International Journal of Innovative Computing, Information and Control
Volume **9**, Number **6**, June **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 2485–2503

EVOLUTIONARY OPTIMIZATION IN DYNAMIC ENVIRONMENTS: BRINGING THE STRENGTHS OF DYNAMIC BAYESIAN NETWORKS INTO BAYESIAN OPTIMIZATION ALGORITHM

Marjan Kaedi[1], Nasser Ghasem-Aghaee[2] and Chang Wook Ahn[3,*]

[1]Department of Information Technology Engineering
[2]Department of Computer Engineering
University of Isfahan
Hezar-Jerib Street, Isfahan 81746-73441, Iran
{ kaedi; aghaee }@eng.ui.ac.ir

[3]Department of Computer Engineering
Sungkyunkwan University
2066 Seobu-ro, Suwon 440-746, Korea
*Corresponding author: cwan@skku.edu

Received February 2012; revised June 2012

Abstract. *In this paper, a new evolutionary algorithm termed DBN-MBOA (Memory-based BOA with Dynamic Bayesian Networks) is proposed for the dynamic optimization. In DBN-MBOA, the knowledge obtained from previously solved problems is encoded in some structures called network translators. The network translators defined on non-stationary Dynamic Bayesian Networks (nsDBNs) describe the correlation between conditional dependencies of candidate solution variables before and after environmental changes. The network translators constructed for the changes are stored in memory. When any change occurs in the environment, a relevant network translator is retrieved from the memory and is used for modifying the dependencies of the current Bayesian network. In the retrieve stage, unlike existing memory-based methods, the relevant network translator is selected based on the characteristic of the change itself, not that of the new environmental state. Experimental results show that DBN-MBOA achieves better performance in random environments as well as cyclic environments.*
Keywords: Bayesian optimization algorithm, Dynamic Bayesian network, Dynamic optimization, Environmental change, Memory-based method, Network translator

1. **Introduction.** In many real-world optimization problems, the environment might change in the course of optimization process. Changes in the environment can take various forms such as changes in the parameters, the objective functions or the problem constraints [1, 2]. When the problem's environment varies, the optimum tends to change as well. *Dynamic optimization problems* take into account any of these uncertain events in the optimization process [3]. Examples include scheduling problems where new jobs arrive over time [4], vehicle routing problems where new requests arrive over time [3], portfolio decisions optimization problem where the stock market changes and the decisions should be updated over time [1], and flight path optimization problem where the flight paths should be adapted to special events [1]. The objective of dynamic optimization is to not only find the optimum solution but also track it over time [6]. If the problem after the change is similar to the previous ones, the knowledge acquired while searching for the solutions of the previous problems could be applied to discover the solution of the new problem [3]; thus, searching for a new solution based on the previous solutions can reduce the searching time [7].

Evolutionary algorithms (EAs) have been inspired by the natural evolution [5]. Generally, they can be used as a proper tool for solving the dynamic optimization problems [6]. In addition, estimation of distribution algorithms (EDAs) [8, 9, 10] have been developed for handling the "building-blocks destruction" problem. Instead of using the standard (genetic) operators, EDAs construct probabilistic models that describe high-quality solutions and then create new solutions by sampling the models [11][1]. In recent years, a variety of methods have been proposed for solving the dynamic optimization problems by using EAs. These methods are classified as follows [12]: a) reaction of the algorithm after the change, b) preserving the diversity during the run, c) the memory-based methods, and d) the multi-population methods. Especially in the memory-based dynamic optimization methods, evolutionary mechanisms are supported by *memory* that stores the useful information about the problems in the previous environments. When any change occurs, the relevant information is retrieved and used for solving the new problem [3].

Meanwhile, EDAs have more potential to be used as a memory-based dynamic optimization method [13] because their probabilistic models contain the abstract form of information about the previous environments. These models can be efficiently stored in memory and then this memory can be managed in a simple and swift manner [13]. In some recent studies, EDAs have been used in the memory-based dynamic optimization. The first work has been presented in [14]. In that work, univariate marginal distribution algorithm (UMDA[2]) [11] is applied to dynamic optimization problems. After the best samples created by the probability vector of UMDA are stored in memory, the memory is re-evaluated at every iteration. If the fitness of any memory sample varies, the environment is detected to be changed. The memory is then merged with the current population to form a new population. Another approach for applying UMDA has been presented in [6]. The probabilistic vectors of UMDA are stored in memory together with the best solutions. Thus, each memory element consists of two parts: the best solution and the probability vector. The best solutions stored in the memory are re-evaluated at every iteration. If any environmental change is detected, the probability vector of the best memory element (i.e., the memory element with the highest fitness) is retrieved. Then, new candidate solutions are created from that probability vector and merged with the current population. Another EDA, Population Based Incremental Learning (PBIL[3]) [15], has been used in [16]. As in the method presented in [6], the best solutions and the probability vectors are stored together in memory. When a change is detected, the probability vector of the best memory element is retrieved and replaces the current probability vector if its related best solution has higher fitness than the best individual created by the current probability vector.

In terms of EDA-based dynamic optimization, a general memory-based framework termed EI-MEDA (EDA with Environment Identification based Memory scheme) has been proposed in [13]. In EI-MEDA, the probabilistic models are stored in memory along with the best solutions. When the environmental change is detected, the memory element with the highest fitness is retrieved. If some of the memory elements have the same fitness, another method is used to choose and retrieve the best element; the probability models of these memory elements are sampled, the average fitness of the samples is calculated, and the memory element with the highest average fitness is retrieved. The probabilistic model of that memory element is used to guide the EDA in the new environment. EI-MEDA

[1]EDAs differ in the probabilistic models used for modeling high-quality solutions.

[2]In UMDA, all the variables are assumed to be independent of each other and a probability vector is used as the probabilistic model.

[3]Similar to UMDA, PBIL uses a probability vector. However, the learning process of PBIL is conducted by an incremental process.

has been tested within the framework of Bayesian optimization algorithm (BOA), termed EI-MBOA. Performance of EI-MBOA has been compared with that of RBOA (i.e., BOA is restarted whenever any change occurs). Empirical results have shown that EI-MBOA is very effective in all cyclic environments, but this method is less effective in random environments whose states are not repetitive.

Generally, in all the methods presented so far for applying EDAs to the memory-based dynamic optimization, their probabilistic models are stored in memory and then retrieved in the future when a similar environmental state is visited. All these methods have high efficiency when the environmental states are repetitive, that is, the environments similar to the new ones have been treated in the past. However, they do not work effectively when they are applied to random dynamic environments. Thus, this paper makes an attempt to improve the memory-based BOA even in random dynamic environments. In this sense, we present a method called "Memory-based BOA with Dynamic Bayesian Networks" (DBN-MBOA) in which, unlike existing memory-based EDAs, the focus is on the characteristic of the environmental changes rather than that of the environmental states. In addition, the probabilistic models are not stored in memory. Instead, the knowledge obtained from the previous solved problems is described in some structural forms which are called *network translators*. We define the network translators based on the concept of non-stationary Dynamic Bayesian Networks (nsDBNs). The network translators represent the correlation between the conditional dependencies of problem variables before and after the change. During the BOA run, when a change occurs in the environment, a proper network translator is retrieved from the memory and then used to adjust the trajectory of BOA toward the proper direction. The experiments show that DBN-MBOA improves the efficiency of the memory-based BOA in random dynamic environments as well. Hence, DBN-MBOA can be applied to solve real-world dynamic optimization problems in which the changes in the problem's parameters or the objective function take place randomly (i.e., they have no repetitive pattern).

The rest of this paper is organized as follows. Section 2 briefly introduces the BOA algorithm. Also, three types of dynamic environments and existing memory-based methods are described. Section 3 presents the proposed DBN-MBOA method. Experimental results are shown in Section 4, and finally, Section 5 concludes this paper.

