the 30-dimensional unimodal problems for all of the EAs for which we have results. For a more detailed discussion of the meaning of the data in the table, see the beginning of Section 4.2. LBBO ranks 4th best out of 13 algorithms on the unimodal benchmarks. LBBO is able to solve all five of the unimodal benchmarks. Also, LBBO is the best algorithm for the F_1, F_2, F_3, and F_6 benchmarks.

Table 5 shows the results of the EAs on the solved 30-dimensional multimodal problems for all of the EAs for which we have results. LBBO ranks 3rd out of the 12 algorithms. LBBO is the best algorithm for the F_9, F_{12}, and F_{15} benchmarks.

Table 6 shows the results of the EAs on the 12 unsolved 30-dimensional multimodal problems for all of the EAs for which we have results. LBBO ranks 12th out of the 15 algorithms.

Table 2
Comparison between LBBO and other EAs on six solved 10-dimensional multimodal problems. The algorithms are listed in order from best to worst. See the text for a more detailed description of this table. LBBO performance is shown in bold font.

	# of solved functions	Success rate (%)	F_7	F_9	F_{10}	F_{11}	F_{12}	F_{15}
CMA-GA-PSO	6	96	1.0 (25)	2.6 (25)	1.1 (24)	0.1 (25)	2.1 (25)	2.8 (20)
PS-CMA-ES	6	75	5.1 (25)	0.4 (25)	0.2 (25)	0.4 (10)	3.4 (25)	27.2 (2)
G-CMA-ES	5	63	1.0 (25)	4.5 (19)	1.2 (23)	1.4 (6)	4.0 (22)	[10] 28.0
DE	5	30	255 (2)	10.6 (11)	[16] 36.0	1.0 (12)	8.8 (19)	75.8 (1)
L-SaDE	4	53	36.2 (6)	1.0 (25)	[6] 5.0	[11] 4.9	3.9 (25)	1.0 (23)
DMS-L-PSO	4	47	126 (4)	2.1 (25)	[5] 3.6	[10] 4.6	6.6 (19)	1.7 (22)
LBBO	**4**	**45**	**17.4 (20)**	**0.2 (25)**	**[11] 14.4**	**[13] 6.1**	**6.5 (23)**	**1.0 (17)**
K-PCX	3	40	[14] 0.23	2.9 (24)	1.0 (22)	[14] 6.7	1.0 (14)	[17] 510
EvLib	3	33	[17] 1270	0.1 (25)	[12] 18.6	[12] 5.2	35.3 (5)	1.7 (19)
BLX-GL50	3	17	12.3 (9)	10.0 (3)	[6] 5.0	[7] 2.3	12.1 (13)	[16] 400
EDA	3	9	404 (1)	[14] 5.4	[8] 5.3	2.9 (3)	4.3 (10)	[14] 365
FEA	2	28	[15] 1.84	0.5 (25)	[13] 23.4	[15] 7.0	[16] 271	0.4 (17)
L-CMA-ES	2	25	1.2 (25)	[17] 44.9	[17] 40.8	[8] 3.7	11.6 (12)	[11] 211
BLX-MA	2	15	[13] 0.20	5.7 (18)	[9] 5.6	[9] 4.6	[14] 74.3	8.5 (5)
SPC-PNX	2	1	383 (1)	[13] 4.0	[10] 7.3	5.8 (1)	[15] 260	[12] 254
PLES	1	1	[16] 4.09	[15] 16.7	[14] 25.6	[17] 9.5	182 (1)	[15] 380
CoEVO	0	0	[12] 0.037	[16] 19.2	[15] 26.8	[16] 9.0	[17] 605	[13] 294

Table 3
Comparison between LBBO and other EAs on 12 unsolved 10-dimensional multimodal problems. The algorithms are listed in order from best to worst. See the text for a more detailed description of this table. LBBO performance is shown in bold font.

	Ave. rank	F_8	F_{13}	F_{14}	F_{16}	F_{18}	F_{19}	F_{20}	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}
G-CMA-ES	4.3	[1] 20.0	[11] 0.70	[11] 3.01	[4] 91	[2] 332	[3] 326	[1] 300	[4] 500	[6] 729	[1] 559	[1] 200	[6] 374
IPOP-CMA-ES	4.3	[9] 20.2	[12] 0.71	[1] 2.03	[2] 89	[3] 360	[2] 320	[3] 340	[4] 500	[5] 728	[1] 559	[1] 200	[9] 403
L-SaDE	6.8	[1] 20.0	[2] 0.22	[10] 2.92	[8] 101	[11] 719	[11] 705	[11] 713	[2] 464	[8] 735	[9] 664	[1] 200	[7] 376
DMS-L-PSO	7.8	[1] 20.0	[4] 0.37	[5] 2.36	[6] 95	[13] 761	[12] 714	[16] 822	[8] 536	[4] 692	[11] 730	[10] 224	[4] 366
LBBO	**8.0**	**[1] 20.0**	**[5] 0.40**	**[14] 3.51**	**[13] 134**	**[10] 511**	**[7] 516**	**[8] 538**	**[7] 506**	**[11] 748**	**[10] 712**	**[9] 220**	**[1] 216**
PS-CMA-ES	8.0	[12] 20.3	[5] 0.40	[13] 3.47	[3] 90	[6] 472	[6] 467	[7] 494	[9] 557	[1] 587	[8] 643	[17] 403	[9] 403
BLX-GL50	8.1	[15] 20.4	[13] 0.75	[3] 2.17	[5] 94	[4] 420	[5] 449	[6] 446	[14] 689	[14] 759	[6] 639	[1] 200	[11] 404
DE	8.2	[15] 20.4	[17] 1.81	[15] 3.52	[18] 173	[1] 300	[1] 300	[1] 300	[4] 500	[7] 734	[1] 559	[1] 200	[17] 928
EvLib	8.8	[1] 20.0	1 (10)	[7] 2.81	[16] 157	[9] 500	[10] 666	[9] 585	[11] 638	[17] 792	[5] 600	[1] 200	[18] 1757
EDA	8.9	[12] 20.3	[18] 1.84	[6] 2.63	[14] 144	[7] 483	[9] 564	[10] 652	[3] 484	[15] 771	[7] 641	[1] 200	[5] 373
SPC-PNX	9.4	[19] 21.0	[15] 0.84	[12] 3.05	[12] 110	[5] 440	[4] 380	[4] 440	[13] 680	[12] 749	[4] 576	[1] 200	[12] 406
L-CMA-ES	9.5	[1] 20.0	[8] 0.49	[18] 4.01	[11] 105	[8] 497	[7] 516	[5] 442	[1] 404	[9] 740	[12]791	[19] 865	[15] 442
BLX-MA	10.8	[9] 20.2	[14] 0.77	[1] 2.03	[10] 102	[16] 803	[15] 763	[14] 800	[15] 722	[3] 671	[15] 927	[10] 224	[8] 396
K-PCX	10.8	[1] 20.0	[10] 0.65	[4] 2.35	[7] 96	[12] 752	[14] 751	[15] 813	[18] 1050	[2] 659	[17] 1060	[18] 406	[12] 406
STS	11.0	[9] 20.2	[7] 0.45	[9] 2.87	[8] 101	[14] 772	[13] 730	[12] 716	[12] 656	[13] 758	[14] 864	[10] 224	N/A
RMA	12.1	[15] 20.4	[9] 0.63	[8] 2.84	[1] 84	[15] 779	[15] 763	[13] 751	[16] 747	[10] 742	[16] 931	[13] 236	[14] 410
FEA	14.1	[8] 20.1	[3] 0.32	[16] 3.53	[17] 158	[18] 909	[18] 900	[18] 951	[17] 1020	[18] 846	[19] 1130	[15] 289	[2] 256
CoEVO	14.4	[12] 20.3	[16] 1.14	[17] 3.71	[19] 177	[17] 902	[17] 845	[17] 863	[10] 635	[16] 779	[13] 835	[16] 314	[3] 257
PLES	17.6	[15] 20.4	[19] 8.7	[19] 4.14	[15] 147	[19] 1015	[19] 1002	[19] 999	[19] 1079	[19] 880	[18] 1114	[14] 282	[16] 692

