Online Learning in Estimation of Distribution Algorithms for Dynamic Environments

André R. Gonçalves*, Fernando J. Von Zuben†
School of Electrical and Computer Engineering
University of Campinas (Unicamp)
Campinas, SP, Brazil
{*andreric, †vonzuben}@dca.fee.unicamp.br

***Abstract*—In this paper, we propose an estimation of distribution algorithm based on an inexpensive Gaussian mixture model with online learning, which will be employed in dynamic optimization. Here, the mixture model stores a vector of sufficient statistics of the best solutions, which is subsequently used to obtain the parameters of the Gaussian components. This approach is able to incorporate into the current mixture model potentially relevant information of the previous and current iterations. The online nature of the proposal is desirable in the context of dynamic optimization, where prompt reaction to new scenarios should be promoted. To analyze the performance of our proposal, a set of dynamic optimization problems in continuous domains was considered with distinct levels of complexity, and the obtained results were compared to the results produced by other existing algorithms in the dynamic optimization literature.**

***Index Terms*—Online learning; mixture model; estimation of distribution algorithms; optimization in dynamic environments.**

I. Introduction

Evolutionary algorithms (EAs) have been applied successfully to a variety of optimization problems. Such problems are generally considered to be stationary, i.e., the objective function and other aspects of the formulation does not change over time. However, it is common to face scenarios in which some aspects of the optimization problem varies with time, such as the presence of a nonstationary objective function and/or a time-varying constraint set. Under these circumstances, the location of the optimal solution in the search space might change with time as well. As practical examples, we may cite a scheduling problem in which new jobs arrive over time, and vehicle routing problems where the cost to travel from one location to another is nonstationary. Thus, standard evolutionary algorithms, which are basically built to deal with stationary problems, tends to produce a poor performance in dynamic environments, due to the absence of reaction mechanisms when facing new conditions.

To be adapted to dynamic environments, EAs should be endowed with specific (implicit or explicit) operators devoted to the maintenance of diversity in the population, and mainly to avoid stagnation. Some of these additional operators, as pointed out by [1], should promote: (i) generation of diversity after a change; (ii) maintenance of diversity throughout the execution; (iii) employment of memory to store and recall useful information from the recent past; and (iv) execution of a multipopulation search to track multiple local optima in the fitness landscape.

Trying to implement at least part of the aforementioned functionalities, several evolutionary approaches have been developed and extended to deal with dynamic environments. Among them, we can cite genetic algorithms [2], [3], particle swarm optimization [4], [5], artificial immune systems [6], [7] and estimation of distribution algorithms (EDAs) [8]–[10].

In this paper, we introduce an inexpensive EDA to the optimization of dynamic environments in continuous domains. This approach employs an online learning version of a mixture model, augmented by an automatic control of the number of components and by an operator to promote population diversity. The purpose is to make the search mechanism effective even in multimodal and time-varying fitness landscapes, by properly applying a concise model of the information from the previous iterations. This paper is organized as follows: Section II presents a brief introduction to estimation of distribution algorithms. In Section III, mixture models, the Expectation-Maximization (EM) algorithm and an online version of EM are described. The details of the proposed algorithm are presented in Section IV. A generator of dynamic environments in continuous search spaces, to be considered in the experiments, is depicted in Section V. Section VI discusses the obtained results on a number of dynamic optimization problems. Concluding remarks and further steps of the research are outlined in Section VII.

II. Estimation of distribution algorithm

Estimation of distribution algorithms (EDAs) are evolutionary methods that use estimation of distribution techniques, instead of genetic operators, to guide the search for better solutions. These algorithms were motivated by some shortcomings of genetic algorithms, more specifically the absence of a proper mechanism for detecting and preserving *building blocks* [11].

The key aspect in EDAs is how to estimate the true distribution of promising solutions. In fact, the estimation of the joint probability distribution associated with the variables that form the search space is the main distinctive component of EDAs. Once estimated the joint probability distribution, new candidate solutions are generated obeying the obtained distribution. Though essential to characterize an EDA, this

978-1-4244-7835-4/11/$26.00 ©2011 IEEE

estimation is generally the most computationally intensive step of the algorithm. So, a compromise between accuracy of estimation and computational cost should be considered. Figure 1 shows the general framework of EDAs.

Step 1. Generate a random initial population $S_0(X)$. Set $t \leftarrow 0$.
Step 2. Evaluate individuals from $S_t(X)$.
Step 3. Select a set of promising solutions, $S' \subset S_t(X)$.
Step 4. Estimate the probability model $P(X)$ from S'.
Step 5. Sample $S_{t+1}(X)$ from the probability model $P(X)$.
Step 6. Set $t \leftarrow t + 1$ and return to step 2.