2. **Background.**

2.1. **Bayesian optimization algorithm.**

BOA evolves a population of candidate solutions by constructing Bayesian networks and sampling the constructed networks [17, 18]. A Bayesian network that is a directed acyclic graph[4] (DAG) represents probabilistic relationships among a set of random variables. Conditional probability tables are used for representing the conditional independencies of the variables [19]. BOA constructs the initial population randomly with a uniform distribution for all possible solutions. Then, the population is updated through several iterations. Each iteration consists of four stages [17, 18]. First, using a selection method (like as what is used in EAs), high-quality solutions are selected from the population. In the second stage, a Bayesian network compatible with these high-quality solutions is constructed. In the third stage, new candidate solutions are created by sampling from the Bayesian network. In the final stage, the new solutions are incorporated with the pre-existing solutions and replace all or some of them. Until reaching the termination condition, these four stages are repeated. The termination condition can be either of the convergence to copies of the same candidate solution, reaching a sufficiently good solution, or reaching the specific number of iterations. Due to

[4]In this graph, each node is related to a variable and the edges correspond to conditional dependencies.

its ability to consider multivariate interactions between variables, BOA is known as one of the powerful EDAs in recent years.

2.2. **Memory-based methods in dynamic environments.** Three types of dynamic environments are studied in dynamic optimization research: *random*, *cyclic* and *cyclic with noise*. An environment is said to be cyclic if it changes periodically in the search space and visits several fixed states in a certain order repeatedly [2]. Thus, as time passes, the environment will return to some old environments precisely [2]. In the cyclic (dynamic) environments with noise, also called partially cyclic environments, the environment will visit some states similar to the old ones in a certain order repeatedly. That is, the new states are noisy versions of the old states. In the real world, there are many dynamic optimization problems that are subject to the cyclic environments or the cyclic environments with noise. Some examples include the annual climatic changes [2], the urban traffic condition daily changes [2], and the topology dynamics in mobile ad-hoc networks [20]. In the random environments, also called non-cyclic environments, the environment moves randomly in the search space [2]. Thus, there is no guarantee that the environment will return to a previous one after certain changes occur [21].

In the memory-based EDAs presented so far, the probabilistic models related to several environmental states are stored in memory and used for similar environmental states that are visited in the future. Hence, these methods work very effectively on the cyclic environments and the cyclic environments with noise. However, they have shown low efficiency for the random environments since the new states are not repetitive [13]. In this paper, we present a new method to improve the efficiency of memory-based BOA in the random environments. This method would be helpful for solving real-world dynamic optimization problems whose changes occur randomly and have no repetitive pattern. An example of such optimization problems is the flight scheduling problem in which the departure or arrival time of some airplanes may be delayed because of unpredicted and random technical defects, etc. This delay may disrupt the timing of other flights and thus, the flights should be rescheduled again after such an event. Another example is the dynamic job shop scheduling problem. Since some machinery may stop working randomly over time, the job shop should be rescheduled. Other examples include the packet routing in computer networks in which some routers may be turned off over time randomly and the traffic signal timing in which the traffic of some streets may change randomly due to car accidents.

3. **Proposed Approach: DBN-MBOA.** This section presents a new evolutionary dynamic optimization algorithm, termed DBN-MBOA, which belongs to the memory-based dynamic optimization approach. The former memory-based methods have focused on the characteristic of the environmental states. In those methods, when the environment changes, the new environmental state is described and then, the most relevant memory element is retrieved from the memory and is used for the new environment. As mentioned in Section 2.2, this approach is efficient for the cyclic environments (with noise).

Unlike the former methods, in order to improve the memory-based approach for the random dynamic environments, DBN-MBOA focuses on the characteristic of the change itself rather than the new environmental states. That is, a previous change whose characteristic is similar to that of the current change is investigated. Another difference between the DBN-MBOA method and the former memory-based methods is related to the form of the knowledge stored in memory. In the existing approaches, the probabilistic models constructed in the former environmental states are stored in memory as the *a priori* knowledge. However, DBN-MBOA describes the knowledge of former environment changes in

the form of the changes which occur in the conditional dependencies of solution variables in the probabilistic models. For this purpose, we define *network translators* based on the concept of nsDBNs. The network translators describe the correlation between conditional dependencies of solution variables in two consecutive environments. The successive stages of DBN-MBOA are shown in Figure 1. When any change is detected in the environment (Figure 1, STAGE 2), the environmental change is described using the criterion presented in Section 3.6 (Figure 1, STAGE 3). Next, all the memory elements available in the memory are evaluated and the network translator related to the most compatible memory element with the current environmental change is selected and retrieved from the memory (Figure 1, STAGE 4). Using this network translator, the evolution trajectory of BOA is adjusted towards the proper direction (Figure 1, STAGE 5). Then, the search continues in the new trajectory until the solution for the new environment is obtained or a new change occurs in that environment. Moreover, in some cases, the knowledge of the new environmental change would be appended to the memory (Figure 1, STAGES 6 and 7). More details for each stage are described below.

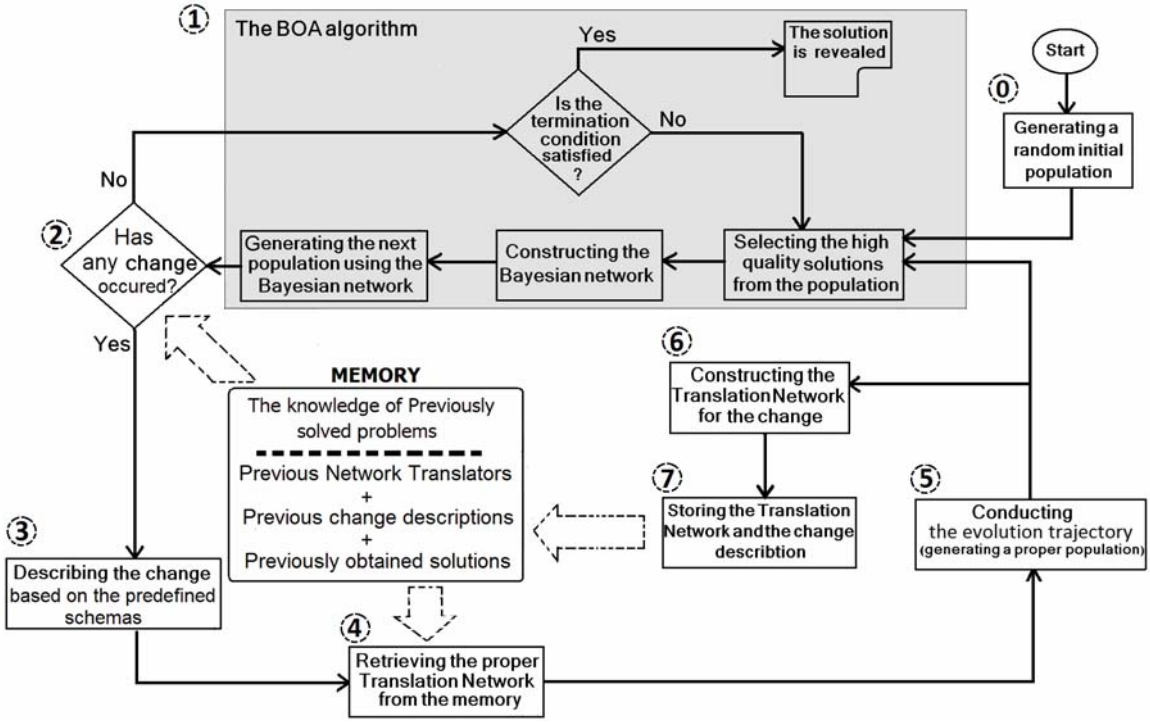


FIGURE 1. The successive stages in the proposed DBN-MBOA method

3.1. **Conduction of the evolution trajectory.** Like all the EAs, the BOA algorithm solves an optimization problem by evolving a population of solutions over time iteratively. The population which gradually evolves towards the target solution constitutes a sequence. In this paper, this sequence is called *evolution trajectory*. An evolution trajectory with the length m can be represented as $(P_0, P_1, \cdots, P_i, P_{i+1}, \cdots, P_m)$, where P_k $(0 \leq k \leq m)$ is a set of candidate solutions generated in the iteration k. While solving a problem using BOA, its evolution trajectory is formed gradually until the termination condition is satisfied. When a change occurs during the BOA run (i.e., a new problem is raised), this new problem is somewhat different from the former one for which BOA was trying to discover a solution. In this case, tracking the current evolution trajectory would not

be advantageous; thus, the trajectory should be conducted towards a proper direction for the solutions of the new problem[5].

