Author's personal copy

Swarm and Evolutionary Computation 10 (2013) 12–24

Contents lists available at SciVerse ScienceDirect

Swarm and Evolutionary Computation

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

Regular Paper

Distributed learning with biogeography-based optimization: Markov modeling and robot control

Dan Simon *, Arpit Shah, Carré Scheidegger

Cleveland State University, Department of Electrical and Computer Engineering, Cleveland, OH, USA

A R T I C L E   I N F O

Article history:
Received 23 May 2012
Received in revised form
11 October 2012
Accepted 27 December 2012
Available online 9 January 2013

Keywords:
Biogeography-based optimization
Distributed learning
Robot control
Markov model

A B S T R A C T

Biogeography-based optimization (BBO) is an evolutionary algorithm that is motivated by biogeography, which is the science that describes how biological species are geographically distributed. We extend the standard BBO algorithm to distributed learning, which does not require centralized coordination of the population. We call this new algorithm distributed BBO (DBBO). We derive a Markov model for DBBO, which provides an exact mathematical model of the DBBO population in the limit as the generation number approaches infinity. We use standard benchmark functions to compare BBO and DBBO with several other evolutionary optimization algorithms, and we show that BBO and DBBO give competitive results, especially for multimodal problems. Benchmark results show that DBBO performance is almost identical to BBO. We also demonstrate DBBO on a real-world application, which is the optimization of robot control algorithms, using both simulated and experimental mobile robots.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

Biogeography is the science and study of the geographical distribution of plant and animal life. Alfred Wallace and Charles Darwin were some of the first to observe patterns of biogeography, and they introduced the subject to the scientific world in the 1800s [27]. Biogeography evolved into a quantitative science with the work of Robert MacArthur and Edward Wilson in the early 1960s [19]. Other scientists also contributed to the quantitative study of biogeography, the earliest being Eugene Munroe in 1948 [21].

Biogeography motivated the development of an evolutionary algorithm called biogeography-based optimization (BBO). Since its introduction in [35], BBO has been theoretically analyzed and modeled using Markov theory [14,36,37] and dynamic system models [38]. BBO has also been applied to several real-world problems, including robot controller tuning [13,34], aircraft engine health sensor selection [35], power system optimization [31], groundwater detection [24], mechanical gear train design [33], satellite image classification [23], antenna design [40], and biomedical signal processing [22,28].

In this paper, we introduce a distributed version of BBO, which we call distributed BBO (DBBO). The primary difference between BBO and DBBO is that BBO is coordinated by a central computer. However, DBBO does not depend on a central computer. Instead, each of the DBBO individuals evaluates its own cost function and communicates directly with other individuals for the purpose of information sharing and mutual improvement.

The over-arching goal of this paper is to introduce DBBO and motivate further research in its theory and application. We achieve this goal through four individual objectives.

1) First, we introduce the DBBO algorithm.
2) Second, we develop a Markov model of DBBO, and confirm the model's validity with simulation results.
3) Third, we use standard benchmark functions to show that BBO and DBBO are competitive with other evolutionary algorithms.
4) Fourth, we implement DBBO on an experimental robot system to demonstrate its practicality in real-world systems.

The development of DBBO is motivated by two observations. First, we see that EAs are powerful optimizers, and can find optimal solutions to many important, real-world problems. Second, we see that EAs typically use a central processor that coordinates selection, recombination, mutation, and any other operations that are involved in the evolutionary process. Although parallel EAs are common [6], most EAs are still implemented sequentially for the sake of convenience. The parallelization of EAs requires additional computational resources and design efforts beyond what is required for sequential EAs. However, in the real world, we may want to solve optimization problems by generating candidate solutions that are relatively independent of each other, and that cannot always communicate with a central processor or with the entire population. This is the case, for example, in peer-to-peer networking strategy optimization, and in many mobile robot applications. Distributed EAs may also be

* Corresponding author.
E-mail address: d.j.simon@csuohio.edu (D. Simon).

2210-6502/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.swevo.2012.12.003

Author's personal copy

important in any application where reliability and robustness are important, and where a single-point failure in the optimization process cannot be tolerated.

The DBBO algorithm that we propose is a form of cooperative intelligence. Each individual has the same goal, intentionally communicates with others, and shares information with others to help one another in the attainment of the goal. Our application is robot controller tuning. Each robot has the same goal, which is maintaining a fixed distance from a wall while traveling at a constant velocity. The robots share each other's control parameters among themselves in an attempt to improve their performance.

Researchers have classified distributed intelligence into several different types. The DBBO algorithm that we propose is a form of cooperative intelligence [43]. That is, each individual has the same goal, intentionally communicates with other individuals, and the individuals mutually share information with each other to help one another attain the goal. Our application is robot controller tuning. Each robot has the same goal, which is maintaining a fixed distance from a wall while traveling at a constant velocity. The robots share each other's control parameters among themselves in an attempt to improve their performance.

Each robot has its own control strategy. Some robots perform well, and others perform poorly. The robots interact with each other through wireless radios, but their interaction is sporadic due to the limitations of radio communication, and due to their physical movements from one location to another. We want the robots to evolve controller solutions as they intermittently communicate with each other. This means that each robot needs to implement an EA in its microprocessor. The EA that is implemented by each robot is not aware of the entire EA population, but is aware only of those robots with which it can communicate. This EA, which is implemented in each robot and which involves a dynamically changing subset of the entire EA population, is called DBBO.

This paper is organized as follows. Section 2 gives an overview of centralized BBO [35], which is the standard BBO algorithm, and extends it to our proposed DBBO algorithm. Section 3 derives a Markov model for DBBO, which is an exact mathematical model in the limit as the generation count approaches infinity. Section 4 investigates the performance of BBO and DBBO on benchmark optimization problems, and also provides a comparison to other EAs. Section 5 provides simulated and experimental results of DBBO performance on mobile robot controller tuning. Section 6 summarizes the results of this paper and suggests directions for future work.

2. Biogeography-based optimization

BBO is an evolutionary algorithm that was introduced in [35], and is modeled on the science of biogeography. Biogeography describes how species migrate between habitats based on environmental factors [12,19]. These environmental factors can be represented quantitatively and are called suitability index variables (SIVs). Examples of SIVs include the amount of rainfall, the amount of available fresh water, the diversity of vegetation, and the temperature range. An area that is highly suitable for the habitation of biological species is considered to have a high habitat suitability index (HSI). Biologists have developed mathematical models of migration, speciation, and extinction.

A high-HSI habitat is likely to have a large number of species. Therefore, because of the accumulation of probabilistic effects on its large population, it has a high probability of emigrating species to other habitats. Because of its dense population, and because it may be saturated with so many species that it is unable to support additional life forms, it has a low probability of immigrating species from other habitats. The opposite situation occurs in low-HSI habitats because of its sparse population. A habitat's emigration and immigration probabilities are therefore proportional to the number of species that live in the habitat.

BBO is modeled on the above description of migration probabilities. BBO includes a population of individuals, each of which is a candidate solution to some optimization problem. A BBO individual with high fitness is analogous to an island with a high HSI. That individual has a high probability of emigrating its features (that is, its decision variables) to other individuals. The individuals that receive those features tend to increase their own fitness. Similarly, a BBO individual with low fitness is analogous to an island with a low HSI. That individual has a high probability of immigrating features from other individuals.

The standard BBO algorithm is called centralized BBO here, to distinguish it from the distributed BBO algorithm. Section 2.1 reviews centralized BBO, and Section 2.2 introduces distributed BBO.

2.1. Centralized BBO

BBO individuals with high fitness have high emigration probability μ, and low immigration probability λ. Migration probabilities are normalized to [0, 1].

If we denote the entire population as $\{P_i\}$, with P_i being the ith individual in the population, then the migration probabilities of P_i are given as

$$\mu_i = \frac{f(P_i)-\min_k f(P_k)}{\max_k f(P_k)-\min_k f(P_k)}$$
$$\lambda_i = 1-\mu_i \tag{1}$$

where $f(P_i)$ is the fitness of P_i. Nonlinear relationships can also be used in BBO [14], but for the purposes of this paper, linear models are sufficient. We see the following from Eq. (1):

$$\text{Most fit individual in population}: \ \mu_i = 1, \quad \lambda_i = 0$$
$$\text{Least fit individual in population}: \ \mu_i = 0, \quad \lambda_i = 1.$$

If all individuals in the population have the same fitness, then the denominator in Eq. (1) is equal to 0. In this case we set the emigration and immigration probabilities to 1/2 for all of the individuals in the population.

As in other EAs, we also implement mutation in BBO to increase the exploration of the search space. Algorithm 1 describes the centralized BBO algorithm. In the "For each individual P_i" loop in Algorithm 1, the immigration decision for each individual and the selection of the emigrating individual are made independently of all previous decisions.