Fig. 1. A general framework for an EDA.

The already presented EDAs differ from each other basically by the way that the estimation of the probability model is made. They are classified according to the complexity of the probability model adopted: *without dependencies*, *bivariate dependencies*, *multivariate dependencies* and *mixture models* [12].

In this paper, the focus is on a class of EDAs for continuous optimization, based on an online version of a mixture model.

III. Mixture models

Considering that we have a dataset x, such that $x = \{x_1, x_2, ..., x_n\}$, we may proceed to divide it into K subsets, also denoted components, so that each one has a mixture coefficient π_k, such that $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$. Subsequently, we define a probability density function (pdf) for each subset, denoted f_k. The parameter vectors Θ_k of each pdf, $k = \{1, ..., K\}$, will composed a single parameter vector Θ and are estimated from the available dataset. Therefore, the proposed probability model combines pdfs, each one associated with a component (subset of the dataset), thus producing a *mixture model*. When a Gaussian pdf is applied to fit each subset, the estimator is known as a *Gaussian mixture model* [13].

The task of obtaining the parameters of each probability density function, denoted parameter estimation, can be solved with several approaches. The maximum likelihood approach is the standard one for mixture models [13]. The likelihood of the dataset, given the parameters of the pdfs, is defined as the conditional probability $p(x|\Theta)$, which we generalize as $f(x;\Theta)$, to denote any density function f with parameters in Θ. Considering that the data are independent and identically distributed (iid), we can define the joint data likelihood as $\prod_j f(x_j;\Theta)$. Using the logarithmic transformation, we define the log-likelihood function as [14]:

$$L(\Theta) = \ln \prod_j f(x_j;\Theta) = \sum_j \ln f(x_j;\Theta). \quad (1)$$

The objective of the estimator is to define a vector of parameters, $\hat{\Theta}$, that maximizes the function in (1). The necessary condition for optimality is:

$$\frac{\partial L(\Theta)}{\partial \Theta} = 0. \quad (2)$$

This concept provides the basis for the derivation of a learning algorithm for the parameters of the mixture model, generally performed by the *expectation-maximization* (EM) algorithm [15]. EM is a powerful method to find maximum likelihood solutions to models with latent variables [13].

A. Online version of EM algorithm

An online version of the EM algorithm is motivated by the necessity of processing data streams, in which the data are coming in a sequential and continuous way. So, the traditional EM algorithm becomes impractical due to the requirement that the whole data be available at each iteration of the algorithm.

The online variant of the EM used here is based on the work of Nowlan [16]. This algorithm summarizes all data item by a vector of sufficient statistics that can be incrementally updated. After that, this vector is used to compute the parameters of the mixture model. Sufficient statistics is a function $s(\cdot)$ of the samples that contains all the information relevant to estimate some parameters [14]. The EM steps of these online variant, described in [17], are shown in Fig. 2, where Z is the unobserved variables, X is the observed data, $\tilde{s}^{(t)}$ is the vector of sufficient statistics at time t, $E_{\tilde{P}_i}$ denotes the expectation with respect to the distribution over the range of unobserved variable Z, and $0 < \gamma < 1$ is a decay constant.

E-step: Select the next data item, i , for updating.
Set $\tilde{s}_i^{(t)} = E_{\tilde{P}_i}[s_i(z_i, x_i)]$, for $\tilde{P}_i(z_i) = P(z_i|x_i, \Theta^{(t-1)})$.
Set $\tilde{s}^{(t)} = \gamma \tilde{s}^{(t-1)} + \tilde{s}_i^{(t)}$.
M-step: Set $\Theta^{(t)}$ to the Θ with maximum likelihood given $\tilde{s}^{(t)}$.

Fig. 2. Online version of EM algorithm [16].

The algorithm uses statistics computed as an exponentially decaying average of recently-visited data points. In accordance with [17], online variants of the EM algorithm will not converge to the exact answer, but are able to converge to the vicinity of the correct answer more rapidly than standard EM, once chosen an appropriate value for γ. In the context of optimization, this drawback can be alleviated by the exploration capability of the search space provided by the evolutionary algorithm.

IV. Mixture model with online learning for optimization in dynamic environments

In this section, we describe an estimation of distribution algorithm based on an *Online Gaussian Mixture Model*, labeled EDA_{OGMM} and designed for continuous optimization in dynamic environments.