We denote the problem before the change as "Problem_1" and the problem after the change as "Problem_2". Suppose that while BOA is solving Problem_1, the environment changes at time t_1 (in which the algorithm is in the generation k of its evolution trajectory) and Problem_2 arises at time $t_1 + 1$. If we knew that at time t_1 the environment has changed, then Problem_2 would have been solved with Problem_1 in parallel and the algorithm would be in the generation k of its evolution trajectories for both problems; hence, under the ideal condition, once a change occurred in the environment, BOA would be able to continue tracking the evolution trajectory of Problem_2 from the generation k. When the environment has gone through a sudden change at time t_1, say Problem_2, in order to make use of all attempts to solve Problem_1, we try to convert all the solutions of the current generation k into the candidate solutions of the corresponding generation in the evolution trajectory of Problem_2. Then, the BOA algorithm that has been conducted towards the evolution trajectory of Problem_2, should continue its run starting from this population of the candidate solutions up to the point where the solution to Problem_2 is achieved or another new environmental change happens. Figure 2 illustrates this conduction process.

In this paper, we propose a tool to adjust the evolution trajectory and translate the current population of BOA into its corresponding population for solving a newly arisen problem when a change occurs in the environment. The proposed tool actually redirects the evolution trajectory by modifying the Bayesian network to achieve another Bayesian network that describes the corresponding candidate solutions in the evolution trajectory of Problem_2. In this sense, this tool is called 'network translators'.

3.2. **Network translators.** Unlike the existing memory-based methods, DBN-MBOA describes the knowledge of former environmental changes in the form of the changes which occur in the conditional dependencies of solution variables for the probabilistic models. For this purpose, a new structure proposed herein, termed the *network translators*, is applied to conduct the evolution trajectory. A network translator demonstrates how the conditional dependencies vary when the environment changes. It describes the correlation between the conditional dependencies of solution variables of Problem_1 and those of Problem_2[6]. Each network translator is a set of modifications that should be applied to the Bayesian network describing the candidate solutions of Problem_1 in order to obtain another Bayesian network describing the candidate solutions of Problem_2.

Assume that the BOA algorithm is in the generation k of its evolution trajectory in solving Problem_1 (i.e., the traversed evolution trajectory is $(\cdots, P_i, P_{i+1}, \cdots, P_k)$). Also, assume that the environment changes at this moment and then Problem_2 arises. The current Bayesian network encoding the conditional dependencies is modified by using a proper network translator. The modified Bayesian network is then sampled to make a population of candidate solutions for Problem_2. We denote this newly generated population as P'_k, which is an element of the evolution trajectory of Problem_2. By evolving this population and tracking the evolution trajectory $(P'_k, P'_{k+1}, \cdots)$, BOA is able to evolve towards the optimal solution of Problem_2. In this way, the evolution trajectory of Problem_1 can be directed towards the evolution trajectory of Problem_2. In order to achieve this purpose, we define the network translators based on the nsDBNs concept. In Section 3.2.1, the nsDBNs are described in detail. The construction of the network translators is explained in Section 3.2.2.

[5]The term 'conduct' can be interchangeable with 'adjust' or 'redirect' in this paper.

[6]The notations of Problem_1 and Problem_2 are same as those in the previous section.

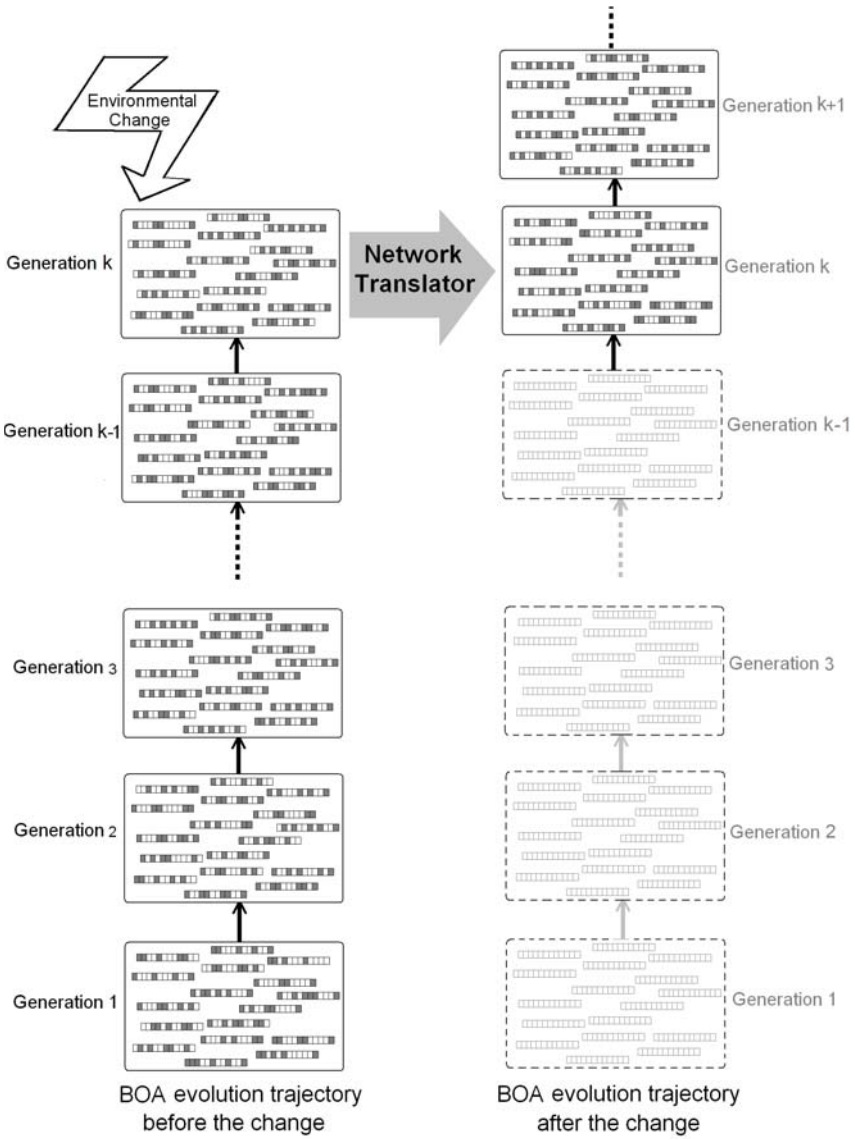


FIGURE 2. Conduction of the evolution trajectory when a change occurs in the environment. The dotted boxes in the right-hand side indicate the generations which are not actually performed by BOA.