Algorithm 1. The centralized BBO algorithm.

```
Generate a population of individuals (that is, candidate
  solutions) $P=\{P_i\}$
While not (termination criterion)
   Calculate the fitness $f(P_i)$ of each individual in $P$
   Use Eq. (1) to calculate the migration probabilities of each
    individual in $P$
   For each individual $P_i$
         For each decision variable $v$ in $P_i$
               Use the immigration probability $\lambda_i$ to decide
                whether to immigrate to $P_i$
               If immigrating to $P_i$ then
                        Use $\{\mu_j\}$ to probabilistically select the
                        emigrating individual $P_k$
                        Migrate from $P_k$ to $P_i$: $P_i(v) \leftarrow P_k(v)$
                     End immigration
               Next decision variable
               Probabilistically mutate $P_i$
         Next individual: $i \leftarrow i+1$
Next generation
```

Author's personal copy

As in other EAs, the "termination criterion" in Algorithm 1 is problem-dependent and might be, for example, generation count, or some quantity that measures the convergence of the population's fitness. As in other EAs, we usually implement elitism in BBO, although this is not depicted in Algorithm 1. Elitism means that we save the best individuals at the end of each generation, and include them in the population at the beginning of the following generation. This ensures that we do not lose the best individuals, and that the best candidate solution never gets worse from one generation to the next. The mutation mentioned in Algorithm 1 can be performed in any way that is used in other EAs.

Algorithm 1 shows that BBO is a relatively simple algorithm. For real-world problems, the vast majority of computational effort and time required by BBO, as in any other EA, is due to fitness function evaluation. The computational effort required for fitness function evaluation "is usually so great that it will rarely pay to give any consideration at all to any other aspect of the run" for a typical EA [10, Appendix H]. As an example, each fitness function evaluation for our robot control example in Section 5 requires about 30 s (the time period for which the robots track the wall), while the BBO algorithm requires computational effort on the order of microseconds.

Fitness function evaluations for benchmark problems often require very little computational effort due to their simplicity. In this case the complexity of an EA might require more consideration. Previous work with benchmark function optimization has provided numerical comparisons of the computational effort required by BBO and other EAs [15,35].

2.2. Distributed BBO

Here we proposed a modified form of BBO, which we call distributed biogeography-based optimization (DBBO). DBBO was proposed conceptually in [13] and the first experimental results were given in [34]. Here we give a more thorough description of DBBO. DBBO has the same goal as centralized BBO, which is to find solutions to an optimization problem. However, DBBO does not use a central computer to control the evolution of the population. DBBO thus allows evolutionary optimization in case a central computer is not available, or in case the individuals in the population are scattered through time or space in such a way that centralized coordination is not possible. As an example, Fig. 1 shows a swarm of robots communicating without centralized coordination.

In DBBO, the evolutionary algorithm is executed by each individual in the population. In the practical application of DBBO that we demonstrate in Section 5, the DBBO individuals will be robots, as shown in Fig. 1, but in general, the DBBO individuals can be any candidate solutions to any optimization problem.

The DBBO algorithm is similar to BBO. However, in BBO the migration probabilities are determined by the entire population, as indicated by the "max" and "min" functions in Eq. (1). In DBBO, the migration probabilities for a group of communicating individuals are determined only by the individuals in that group.

Suppose that we have a subset G of the population P that is communicating with each other, where $G \subset P$, and G_i is the ith individual in G. We call G a peer group. Since we do not have a central computer, we do not know the minimum and maximum fitness values in the population, which are used in Eq. (1) to calculate migration rates for centralized BBO. However, due to the fact that DBBO individuals regularly communicate with each other, they can estimate the minimum and maximum fitness values of the population based on the fitness values of those individuals with whom they have communicated in the past. We use W_i and B_i to denote the ith individual's estimate of the worst and best fitness in the population, respectively. Then each time the ith individual G_i in G communicates within its group, it updates its estimates W_i and B_i as follows:

$$W_i = \min\left[W_i, \min_k f(G_k)\right]$$
$$B_i = \max\left[B_i, \max_k f(G_k)\right]. \tag{2}$$

Then instead of using Eq. (1), individual G_i sets its migration probabilities as

$$\mu_i = \frac{f(G_i) - W_i}{B_i - W_i}$$
$$\lambda_i = 1 - \mu_i. \tag{3}$$

We see the following from Eq. (3):

$$\text{Most fit individual in } G : \mu_i = 1, \quad \lambda_i = 0$$
$$\text{Least fit individual in } G : \mu_i = 0, \quad \lambda_i = 1.$$

In case the denominator is equal to 0 in Eq. (3), we set the emigration and immigration probabilities to 1/2 for all of the individuals in G. Algorithm 2 describes the DBBO algorithm. Similar to Algorithm 1, in the "For each individual G_i" loop in Algorithm 2, the immigration decision for each individual and the selection of the emigrating individual are made independently of all previous decisions. Also, the stochastic "Select M individuals" statement in Algorithm 2 is also made independently of all previous decisions.

Algorithm 2. The distributed BBO algorithm.

```
Generate a population of individuals (that is, candidate
solutions) $P=\{P_i\}$
Initialize the best fitness estimate of each individual: $B_i=-\infty$
Initialize the worst fitness estimate of each individual:
$W_i=+\infty$
While not (termination criterion)
    Select $M$ individuals $G=\{G_i\}$ to communicate with each
    other, where $G \subset P$
    Calculate the fitness $f(G_i)$ of each individual in $G$
    Use Eq. (2) to update the worst and best estimates for
    each individual in $G$
    Use Eq. (3) to calculate the migration probabilities of
    each individual in $G$
    For each individual $G_i$
        For each decision variable $v$ in $G_i$
            Use the immigration probability $\lambda_i$ to decide
            whether to immigrate to $G_i$
            If immigrating to $G_i$ then
                Use $\{\mu_j\}$ to probabilistically select
                the emigrating individual $G_k$
                Migrate from $G_k$ to $G_i$: $G_i(v) \leftarrow G_k(v)$
            End immigration
        Next decision variable
        Probabilistically mutate $G_i$
    Next individual: $i \leftarrow i+1$
Next generation
```

2.3. Peer group selection

The first statement inside the loop in Algorithm 2, "Select M individuals G to communicate with each other," is problem dependent. The number M depends on how many DBBO individuals are in contact with each other at any given generation. In some applications M will be fixed, while for other applications it will change from one generation to the next. The method for

Author's personal copy

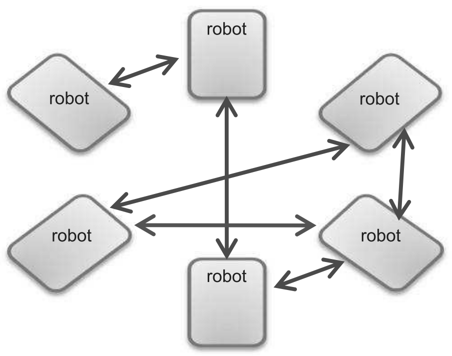

Fig. 1. Robots communicating with each other. If the robots are not in communication range of a central computer, then it is not possible to implement the standard BBO algorithm to evolve robot behavior. Instead, distributed BBO can be used to control the evolution of the population.

selecting the subset G is also problem dependent. For some applications the selection of G might be deterministic while for other applications the selection might be random.

In the Markov modeling and robot application of DBBO later in this paper, we randomly select individuals for membership in G each generation. This is motivated by our application, which is robot control optimization. A typical industrial task for a mobile robot might be to move along a hallway, looking for a particular office to deliver mail. As robots implement their control strategies, they move from one location to another. By the time they have completed their task, they have enough information to quantify their tracking performance, and to try to improve their control strategy. However, the group of robots that are in radio range of each other will have completely changed since the beginning of the task. That is, the peer group G randomly changes each generation.

We can view the population of robots as a connected graph where each vertex is a robot, and edges connect robots that are within radio range of each other. A peer group is a clique (that is, a complete subgraph). In practice the peer group does not need to be a clique as long as they form a strongly connected component (that is, as long as unconnected robots can communicate with each other through other robots). But for practical purposes we can consider a peer group to be a clique. Algorithm 2 assumes that DBBO runs in only one peer group at a time. In future work we might want to extend Algorithm 2 to allow multiple peer groups to simultaneously execute DBBO. This possibility introduces additional complications such as synchronization issues, and the possibility of a single robot belonging to multiple peer groups. We defer these issues to future research.

3. Markov modeling

In this section we use Markov theory to model DBBO and provide some mathematical tools for its analysis. We also use some simple simulations to verify the Markov model. Since Markov models of DBBO are mathematically exact in the limit as the generation count approaches infinity, they can be used in place of Monte Carlo simulations to quantify DBBO performance. The probability of an event with a very low likelihood may be difficult to measure with simulations. However, we can exactly obtain these probabilities, no matter how small, with Markov models (and also with dynamic system models, whose development follows that of Markov models [38]). We will see that Markov models are not computationally tractable for large problems; but for small problems, they provide exact results, and can therefore provide exact numerical comparisons between different algorithms and their variations.

A Markov model is a state sequence for which the probability of changing from a certain state s_i to another state s_k is given by the probability p_{ik}, which does not depend on any of the previous states of the system. The matrix $P=[p_{ik}]$ is called the Markov transition matrix. The set of possible states is $S=\{s_1,s_2,\ldots,s_r\}$.

In BBO Markov modeling, we assume that the BBO search space is discrete, and that a state represents a population distribution. That is, if the search space of BBO is represented as $\{x_i\}$, then the BBO state indicates how many of each x_i individual there are in the population.

Section 3.1 reviews previous research on Markov modeling for centralized BBO [37]. Section 3.2 extends the centralized BBO Markov model to DBBO. Section 3.3 presents some simulations to verify the DBBO Markov model.

3.1. Markov modeling for centralized BBO

This section reviews the Markov model for centralized BBO [37]. First, in Section 3.1.1, we define the notation. Then in Section 3.1.2 we show how BBO migration can be modeled with a Markov transition matrix. In Section 3.1.3 we show how the incorporation of mutation in the BBO algorithm modifies the Markov transition matrix.