Table 4
Comparison between LBBO and other EAs on five 30-dimensional unimodal problems. The algorithms are listed in order from best to worst. See the text for a more detailed description of this table. LBBO performance is shown in bold font.

	# of solved functions	Success rate (%)	F_1	F_2	F_3	F_5	F_6
G-CMA-ES	5	100	1.7 (25)	1.1 (25)	1 (25)	1 (25)	1 (25)
L-CMA-ES	5	100	1.8 (25)	1.2 (25)	1 (25)	1.1 (25)	1.1 (25)
CMA-GA-PSO	5	99	1.7 (25)	1.2 (25)	1.2 (25)	1.5 (24)	3.3 (25)
LBBO	**5**	**83**	**0.99 (25)**	**0.4 (25)**	**0.2 (25)**	**[4] 1.7**	**0.8 (25)**
DMS-L-PSO	4	76	1.9 (25)	10.8 (25)	7.9 (21)	N/A	5.5 (24)
EDA	3	60	55.6 (25)	13.3 (25)	5.1 (25)	[5] 0.055	[10] 21.1
BLX-GL50	3	60	21.5 (25)	13.3 (25)	[9] 3112	[7] 332.6	3.7 (25)
K-PCX	3	51	1 (25)	1 (25)	[7] 57.9	[8] 2040	1.1 (14)
DE	2	40	51.9 (25)	20.5 (25)	[10] 3E5	[6] 235	[9] 3.77
SPC-PNX	3	38	11.1 (25)	26.7 (22)	[12] 1E6	[10] 4237	86.7 (1)
BLX-MA	1	20	11.9 (25)	[12] 8E−6	[11] 9E5	[9] 2183	[11] 49.5
CoEVO	2	9	519 (3)	70.0 (8)	[8] 367	[11] 8344	[13] 1211
FEA	0	0	[13] 70.6	[13] 982	[13] 7E6	[12] 9913	[12] 258

Table 5
Comparison between LBBO and other EAs on six solved 30-dimensional multimodal problems. The algorithms are listed in order from best to worst. See the text for a more detailed description of this table. LBBO performance is shown in bold font.

	# of solved functions	Success rate	F_7	F_9	F_{10}	F_{11}	F_{12}	F_{15}
CMA-GA-PSO	6	52	1.3 (25)	3.7 (23)	7.6 (3)	0.03 (22)	6.7 (2)	1.0 (3)
G-CMA-ES	5	31	1.0 (25)	8.0 (9)	5.3 (3)	1.0 (1)	1.3 (8)	[4] 208
LBBO	**4**	**38**	**12.4 (24)**	**0.15 (25)**	**[8] 172**	**[6] 28.1**	**0.5 (5)**	**1.0 (3)**
DE	2	31	32.8 (22)	0.7 (25)	[6] 61.6	[10] 32.6	[7] 8430	[11] 484
K-PCX	4	31	2.5 (10)	3.3 (18)	1.0 (14)	[7] 29.5	1.0 (5)	[12] 876
EDA	1	17	21.3 (25)	[11] 179	[10] 189	[12] 39.5	[6] 6054	[9] 396
L-CMA-ES	1	17	1.1 (25)	[12] 291	[12] 563	[3] 15.2	[10] 13,200	[3] 21.6
BLX-GL50	1	17	10.2 (25)	[7] 15.1	[4] 35.2	[5] 24.7	[8] 9521	[5] 304
SPC-PNX	1	11	60.7 (16)	[8] 23.9	[5] 60.3	[4] 18.1	[9] 13,134	[8] 368
CoEVO	1	7	93.4 (11)	[10] 131	[11] 232	[11] 37.7	[12] 1E5	[10] 411
BLX-MA	1	6	[11] 0.01	6.7 (9)	[7] 90.6	[8] 31.1	[5] 4391	[7] 356
FEA	0	0	[12] 3.6	[9] 24.1	[9] 186	[9] 31.4	[11] 14,819	[6] 341

4.3.1. Summary of 30-dimensional benchmark results

Tables 4–6 show that for the 30-dimensional benchmarks, LBBO ranks 4th best on the unimodal problems, 3rd best on the solved multimodal problems, and 12th best on the unsolved problems. Combining the ranks of the algorithms that appear in all three of these tables, we see that overall, LBBO ranks 4th best out of the 11 algorithms.

Table 6
Comparison between LBBO and other EAs on 12 unsolved 30-dimensional multimodal problems. The algorithms are listed in order from best to worst. See the text for a more detailed description of this table. LBBO performance is shown in bold font.