3.2.1. *The non-stationary dynamic Bayesian networks.* Dynamic Bayesian networks (DBNs) are the extension of the Bayesian networks that the time dimension is added to the current uncertainty [22, 23]. To introduce the time dimension into the Bayesian networks, a variety of methods have been proposed [22]. The nsDBNs are a specific kind of DBNs, which represents a new framework for investigating problems where the structure of a Bayesian network evolves over time. Some examples of nsDBNs are introduced in [24, 25, 26, 27]. Among these works, we choose the method proposed by Robinson and Hartemink [25] since it is capable of modeling the discrete data and reasoning the pattern through which the dependencies change over time. In the followings, we briefly explain how the nsDBN is defined and learned.

Let us assume that the state of n random variables is observed at N discrete times. This multivariate time-series data is denoted by D. We further assume that D is generated according to an unknown non-stationary process. This process is non-stationary in the sense that the network of conditional dependencies is changing over time [25]. The initial

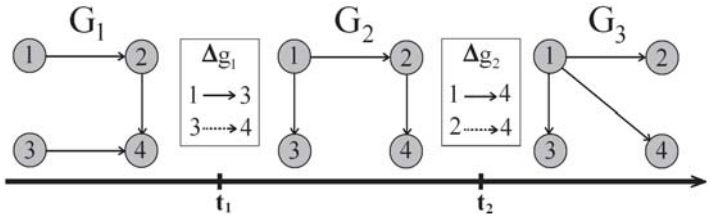


FIGURE 3. An example of nsDBN. In each Δg, the dashed/solid arrows represent the edges that are removed/added at the next transition time [25].

network of the conditional dependencies is denoted by G_1 and thus subsequent networks are naturally represented by $G_2, G_3, \cdots, G_m$. In this model, Δg_i is defined as the set of edges that change (either added or deleted) between G_i and G_{i+1}. The transition time t_i is defined as the time at which G_i is replaced by G_{i+1} in the data-generation process. The period between two consecutive transition times is called an *epoch*. Thus, G_1 prevails during the first epoch, G_2 prevails during the second epoch, and so forth. We will refer to the entire series of the prevailing networks as the structure of nsDBN. An example of nsDBN is shown in Figure 3.

In this work, we want to apply this nsDBN structure to describe the correlation of the problem's solutions in the consecutive times. The solutions represent the time-series data D. We consider the networks of conditional dependencies in two epochs (i.e., $m = 2$). Assume that the environment is changed at time t_1; thus, the structure of the nsDBN evolves from the network G_1 to the network G_2 at the transition time t_1. The task is to find the networks G_1 and G_2 that have the highest probability given from the observed time-series data. That is, we want to discover the networks that maximize Equation (1).

$$P(G_1, G_2 \mid D, t_1) = \frac{P(D \mid G_1, G_2, t_1) P(G_1, G_2 \mid t_1)}{P(D \mid t_1)} \propto P(D \mid G_1, G_2, t_1) P(G_1, G_2 \mid t_1) \quad (1)$$

Since each network differs from the previous one by a set of edge changes, the problem can be rephrased as the initial network and a set of edge changes by Equation (2) [25].

$$P(G_1, G_2 \mid T) = P(G_1, \Delta g_1 \mid T) \quad (2)$$

The prior networks can be further split into components describing the initial network and the subsequent edge changes, thereby leading to the final form of the posterior as given in Equation (3)[7] [25].

$$P(G_1, \Delta g_1 \mid D, T) \propto P(D \mid G_1, \Delta g_1, T) P(G_1) P(\Delta g_1 \mid T) \quad (3)$$

Thus, the above equation should be maximized to identify the most likely initial network G_1 and a set of edge changes, Δg_1. In order to evaluate the marginal likelihood, a non-stationary version of the BDe (Bayesian-Dirichlet equivalent) score is employed [25]. Due to the computational inefficiency in maximizing Equation (3), a greedy search algorithm is used, by which the elementary graph operations (i.e., edge addition, edge removal, and edge reversal) are applied to create and modify the initial network. In other words, some operations that increase the value of $P(G_1, \Delta g_1 | D, T))$ are selected and applied to G_1 and Δg_1. This greedy algorithm continues until no further improvement is achieved.

[7]In this equation, a uniform prior probability is assumed on $P(G_1)$

3.2.2. *The construction of network translators.* Creating the network translators and storing them in memory are accomplished when the environmental changes occur in the course of solving a problem (Figure 1, STAGES 6 and 7). As mentioned earlier, during the BOA run, when any change occurs in the environment, the algorithm stops running in the current evolution trajectory; the trajectory is then redirected to a new direction and the searching process continues in the new trajectory. Together with the conduction of evolution trajectory, a new network translator is created to describe the change of conditional dependencies in this environmental change. This new network translator is then stored in memory for later use of adjusting the evolution trajectory in the future similar changes.

To this end, we define the network translators on the concept of nsDBNs. To create a network translator, a set of candidate solutions to `Problem_1` as well as a set of candidate solutions to `Problem_2` are necessary. These solutions should be from a generation in almost half way through the BOA run, in which proper diversity exists in the candidate solutions. This generation is denoted by M. The set of candidate solutions in the generation M of the algorithm that runs for solving `Problem_1` and `Problem_2` are regarded as data at time t_1 and $t_1 + 1$, respectively. Thus, these solutions represent the time-series data D in the two consecutive times.

The objective is to identify the differences between the conditional dependencies of data at time t_1 and data at time $t_1 + 1$. Based on the nsDBN model reviewed in Section 3.2.1, these differences can be expressed as a set of edge changes (i.e., the form of Δg_1). Thus, we create a Δg_1 that best describes the time-series data by using a greedy search algorithm. The obtained Δg_1, a network translator, which represents the set of modifications is applied to the Bayesian network describing the candidate solutions of `Problem_1`, thereby obtaining another Bayesian network describing the candidate solutions of `Problem_2`. Then, the obtained network translator (i.e., the proper Δg_1) is stored in memory along with some other parts as a new memory element (Figure 1, STAGE 7). Each memory element includes four parts: the network translator, the best solution of the pre-change problem called BoSBC (the Best of Solutions Before the Change), the best solution of the after-change problem called BoSAC (the Best of Solutions After the Change), and DSF (Difference of the Schemas Fitness) vector[8] that describes the change. During solving a problem, whenever a certain change similar to the change from `Problem_1` to `Problem_2` occurs, the network translator, Δg_1, can be used to redirect the evolution trajectory towards a proper direction. How to measure the similarity of the two changes and how to conduct the evolution trajectory using a network translator are described in Sections 3.4 and 3.6, respectively.

It should be mentioned that in some cases, when a change happens, the BOA algorithm that solves `Problem_1` has not yet reached a sufficiently converged population. In this case, it is necessary that after solving `Problem_2` or in parallel, the algorithm resume its run to solve `Problem_1` from that point stopped until reaching a sufficient solution.

3.3. **Detection of changes in the environment.** To detect an environmental change during the BOA run, we employ the method used in EI-MBOA [13]. At each iteration of BOA, the best solutions of all memory elements should be re-evaluated in the current environment; in the case that the fitness of at least one of the best solutions stored in memory has changed, it implies that the environment has changed (Figure 1, STAGE 2). In order to detect the first change in the environment, meanwhile, a set of random chromosomes are generated and stored in memory at the initialization phase. When the fitness of at least one of these chromosomes varies, it denotes that the first change has occurred in the environment.

[8]The DSF vector will be introduced in Section 3.4.

3.4. **Selecting the proper network translator when a change occurs.** When the environment changes during the BOA run, the most appropriate element should be selected from the memory (Figure 1, STAGE 4). The network translator of that memory element is used for translating the evolution trajectory (Figure 1, STAGE 5). As mention earlier, in order to retrieve the appropriate memory element in DBN-MBOA, we focus on the characteristic of the change rather than the new environmental state itself. As such, any investigation on the previous environments similar to the current environment is not required in our method. Thus, an appropriate element for a certain previous change is able to effectively describe the current change as well, due to their similar characteristics. However, we need to define a criterion in order to measure the similarity between changes. (Figure 1, STAGE 3).