3.1.1. Markov model notation

Suppose that we are trying to find a q-bit string to optimize some problem. The search space of BBO is represented as $\{x_i\}$, where each x_i is a bit string that consists of q-bits. For example, in an optimization problem for which the search space consists of three-bit strings, the search space could be represented as

$$\{x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8\}=\{000,001,010,011,100,101,110,111\}. \tag{4}$$

In this example, the cardinality of the state space is $n=8$.

In general, we use N to denote the BBO population size, and we use v to denote the population vector. The population vector contains the count of each bit string in the population, so

$$\sum_{i=1}^{n} v_i = N. \tag{5}$$

We use y_i to denote the ith individual (out of N total) in the BBO population. The population consists of a number of x_i components, and we order the BBO individuals in the same order as the elements in $\{x_i\}$, so the population can be written as

$$\text{Population}=\{y_1,\ldots,y_N\}=\{x_1,x_1,\ldots,x_2,x_2,\ldots,x_n,x_n\} \tag{6}$$

Recall that there are v_1 copies of x_1 in the population, v_2 copies of x_2, and so on. We use λ_i to represent the probability of immigration to x_i, and μ_i to represent the probability of emigration from x_i. We use $x_i(s)$ to represent the sth bit of x_i, where the bits are numbered from left to right, beginning with number 1. We use $\mathfrak{I}_i(s)$ to represent the search space indices that have the same sth bit values as the sth bit of x_i; that is,

$$\mathfrak{I}_i(s)=\{j : x_j(s)=x_i(s)\}. \tag{7}$$

The kth element in the population, y_k, can be written as

$$y_k = x_{m(k)} \quad \text{for} \quad k=1,\ldots,N \tag{8}$$

where $m(k)$ is given as

$$m(k)=\min r \quad \text{such that} \quad \sum_{i=1}^{r} v_i \geq k. \tag{9}$$

Example. To more clearly explain the above notation, we consider a simple example. Suppose we have the three-bit search

Author's personal copy

space of Eq. (4), and a BBO population size $N=3$, with $y_1=x_2=001$, $y_2=x_7=110$, and $y_3=x_7=110$. That is, $\{y_k\}=\{y_1y_2y_3\}=\{x_2x_7x_7\}$.

In this case, the elements of the population vector v are given as

$$v_1=0,\quad v_2=1,\quad v_3=0,\quad v_4=0,\quad v_5=0,\quad v_6=0,\quad v_7=2,\quad v_8=0.$$

That is, the population consists of one x_2 individual, and two x_7 individuals. We can derive $\Im_i(s)$ for the different bits in the various individuals in the population. For example, from Eq. (7),

$$\Im_1(1)=\{j: x_j(1)=x_1(1)\} \tag{10}$$

Since $x_1=000$, we see that $x_1(1)=0$. Eq. (10) can then be written as

$$\Im_1(1)=\{j: x_j(1)=0\}.$$

Note that the first bit of x_j is zero for $x_2=001$, $x_3=010$, and $x_4=011$. From this we see that $\Im_1(1)=\{1, 2, 3, 4\}$. If we continue this process we obtain

$$\begin{aligned}
&\Im_1(1)=\{1,2,3,4\}, \quad \Im_1(2)=\{1,2,5,6\}, \quad \Im_1(3)=\{1,3,5,7\}\\
&\Im_2(1)=\{1,2,3,4\}, \quad \Im_2(2)=\{1,2,5,6\}, \quad \Im_2(3)=\{2,4,6,8\}\\
&\Im_3(1)=\{1,2,3,4\}, \quad \Im_3(2)=\{3,4,7,8\}, \quad \Im_3(3)=\{1,3,5,7\}\\
&\Im_4(1)=\{1,2,3,4\}, \quad \Im_4(2)=\{3,4,7,8\}, \quad \Im_4(3)=\{2,4,6,8\}\\
&\Im_5(1)=\{5,6,7,8\}, \quad \Im_5(2)=\{1,2,5,6\}, \quad \Im_5(3)=\{1,3,5,7\}\\
&\Im_6(1)=\{5,6,7,8\}, \quad \Im_6(2)=\{1,2,5,6\}, \quad \Im_6(3)=\{2,4,6,8\}\\
&\Im_7(1)=\{5,6,7,8\}, \quad \Im_7(2)=\{3,4,7,8\}, \quad \Im_7(3)=\{1,3,5,7\}\\
&\Im_8(1)=\{5,6,7,8\}, \quad \Im_8(2)=\{3,4,7,8\}, \quad \Im_8(3)=\{2,4,6,8\}.
\end{aligned}$$

3.1.2. Migration in centralized BBO

As seen in the previous section, each decision variable in individual y_k in a BBO population has a $\lambda_{m(k)}$ probability of being selected for immigration. We use $y_k(s)_t$ to denote the sth bit of y_k at generation t. If the sth bit of y_k is not chosen for immigration during generation t, then $y_k(s)$ does not change from generation t to generation $t+1$:

$$y_k(s)_{t+1}=x_{m(k)}(s) \quad \text{if no immigration to } y_k. \tag{11}$$

Therefore, the conditional probability that $y_k(s)_{t+1}$ is equal to $x_i(s)$ can be written as

$$\Pr[y_k(s)_{t+1}=x_i(s)]=1_0\left(x_{m(k)}(s)-x_i(s)\right) \quad \text{if no immigrationto } y_k \tag{12}$$

where 1_0 is the indicator function on the set {0}. That is, $1_0(x)=1$ if $x=0$, and $1_0(x)=0$ if $x\neq 0$.

If the sth bit of y_k is chosen for immigration during generation t, then the probability that $y_k(s)_{t+1}=x_i(s)$ is proportional to the sum of all of the emigration probabilities of the individuals whose sth bit is equal $x_i(s)$:

$$\Pr\left[y_k(s)_{t+1}=x_i(s)\right]=\frac{\sum_{j\in\Im_i(s)} v_j\mu_j}{\sum_{j=1}^{n} v_j\mu_j} \quad \text{if immigration to } y_k \tag{13}$$

Eqs. (12) and (13) can be combined to obtain

$$\begin{aligned}
\Pr[y_k(s)_{t+1}=x_i(s)] &= \Pr(\text{no immto } y_k)\Pr[y_k(s)_{t+1}=x_i(s)\,|\,\text{no imm to } y_k]+\\
&\quad \Pr(\text{imm to } y_k)\Pr[y_k(s)_{t+1}=x_i(s)\,|\,\text{imm to } y_k]\\
&= \left(1-\lambda_{m(k)}\right)1_0\left(x_{m(k)}(s)-x_i(s)\right)+\lambda_{m(k)}\frac{\sum_{j\in\Im_i(s)} v_j\mu_j}{\sum_{j=1}^{n} v_j\mu_j}.
\end{aligned} \tag{14}$$

Now we use $P_{ki}(v)$ to denote the probability that, given the population vector v, the kth individual in the population, y_k, is equal to the ith individual in the search space, x_i, at the $(t+1)$st generation. Taking into consideration that there are q-bits in each BBO individual, and that migration to each bit is independent (as seen from Algorithm 1 in Section 2.1), $P_{ki}(v)$ can be computed from Eq. (14) as

$$\begin{aligned}
P_{ki}(v) &= \Pr\left(y_{k,t+1}=x_i\right)\\
&= \prod_{s=1}^{q}\left[\left(1-\lambda_{m(k)}\right)1_0\left(x_{m(k)}(s)-x_i(s)\right)+\lambda_{m(k)}\frac{\sum_{j\in\Im_i(s)} v_j\mu_j}{\sum_{j=1}^{n} v_j\mu_j}\right].
\end{aligned} \tag{15}$$

Computing this probability for each $k\in[1, N]$ and for each $i\in[1,n]$, we can find the $N\times n$ transition matrix $P(v)$. This is not yet the Markov transition matrix, but the element in the kth row and ith column of $P(v)$, which is denoted as $P_{ki}(v)$, gives the probability that kth individual in the population at the $(t+1)$st generation is equal to x_i.

The probability that the BBO population transitions from a given population vector v to another population vector u can be calculated with the multinomial theorem [37]. The multinomial theorem tells us how to calculate the probability that the ith possible outcome occurs exactly u_i times after N trials, where the probability of the ith outcome at the kth trial is given as $P_{ki}(v)$. Applying the multinomial theorem to BBO allows us to calculate $\Pr(u|v)$, which is the probability that the BBO population transitions from a given population vector v to another population vector u:

$$\begin{aligned}
&\Pr(u|v)=\sum_{J\in Y}\prod_{k=1}^{N}\prod_{i=1}^{n}[P_{ki}(v)]^{J_{ki}}\\
&\quad\text{where}\quad Y=\{J\in R^{N\times n}\ :\ J_{ki}\in\{0,1\}, \sum_{i=1}^{n}J_{ki}=1 \ \text{for all } k,\\
&\quad\text{and}\ \sum_{k=1}^{N}J_{ki}=u_i \ \text{for all } i\}.
\end{aligned} \tag{16}$$

Details for the derivation of Eq. (16), along with an example, are given in [37].