In order to achieve high performance in multimodal problems, it is necessary to use multiple Gaussian distributions, each one devoted to one promising region of the search space, thus producing a mixture model. However, when these models are implemented for dynamic problems, there is a concern about the computational cost of the training algorithm for the mixture model.

In accordance with [18], unless the dynamic change is of a high intensity, it is expected that better solutions are achieved by continuously adapting the current solutions to a changing environment, usually reusing the information gained in the near past. Thus, in Fig. 3 we propose an online learning algorithm capable of aggregating information from the past and current conditions of the environment, incorporating it into the mixture model.

The use of past information, besides the current one, seems to be relevant to mixture models, because the more pertinent data is available to estimate the fitness landscape, the more accurate the model tends to be.

The parameter γ (see Fig. 2) defines for how long the information will be preserved to estimate the vector of sufficient statistics. Reducing γ, less information from the past will be stored in the mixture model. So, we can choose a γ value coherent with the ratio of change in the environmental conditions. The contribution of each data item to the mixture model, over time, will be reduced by a geometric progression with ratio γ.

The EDA_{OGMM} algorithm is outlined in Fig. 3, where N is the number of individuals in population, η is the percentage of population used to estimate the promising regions in the search space and K is the number of components in the mixture model.

Starting with a single component, $K = 1$, the EDA_{OGMM} tries to add more components, in a constructive fashion whilst it performs a continuous search for new promising regions, keeping in the population the best solutions found so far.

Two other important features of the EDA_{OGMM} should be highlighted. First, in Step 5, after updating the vector of sufficient statistics, just a single M-step is applied, instead of the complete execution of the online EM algorithm. This is motivated by the work of [10], which has shown an overtime learning mixture model, where the convergence takes place along iterations and not at each iteration. With a single M-step, the computational time of the algorithm is significantly reduced.

Secondly, a key feature is the continued analysis of the overlapping components in the mixture model. Overlapping components are Gaussian terms centred at the same promising region of the search space. So, one of the components can be eliminated without reducing the representation capability of the mixture model, thus saving computational resources. We consider that two components are overlapped if the Euclidian distance between their centers is lower than a threshold ϵ. However, any other procedure to detect that two centers are located at the same basin of attraction of the optimization surface may be considered here.

The control of the number of components in the mixture model is made by a rule (Step 5), which is based on the Bayesian information criterion (BIC) score. BIC is usually applied in model selection tasks, i.e., given two models, it indicates which one maximizes the likelihood of the data, given the parameters and complexity of the model (number of parameters to be learned).

Step 1. Generate a random initial population and set $K \leftarrow 1$.

Step 2. Evaluate individuals in the current population.

Step 3. Select $\eta \cdot N$ of the best individuals from the population, using a fitness proportional selection method, like *tournament selection*.

Step 4. Set $K \leftarrow K + 1$, update the vector of sufficient statistics and estimate the distribution of the selected data by applying a single M-step.

Step 5. Calculate the BIC for the new estimated model and verify if this is greater than or equal to the BIC of the best model so far. If so, set $K \leftarrow K - 1$.

Step 6. Remove, if existent, components which have converged to the same promising region (overlapped components), keeping only one of them.

Step 7. Sample $\eta \cdot N$ new individuals from the estimated mixture model.

Step 8. Preserve $\frac{(1-\eta) \cdot N}{2}$ from the best solutions deterministically selected.

Step 9. Randomly sample $\frac{(1-\eta) \cdot N}{2}$ new individuals.

Step 10. Set the new population as the union of these three subpopulations: sampled by the model, preserved, and sampled randomly. Return to Step 2.

Fig. 3. Outline of the EDA_{OGMM} algorithm.

EDA_{OGMM} uses a user-controllable parameter η that defines the proportion of individuals that will be considered on the estimation of distribution and, consequently, how many random immigrants will be introduced in the population (Step 9). If η is a high value, EDA_{OGMM} tends quickly to a local optimum solution, because there is an insufficient number of individuals to promote global exploration. However, for low values of η, EDA_{OGMM} possibly faces a slow convergence, which is not suitable for applications in dynamic environments.

An elitist approach is employed (Step 8) to preserve good solutions found so far. Besides that, random immigrants are responsible for the exploratory behavior of the algorithm, aiming at finding new promising regions not yet considered in the mixture model.