	Ave. rank	F_8	F_{13}	F_{14}	F_{16}	F_{18}	F_{19}	F_{20}	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}
ADE	1.7	[1] 20.0	[1] 1.11	[4] 12.5	[3] 52.3	[1] 716	[1] 806	[1] 619	[2] 500	[1] 500	[1] 534	[1] 200	[3] 210
IPOP-CMA-ES	4.1	[9] 20.8	[7] 2.53	[1] 11.0	[1] 11.1	[8] 904	[8] 904	[7] 904	[2] 500	[3] 817	[1] 534	[1] 200	[1] 209
G-CMA-ES	5.3	[5] 20.1	[5] 2.49	[7] 12.9	[2] 35.0	[8] 904	[8] 904	[7] 904	[2] 500	[2] 803	[1] 534	[13] 910	[4] 211
RMA	5.3	[10] 20.9	[6] 2.50	[5] 12.6	[7] 85.7	[7] 902	[5] 902	[11] 906	[2] 500	[4] 867	[1] 534	[1] 200	[4] 211
EDA	5.6	[10] 20.9	[15]15.3	[12] 13.3	[10] 208	[3] 847	[3] 842	[3] 851	[2] 500	[6] 872	[1] 534	[1] 200	[1] 209
BLX-MA	6.3	[8] 20.7	[9] 3.96	[5] 12.6	[14] 326	[4] 878	[4] 880	[4] 879	[2] 500	[9] 908	[8] 559	[1] 200	[7] 212
STS	6.4	[7] 20.5	[2] 1.72	[2] 11.8	[9] 121	[6] 891	[5] 902	[6] 897	[2] 500	[11] 991	[9] 575	[11] 445	N/A
L-CMA-ES	6.8	[1] 20.0	[4] 2.32	[15] 14.0	[4] 58.4	[5] 890	[7] 903	[5] 889	[1] 485	[5] 871	[7] 535	[15] 1410	[12] 691
SPC-PNX	7.2	[10] 20.9	[8] 3.59	[10] 13.1	[6] 74.0	[11] 905	[11] 905	[10] 905	[2] 500	[8] 880	[1] 534	[1] 200	[8] 213
BLX-GL50	8.0	[15] 21.0	[11] 5.15	[3] 12.1	[8] 88.7	[8] 904	[8] 904	[7] 904	[2] 500	[7] 874	[11] 587	[12] 877	[4] 211
K-PCX	8.2	[1] 20.0	[14] 11.9	[14] 13.8	[5] 71.5	[2] 830	[2] 831	[2] 831	[15] 859	[15] 1560	[13] 866	[7] 213	[8] 213
LBBO	**8.8**	**[1] 20.0**	**[3] 2.26**	**[9] 13.1**	**[12] 273**	**[13] 933**	**[13] 930**	**[13] 929**	**[2] 500**	**[12] 1073**	**[10] 582**	**[8] 264**	**[10] 270**
DE	11.4	[10] 20.9	[10] 4.51	[12] 13.3	[13] 282	[12] 913	[12] 913	[12] 913	[12] 581	[10] 964	[12] 621	[9] 314	[13] 786
FEA	11.8	[6] 20.3	[12] 5.57	[8] 12.9	[11] 270	[14] 964	[14] 943	[14] 967	[14] 695	[13] 1080	[14] 910	[10] 328	[11] 517
CoEVO	13.7	[10] 20.9	[13] 9.02	[11] 13.2	[15] 381	[15] 1061	[15] 1049	[15] 1059	[13] 604	[14] 1155	[15] 922	[14] 1097	[14] 1027

4.4. Discussion of the 2005 CEC benchmark results

The results of the previous sections reveal some interesting characteristics of LBBO.

- In general, LBBO performs well on unimodal benchmarks. It ranks 7th out of 17 algorithms for the 10-dimensional problems, and it ranks 4th out of 13 for the 30-dimensional problems. In addition, it obtained the best performance on two of the five problems in 10 dimensions, and on four of the five problems in 30 dimensions.
- LBBO performs well for multimodal benchmarks with known solutions. LBBO ranks 7th out of 17 algorithms for the 10-dimensional problems, and 3rd out of 12 algorithms for the 30-dimensional problems. This implies that LBBO is suitable for multimodal problems that are not too difficult.
- LBBO's performance was mediocre for high-dimensional unsolved multimodal problems. LBBO ranks 5th out of 19 algorithms for the 10-dimensional problems, but only 12th out of 15 algorithms for the 30-dimensional problems. This implies that LBBO may not be the best choice for extremely difficult, high-dimensional, multimodal problems. However, we will see in the following section that this does not imply that LBBO is unsuitable for real-world problems. The unsolved multimodal problems may be artificially difficult, and may therefore not be a good indication of real-world performance.
- Next we compare LBBO performance on F_9 with F_{10} (a rotated version of F_9), and F_{15} with F_{16} (a rotated version of F_{15}). LBBO ranks first on the non-rotated functions F_9 and F_{15} in both 10 dimensions and 30 dimensions. However, LBBO ranks much worse on the rotated functions F_{10} and F_{16}. Superficially, this may lead one to conclude that LBBO does not perform well on rotated functions. However, a rotated function may be quite different than its original version, and so the comparison of performances may not be meaningful. For example, the solution of the 10-dimensional Rastrigin benchmark F_9 is within the search space, but the solution of the rotated version F_{10} is outside the search space. Also, since we obtain F_{10} by rotating F_9 with rotation matrix M, that means we obtain F_9 by rotating F_{10} with rotation matrix M^{-1}. Therefore, in general, it is not meaningful to say that an optimization algorithm performs well, or poorly, on rotated functions. There are exactly the same number of functions for which rotation deteriorates the performance of an algorithm, as there are functions for which rotation improves the performance. Note that this is not the same as saying that an algorithm is rotationally invariant – otherwise all algorithms would be rotationally invariant. Many benchmarks are separable under a specific rotation, and certain algorithms perform well only on separable functions. Those algorithms will thus perform well only if the benchmark is rotated with a specific rotation matrix. Rotational invariance means that an algorithm performs equally well on separable and nonseparable functions.
- LBBO performs just as well on functions whose solution lies on the search domain boundary, as on functions whose solution lies within the boundary. F_{20} is a version of F_{18} with the global optimum on the domain boundary. In both 10 and 30 dimensions, LBBO performed slightly better on F_{20} than it did on F_{18}. LBBO's slightly better performance on F_{20} may be due to the boundary search logic described in Section 3.3.
- LBBO performs just as well on functions with narrow optimum basins as on functions with wider basins. F_{19} is a version of F_{18} with a narrow global optimum basin. LBBO performed relatively better on F_{19} than on F_{18} in 10 dimensions, and its performance was the same on F_{19} and F_{18} in 30 dimensions. LBBO's slightly better performance on F_{19} may be due to the local gradient search logic described in Section 3.2.
- LBBO performs just as well on functions whose optimum is outside the initialization domain, as on functions whose optimum is within the initialization range. F_{25} is a version of F_{24} with its optimum outside the initialization range. In both 10 and 30 dimensions, LBBO performed about the same on F_{25} as on F_{24}. In fact, its equivalent performance on F_{25} and F_{24} resulted in its being ranked 1st on F_{25} in 10 dimensions. This may be due to several factors. First, the migration logic

of Eq. (4) allows an LBBO offspring variable to move outside of the range of its parents. Second, the mutation described in Section 2.2, and the Latin hypercube search and re-initialization described in Sections 3.5 and 3.6, are allowed to search outside the initialization boundaries if the optimum is known a priori to lie outside those boundaries.

- The inclusion of seven new components to BBO to obtain the LBBO procedure of Algorithm 8 improves performance, but results in a higher computational cost. The increased computational cost varies depending on the particular random number sequence realization for a particular simulation, and depending on the cost function. However, the vast majority of computational effort in interesting, real-world problems is consumed with fitness function evaluations, and so the computational effort of the optimization algorithm is rarely significant as discussed in Section 21.1 of [63]. This consideration leads us to study LBBO performance on real-world benchmarks in the following section.