3.1.3. Mutation in centralized BBO

The previous section obtained the Markov transition matrix for centralized BBO when migration was the only BBO operation. However, in BBO, we also use mutation. We include mutation in the Markov model with an $n\times n$ mutation matrix U, where U_{ik} is the probability that the individual x_k mutates to x_i [29]. We used $P_{ki}(v)$ in the previous section to denote the probability that $y_k=x_i$ when only migration is taken into account. In this section, we use $P_{ki}^{(2)}(v)$ to denote the same probability with both migration and mutation taken into account. The equation for this probability is

$$P_{ki}^{(2)}(v)=\sum_{j=1}^{n}U_{ij}P_{kj}(v) \tag{17}$$

which gives

$$P^{(2)}(v)=P(v)U^T \tag{18}$$

where the components of $P(v)$ are given in Eq. (15). Again using the multinomial theorem as in the previous section, we obtain the probability of transitioning from population vector v to population vector u in a single generation as

$$\Pr^{(2)}(u|v)=\sum_{Y}\prod_{k=1}^{N}\prod_{i=1}^{n}\left[P_{ki}^{(2)}(v)\right]^{J_{ki}}. \tag{19}$$

Author's personal copy

This equation can be used to determine the transition probability between all possible population distributions u and v. Combining all of these transition probabilities gives a matrix Q which contains the probabilities of transitioning between each population distribution. Q is a $T \times T$ matrix, where T is the number of possible BBO population distributions [37], which depends on the population size and the search space cardinality.

3.2. Markov modeling for distributed BBO

This section extends BBO Markov modeling to DBBO. Recall the step in the DBBO procedure of Algorithm 2 that says, "Select M individuals G to communicate with each other." In this section, we assume that M is a constant, and we also assume that the selection of the subset G is random. Therefore, we can use the previously-developed Markov model by realizing that in DBBO, $\lambda_k = 0$ for the $(N-M)$ randomly chosen individuals that are not part of G, where N is the population size. We refer to G as the peer group, which includes the M randomly chosen individuals:

$$G = \{y_{i1}, \ldots, y_{iM}\} \tag{20}$$

where $\{i_1, \ldots, i_M\}$ are unique random integers from $\{1, \ldots, N\}$. From combinatorics, we know that there are

$$C_{N,M} = \frac{N!}{M!(N-M)!} \tag{21}$$

unique possible peer groups.

As an example, suppose that we have population size $N=4$, that the population $P=\{y_1, y_2, y_3, y_4\}$, that the search space cardinality $n=8$ (that is, the search space consists of three-bit individuals), and that the peer group size $M=3$. In this case, there are four possible peer groups:

$$\{y1, y2, y3\}, \{y1, y3, y4\}, \{y2, y3, y4\} \quad \text{and} \quad \{y1, y2, y4\}.$$

Now we consider, for example, only peer groups that include y_2. There are three such possible peer groups:

$$G_1 = \{y_1, y_2, y_3\}, \quad G_2 = \{y_2, y_3, y_4\}, G_3 = \{y_1, y_2, y_4\}.$$

We can calculate how many unique peer groups include y_2 by using a modified form of Eq. (21):

$$C_{N-1,M-1} = \frac{(N-1)!}{(M-1)!(N-1-(M-1))!} = \frac{(N-1)!}{(M-1)!(N-M)!} \tag{22}$$

In DBBO, each individual y_k in the BBO population has a a specific probability of immigration. The probability of not immigrating to y_k includes: (1) the probability of not immigrating given that y_k is randomly selected as part of peer group G, and (2) the probability of not immigrating given that y_k is *not* randomly selected as part of G:

$$\begin{aligned}\Pr(\text{no imm to } y_k) &= \Pr(\text{no imm} \mid y_k \in G)\Pr(y_k \in G) \\ &\quad + \Pr(\text{no imm} \mid y_k \notin G)\Pr(y_k \notin G) \\ &= (1-\lambda_{m(k)})\left(\frac{M}{N}\right) + (1)\left(1-\frac{M}{N}\right) = 1-\left(\frac{M}{N}\right)\lambda_{m(k)}.\end{aligned} \tag{23}$$

Similarly, the probability of immigrating to y_k is

$$\begin{aligned}\Pr(\text{imm to } y_k) &= \Pr(\text{imm} \mid y_k \in G)\Pr(y_k \in G) + \Pr(\text{imm} \mid y_k \notin G)\Pr(y_k \notin G) \\ &= (\lambda_{m(k)})\left(\frac{M}{N}\right) + (0)\left(1-\frac{M}{N}\right) \\ &= \left(\frac{M}{N}\right)\lambda_{m(k)}.\end{aligned} \tag{24}$$

Comparing these immigration probabilities to those of centralized BBO, in which the probability of migration is simply $\lambda_{m(k)}$, we see that the probability of immigration has been modified by the factor M/N. This factor is the ratio of the peer group size to the population size. We call this probability of immigration $\lambda'_{m(k)}$ for distributed BBO. Using this notation, we rewrite Eqs. (23) and (24) for the no-immigration probability and the immigration probability as

$$\begin{aligned}\Pr(\text{no imm.to } y_k) &= 1-\lambda'_{m(k)} \\ \Pr(\text{imm. to } y_k) &= \lambda'_{m(k)}.\end{aligned} \tag{25}$$

Now recall that the population vector is denoted as v. We denote the population vector of the peer group G as v'. With this notation, we use an analysis similar to the centralized BBO analysis of Eq. (13). We note that if immigration to $y_k(s)$ occurs, then the probability that $y_k(s)_{t+1} = x_i(s)$ is proportional to the sum of all of the emigration probabilities of those individuals in G whose sth bit is equal $x_i(s)$:

$$\Pr(y_k(s)_{t+1} = x_i(s) \mid \text{imm}) = \frac{\sum_{j \in \mathfrak{I}_i(s)} v'_j \mu_j}{\sum_{j=1}^{M} v'_j \mu_j}. \tag{26}$$

Note that the DBBO peer group population vector v' has n elements and satisfies the following properties:

$$\begin{aligned}&\sum_{i=1}^{n} v_i{}' = M \\ &v_i{}' \in [0, M] \text{ for all } i \in [1, n].\end{aligned} \tag{27}$$

As we noted above in Eq. (22), in DBBO there are $C(N-1, M-1)$ possible peer groups, all of which are equally likely, which contain a given individual y_k. There are thus $C_{N-1,M-1}$ possible v' vectors. We denote the αth possible v' vector as $v'(\alpha)$. With this notation, we can write the DBBO counterpart to Eq. (14) as

$$\begin{aligned}&\Pr(y_k(s)_{t+1} = x_i(s)) = \Pr(\text{no imm to } y_k)\Pr(y_k(s)_{t+1} = x_i(s) \mid \text{no imm to } y_k) + \\ &\Pr(\text{imm to } y_k)\Pr(y_k(s)_{t+1} = x_i(s) \mid \text{imm to } y_k) \\ &\quad = \left(1-\lambda'_{m(k)}\right) 1_0\left(x_{m(k)}(s) - x_i(s)\right) + \frac{\lambda'_{m(k)}}{C_{N-1,M-1}} \sum_{\alpha=1}^{C_{N-1,M-1}} \left(\frac{\sum_{j \in \mathfrak{I}_i(s)} v_j{}'(\alpha)\mu_j}{\sum_{j=1}^{n} v_j{}'(\alpha)\mu_j} \right).\end{aligned} \tag{28}$$

Now, as in Eq. (15) of the centralized BBO analysis, we use $P_{ki}(v)$ to denote the probability that, given the population vector v, the kth individual in the DBBO population, y_k, is equal to the ith individual in the search space, x_i, at the $(t+1)$st generation:

$$\begin{aligned}P_{ki}(v) &= \Pr(y_{k,t+1} = x_i) \\ &= \prod_{s=1}^{q} \left[\left(1-\lambda'_{m(k)}\right) 1_0\left(x_{m(k)}(s) - x_i(s)\right) + \frac{\lambda'_{m(k)}}{C_{N-1,M-1}} \sum_{\alpha=1}^{C_{N-1,M-1}} \left(\frac{\sum_{j \in \mathfrak{I}i(s)} v_j{}'(\alpha)\mu_j}{\sum_{j=1}^{n} v_j{}'(\alpha)\mu_j} \right) \right].\end{aligned} \tag{29}$$

Just as in the centralized BBO analysis at the end of Section 3.1.2, we can use the above $P_{ki}(v)$ to find the probability that the DBBO population transitions from a given population vector v to another population vector u. Eq. (16) therefore holds for DBBO, if we use the $P_{ki}(v)$ quantities from Eq. (29) in Eq. (16). The incorporation of mutation in DBBO exactly follows the development of Section 3.1.3, assuming that we allow the possibility of mutation for each individual in the population at each generation.

3.3. Simulation results and Markov model confirmation

In this section, we use some simple simulations to confirm the DBBO Markov model.

We use three-bit problems with a search space cardinality of eight and a population size of five ($n=8$ and $N=5$). From [37,

Author's personal copy

Appendix B] we calculate the total number of possible populations as

$$T = \binom{N+n-1}{N} = \binom{12}{5} = 792. \tag{30}$$

Therefore, the state transition matrix Q is a 792×792 matrix. The element in the ith row and kth column of Q gives the probability that the population transitions from the ith population distribution v_i to the kth population distribution v_k. Repeatedly multiplying Q by itself results in the matrix Q^∞ in the limit as the number of multiplications approaches ∞, and each column of Q^∞ is identical [29]. The element in the ith row of each column of Q^∞ gives the probability that the DBBO population distribution after an infinite number of generations is equal to the ith possible population distribution v_i. Other more efficient and numerically robust methods also exist for finding these probabilities, but the above description is meant to give the general idea.