We present a new criterion called DSF that is inspired from the schema theorem[9]. A schema is a *similarity template* describing a set of strings with similarities at certain positions [29]. In the case of binary representation, a schema (or similarity template) is a string of symbols taken from the alphabets {0,1,*}. Each schema represents the set of bit strings containing the indicated 0s and 1s, with each '*' interpreted as a 'do not care'[10]. For example, the schema 0*10 represents the set of bit strings that includes exactly 0010 and 0110 [30]. It denotes that a population of bit strings can be described in terms of a set of schemas and a number of bit strings matching each of these schemas [30]. Therefore, the criterion for describing the environmental changes can be defined on the basis of the schema concept. At first, initial N schemas are generated randomly before running BOA and remain unchanged during several environmental changes. Then, when a change is detected in the environment, the following steps are carried out:

- **Step 1:** The last generation of BOA, which has been created before the change, is accounted for here. First, the candidate solutions of this generation matching each of the N schemas are selected and then, their average fitness values are computed. The obtained N fitness values are put into a vector called BCSF (Before Change Schemas Fitness).
- **Step 2:** Under the condition that the environment has changed, the fitness of candidate solutions would be changed in the new environment. For the current generation of BOA (i.e, at the moment of change), the candidate solutions of that generation matching each of the N schemas are selected and then, their average fitness values are determined. The obtained N fitness values are put into a vector called ACSF (After Change Schemas Fitness). Notice that, to reduce computation time, Step 1 and Step 2 can be performed on a randomly selected subset of the chromosomes rather than all the chromosomes (e.g., 30% of the whole chromosomes).
- **Step 3:** By subtracting the BCSF vector from the ACSF vector, we can obtain another vector termed DSF (Difference of the Schemas Fitness). This vector is used as a criterion for describing the change since it represents how much the average fitness of each schema varies according to the environmental chance.

We employ the DSF criterion to select the most appropriate memory element. Considering the Euclidean distances of all the DSF vectors, termed SBD (Schema Based Distance), the memory element with the least SBD is retrieved and its corresponding network translator is used to conduct the evolution trajectory. More detailed procedures for conducting this trajectory are presented in Section 3.6.

3.5. **The memory management.** In dynamic environments, several environmental changes occur over time. If the knowledge of every environmental change is stored in

[9]The theorem was presented by Holland [28] in order to explain the power of genetic algorithms.

[10]The asterisk '*' matches either 0 or 1.

memory, the number of memory elements increases gradually. As a result, the searching process presented in Section 3.4 to find the best memory element would be slowed down. Moreover, the memory capacity for storing the network translators is limited. Thus, inserting new elements into memory should be properly managed. Various policies for such management have been introduced in [31]. We choose one of these polices which best fits our purpose. A limited memory capacity is considered here. Whenever a new element is to be added to memory but there is no free space, an element with the sufficient similarity to the new element is retrieved. If its quality is lower than that of the new element, it is replaced by the new element. Otherwise, the new element is discarded. The SBD criterion mentioned in Section 3.4 is used to compare the new element with the elements stored in memory, thereby obtaining the most similar memory element. Furthermore, we need a quality criterion to decide whether or not the most similar element should be replaced by the new element. The quality of an element is measured by adding the fitness values of BoSBC and BoSAC of that element.

3.6. **Using the network translator for conducting the evolution trajectory.** In the course of solving a problem by BOA, if a new problem arises due to the environmental change, the most appropriate network translator would be retrieved by the procedures described in Section 3.4. Suppose that when a change happens, the algorithm is in the generation i. Using the retrieved network translator, the evolutionary trajectory of BOA is conducted (i.e., adjusted) towards the proper direction (Figure 1, STAGE 5). For conducting the evolution trajectory, the algorithm goes through the following steps:

- **Step 1:** The structure of the Bayesian network of the most recent generation of BOA is modified by the retrieved network translator. In this process, inserting the edges that cause cycles is prevented. The obtained Bayesian network estimates the dependencies of variables of candidate solutions in the generation i in terms of the new problem.
- **Step 2:** The obtained Bayesian network is sampled. The obtained samples are the expected candidate solutions that the algorithm would reach in its generation i, provided that it initiates its run to solve the new problem.
- **Step 3:** BOA continues its operation not from the pre-change generation but from the new population (i.e., the search continues in this new trajectory) until the solution in the new environment is obtained or another new change takes place in that environment (Figure 1, STAGE 1).

3.7. **An illustrative example.** We exemplify the process of appending a new element to memory and then retrieving a promising element in the future changes.

Example 3.1. Appending a new element to memory *Assume that an environmental change occurs during solving a problem by DBN-MBOA. After each environmental change, both the description on the change and the suitable network translator should be obtained and be stored in memory. These two procedures are explained through an example below.*

Consider that the number of problem variables is 7, the population size is 100, and three random schemas have been predefined to describe the environmental changes. These three schemas are presented in Table 1. In order to describe the change, a group of chromosomes that match each schema should be found in the population after and before the change and then, the average fitness of each group should be calculated. If these data are given like as Table 1, the DSF vector that describes the change is obtained by

$$DSF = [-1.68 \qquad 1.12 \qquad -1.05] \tag{4}$$

TABLE 1. The predefined schemas and their DSF

Schema	The average fitness before the change (out of 7)	The average fitness after the change (out of 7)	Difference of the Schemas average Fitness (DSF)
101****	4.55	2.87	–1.68
01**0**	2.94	4.06	1.12
1**01**	4.97	3.92	–1.05

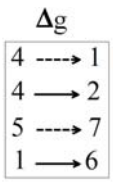


FIGURE 4. A sample network translator (Δg) that is created after the environmental change. In Δg, the dashed/solid arrows indicate the edges that should be removed/added after the change.

Assume that, using the nsDBN concept introduced in Section 3.2, the network translator (Δg) depicted in Figure 4 is created to represent the change that happens in the conditional dependencies of problem variables due to the environmental change. This network translator is converted to an adjacent matrix as follows:

$$\Delta G = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

At this juncture, the obtained network translator and the DSF vector along with the best solution obtained before and after the change (i.e., BoSBC and BoSAC) are stored in memory as a new element.

Example 3.2. Retrieving a proper element from the memory *When the environment changes, in addition to appending a new element to memory, the memory is searched for choosing a suitable memory element that is relevant to the change similar to the new change. To do this, the DSF vector for the new change is calculated by the method explained in Example 3.1. Assume that the DSF vector of the current change is given by*

$$DSF_{new} = [-1.8 \qquad 1.01 \qquad -0.91] \tag{6}$$

Next, the distances (i.e., SBDs) of the new DSF and the DSF of all the memory elements are computed. The element with the least SBD is retrieved. Also, assume that the element appended to memory in Example 3.1 has the least SBD; the memory element is retrieved. Its network translator shown in Figure 4 is used to modify the current Bayesian network of BOA. If the directed edges $(4, 1)$ *and* $(5, 7)$ *exist in the current Bayesian network, they should be removed. If the directed edges* $(4, 2)$ *and* $(1, 6)$ *do not exist in the current Bayesian network, they should be inserted. Then, the BOA algorithm continues the optimization process on the basis of the modified Bayesian network.*

4. **Experiments and Results.** To evaluate the DBN-MBOA method, we compare the results of DBN-MBOA with those of EI-MBOA [13] which is one of the most up-to-date methods applying BOA to the dynamic environments. For this evaluation, a dynamic problem generator that constructs dynamic environments onto the stationary optimization problems is used.