4.5. 2011 CEC benchmarks

Next we test the performance of the proposed LBBO method on the 22 real-world benchmark functions from the 2011 CEC. These problems include a parameter estimation problem for frequency-modulated sound saves (T_1); a Lennard-Jones potential problem (T_2); a bifunctional catalyst blend control problem (T_3); a stirred tank reactor control problem (T_4); two Tersoff potential minimization problems (T_5 and T_6); a radar polyphase code design problem (T_7); a transmission network expansion problem (T_8); a transmission pricing problem (T_9); an antenna array design problem (T_{10}); two dynamic economic dispatch problems ($T_{11.1}$ and $T_{11.2}$); five static economic dispatch problems ($T_{11.3} - T_{11.7}$); three hydrothermal scheduling problems ($T_{11.8} - T_{11.10}$); and two spacecraft trajectory optimization problems (T_{12} and T_{13}). The dimensions of these problems range from a minimum of 1 to a maximum of 216. The benchmarks include unconstrained problems, equality-constrained problems, and inequality constrained problems. Constraints are augmented to the cost functions as penalty terms. For details about these functions, the reader is referred to [14,56]. As in the 2011 CEC, in this paper we limit each simulation to 150,000 function evaluations. LBBO parameters are the same as those described at the beginning of Section 4.1.

We compared LBBO algorithms to the 14 algorithms that were accepted for the 2011CEC competition.

1. GAMPC, which is a GA with multi-parent crossover [18].
2. SAMODE, which is differential evolution with a mixture of search operators [19].
3. ENSDE, which is ensemble-based differential evolution [41].
4. EADEMA, which is a hybrid of a memetic algorithm and a differential evolution variant [58].
5. AdapDE, which is adaptive differential evolution [3].
6. EDDE, which is a hybrid of estimation of distribution and differential evolution [71].
7. OXDE, which is differential evolution with orthogonal crossover [33].
8. DERHC, which is a hybrid of differential evolution and random hill climbing [31].
9. RGA, which is a real-coded GA [55].
10. CDASA, which is an ant system [28].
11. mSBXGA, which is a GA with simulated binary crossover [7].
12. DEcr, which is differential evolution with adaptive crossover and local search [52].
13. WIDE, which is a hybrid of invasive weed optimization and differential evolution [24].
14. ModDELS, which is differential evolution with local search [2].

We collected benchmark performance data for these EAs from [67,68]. As in the 2011 CEC competition, we ran 25 Monte Carlo simulations of LBBO for each benchmark, and then ranked all of the algorithms based on both the best cost achieved out of 25 simulations, and the average cost achieved over 25 simulations. Tables 7 and 8 show the results. Note that the rankings in these tables differ slightly from those reported in [68]. This is because we used all reported significant digits in determining the rankings reported in this paper, whereas [68] rounded the results to a given number of significant digits for each benchmark. Table 7 shows that in terms of the best cost achieved, LBBO performs third best out of 15 algorithms. Table 8 shows that in terms of the average cost achieved, LBBO performs fourth best.

Tables 7 and 8 show that LBBO performs relatively poorly on benchmarks T_1, T_{10}, $T_{11.9}$, and $T_{11.10}$. LBBO's poor performance on T_1 could simply be a matter of reporting precision. The best seven algorithms reported a best cost of exactly 0 for T_1, while LBBO reported a best cost of 1.83×10^{-15}. So although it appears from Tables 7 and 8 that LBBO has poor performance on T_1, its performance may be the best for all practical purposes. We see a similar phenomenon with T_{10}. However, LBBO's poor performance on $T_{11.9}$ and $T_{11.10}$ is more difficult to explain. LBBO clearly performs poorly on these two benchmarks, with average costs of 2.2×10^6 and 1.8×10^6 respectively, compared to average costs by the best algorithm (DEcr) of 9.3×10^5 and 9.2×10^5 respectively. $T_{11.9}$ and $T_{11.10}$ are hydrothermal scheduling problems, but so is $T_{11.8}$, for which LBBO attained the best ranking.

Tables 7 and 8 shows that LBBO performs exceptionally well on benchmarks $T_{11.1}$, $T_{11.2}$, $T_{11.5}$, $T_{11.6}$, $T_{11.7}$, and $T_{11.8}$, where LBBO's performance ranks at the top in terms of both the best cost attained and the average cost attained. In general, these six benchmarks have the common feature of high dimensionality, with problem dimensions of 120, 216, 15, 40, 140, and 96 respectively. These problems include the ones with the highest, third highest, fourth highest, and fifth highest dimensions. Also, LBBO performs third best (on average) on the problem with the second highest dimension (T_9). So in general, LBBO performs very well on high-dimension problems.

Table 7
Comparison between LBBO and 14 other EAs on real-world benchmark problems in terms of the best results obtained after 25 Monte Carlo simulations. The table shows the rank of each algorithm, and the algorithms are listed in order from best to worst. This table shows the ranks of the algorithms in terms of best results, while Table 8 shows the ranks in terms of average results. LBBO performance is shown in bold font.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	$T_{11.1}$	$T_{11.2}$	$T_{11.3}$	$T_{11.4}$	$T_{11.5}$	$T_{11.6}$	$T_{11.7}$	$T_{11.8}$	$T_{11.9}$	$T_{11.10}$	T_{12}	T_{13}	Ave.
GAMPC	1	4	1	3	7	2	1	1	5	3	4	4	1	2	3	5	11	10	4	7	3	1	3.8
DEcr	11	4	1	3	7	1	9	1	13	10	2	2	1	1	3	2	2	2	1	1	10	9	4.4
LBBO	**10**	**4**	**1**	**3**	**2**	**2**	**8**	**1**	**4**	**10**	**1**	**1**	**1**	**6**	**1**	**1**	**1**	**1**	**14**	**14**	**5**	**8**	**4.5**
SAMODE	1	4	1	3	7	2	1	1	9	3	7	4	1	9	9	6	11	9	7	8	2	2	4.9
OXDE	1	1	1	1	6	14	1	1	13	1	10	4	1	2	3	6	4	7	8	12	6	5	4.9
EDDE	1	4	1	3	2	2	7	1	15	3	7	4	1	6	9	6	6	4	6	3	8	13	5.1
WIDE	1	4	14	3	7	2	1	1	10	3	7	4	1	2	3	9	7	11	5	9	1	7	5.1
AdapDE	1	2	13	14	5	10	1	1	1	2	3	9	10	8	7	3	5	3	2	2	11	3	5.3
ENSDE	1	15	1	3	15	15	15	1	7	3	5	3	1	2	2	4	7	4	3	3	14	6	5.9
RGA	15	11	1	3	7	2	11	1	3	13	12	13	14	10	9	14	7	6	10	5	7	10	8.4
CDASA	9	13	1	3	7	2	10	1	6	15	6	11	14	12	14	11	3	7	9	5	13	14	8.5
EADEMA	12	3	1	12	1	11	1	1	2	8	15	15	11	11	8	15	14	15	12	15	4	4	8.7
DERHC	8	4	1	15	2	2	14	1	8	14	11	14	1	12	15	13	15	14	15	13	15	15	10.1
mSBXGA	14	12	12	13	13	12	12	1	11	12	14	10	13	14	9	10	10	12	11	10	12	12	11.3
ModDELS	13	14	14	2	14	13	13	1	12	9	13	12	12	15	13	12	13	13	13	11	9	11	11.5