Note that Q must be positive recurrent in order for Q^∞ to exist; that is, every state must have a finite average time between occurrences. This property is guaranteed for the BBO and DBBO Markov chains if every element of the mutation matrix U is positive [17].

We also note that the beginning of Section 3.2 assumed that M is constant, and that the selection of the subset G is random. The first assumption can probably be relaxed; that is, the Markov model of Section 3.2 can probably be rederived without much difficulty if M varies from one generation to the next. However, if G is not random, then Q will be time-varying; it will change from one generation to the next as the peer group changes. In this case, the limiting population distribution Q^∞ might not exist. We can still implement DBBO in this case, but additional research is required to determine the properties of the Markov model for this situation.

We use the one-max problem and a deceptive problem to confirm the DBBO Markov model of Section 3.2. The one-max problem has a fitness function that is proportional to the number of ones in the individual, and is a popular test function in EA research [1]. The fitness values of the three-bit one-max problem are

$$f(000)=1, \quad f(001)=2, \quad f(010)=2, \quad f(011)=3,$$
$$f(100)=2, \quad f(101)=3, \quad f(110)=3, \quad f(111)=4.$$

The fitness values of the three-bit deceptive problem are the same as those of the one-max problem, except that the most fit individual is comprised of all zeros. The fitness values of the deceptive problem are

$$f(000)=5, \quad f(001)=2, \quad f(010)=2, \quad f(011)=3,$$
$$f(100)=2, \quad f(101)=3, \quad f(110)=3, \quad f(111)=4.$$

Table 1 shows a comparison of the analytical DBBO Markov model results and simulation results. The simulation results are averaged over 10 Monte Carlo simulations, with each simulation running for 10,000 generations. We used a mutation rate of 10% per bit per generation. We see that the simulation results match the Markov model results to within one standard deviation, thus confirming the Markov model equations derived in Section 3.2. Since the population size is five, $M=5$ in Table 1 is equivalent to the centralized BBO algorithm in which the entire population is in communication with each other.

Table 1 shows that the one-max problem results in a greater probability than the deceptive problem of finding at least one optimum, which is as expected. Table 1 also shows that as the size of the peer group increases, the performance of DBBO also increases. However, the point of Table 1 is not to investigate performance, but rather to confirm the Markov model of Section 3.2.

Table 1
This table shows the probability (percent) that the three-bit DBBO algorithm with a population size of five contains at least one optimal individual. The simulation results show the average and standard deviation of 10 simulations. M is the number of individuals in the randomly-selected peer group. The mutation rate was 10% per bit.

M	One-max problem		Deceptive problem	
	Markov model	Simulation	Markov model	Simulation
2	60.8	61.5 ± 1.6	51.9	53.2 ± 2.7
3	69.3	70.0 ± 1.1	53.1	56.1 ± 3.9
4	75.7	76.2 ± 1.3	54.8	57.4 ± 3.9
5	80.1	80.4 ± 0.8	57.3	55.2 ± 4.8

Finally, we note that the Markov model and simulation results both use the mutation matrix described in Section 3.1.3. This assumes that even though communication occurs only among a peer group of size M at each generation, mutation still occurs throughout the entire population at each generation. This assumption may or may not be valid, depending on the particular DBBO implementation. If mutation occurs only among the peer group at each generation, then the mutation matrix U needs to be modified accordingly.

The Markov model developed here can be used to provide exact quantitative comparisons between BBO, DBBO with various parameter settings, and other EAs [39]. It can also be used to obtain convergence conditions [16] and dynamic system models [38], which are exact in the limit as the population count approaches infinity. We leave these Markov model applications as important tasks for future research.

4. Benchmark simulation results

In this section we test BBO and DBBO on some benchmark problems from the 2005 competition of the IEEE Congress on Evolutionary Computation (CEC) [42]. The CEC 2005 benchmarks are continuous-domain problems. The Markov models presented in the previous section apply only to binary encodings. They can be extended in a straightforward way to other discrete encodings, but not to continuous state spaces. For continuous state spaces, the Markov transition matrix is replaced with a transition kernel K, where $K(x, A)$ is the probability of transitioning from the state x to the region A in state space. With this change we could obtain results that are analogous to those in Section 3, but the mathematics of continuous-state-space Markov processes are more involved than those of discrete-state-space Markov processes [32]. We therefore relegate the development of Markov models for continuous-domain BBO and DBBO to future work.

The CEC 2005 benchmarks that we use in this paper are as follows.

F_1: Shifted sphere function.
F_2: Shifted Schwefel's problem 1.2.
F_3: Shifted Rotated High Conditioned Elliptic Function.
F_5: Schwefel's problem 2.6 with global optimum on bounds.
F_6: Shifted Rosenbrock's function.
F_7: Shifted rotated Griewank's function without bounds.
F_9: Shifted rotated Ackley's function with global optimum on bounds.
F_{10}: Shifted Rastrigin's function.
F_{11}: Shifted rotated Rastrigin's function.
F_{12}: Shifted rotated Weierstrass function.
F_{15}: Schwefel's problem 2.13.

We choose these functions because they have known solutions. We do not test with F_4 because it is a noisy function, and we

Author's personal copy

Table 2
Comparison between BBO and 11 other EAs on five unimodal problems. The algorithms are listed in order from best to worst. See the text for a more detailed description of this table.

	Number of solved functions	Success rate (%)	F_1	F_2	F_3	F_5	F_6
G-CMA-ES	5	100	1.6 (25)	1 (25)	1 (25)	1 (25)	1.5 (25)
L-CMA-ES	5	100	1.7 (25)	1.7 (25)	1.1 (25)	1 (25)	1.3 (25)
EDA	5	97	10 (25)	4.6 (25)	2.5 (23)	4.2 (25)	9.6 (22)
DE	5	96	29 (25)	19.2 (25)	18.5 (20)	6.9 (25)	6.6 (24)
DMS-L-PSO	5	96	12.0 (25)	5.0 (25)	1.8 (25)	18.6 (20)	7.7 (25)
L-SaDE	5	93	10.0 (25)	4.2 (25)	8.0 (16)	10.7 (25)	6.8 (25)
BBO	**5**	**84**	**1.7 (25)**	**0.8 (25)**	**0.3 (25)**	**8.1 (5)**	**1.1 (25)**
BLX-GL50	4	83	19.0 (25)	17.1 (25)	[10] 932	4.7 (25)	7.3 (25)
SPC-PNX	3	67	6.7 (25)	12.5 (25)	[12] 10806	6.8 (25)	[11] 18.9
CoEVO	3	67	23.0 (25)	11.3 (25)	6.8 (25)	[11] 2.13	[12] 12.5
K-PCX	3	58%	1.0 (25)	1 (25)	[9] 0.42	[12] 48.5	1.0 (22)
BLX-MA	2	40	12.0 (25)	15.4 (25)	[11] 4771	[10] 0.02	[10] 1.49

have not implemented any noise-handling capabilities in BBO. Each of these problems can be implemented with any number of dimensions. F_1–F_6 are unimodal problems, and F_7–F_{15} are multimodal problems. Problems F_1–F_5 are considered to be solved if we come within 10^{-6} of the global minimum, and problems F_6–F_{15} are considered to be solved if we come within 10^{-2} of the global minimum.

We compared our BBO and DBBO algorithms to the 11 algorithms that were accepted for the CEC 2005 competition, which we refer to as the *baseline algorithms:*

1. BLX-GL50, which is a two-sex GA with original crossover operators [9].
2. BLX-MA, which is an adaptive memetic algorithm [20].
3. CoEvo, which is a co-evolutionary algorithm [25].
4. DE, which is differential evolution [30].
5. DMS-L-PSO, which is a multi-swarm particle swarm method [11].
6. EDA, which is an estimation of distribution algorithm [44].
7. G-CMA-ES, which is a covariance matrix adaptation evolution strategy [3].
8. K-PCX, which is an amalgamation of various EA strategies [41].
9. L-CMA-ES, which is another covariance matrix adaptation evolution strategy [4].
10. L-SaDE, which is an adaptive differential evolution algorithm [26].
11. SPC-PNX, which is a continuous genetic algorithm [5].

We collected benchmark performance data for these EAs from the above references.

4.1. Centralized BBO results

First we compare the results of centralized BBO with the 11 algorithms listed above. We run 25 Monte Carlo simulations, use a population size of 100, use a 1% mutation probability, and implement elitism by replacing the two worst individuals each generation with the two best individuals from the previous generation. We also augment BBO with a standard local search algorithm. We limit our benchmark study to 10-dimensional problems, and we impose a function evaluation limit of 100,000.

As in the CEC 2005 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 solves the problems the most times is better.

Table 2 shows the results of BBO and the baseline algorithms on the unimodal problems. The "number 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 how many of the total number of (25×5) simulations were successful at solving a problem.

Each cell in Table 2 corresponds to a given algorithm and a given problem, and contains two numbers. If the algorithm was successful in solving the problem at least once, then the number of successes is in parentheses. If the algorithm was *not* successful even once in solving the problem, then the number in square brackets indicates the performance rank of the algorithm for that particular problem, out of a total of 12 algorithms. The number outside of the parentheses or brackets in each cell shows the normalized function value that was achieved by the algorithm, averaged over 25 Monte Carlo simulations.

Table 2 shows that BBO 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 BBO. On the other hand, BBO is the best algorithm for the F_2, F_3, and F_6 benchmarks.