4.1. **Dynamic problem generator.** The dynamic optimization problem (DOP) generator based on a bitwise exclusive-or (XOR) operator, termed XOR-DOP, was introduced in [32, 33]. The generator was further extended to generate three kinds of dynamic environments called *random*, *cyclic*, and *cyclic with noise* from any binary-encoded stationary function $f(x)$, $x \in \{0,1\}^L$ where L is the length of the binary input vector [2, 14]. In the XOR-DOP generator, it is assumed that the environment periodically changes at every T fitness evaluations. Each environment is characterized by an `XOR` mask. Thus, at each environment, the corresponding `XOR` mask is used to evaluate the fitness of candidate solutions. The fitness evaluation at generation t is done by

$$f(x,t) = f(x \texttt{ XOR } M(k)) \tag{7}$$

where $k = \lceil t/T \rceil$ is the environmental period index [32, 33].

For generating a random environment, the `XOR` mask is set to a zero vector and the next environment is generated by a random change in the current environment [32, 33]. To this end, the `XOR` mask for an environmental period k, denoted by $M(k)$, is generated incrementally as follows:

$$M(k) = M(k-1) \texttt{ XOR } T(k) \tag{8}$$

Here, $T(k)$ is an intermediate binary template randomly created with $\rho \times L$ 1s for an environmental period k where $\rho \in (0,1)$.

To generate a cyclic environment, a limited number of predetermined environmental states are produced and the environment enters into these states sequentially [2, 14]. More specifically, $2k$ `XOR` masks are generated randomly as the base states in the search space and then the environment cycles among these base states in a fixed logical ring. To generate a cyclic environment with noise, meanwhile, the `XOR` mask $M(k)$ moves to the next state after a bit-wise flipping with a small probability of p_n, called the noise rate [2, 14]. In the XOR-DOP generator, the parameters T and ρ control the speed and the severity of the environmental changes, respectively. More details on the XOR-DOP generation can be found in [2, 14].

4.2. **Stationary test problems.** In the performance evaluation of DBN-MBOA using the XOR-DOP generator, some binary-encoded functions should be chosen as the stationary base functions that are constituents of the dynamic test environments. Similar to [13], three decomposable unitation-based functions (DUFs) are of use to this evaluation. Each DUF consists of 25 copies of 4-bit building blocks, in which each building block is a unitation-based function contributing a maximum value of 4 to the total fitness [13]. The building blocks of the three DUFs are defined in Equations (9)-(11).

$$f_{DUF1}(x) = u(x) \tag{9}$$

$$f_{DUF2}(x) = \begin{cases} 4 & \text{if } u(x) = 4 \\ 2 & \text{if } u(x) = 3 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$f_{DUF3}(x) \begin{cases} 3 - u(x) & \text{if } u(x) < 4 \\ 4 & \text{otherwise} \end{cases} \tag{11}$$

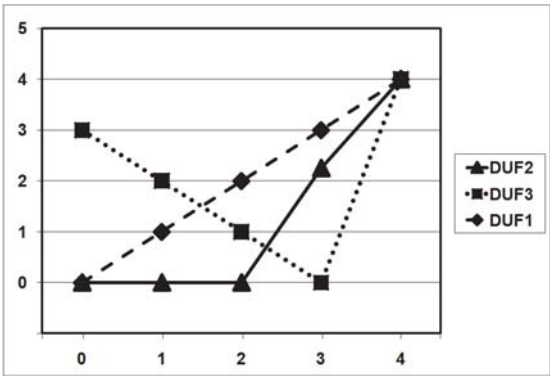


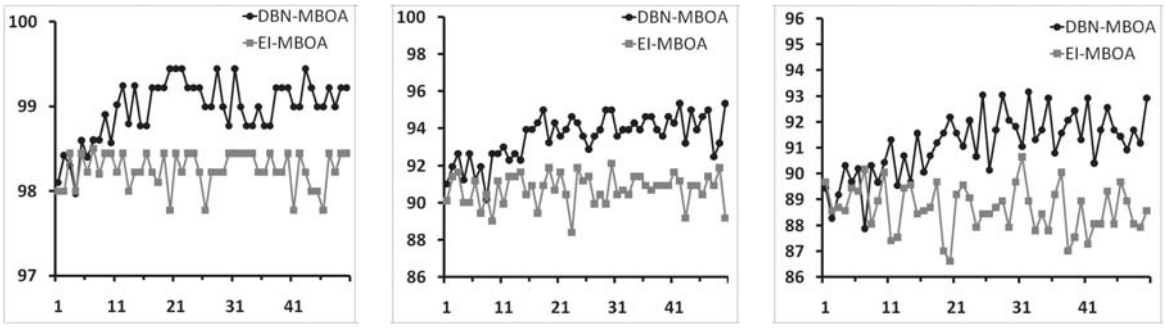
FIGURE 5. Building blocks of the three DUFs [13]

TABLE 2. The control parameter settings for DBN-MBOA

Parameter	Value	Comments
Size of population	100	–
Size of each chromosome (L)	100	Each DUF is a 100-bit binary vector
Severity of change (ρ)	0.2	–
Noise rate (p_n)	0.01	–
Speed of chage (T)	1000	The environment changes after every 1000 fitness evaluations
Total number of environment changes	50	–
Number of runs for each environment	50	–
Memory size	20	–
Truncation selection rate (p_s)	0.5	–
Number of intermediate binary templates in the XOR-DOP generator	10	In cyclic environments with/without noise, the environment re-cycles after it changes 10 times

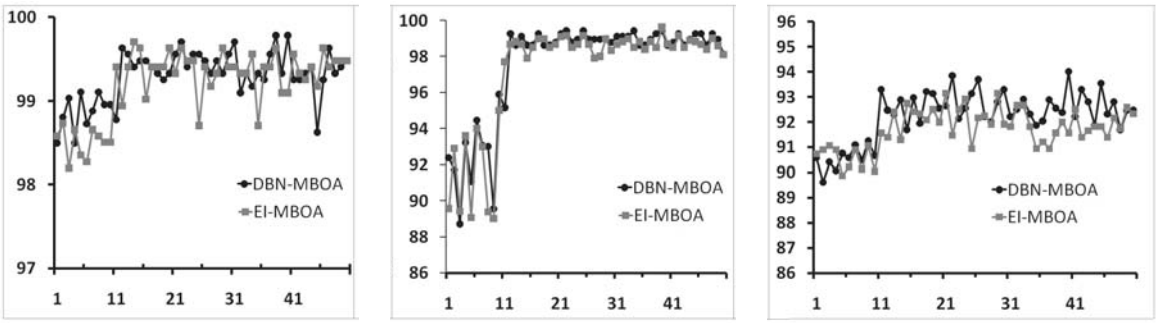
where $u(x)$ denotes the number of 1s in each building block. Also, these building blocks are shown in Figure 5. The fitness of a bit string is the sum of contributions from all building blocks. The optimal fitness for all the three DUFs is 100 [2]. Notice that DUF1 is a less difficult problem and DUF3 is a very difficult problem.

4.3. **Experimental results.** Comparing the DBN-MBOA results with the EI-MBOA results, the LC+BC method [34] used in EI-MBOA to resolve the problem of "diversity loss" is also employed in DBN-MBOA. The control parameters of DBN-MBOA are set to the values given in the EI-MBOA method [13]. These parameters are listed in Table 2. In addition, the number of randomly generated schemas in the DBN-MBOA method is set to 10. As the number of schemas increases, the environmental change can be described more precisely but the computation time becomes worse (which makes trouble under the fast environmental changes). On the other hand, the memory size is 20. It means that whenever a new element is to be added to memory, if the number of elements existing in the memory is already 20, the replacement strategy explained in Section 3.5 proceeds to keep the memory size constant. By increasing the memory size, the knowledge about more environmental changes can be stored in the memory. However, the problem is that the larger size of memory would increase the required search time to find a proper element out of the memory. Fortunately, the replacement strategy ensures that even though the size of memory is limited, in all the times the memory can contain the elements with a high degree of variety. This property leads to a higher probability to find suitable memory elements. In this way, the limitation of finite memory can be compensated.