Table 8
Comparison between LBBO and 14 other EAs on real-world benchmark problems in terms of the average results obtained by 25 Monte Carlo simulations. The table shows the rank of each algorithm, and the algorithms are listed in order from best to worst. This table shows the ranks of the algorithms in terms of average results, while Table 7 shows the ranks in terms of best results. LBBO performance is shown in bold font.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	$T_{11.1}$	$T_{11.2}$	$T_{11.3}$	$T_{11.4}$	$T_{11.5}$	$T_{11.6}$	$T_{11.7}$	$T_{11.8}$	$T_{11.9}$	$T_{11.10}$	T_{12}	T_{13}	Ave.
GAMPC	1	2	1	1	5	5	5	1	6	1	4	3	1	4	3	5	7	10	4	8	2	1	3.6
DEcr	4	2	1	13	6	1	9	1	14	13	2	2	1	1	3	2	2	2	1	1	10	7	4.5
SAMODE	5	4	1	5	12	4	7	1	8	1	7	3	1	7	3	5	9	8	7	7	1	2	4.9
LBBO	**10**	**1**	**1**	**1**	**1**	**7**	**6**	**1**	**3**	**9**	**1**	**1**	**10**	**7**	**1**	**1**	**1**	**1**	**14**	**15**	**6**	**14**	**5.1**
EDDE	1	9	1	1	11	7	14	1	15	5	4	7	1	4	8	5	3	4	5	3	7	9	5.7
OXDE	12	5	1	11	9	14	4	1	13	4	8	3	1	4	3	8	4	6	8	13	4	6	6.5
AdapDE	9	6	1	14	10	11	1	1	1	8	12	9	10	9	8	3	6	3	3	2	14	3	6.5
WIDE	8	11	14	8	4	10	3	1	10	1	11	7	1	3	3	10	12	12	6	11	9	4	7.2
ENSDE	6	14	1	12	14	15	15	1	9	14	6	3	10	1	2	4	4	5	2	4	12	5	7.2
RGA	14	12	1	5	7	3	11	1	2	12	9	13	10	11	8	14	14	9	10	6	8	8	8.5
ModDELS	3	13	14	1	8	11	8	1	12	7	13	12	7	12	13	11	7	11	13	9	3	11	9.1
EADEMA	7	10	1	9	2	6	2	1	4	6	15	15	8	14	12	15	15	15	15	14	5	12	9.2
DERHC	13	7	1	15	3	2	13	1	7	11	9	14	10	13	14	12	12	13	11	10	13	13	9.9
CDASA	15	15	1	5	15	9	10	1	5	15	3	11	10	15	15	13	11	6	9	5	15	15	10.0
mSBXGA	11	8	13	10	13	13	12	1	11	10	14	10	9	10	8	9	10	14	12	12	11	10	10.5

Another common feature of the benchmarks for which LBBO performs well is inequality constraints. The 2011 CEC benchmarks augment inequality constraints to the original cost function with a penalty function. This results in augmented cost functions with deep, narrow valleys at their constrained minima. Of the 22 benchmarks, 10 of them have only inequality constraints (no equality constraints), and all six of the benchmarks for which LBBO performs best belong to that group.

In summary, we conclude that, in general, LBBO performs particularly well on high-dimensional problems with the type of deep, narrow optima that characterizes cost functions that are augmented with inequality constraints.

5. The relative importance of LBBO components

Section 3 shows that LBBO includes seven components that are added to BBO: migration (Section 3.1), gradient descent (Section 3.2), boundary search (Section 3.3), grid search (Section 3.4), Latin hypercube search (Section 3.5), re-initialization (Section 3.6), and restart (Section 3.7). The issue that we address in this section is the relative importance of each of these components.

First we run LBBO with all seven components except the first one; that is, we replace the unique LBBO migration logic of Section 3.1 with the standard BBO migration logic of Section 2.1. Next, we run LBBO with all seven components except the second one; that is, we skip the gradient descent logic of Section 3.2. In general, we run LBBO with all seven components except the nth one, for $n \in [1,7]$.

We repeat the 10-dimensional benchmark tests of Section 4.2 under these conditions. Results for the unimodal problems, the solved multimodal problems, and the unsolved multimodal problems are shown in Tables 9–11 respectively.

Table 12 shows the results of pair-wise Wilcoxon signed rank tests [45] between the complete LBBO algorithm with all seven additional components, and each algorithm that has a single component removed. Each cell in Table 12 indicates whether the given algorithm with a single component removed performs better than, worse than, or statistically equal to the complete LBBO algorithm on the given benchmark, where we used 95% as the confidence threshold.

Table 9
Comparison between LBBO, and LBBO with the nth component omitted for $n \in [1,7]$, on 10-dimensional unimodal problems. The algorithms are listed in order from best to worst. See Sections 4.2 and 5 for a more detailed description of this table. Note that the last seven rows of the table show the performance of the complete LBBO algorithm except for the omission of a single algorithmic component.

	# of solved functions	Success rate (%)	F_1	F_2	F_3	F_5	F_6
LBBO	5	90	1.7 (25)	0.9 (25)	0.4 (25)	18.1 (13)	1.5 (25)
No grid search	5	88	1.7 (25)	0.8 (25)	0.3 (25)	23.9 (10)	1.2 (25)
Std. migration	5	84	3.5 (25)	1.9 (25)	0.8 (25)	19.5 (5)	1.6 (25)
No Latin	5	84	3.5 (25)	1.8 (25)	0.9 (25)	51.4 (5)	1.7 (25)
No restart	5	83	4.2 (25)	1.7 (25)	0.9 (25)	32.2 (4)	1.6 (25)
No re-init	5	82	3.3 (25)	2.1 (25)	0.9 (25)	73.9 (3)	1.7 (25)
No boundary	4	80	3.9 (25)	2.2 (25)	0.9 (25)	[7] 0.59	1.5 (25)
No gradient	0	0	[8] 508	[8] 2405	[8] 4.9E5	[8] 3725	[8] 5.3E6

Table 10
Comparison between LBBO, and LBBO with the nth component omitted for $n \in [1,7]$, on solved 10-dimensional multimodal problems. The algorithms are listed in order from best to worst. See Sections 4.2 and 5 for a more detailed description of this table. Note that the last seven rows of the table show the performance of the complete LBBO algorithm except for the omission of a single algorithmic component.