Table 3 shows the results of BBO and the baseline algorithms on the multimodal problems that we tested. BBO is ranked third out of the 12 algorithms. However, BBO has the best success rate (65%). BBO is the best algorithm for the F_9 and F_{12} benchmarks. In addition, BBO is the only algorithm that solved F_{15} 100% of the time.

We realize that we could perform many more benchmark tests, including additional benchmark functions, higher dimensions, and additional EAs. The results we present here on a widely-accepted set of benchmark functions confirm the competitive ability of BBO. Also, by extension, they confirm the competitive ability of DBBO, as discussed in the following section.

4.2. Distributed BBO results

Next we test distributed BBO on the benchmark functions. We implement DBBO with three different peer group sizes: 2, 4, and 6. We call these algorithms DBBO/2, DBBO/4, and DBBO/6. Tables 4 and 5 show the comparison between BBO and DBBO. We see that the performance of the centralized and distributed versions of BBO are very similar. For the unimodal functions in Table 4, DBBO/6 performs slightly better than BBO, although the difference is probably not statistically significant. For the multimodal functions in Table 5, BBO performs slightly better than any of the DBBO versions, although, again, the difference between the performance levels is probably not statistically significant.

We intuitively expect BBO to outperform DBBO. But just because BBO has more individuals to choose from when performing migration, that does not guarantee that it will outperform DBBO. The fact that BBO has more individuals to choose from might increase the probability of a detrimental migration. Further

Author's personal copy

Table 3
Comparison between BBO and 11 other EAs on five multimodal problems. The algorithms are listed in order from best to worst. See the text for a more detailed description of this table.

	Number of solved functions	Success rate (%)	F_7	F_9	F_{10}	F_{11}	F_{12}	F_{15}
G-CMA-ES	5	63	1.0 (25)	4.5 (19)	1.2 (23)	1.4 (6)	4.0 (22)	[9] 400
DE	5	30	255 (2)	10.6 (11)	[9] 13.7	1.0 (12)	8.8 (19)	75.8 (1)
BBO	**4**	**66**	**9.9 (24)**	**0.2 (25)**	**[8] 12.4**	**[10] 5.1**	**0.3 (25)**	**0.06 (25)**
L-SaDE	4	53	36.2 (6)	1.0 (25)	[4] 5.0	[8] 4.9	3.9 (25)	1.0 (23)
DMS-L-PSO	4	47	126 (4)	2.1 (25)	[3] 3.6	[7] 4.6	6.6 (19)	1.7 (22)
K-PCX	3	40	[12] 0.23	2.9 (24)	1.0 (22)	[11] 6.7	1.0 (14)	[11] 510
BLX-GL50	3	17	12.3 (9)	10.0 (3)	[4] 5.0	[5] 2.3	12.1 (13)	[9] 400
EDA	3	9	404 (1)	[11] 32.3	[11] 32.3	2.9 (3)	4.3 (10)	[12] 511
L-CMA-ES	2	25	1.2 (25)	[12] 420	[12] 1270	[6] 2.7	11.6 (12)	[6] 115
BLX-MA	2	15	[11] 0.20	5.7 (18)	[6] 5.6	[9] 4.6	[10] 74.3	8.5 (5)
SPC-PNX	2	1	383 (1)	[9] 4.0	[7] 7.3	5.8 (1)	[11] 260	[7] 254
CoEVO	0	0	[10] 0.037	[10] 19.2	[10] 26.8	[12] 9.0	[12] 605	[8] 294

Table 4
Comparison between BBO and DBBO on five unimodal benchmarks.

	Number of solved functions	Success rate (%)	F_1	F_2	F_3	F_5	F_6
BBO	5	84	1.7 (25)	0.80 (25)	0.3 (25)	8.1 (5)	1.1 (25)
DBBO/2	5	82	1.7 (25)	0.82 (25)	0.4 (25)	2.5 (2)	1.1 (25)
DBBO/4	5	84	1.7 (25)	0.82 (25)	0.4 (25)	5.2 (5)	1.1 (25)
DBBO/6	5	85	1.7 (25)	0.82 (25)	0.4 (25)	9.1 (6)	1.2 (25)

Table 5
Comparison between BBO and DBBO on six multimodal benchmarks.

	Number of solved functions	Success rate (%)	F_7	F_9	F_{10}	F_{11}	F_{12}	F_{15}
BBO	4	66	9.9 (24)	0.2 (25)	12.4	5.1	0.3 (25)	0.06 (25)
DBBO/2	4	66	9.6 (24)	0.2 (25)	28.1	5.3	0.4 (25)	0.08 (25)
DBBO/4	4	65	9.2 (23)	0.2 (25)	25.0	5.7	0.4 (25)	0.08 (25)
DBBO/6	4	63	10.1 (20)	0.2 (25)	24.5	5.1	0.4 (25)	0.08 (25)

research is recommended to determine the conditions under which BBO or DBBO give better optimization results.

The benchmark tests in this section are not extensive. For further research we recommend a more complete set of benchmarks, including problems with higher dimensions, and also comparisons to a broader set of optimization algorithms. However, the preliminary comparisons in this section are sufficient to show that BBO and DBBO are competitive optimization algorithms.

5. Robot optimization using BBO and DBBO

This section discusses the use of BBO and DBBO for robot controller optimization. Section 5.1 discusses the robot hardware that we used, Section 5.2 gives an overview of the robot control task, Section 5.3 presents simulation results, and Section 5.4 presents experimental hardware results.

5.1. Robot hardware

The mobile robots that we use for this research were developed for a mapping application [7]. They were first used in BBO research in [13], and were used for preliminary DBBO research in [34]. This section gives some information about the robot hardware.

The robots are equipped with two DC motors (one for each rear wheel), and eight AA batteries that provide two separate power supplies. The batteries power the two motors (first power supply) and the digital electronics (second power supply). The robots are controlled by a Microchip PIC18F4520 microcontroller. The microcontroller controls the robot wheel motors, and controls radio communication with a personal computer (PC). Two voltage regulators are on each robot, one to distribute 5 V to the microcontroller and other digital electronics, and one to power the motors. A digital signal from the microcontroller switches the motor power supply using an H-bridge SN754410NE. Infrared sensors measure the distance of the robot from the wall using a light-emitting sensor and a light-detecting sensor. The robot's task is to track the wall as closely as possible while traveling at a constant velocity. When running centralized BBO, the microcontroller communicates with a PC using a wireless radio, the

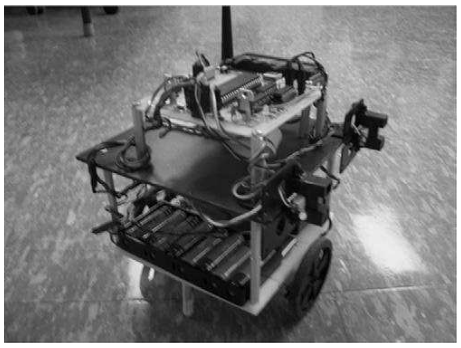
Fig. 2. Photograph of a mobile robot used for BBO and DBBO testing.

Author's personal copy

MaxStream 9Xtend radio, and the PC runs the BBO algorithm. When running DBBO, the robots communicate with each other using the 9Xtend radio. Fig. 2 shows a photograph of one of the robots.

Next we give some details about the DBBO logic of Algorithm 2 as it is implemented by the robots.

- "Select M individuals $G=\{G_i\}$ to communicate with each other"—This statement in Algorithm 2 is implemented by starting with a single randomly-selected robot, and then continuing the selection process through a random sequence of robots. In our case, the first robot is selected by a human operator pressing a switch on a randomly-selected robot. The first robot then randomly selects a second robot with which it is in radio contact. This process continues until M robots are in radio contact with each other.
- "Calculate the fitness $f(G_i)$ of each individual in G"—This statement in Algorithm 2 is implemented by each robot keeping track of its own control performance as discussed later in Section 5.3. Each robot in G transmits its fitness $f(G_i)$ to every other robot in G.
- Each robot also transmits its decision variables (that is, its control variables) to every other robot in G. Therefore, each robot in G is aware of the decision variables of every other robot in G.
- "Use Eq. (2)… Use Eq. (3)…"—These two lines in Algorithm 2 are implemented by each robot in G. Therefore, each robot in G has an identical copy of the migration probabilities for its peer group.
- "For each individual G_i"—This loop in Algorithm 2 is implemented in parallel by each robot in peer group G. The migration in this loop is possible because each robot previously transmitted its decision variables to every other robot in G, as described above. The robots' migration of decision variables, and the possible mutation of decision variables, comprises the robot controller tuning algorithm.
- The clock speed of each robot's microcontroller is 40 MHz, which gives an instruction frequency of 10 MHz. The DBBO function was written in C and compiled to 1866 assembly code instructions. A worst-case timing analysis shows that a maximum of 3064 instruction cycles execute during the DBBO logic, which requires 31 μs to execute. The remainder of the robot code, also written in C, consists of infrared sensor processing, motor control logic that executes at a rate of 10 Hz, radio communications, and LCD output logic, and compiles to 20184 assembly code instructions.