(a) Results for DUF1 (b) Results for DUF2 (c) Results for DUF3

FIGURE 6. Performance comparison of DBN-MBOA and EI-MBOA in the random environments



(a) Results for DUF1 (b) Results for DUF2 (c) Results for DUF3

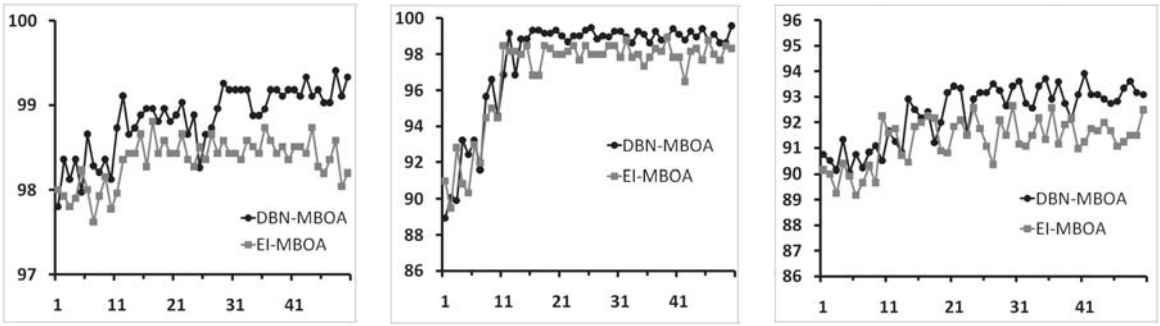
FIGURE 7. Performance comparison of DBN-MBOA and EI-MBOA in the cyclic environments

TABLE 3. Improvement in CMF obtained by DBN-MBOA compared with EI-MBOA

Function / Environments	**DUF1**	**DUF2**	**DUF3**
Random	0.75%	2.86%	2.48%
Cyclic	0.10%	0.12%	0.42%
Cyclic with noise	0.53%	0.78%	1.11%

As the performance measure, the best fitness values obtained by the DBN-MBOA and EI-MBOA methods in each of 50 environments over 50 runs are compared. The results of the algorithms in the random environments for the functions DUF1, DUF2, and DUF3 are shown in Figure 6. Similarly, the results of the algorithm in the cyclic environments and the cyclic environments with noise for DUF1, DUF2, and DUF3 are depicted in Figure 7 and Figure 8, respectively. Furthermore, the collective mean fitness (CMF) presented in [35] is also used to evaluate the algorithms. This measure calculates the averaged fitness of the best solutions over all the generations as well as over all the runs. The improvement in the CMF measure achieved by DBN-MBOA compared with EI-MBOA under the different environments is demonstrated in Table 3.

The results show that in random environments, DBN-MBOA achieves much higher performance than EI-MBOA, while in cyclic environments without/with noise the performance of DBN-MBOA is similar to that of EI-MBOA. The reason can be explained

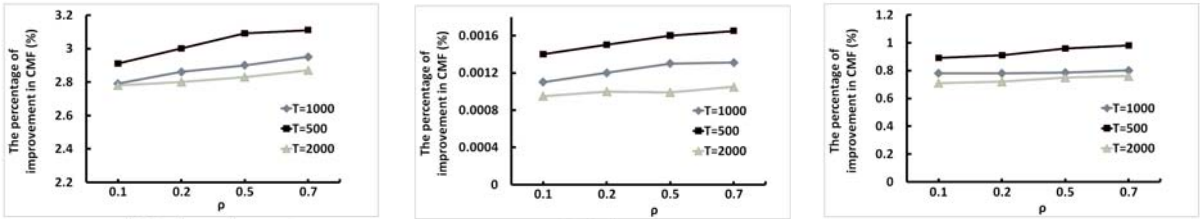


(a) Results for DUF1 (b) Results for DUF2 (c) Results for DUF3

FIGURE 8. Performance comparison of DBN-MBOA and EI-MBOA in the cyclic environments with noise

as follows. In case that the environmental changes are cyclic, the new environments have been exactly investigated in previous times. Hence, the EI-MBOA method that stores the probability models of the previously observed environments in its memory, is definitely efficient. Due to the similar property of storing the information about the environmental changes in memory, the DBN-MBOA method is comparable to the EI-MBOA in these cyclic environments. For the cyclic environments with noise, the new environments are not same as the ones that have been previously observed. Although the probability models on the environments are stored in memory, the efficiency in these environments is not as good as that in the cyclic environments. However, the EI-MBOA and DBN-MBOA methods achieve acceptable performance. On the other hand, the new environments occur randomly in the random environments. In other words, they have not been visited in previous times; thus, any useful information on the environmental changes is not stored in memory. As such, the EI-MBOA method in such environments is not so efficient. Notice that in the environments with random changes, the DBN-MBOA method focuses on the characteristic of the change rather than the environment itself. In other words, DBN-MBOA discovers a similar change occurred already in the environment and then uses it as a pattern for the current change. More specifically, it connects the solutions before and after the change in the current environment based on that pattern. Thus, the DBN-MBOA method does not need to go through the new environment in advance; discovering a previous change that has the most similar characteristic of the current change is beneficial to achieve better performance. As a result, the DBN-MBOA method is more efficient than the EI-MBOA method in the random environments.

In order to study the effect of the speed of change T and the severity of change ρ on the DBN-MBOA performance in dynamic environments, a various setting of the two parameters is tested. The value of T is set to 500, 1000 and 2000, and the value of ρ is set to 0.1, 0.2 and 0.5. The other control parameters are adjusted according to Table 2. Each algorithm runs 50 times for each parameter setting. The improvement results in CMF obtained by DBN-MBOA compared with EI-MBOA in different environments for DUF2 are represented in Figure 9. It is seen that the improvement attained by our method increases as the value of ρ increases. The reason is that in the EI-MBOA method, after an environmental change, it tries to search for an environment similar to this newly arisen environment; for the severe changes (i.e., the value of ρ is high), the probability of discovering such an environment is low, especially for random environments. On the other hand, in the DBN-MBOA method, after an environmental change, it looks for a similar environmental change occurred in the past. For higher/lower values of ρ, the



(a) Random environment (b) Cyclic environment (c) Cyclic with noise

FIGURE 9. Improvement in CMF achieved by DBN-MBOA compared with EI-MBOA for a various setting of T and ρ in different environments

TABLE 4. The t-test results for comparing DBN-MBOA and EI-MBOA on DUF2 for several values of parameters T and ρ in different types of environments (S: Significant, N: Non-significant)

T	500				1000				2000			
ρ	0.1	0.2	0.5	0.7	0.1	0.2	0.5	0.7	0.1	0.2	0.5	0.7
Random	S	S	S	S	S	S	S	S	S	S	S	S
Cyclic	S	S	S	S	N	N	N	N	N	N	N	N
Cyclic with noise	S	S	S	S	S	S	S	S	S	S	S	S

environment undergoes severe/gentle changes and the previous environmental changes have been intensive/slight as well. In the two cases, the probability of finding a similar environmental change in the memory does not vary. Figure 9 illustrates that under the same value of ρ, the impact of DBN-MBOA decreases as the value of T increases. It is also observed that the DBN-MBOA method achieves better improvement in the random environments.