	# of solved functions	Success rate (%)	F_7	F_9	F_{10}	F_{11}	F_{12}	F_{15}
LBBO	4	57	17.4 (20)	0.2 (25)	[4] 14.4	[8] 6.1	6.3 (23)	1.0 (17)
Std. migration	3	50	[5] 1.2	0.2 (25)	[1] 11.9	[3] 5.5	3.1 (25)	0.5 (25)
No boundary	3	50	[5] 1.2	0.2 (25)	[2] 12.6	[6] 5.8	3.1 (25)	0.6 (25)
No re-init.	3	50	[4] 1.1	0.2 (25)	[6] 16.0	[5] 5.7	5.4 (25)	0.6 (25)
No Latin	3	49	[3] 1.0	0.2 (25)	[5] 15.4	[7] 5.9	4.2 (25)	0.8 (23)
No restart	3	36	[7] 1.3	0.2 (25)	[7] 17.9	[1] 5.1	6.6 (17)	0.6 (12)
No grid search	2	27	14.2 (16)	[7] 7.0	[3] 14.2	[2] 5.4	4.5 (25)	[7] 138
No gradient	0	0	[8] 18.6	[8] 15.9	[8] 27.3	[4] 5.6	[8] 1.0E4	[8] 142

Table 11
Comparison between LBBO, and LBBO with the nth component omitted for $n \in [1,7]$, on unsolved 10-dimensional multimodal problems. The algorithms are listed in order from best to worst. See Sections 4.2 and 5 for a more detailed description of this table. Note that the last seven rows of the table show the performance of the complete LBBO algorithm except for the omission of a single algorithmic component.

	F_8	F_{13}	F_{14}	F_{16}	F_{18}	F_{19}	F_{20}	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}
LBBO	[1] 20.0	[1] 0.40	[3] 3.51	[2] 134	[1] 511	[1] 516	[1] 538	[2] 506	[2] 748	[5] 712	[6] 220	[4] 216
No Latin	[1] 20.0	[4] 0.45	[1] 3.35	[6] 139	[3] 762	[3] 766	[3] 747	[1] 491	[5] 760	[2] 598	[1] 205	[1] 205
No re-init	[1] 20.0	[2] 0.41	[2] 3.41	[4] 137	[2] 754	[2] 762	[2] 744	[5] 574	[7] 797	[3] 618	[2] 206	[1] 205
Std. migration	[1] 20.0	[6] 0.50	[6] 3.56	[1] 133	[5] 794	[5] 826	[6] 818	[4] 541	[6] 784	[1] 588	[4] 215	[4] 216
No boundary	[1] 20.0	[7] 0.51	[8] 3.91	[5] 138	[6] 797	[4] 783	[4] 776	[3] 513	[1] 728	[4] 662	[3] 212	[3] 213
No grid search	[1] 20.0	[5] 0.48	[5] 3.54	[2] 134	[4] 782	[7] 831	[4] 776	[6] 577	[3] 749	[7] 882	[5] 219	[6] 223
No restart	[1] 20.0	[2] 0.41	[4] 3.52	[7] 144	[8] 902	[8] 912	[8] 906	[8] 902	[4] 756	[8] 925	[7] 296	[7] 380
No gradient	[8] 20.3	[8] 0.52	[7] 3.57	[8] 167	[7] 826	[6] 827	[7] 858	[7] 819	[8] 837	[6] 746	[8] 441	[8] 488

Tables 9–12 show that the worst-performing algorithm is LBBO without gradient descent, which performs worse than LBBO on 20 of the 23 benchmarks. This means that gradient descent is the most important feature of our LBBO algorithm. We conclude that LBBO is not very effective unless it is augmented with a local search method such as gradient descent.

Tables 9–12 also show that some of the algorithms perform better than LBBO on certain benchmarks. This is most clear from Table 12, where we see that LBBO without grid search performs better than LBBO on five benchmarks, although LBBO still performs better on seven benchmarks.

In general, we conclude that gradient descent is the most important feature of LBBO, and grid search is the least important feature. For the simplest functions like the unimodal functions of Table 9, all of the features are of secondary importance compared to gradient descent. For solved multimodal functions (Table 10), the restart and grid search features are also important, but still less important than gradient descent. For the most difficult problems (the unsolved multimodal functions of Table 11), all of the LBBO features appear to be important except Latin hypercube search and periodic re-initialization.

The only gradient descent algorithm that we tested was the interior point algorithm as implemented by MATLAB's **fmincon** function. We have not implemented our own local optimization algorithm, or tested other algorithms as alternatives to **fmincon**, but that could be an important task for future research. How important is the particular gradient descent algorithm that we used? The gradient descent algorithm, along with its settings, can be considered to be tuning parameters of LBBO.

This section has shown the relative importance of each component of LBBO by eliminating one component at a time. It would also be interesting to study this question from the opposite perspective; that is, start with standard BBO and add one component at a time to see how each component affects performance in isolation from the other components. Such a study may give different results than what we conclude here, and we suggest that idea for future research.

Table 12
Statistical comparison between LBBO, and LBBO with a single component omitted. Each cell indicates whether the given algorithm is better than (B), worse than (W), or equal to (≈) LBBO, based on a Wilcoxon signed rank test with a confidence threshold of 95%. The number at the bottom of each column is equal to the number of W cells minus the number of B cells. Note that the seven right-most columns of the table compare the performance of LBBO, with the complete LBBO algorithm except for the omission of a single algorithmic component.

	Standard migration	No boundary	No gradient	No grid	No Latin	No reinit.	No restart
F_1	≈	W	W	B	≈	≈	W
F_2	W	W	W	B	W	W	W
F_3	W	W	W	B	W	W	W
F_5	≈	W	W	≈	≈	≈	≈
F_6	W	W	W	≈	W	W	W
F_7	W	W	W	≈	W	W	W
F_8	≈	≈	W	≈	≈	≈	≈
F_9	≈	B	W	W	B	B	B
F_{10}	≈	≈	W	B	W	W	W
F_{11}	≈	≈	≈	≈	≈	≈	≈
F_{12}	≈	≈	W	≈	≈	≈	W
F_{13}	W	≈	W	≈	≈	≈	≈
F_{14}	≈	W	≈	≈	≈	≈	≈
F_{15}	≈	≈	W	W	≈	≈	≈
F_{16}	≈	≈	W	≈	≈	≈	≈
F_{18}	W	W	W	W	W	W	W
F_{19}	W	W	W	W	W	W	W
F_{20}	W	W	W	W	W	W	W
F_{21}	≈	≈	W	≈	≈	≈	W
F_{22}	≈	≈	W	≈	≈	≈	≈
F_{23}	B	≈	≈	W	B	B	W
F_{24}	B	≈	W	B	B	≈	≈
F_{25}	≈	≈	W	W	B	B	≈
Net # Worse than LBBO	6	9	20	2	4	5	11

6. Conclusions and future work

A linearized version of BBO, LBBO, was introduced in this paper. LBBO modifies the BBO migration operator to reduce rotational variance. In addition, we included global and local search operators, along with re-initialization logic, to improve performance.