5.2. Robot control

The robot controller is a proportional-integral-derivative (PID) controller [2]. A PID controller includes a proportional gain, an integral gain, and a derivative gain. The proportional gain, K_p, is the primary determinant of the control signal magnitude, and thus is the primary determinant of the responsiveness of the control system. The higher the value of K_p, the faster the controller responds to tracking errors. The integral gain, K_i, helps reduce steady-state error. The derivative gain, K_d, is multiplied by the rate of change of the tracking error, and the product is added to the control signal command. The derivative gain helps maintain controller stability, and decreases the amount of overshoot due to K_i and K_p. The PID controller can be described as follows:

$$\Delta u = K_p e + K_d \dot{e} + K_i \int e \, dt \qquad (31$$

where Δu is the change in the control signal from one time step to the next, e is the tracking error, and the limits of the integral depend on the user's implementation.

In the simulations for our robots, we found that the integral term did not provide significant improvement compared to the proportional and derivative terms [13]. Although integral control reduces steady-state error, it also tends to increase overshoot and settling time. Because of this fact and the relatively short duration of our robot task (20 s), the integral term degrades performance for the robot control problem. Therefore, we use only the K_p and K_d terms; that is, we use PD controllers. If we apply our research to other control tasks, or to the same control task with a longer duration, we will need to use the integral term to get better performance.

The robots are programmed to follow a wall using the PD controller to maintain a specified distance from the wall. The robot uses infrared sensors to measure its distance from the wall and its angle relative to the wall. The robot controller uses the tracking error and the PD controller to correct the motor commands and maintain a fixed distance from the wall. See [13] for more details about the robot control algorithm. Fig. 3 shows a simplified diagram of the robot control task.

PD control is relatively simple; in practice we do not need a tuning algorithm as complex as DBBO to optimize PD parameters. We use the PD control task as a simple but realistic example of the ability of DBBO to optimize an experimental system. Furthermore, even though PD control tuning could be done by human operators, computer and robot time might be much cheaper than human effort. DBBO and other automatic algorithms can be used to tune simple systems like PD controllers with very little human intervention, which could result in cost savings.

5.3. Robot simulation results

This section discusses our MATLAB® simulation of the robot and its controller as described in the previous section, and our use of BBO and DBBO to optimize the K_p and K_d terms of the robots' PD controllers.

We used 500 function evaluations with a population size of 50 and a mutation rate of 1% in our BBO and DBBO algorithms. Each individual in the population is a mobile robot. Our function evaluation limit is much smaller than is typically used in EA simulations. This is motivated by the low number of function evaluations that are often used in real-world optimization problems for which the system needs to optimize itself on-line.

To use BBO and DBBO, we need to set the search domain of each independent variable. The domain of K_p was set to [0, 2], and the domain of K_d was set to [0, 10]. These domains were chosen

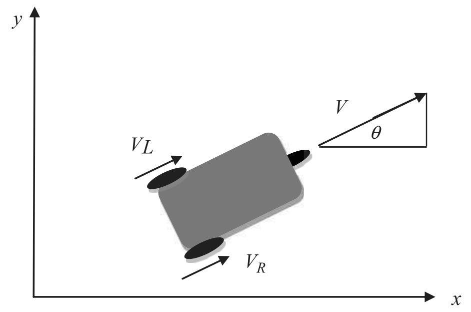

Fig. 3. Diagram of a wall-following robot. x and y are horizontal coordinates. V_L is the velocity of the left wheel, V_R is the velocity of the right wheel, V is the velocity of the robot's center of mass, and θ is the heading angle error of the robot. The controller adjusts V_L and V_R to maintain a fixed distance from the x-axis, and to maintain $\theta=0$.

Author's personal copy

on the basis of initial simulation tests. The cost function that is minimized by BBO and DBBO is given as follows:

$$f(x) = c\int |e(t)|dt + T_r. \tag{32}$$

In Eq. (32), x represents one BBO or DBBO individual, and includes K_p and K_d values as its independent variables. $e(t)$ is the tracking error in millimeters as a function of time and is integrated starting when the rise time is reached, and ending at the end of the test, which is 20 s. T_r is the rise time in seconds, which is the time that it takes the robot to reach the commanded distance from the wall with an error of 5% or less. In Eq. (32), c is a parameter that weights the relative importance of tracking error compared to rise time. Based on typical control results, we used $c=5$ to obtain a good balance between tracking error and rise time performance.

For practical control problems we need a way to limit control effort, but we do not include control effort in Eq. (32). This is because the search domains of K_p and K_d are specified as part of the BBO algorithms ([0, 2] and [0, 10], respectively). BBO never finds candidate solutions outside of those domains because K_p and K_d are obtained by migration and mutation within the search domain. Control chattering is avoided by a combination of the natural PD dynamics of Eq. (31) and the natural damping of the mechanical robot dynamics. However, for other control optimization problems, we may need to include a penalty in the cost function for control effort or chattering.

The results of the BBO and DBBO simulations are shown in Table 6. As with the benchmark problems of Section 4, we ran 100 Monte Carlo simulations. We thus obtain 100 minimum costs and 100 optimal sets of PD parameters by each algorithm. Table 6 shows the minimum (best) of those 100 costs, the maximum (worst) of the 100 costs, the average of the 100 costs, and the standard deviation of the 100 costs.

Interestingly, the DBBO versions all found better PD solutions than BBO (the "minimum cost" row of Table 6). However, on average, BBO performed the best (the "average cost" row of Table 6). In addition, BBO was more consistent and more robust than DBBO, as seen from the "maximum cost" and "standard deviation" rows of Table 6. The three DBBO versions all performed similarly to each other.

5.4. Robot experiment results

We use four robots and a mutation rate of 15% for our experimental testing of DBBO. The small population size is comparable to many real-world applications in which a limited number of BBO individuals participate (four robots in this application). The mutation rate is relatively high compared to most EAs because of the low population size and the low generation limit. For our experiments, we use a different version of Eq. (32) for the cost function. For our simulations, the integral of the error in Eq. (32) begins when the rise time is reached; but for our hardware tests, the integral of the error starts at the initial time. This gives more emphasis to rise time, and also results in cost values that are much higher than those obtained by the simulations.

We set the robots' initial K_p and K_d values randomly as follows.

Robot 1: $K_p=0.93$, $K_d=4.26$.
Robot 2: $K_p=0.07$, $K_d=6.36$.
Robot 3: $K_p=0.18$, $K_d=2.45$.
Robot 4: $K_p=0.12$, $K_d=2.21$.

At each generation, two randomly-selected robots communicate with each other via wireless radio. The PD parameters gradually change from one generation to the next as the robots share information with each other using the DBBO algorithm. We run DBBO for eight generations, obtaining a cost value from Eq. (32) for each robot at each generation. The final PD parameters for the robots after eight generations are as follows.

Robot 1: $K_p=0.82$, $K_d=9.03$.
Robot 2: $K_p=0.07$, $K_d=3.41$ ← Best Performance.
Robot 3: $K_p=0.67$, $K_d=4.32$.
Robot 4: $K_p=0.02$, $K_d=2.03$.

Fig. 4 displays the experimental DBBO results. The cost function value of the best robot, and the average cost of all four robots, decrease throughout the DBBO process. The decreasing cost values show that the robots are learning to have faster rise times and smaller fluctuations in their paths as they attempt to maintain a given distance from the wall.

Recall that we built the robots by hand (see Fig. 2). Although we tried to build them identically, there are differences between them due to variations in hardware and manufacturing tolerances. This means that the optimal PD parameters vary from one robot to the next. However, the robots are similar enough to each other that they can still learn from each other. This is similar to what we see in human learning. Individual humans are much different from each other, so optimal success strategies vary from one individual to the next. However, we are similar enough to each other that we can still learn from each other.

Our experimental results are preliminary due to hardware challenges. These results are not intended to be exhaustive, but are intended to demonstrate that DBBO is a viable option for distributed evolutionary optimization in practical real-world systems. For further research we hope to obtain more extensive experimental results. This will include the use of many more evolving robots, and more generations. It will also include more experiments so that we can study the repeatability of DBBO and compare it with other distributed EAs. It will include the study of

Table 6
Cost values from 100 Monte Carlo simulations of robot control optimization using BBO and DBBO. The best value of each metric is shown in bold font.

	BBO	DBBO/2	DBBO/4	DBBO/6
Minimum cost	7.48	**7.23**	7.30	7.16
Average cost	**7.68**	7.78	7.77	7.76
Maximum cost	**7.99**	8.12	8.07	8.10
Standard deviation	**0.12**	0.17	0.15	0.19

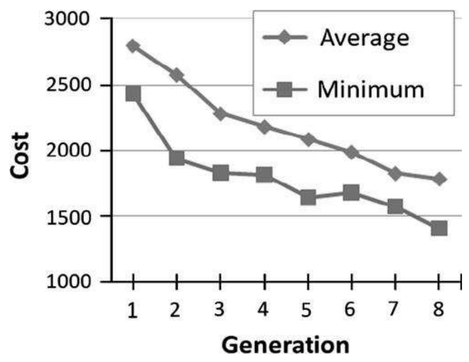

Fig. 4. Minimum cost and average cost of DBBO/2 as it optimizes the control performance of four experimental mobile robots. Both the minimum cost and the average cost of the four robots consistently decreases from one generation to the next.

Author's personal copy

various peer group sizes, including peer group sizes that change from one generation to the next.

Finally, we note that we did not implement any collision avoidance strategies in our algorithms. We started our robot tracking experiments with the robots sufficiently separated from each other so that there was no danger of collision. If we implement the ideas in this paper in a real-world setting (for example, in an office setting or on a factory floor), we would need to program robot collision avoidance in conjunction with the robots' control algorithms.