Lastly, we have applied a one-tailed t-test at a 0.05 level of significance to investigate whether there is any statistically significance difference between the DBN-MBOA and EI-MBOA methods. It has been applied to the CMF values obtained by both the methods for DUF2 and for the various parameters, T and ρ, in different types of environments. The t-test results are demonstrated in Table 4. It is seen that the performance of DBN-MBOA is significantly better than that of EI-MBOA in random environments and cyclic environments with noise. However, no significant difference exists in the two methods in cyclic environments when the value of T is high; they perform very well unless the speed of change is fast.

5. **Conclusion.** In this paper, using dynamic Bayesian networks, a new method called DBN-MBOA has been proposed for optimization in dynamic environments. In the DBN-MBOA method, we have focused on the characteristic of the change in the environment rather than the characteristic of the environments before and after the change. An nsDBN model has been adopted to describe the correlations between the conditional dependencies of solution variables before and after the occurrence of the environmental change. These correlations are stored in memory and when any change occurs in the environment, they are retrieved subject to the characteristic of the change. Then the evolutionary trajectory of BOA is conducted/adjusted towards the proper direction, by which the environment can be effectively adapted. The experimental results showed that the DBN-MBOA method

leads to better performance in the non-cyclic (i.e., random) environments that the EI-MBOA method is not efficient. Thus, the DBN-MBOA method works well for a set of real-world dynamic problems whose changes in their parameters or their objectives take place randomly (i.e., their changes have no repetitive pattern). As the future work, the extension of the proposed approach into hierarchical BOA (hBOA), the most powerful method for solving nearly decomposable and hierarchical problems, would be very promising.

Acknowledgment. This work was supported by the NRF funded by the Korea government (MEST) (No. 2012-013-735).

REFERENCES

[1] L. T. Bui, H. A. Abbass and J. Branke, *Multiobjective Optimization for Dynamic Environments*, pp.2349-2356, 2005.

[2] S. Yang and X. Yao, Population-based incremental learning with associative memory for dynamic environments, *IEEE Transactions on Evolutionary Computation*, vol.12, no.5, pp.542-561, 2008.

[3] Y. Jin and J. Branke, Evolutionary optimization in uncertain environments – A survey, *IEEE Transactions on Evolutionary Computation*, vol.9, no.3, pp.303-317, 2005.

[4] J. Branke, E. Salihoglu and S. Uyar, Towards an analysis of dynamic environments, *Genetic and Evolutionary Computation Conference (GECCO'05)*, Washington, DC, USA, pp.1433-1440, 2005.

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Boston, 1989.

[6] S. Yang, Explicit memory schemes for evolutionary algorithms in dynamic environments, *Evolutionary Computation in Dynamic and Uncertain Environments*, Springer, Heidelberg, vol.51, pp.3-28, 2007.

[7] R. Tinos and S. Yang, Genetic algorithms with self-organizing behaviour in dynamic environments, *Evolutionary Computation in Dynamic and Uncertain Environments*, pp.105-127, 2007.

[8] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, Boston, MA, USA, 2002.

[9] M. Pelikan, D. E. Goldberg and F. Lobo, A survey of optimization by building and using probabilistic models, *Computational Optimization and Applications*, vol.21, no.1, pp.5-20, 2002.

[10] M. Pelikan, K. Sastry and E. Cantù-Paz, *Scalable Optimization via Probabilistic Modelling: From Algorithms to Applications*, Springer, 2006.

[11] H. Mühlenbein and G. Paaß, From recombination of genes to the estimation of distributions: Binary parameters, *Proc. of the 4th International Conference on Parallel Problem Solving from Nature*, pp.178-187, 1996.

[12] J. Brank, *Evolutionary Optimization in Dynamic Environments*, Kluwer Academic Publishers, Boston, 2002.

[13] X. Peng, X. Gao and S. Yang, Environment identification based memory scheme for estimation of distribution algorithms in dynamic environments, *Soft Computing*, vol.15, no.2, pp.311-326, 2010.

[14] S. Yang, Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems, *IEEE Congress on Evolutionary Computation (CEC'05)*, pp.2560-2567, 2005.

[15] S. Baluja, Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning, *Technical Report CMU-CS-94-163*, Carnegie Mellon University, USA, 1994.

[16] S. Yang, Population-based incremental learning with memory scheme for changing environments, *Genetic and Evolutionary Computation Conference (GECCO'05)*, Washington, DC, USA, pp.711-718, 2005.

[17] M. Pelikan, *Bayesian Optimization Algorithm: From Single Level to Hierarchy*, Ph.D. Thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 2002.

[18] M. Pelikan, *Hierarchical Bayesian Optimization Algorithm, Toward a New Generation of Evolutionary Algorithms*, Springer-Verlag, Berlin Heidelberg, 2005.

[19] P. Lucas, J. Gámez and A. Salmerón, *Advances in Probabilistic Graphical Models (Studies in Fuzziness and Soft Computing)*, Springer-Verlag, 2007.

[20] S. Yang, H. Cheng and F. Wang, Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks, *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews*, vol.40, no.1, pp.52-63, 2010.

[21] S. Yang, Genetic algorithms with memory- and elitism-based immigrants in dynamic environments, *Evolutionary Computation*, vol.16, no.3, pp.385-416, 2008.
[22] V. Mihajlovic and M. Petkovic, Dynamic Bayesian networks: A state of the art, *Technical Report TR-CTIT-01-34*, Centre for Telematics and Information Technology University of Twente, Enschede, 2001.
[23] H. C. Cho, K. S. Lee and M. S. Fadali, Online learning algorithm of dynamic Bayesian networks for nonstationary signal processing, *International Journal of Innovative Computing, Information and Control*, vol.5, no.4, pp.1027-1041, 2009.
[24] F. Dondelinger, S. Lebre and D. Husmeier, Heterogeneous continuous dynamic Bayesian networks with flexible structure and inter-time segment information sharing, *International Conference on Machine Learning (ICML'10)*, pp.303-310, 2010.
[25] J. W. Robinson and A. J. Hartemink, Learning non-stationary dynamic Bayesian networks, *Journal of Machine Learning Research*, vol.11, pp.3647-3680, 2010.
[26] L. Song, M. Kolar and E. Xing, Time-varying dynamic Bayesian networks, *Advances in Neural Information Processing Systems (NIPS)*, vol.22, pp.1732-1740, 2009.
[27] M. Grzegorczyk and D. Husmeier, Non-stationary continuous dynamic Bayesian networks, *Advances in Neural Information Processing Systems (NIPS)*, vol.22, pp.682-690, 2009.
[28] J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA, 1975.
[29] S. Rajasekaran and G. A. V. Pai, *Neural Networks, Fuzzy Logic and Genetic Algorithms: Synthesis and Applications*, Prentice-Hall of India Pvt. Ltd., 2004.
[30] T. M. Mitchell, *Machine Learning*, Mc Graw-Hill, 1999.
[31] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, *IEEE Congress on Evolutionary Computation (CEC'99)*, pp.1875-1882, 1999.
[32] S. Yang, Non-stationary problem optimization using the primal-dual genetic algorithm, *IEEE Congress on Evolutionary Computation (CEC'03)*, pp.2246-2253, 2003.
[33] S. Yang and X. Yao, Experimental study on population-based incremental learning algorithms for dynamic optimization problems, *Soft Computing*, vol.9, no.11, pp.815-834, 2005.
[34] J. Branke, C. Lode and J. L. Shapiro, Addressing sampling errors and diversity loss in UMDA, *Genetic and Evolutionary Computation Conference (GECCO'07)*, pp.508-515, 2007.
[35] R. Morrison and K. De Jong, A test problem generator for nonstationary environments, *IEEE Congress on Evolutionary Computation (CEC'99)*, pp.2047-2053, 1999.