LBBO was compared with the entries to the 2005 Congress on Evolutionary Computation on 23 benchmark problems. The results show that LBBO provides competitive performance. LBBO performs particularly well for certain types of multimodal problems. Also, LBBO is insensitive to whether or not the solution lies on the search domain boundary, to whether or not the global optimum lies in a wide or narrow basin, and to whether or not the global optimum lies within or outside of the initialization domain. Compared to 11 other EAs, LBBO performed the best for 5 out of 23 benchmarks in 10 dimensions, and it performed the best for 9 out of 23 benchmarks in 30 dimensions. The improved relative performance of LBBO in 30 dimensions indicates that it may perform even better on higher dimensional problems, which we suggest for future research.

LBBO was also compared with the entries to the 2011 Congress on Evolutionary Computation on 22 real-world problems. The results show that LBBO performs particularly well for high dimensional problems, and for problems with inequality constraints. Compared to 14 other EAs, LBBO performed third best in terms of best performance, and fourth best in terms of average performance.

Tables 9–11 showed that local search is the most important new component that we added to BBO to obtain LBBO. We used gradient descent as a default local search operator, but our results indicate that it may be fruitful to experiment with other local search operators to improve LBBO performance.

One area in which LBBO fell short was its performance on extremely difficult (unsolved) multimodal problems, including highly nonseparable problems. This reveals an area where LBBO could be improved. Other future work could include testing on additional benchmark functions, testing with higher dimensions, noise handling, comparing LBBO with additional EAs, and hybridizing LBBO with other successful EAs. The MATLAB software that we used to generate the results in this paper is available at http://www.academic.csuohio.edu/simond/bbo/linearized.

Acknowledgments

The comments of the reviewers were helpful in improving the original version of this paper. The material in this paper is based on work supported by the National Science Foundation under Grant No. 0826124.

References

[1] S. Alonso, J. Jimenez, H. Carmona, B. Galvan, G. Winter, Performance of a flexible evolutionary algorithm, Technical Report, University of Las Palmas de Gran Canaria, Canary Islands, Spain, 2005. <http://www.ntu.edu.sg/home/epnsugan/index_files/CEC-05/CEC05.htm>.
[2] M. Ankush, A. Das, P. Mukherjee, S. Das, P. Suganthan, Modified differential evolution with local search algorithm for real world optimization, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1565–1572.
[3] M. Asafuddoula, T. Ray, R. Sarker, An adaptive differential evolution algorithm and its performance on real world optimization problems, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1057–1062.
[4] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 1769–1776.
[5] A. Auger, N. Hansen, Performance evaluation of an advanced local search evolutionary algorithm, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 1777–1784.
[6] P. Ballester, J. Stephenson, J. Carter, K. Gallagher, Real-parameter optimization performance study on the CEC-2005 benchmark with SPC-PNX, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 498–505.
[7] S. Bandaru, R. Tulshyan, K. Deb, Modified SBX and adaptive mutation for real world single objective optimization, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1335–1342.
[8] W. Becker, X. Yu, J. Tu, EvLib: A Parameterless Self-Adaptive Real-Valued Optimisation Library, Technical Report, RMIT University, Melbourne, Australia, 2005. <http://www.ntu.edu.sg/home/epnsugan/index_files/CEC-05/CEC05.htm>.
[9] D. Bernstein, Optimization r us, IEEE Cont. Syst. Mag. 26 (5) (2006) 6–7.
[10] I. Boussaid, A. Chatterjee, P. Siarry, M. Ahmed-Nacer, Hybridizing biogeography-based optimization with differential evolution for optimal power allocation in wireless sensor networks, IEEE Trans. Veh. Technol. 60 (5) (2011) 2347–2353.
[11] J. Brown, M. Lomolino, Independent discovery of the equilibrium theory of island biogeography, Ecology 70 (6) (1989) 1954–1957.
[12] M. Clerc, Beyond standard particle swarm optimisation, Int. J. Swarm Intell. Res. 1 (4) (2010) 46–61.
[13] L. Costa, A parameter-less evolution strategy for global optimization, in: WSEAS International Conference on Simulation, Modelling and Optimization, Lisbon, Portugal, 2006, pp. 622–627.
[14] S. Das, P. Suganthan, Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems, Technical Report, Jadavpur University, Nanyang Technological University, 2010.
[15] M. Dorigo, Optimization, Learning and Natural Algorithms, PhD Thesis, Politecnico di Milano, Italy, 1992.
[16] A. Duarte, R. Martí, F. Glover, F. Gortazar, Hybrid scatter tabu search for unconstrained global optimization, Ann. Oper. Res. 183 (1) (2011) 95–123.
[17] A. Eiben, J. Smith, Introduction to Evolutionary Computing, Springer, 2010.
[18] S. Elsayed, R. Sarker, D. Essam, GA with a new multi-parent crossover for solving IEEE-CEC2011 competition problems, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1034–1040.
[19] S. Elsayed, R. Sarker, D. Essam, Differential evolution with multiple strategies for solving CEC2011 real-world numerical optimization problems, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1041–1048.
[20] M. Ergezer, D. Simon, D. Du, Oppositional biogeography-based optimization, in: IEEE Conference on Systems, Man, and Cybernetics, San Antonio, Texas, 2009, pp. 1035–1040.
[21] C. García-Martínez, M. Lozano, Hybrid real-coded genetic algorithms with female and male differentiation, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 896–903.
[22] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.
[23] W. Gong, Z. Cai, C. Ling, DE/BBO: a hybrid differential evolution with biogeography-based optimization for global numerical optimization, Soft. Comput. 15 (4) (2011) 645–665.
[24] U. Haider, S. Das, D. Maity, A. Abraham, P. Dasgupta, Self adaptive cluster based and weed inspired differential evolution algorithm for real world optimization, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 750–756.
[25] N. Hansen, Compilation of results on the 2005 CEC benchmark function set, Technical Report, Computational Laboratory (CoLab), Institute of Computational Science, 2006.
[26] S. Islam, S. Das, S. Ghosh, S. Roy, P. Suganthan, An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization, IEEE Trans Syst, Man, Cybern Part B: Cybern 42 (2) (2012) 482–500.
[27] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE International Joint Conference on Neural Networks, Perth, Western Australia, 1995, pp. 1942–1948.
[28] P. Korosec, J. Silc, The continuous differential ant-stigmergy algorithm applied to real-world optimization problems, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1327–1334.
[29] H. Kundra, M. Sood, Cross-country path finding using hybrid approach of PSO and BBO, Int. J. Comput. Appl. 7 (6) (2010) 15–19.
[30] B. Lacroix, D. Molina, F. Herrera, Region based memetic algorithm with LS chaining, in: World Congress on Computational Intelligence, Brisbane, Australia, 2012, pp. 1474–1479.
[31] A. LaTorre, S. Muelas, J.-M. Peña, Benchmarking a hybrid DE-RHC algorithm on real world problems, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1027–1033.
[32] G. Leguizamón, C. Coello Coello, Boundary search for constrained numerical optimization problems, in: E. Mezura-Montes (Ed.), Constraint-Handling in Evolutionary Optimization, Springer, 2009, pp. 25–49.
[33] X. Li, M. Yin, Enhancing the Exploration Ability of Composite Differential Evolution through Orthogonal Crossover, 2011 (unpublished).
[34] J. Liang, P. Suganthan, Dynamic multi-swarm particle swarm optimizer with local search, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 522–528.
[35] T. Liao, M. Montes de Oca, T. Stützle, Computational results for an automatically tuned IPOP-CMA-ES on the CEC'05 benchmark set, Technical Report TR/IRIDIA/2011-022, Université Libre de Bruxelles, Brussels, Belgium, 2011.
[36] P. Lozovyy, G. Thomas, D. Simon, Biogeography-based optimization for robot controller tuning, in: B. Igelnik (Ed.), Computational Modeling and Simulation of Intellect: Current State and Future Perspectives, IGI Global, 2011, pp. 162–181.
[37] H. Ma, D. Simon, Analysis of migration models of biogeography-based optimization using Markov theory, Eng. Appl. Artif. Intell. 24 (2011) 1052–1060.
[38] H. Ma, D. Simon, Blended biogeography-based optimization for constrained optimization, Eng. Appl. Artif. Intell. 24 (3) (2011) 517–525.
[39] H. Ma, D. Simon, M. Fei, Z. Xie, Variations of biogeography-based optimization and Markov analysis, Inf. Sci. 220 (2013) 492–506.
[40] R. MacArthur, E. Wilson, The Theory of Island Biogeography, Princeton University Press, Princeton, New Jersey, 1967.
[41] R. Mallipeddi, P. Suganthan, Ensemble differential evolution algorithm for CEC2011 problems, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1557–1564.
[42] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. 8 (1) (1998) 3–30.
[43] D. Molina, F. Herrera, M. Lozano, Adaptive local search parameters for real-coded memetic algorithms, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 888–895.
[44] C. Müller, B. Baumgartner, I. Sbalzarini, Particle swarm CMA evolution strategy for the optimization of multi-funnel landscapes, in: IEEE Congress on Evolutionary Computation, Trondheim, Norway, 2009, pp. 2685–2692.
[45] J. Neter, W. Wasserman, G. Whitmore, Applied Statistics, Allyn & Bacon, 1992.