6. Conclusion

We have developed a distributed version of biogeography-based optimization that is based on cooperative intelligence. This new optimization algorithm, which we call DBBO, does not require a central computer. DBBO rather involves each individual running the evolutionary algorithm by itself, based on its communication with a small subset of the population. We have developed a Markov model for DBBO and have verified it by testing it against simulations. We have tested DBBO against a common set of benchmark problems, and have used it to optimize the parameters of a group of experimental robots.

Our results show that DBBO performs at about the same level as BBO, and that both algorithms are competitive with other EAs. The advantage of DBBO is that it does not require a central computer to run the algorithm, and so it is suitable for optimization problems in which communication between individuals is not possible due to geographic or temporal limitations. We have not compared DBBO in this paper with other distributed EAs, but such comparisons are important for future work.

This paper opens up many possibilities for future work. We could conduct additional mathematical modeling to obtain additional insights into DBBO behavior. Some additional modeling and analysis tools that could be applied to DBBO include dynamic system modeling [38], and statistical mechanics and Walsh transform approximations [29]. The Markov models in this paper are specific tools that can be theoretically applied to any population size, but they are limited to the determination of population distributions in the limit as the generation count approaches infinity. Dynamic system models provide population distributions at each generation in the limit as the population size approaches infinity. A dynamic system model of BBO is derived in [38] and could be extended in future work to DBBO. Although Markov and dynamic system models provide population distributions, they do not provide the variance of the population distribution. EA population variances can be obtained with statistical mechanics models. A statistical mechanics model of BBO is derived in [18] and could be extended in future work to DBBO. Although we used Markov models in this paper to find the population distribution of DBBO, we did not study the convergence behavior of DBBO. However, Markov models can be used to study convergence behavior. Such a study is conducted in [17] for BBO and could be extended in future work to DBBO.

More work could be done on finding appropriate tuning parameters for DBBO. These tuning parameters could be obtained using either simulation results or the Markov model. For example, suppose that we are interested in solving some specific real-world problem that has a computationally expensive fitness function. We could use the Markov model on a small-scale version of the problem to find the best values for mutation rate, migration curve parameters, and so on, rather than relying on simulation results.

We suggest further research to provide problem-dependent guidance for when to use DBBO instead of BBO, and vice versa. We also note that the distributed learning approach proposed here could also, with appropriate modifications, be implemented in many other evolutionary algorithms (genetic algorithms, particle swarm optimization, differential evolution, and so on). The Markov model developed here could be used to obtain numerical comparisons between BBO, DBBO with various parameter settings, and other EAs. Finally, we mention that the development of Markov models for BBO and DBBO in continuous state-spaces is an important direction for future research.

Acknowledgments

This material is based on work supported by the Cleveland State University Provost's Office and by the National Science Foundation under Grant No. 0826124.

References

[1] C. Ahn, Advances in Evolutionary Algorithms: Theory, Design and Practice, Springer, 2006.
[2] K. Astrom, T. Hagglund, PID controllers: theory, design, and tuning, International Society for Measurement and Control (1995).
[3] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 1769–1776, 2005a.
[4] A. Auger, N. Hansen, Performance evaluation of an advanced local search evolutionary algorithm, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 1777–1784, 2005b.
[5] P. Ballester, J. Stephenson, J. Carter, K. Gallagher, Real-parameter optimization performance study on the CEC-2005 Benchmark with SPC-PNX, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 498–505, 2005.
[6] E. Cantu-Paz, Efficient and Accurate Parallel Genetic Algorithms, Kluwer Academic Publishers, 2000.
[7] C. Churavy, M. Baker, S. Mahta, I. Pradhan, N. Scheidegger, S. Shanfelt, R. Rarick, D. Simon, Effective implementation of a mapping swarm of robots, IEEE Potentials 27 (4) (2008) 28–33, July.
[8] G. Fischer, Distributed intelligence: extending the power of the unaided, individual human mind, in: Proceedings of the Advanced Visual Interfaces (AVI) Conference (A. Celentano, Ed.) ACM Press, pp. 7–14, 2006.
[9] C. García-Martínez, M. Lozano, Hybrid real-coded genetic algorithms with female and male differentiation, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 896–903, 2005.
[10] J. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.
[11] J. Liang, P. Suganthan, Dynamic multi-swarm particle swarm optimizer with local search, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 522–528, 2005.
[12] M. Lomolino, J. Brown, The reticulating phylogeny of island biogeography theory, The Quarterly Review of Biology 84 (4) (2009) 357–390.
[13] P. Lozovyy, G. Thomas, D. Simon, Biogeography-based optimization for robot controller tuning, in: B. Igelnik (Ed.), Computational Modeling and Simulation of Intellect, IGI Global, 2011, pp. 162–181.
[14] H. Ma, D. Simon, Analysis of migration models of biogeography-based optimization using Markov theory, Engineering Applications of Artificial Intelligence 24 (6) (2011) 1052–1060.
[15] H. Ma, M. Fei, Z. Ding, J. Jin, Biogeography-based optimization with ensemble of migration models for global numerical optimization, IEEE World Congress on Computational Intelligence, Brisbane, Australia, 2981–2988, June 2012.
[16] H. Ma, D. Simon, Z. Xie, Variations of biogeography-based optimization and Markov analysis Information Sciences 220 (2013) 492–506.
[17] H. Ma, D. Simon, On the convergence of biogeography-based optimization, submitted for publication.
[18] H. Ma, D. Simon, M. Fei, On the statistical mechanics approximation of biogeography-based optimization, submitted for publication.
[19] R. MacArthur, E. Wilson, The Theory of Island Biogeography, Princeton University Press, 1967.
[20] D. Molina, F. Herrera, M. Lozano, Adaptive local search parameters for real-coded memetic algorithms, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 888–895, 2005.

Author's personal copy

[21] E. Munroe, The Geographical Distribution of Butterflies in the West Indies, Ph.D. Dissertation, Cornell University, Ithaca, New York, 1948.
[22] M. Ovreiu, D. Simon, Biogeography-based optimization of neuro-fuzzy system parameters for diagnosis of cardiac disease, Genetic and Evolutionary Computation Conference, Portland, Oregon, 1235–1242, 2010.
[23] V. Panchal, P. Singh, N. Kaur, H. Kundra, Biogeography based satellite image classification, International Journal of Computer Science and Information Security 6 (2) (2009) 269–274.
[24] V. Panchal, E. Kundra, A. Kaur, Biogeography based groundwater exploration, International Journal of Computer Applications 1 (8) (2010) 87–91.
[25] P. Posik,Real parameter optimisation using mutation step co-evolution, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 872–879, 2005.
[26] A. Qin, P. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 1785–1791, 2005.
[27] D. Quammen, The Song of the Dodo: Island Biogeography in an Age of Extinction, Simon & Schuster, 1997.
[28] 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, Physiological Measurement 32 (7) (2011) 767–796.
[29] C. Reeves, J. Rowe, Genetic Algorithms—Principles and Perspectives, Springer, 2002.
[30] J. Rönkkönen, S. Kukkonen, K. Price, Real-parameter optimization with differential evolution, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 506–513, 2005.
[31] P. Roy, S. Ghoshal, S. Thakur, Biogeography-based optimization for economic load dispatch problems, Electric Power Components and Systems 38 (2) (2010) 166–181.
[32] G. Rudolph, Convergence Properties of Evolutionary Algorithms, Verlag Dr. Kovac, 1997.
[33] V. Savsani, R. Rao, D. Vakharia, Discrete optimisation of a gear train using biogeography based optimisation technique, International Journal of Design Engineering 2 (2) (2009) 205–223.
[34] C. Scheidegger, A. Shah, D. Simon, Distributed learning with biogeography-based optimization, in: 24th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, Syracuse, New York, pp. 203–215, 2011.
[35] D. Simon, Biogeography-based optimization, IEEE Transactions on Evolutionary Computation 12 (6) (2008) 702–713.
[36] D. Simon, A probabilistic analysis of a simplified biogeography-based optimization algorithm, Evolutionary Computation 19 (2) (2011) 167–188.
[37] D. Simon, M. Ergezer, D. Du, R. Rarick, Markov models for biogeography-based optimization, IEEE Transactions on Systems, Man, and Cybernetics Part B Cybernetics 41 (1) (2011) 299–306.
[38] D. Simon, A dynamic system model of biogeography-based optimization, Applied Soft Computing 11 (8) (2011) 5652–5661.
[39] D. Simon, R. Rarick, M. Ergezer, D. Du, Analytical and numerical comparisons of biogeography-based optimization and genetic algorithms, Information Sciences 181 (7) (2011) 1224–1248.
[40] U. Singh, H. Singla, T. Kamal, Design of Yagi-Uda antenna using biogeography based optimization, IEEE Transactions on Antennas and Propagation 58 (10) (2010) 3375–3379.
[41] A. Sinha, S. Tiwari, K. Deb, A population-based, steady-state procedure for real-parameter optimization, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 514–521, 2005.
[42] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization, Technical Report, Nanyang Technological University, Singapore, and KanGAL Report number 2005005, May 2005.
[43] K. Van Dam, Z. Verwater-Lukszo, J. Ottjes, G. Lodewijks, Distributed intelligence in autonomous multi-vehicle systems, International Journal of Critical Infrastructures 2 (2–3) (2006) 261–272.
[44] B. Yuan, M. Gallagher, Experimental results for the special session on real-parameter optimization at CEC 2005: a simple, continuous EDA, IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, 1792–1799, 2005.