[46] M. Ovreiu, D. Simon, Biogeography-based optimization of neuro-fuzzy system parameters for diagnosis of cardiac disease, in: Genetic and Evolutionary Computation Conference, Portland, Oregon, 2010, pp. 1235–1242.
[47] V. Panchal, P. Singh, N. Kaur, H. Kundra, Biogeography based satellite image classification, Int. J. Comput. Sci. Inf. Secur. 6 (2009) 269–274.
[48] P. Pošik, Real parameter optimisation using mutation step co-evolution, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 872–879.
[49] A. Qin, P. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 1785–1791.
[50] D. Quammen, The Song of the Dodo: Island Biogeography in an Age of Extinction, Simon & Schuster, New York, 1997.
[51] A. Rashid, B. Kim, A. Khambampati, S. Kim, K. Kim, An oppositional biogeography-based optimization technique to reconstruct organ boundaries in the human thorax using electrical impedance tomography, Physiol. Meas. 32 (2011) 767–796.
[52] G. Reynoso-Meza, J. Sanchis, X. Blasco, J. Herrero, Hybrid DE algorithm with adaptive crossover operator for solving real-world numerical optimization problems, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1551–1556.
[53] J. Rönkkönen, S. Kukkonen, K. Price, Real-parameter optimization with differential evolution, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 506–513.
[54] P. Roy, D. Mandal, Quasi-oppositional biogeography-based optimization for multi-objective optimal power flow, Electr. Power Comp. Syst. 40 (2) (2011) 236–256.
[55] A. Saha, T. Ray, How does the good old genetic algorithm fare at real world optimization? in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1049–1056.
[56] E. Sandgren, Non linear integer and discrete programming in mechanical design optimization, J. Mech. Des. 112 (2) (1990) 223–229.
[57] V. Savsani, R. Rao, D. Vakharia, Discrete optimisation of a gear train using biogeography based optimisation technique, Int. J. Des. Eng. 2 (2009) 205–223.
[58] H. Singh, T. Ray, Performance of a hybrid EA-DE-Memetic algorithm on CEC 2011 real world optimization problems, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1322–1326.
[59] U. Singh, T. Kamal, Design of non-uniform circular antenna arrays using biogeography-based optimization, IET Microwaves Antennas Propag. 5 (11) (2011) 1365–1370.
[60] A. Sinha, S. Tiwari, K. Deb, A population-based, steady-state procedure for real-parameter optimization, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 514–521.
[61] D. Simon, Biogeography-based optimization, IEEE Trans. Evol. Comput. 12 (6) (2008) 702–713.
[62] D. Simon, A dynamic system model of biogeography-based optimization, Appl. Soft Comput. 11 (2011) 5652–5661.
[63] D. Simon, Evolutionary Optimization Algorithms, John Wiley & Sons, 2013.
[64] D. Simon, M. Ergezer, D. Du, R. Rarick, Markov models for biogeography-based optimization, IEEE Trans. Syst. Man Cybern. Part B: Cybern. 41 (2011) 299–306.
[65] R. Storn, K. Price, Differential Evolution – A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces, Technical Report TR-95-012, International Computer Science Institute, Berkeley, California, 1995.
[66] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC2005 special session on real-parameter optimization, Technical Report, Nanyang Technological University, Singapore, 2005.
[67] P. Suganthan, Competition on Testing Evolutionary Algorithms on Real-world Numerical Optimization Problems @ CEC11, New Orleans, USA, June 2011. <http://www3.ntu.edu.sg/home/epnsugan/index_files/CEC11-RWP/CEC11-RWP.htm>.
[68] P. Suganthan, Testing Evolutionary Algorithms on Real-World Numerical Optimization Problems, Technical Report, Nanyang Technological University, Singapore, 2011. <http://www3.ntu.edu.sg/home/epnsugan/index_files/CEC11-RWP/CEC11-RWP.htm>.
[69] G. Thomas, T. Wilmot, S. Szatmary, D. Simon, W. Smith, Evolutionary optimization of artificial neural networks for prosthetic knee control, in: B. Igelnik, J. Zurada (Eds.), Efficiency and Scalability Methods for Computational Intellect, IGI Global, 2011, pp. 142–161.
[70] J. Vrugt, B. Robinson, J. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Trans. Evol. Comput. 13 (2) (2009) 243–259.
[71] Y. Wang, B. Li, K. Zhang, Estimation of distribution and differential evolution cooperation for real-world numerical optimization problems, in: IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, 2011, pp. 1315–1321.
[72] J. Weickert, H. Scharr, A scheme for coherence-enhancing diffusion filtering with optimized rotation invariance, J. Vis. Commun. Image Represent. 13 (1–2) (2002) 103–118.
[73] B. Yuan, M. Gallagher, Experimental results for the special session on real-parameter optimization at CEC 2005: a simple, continuous EDA, in: IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 2005, pp. 1792–1799.