V. Experimental settings

In order to evaluate the performance of the EDA_{OGMM} and to compare with contender algorithms in dynamic optimization problems, we will define a test suite based on the well known Moving Peaks benchmark (MPB) generator introduced by [18], plus a rotation method, instead of random shifting the location of the peaks, as proposed by [19]. The rotation method is motivated by the MPB disadvantages of unequal challenge per change when the position of a peak bounces back from the search boundary [19].

A. Specification of the dynamic environment generator

The moving peaks generator uses, besides the specific variables to control the changes, six basic parameters to define the dynamic environment: (1) search space range, (2) number of peaks, (3) period between consecutive changes, (4) peaks height range, (5) peaks width range, and (6) dimension range.

The function $f(x, \phi, t)$ to be maximized, with $\phi = (\vec{H}, \vec{W}, \vec{X})$ and where $\vec{H}$, $\vec{W}$ and $\vec{X}$, and denote the peak height, width and position, respectively, can be expressed as follows:

$$f(x, \phi, t) = \max_{m} \frac{\vec{H}_m(t)}{1 + \vec{W}_m(t) \cdot \sqrt{\sum_{j=1}^{n} \frac{(x_j - \vec{X}_j^m(t))^2}{n}}}, \quad (3)$$

where m is the number of peaks and n is the number of dimensions. Here, $\vec{H}_m$ and $\vec{W}_m$ are the height and the width of the m-th peak, respectively, and $\vec{X}_j^m$ is the position in the j-th dimension of the m-th peak. The parameters $\vec{W}$ and $\vec{X}$ changes according to seven distinct behaviors ($T_1 \sim T_7$): *small step, large step, random, chaotic, recurrent, recurrent with noise* and *random with dimensional changes* (to simulate structural variations). The mathematical definition and parameter values of all change profiles can be seen in [19]. The changes in position ($\vec{X}$) and width ($\vec{W}$) of the peaks are responsible for the dynamic nature of the environment.

B. Varying period between changes and number of peaks

To simulate several types of dynamic environments with several complexity levels, in relation to the frequency of change (measured in terms of the number of fitness evaluations - FEs) and the multimodality (number of peaks), which are controlled in MPB by the *period between changes* and *number of peaks* parameters, we build six different scenarios, which are specified in Tab. I.

For each one of the above scenarios, the seven change profiles ($T_1 - T_7$) are applied, thus yielding a complete suite of 42 test cases. Scenarios 1 and 2 are highly dynamic, i.e., the fitness landscape changes very quickly and if the algorithm has a very slow convergence rate, probably will get a poor performance. On the other hand, in scenarios 5 and 6 we have more time to reach the optimum, but the algorithms should be able to solve a global optimization problem.

TABLE I
Settings of dynamic environments.

	Period between changes (# of FEs)	**Multimodality** (# of peaks)
Scenario 1	5000	10
Scenario 2	5000	50
Scenario 3	20000	10
Scenario 4	20000	50
Scenario 5	50000	10
Scenario 6	50000	50

All test problems were defined in a 10-dimension search space and each decision variable was restricted to belong to the interval [-5, 5]. The remaining parameters are set as follows: peaks height range is defined in [10, 100], peaks width range in [1, 10] and dimension range in [5, 15]. The parameter settings are the same adopted in [20]. The benchmark used in our experiments was, originally, developed to low frequency dynamic environments, where the changes occur each 100.000 FEs. However, the scenarios defined in our experiments vary 20, 5 and 4 times faster, respectively. The objective is to analyze the performance of the algorithms in complex dynamic environments.

C. Comparative analysis

The following approaches to deal with environments have been considered as contenders:

- Improved Univariate Marginal Distribution Algorithm (IUMDA) [9];
- Three components EDA (Tri-EDA_G) [21];
- Hypermutation genetic algorithm (HGA) [22].

The first two methods are EDAs and the third is a genetic algorithm which generates diversity just after detecting a change, by means of increasing the mutation probability. For selection of the best individuals, HGA and EDA_{OGMM} use tournament selection with five players, while IUMDA and Tri-EDA_G use deterministic selection. HGA employs a Gaussian mutation operator, being necessary to define the standard deviation. The free parameters of each algorithm, used in the experiments, are presented in Tab. II.

For the IUMDA algorithm, μ is the percentage of the best individuals to be used in an interim population of the algorithm. Several μ values were tested and μ = 0.2 (20 percent of the best individuals) yielded the best results, given the population size considered. The mutation parameter ($P_{mut.}$) was chosen using a similar procedure.

Concerning the Tri-EDA_G algorithm, ρ is the percentage of best individuals selected for updating the parameters of the local exploration multivariate Gaussian distribution (primary model), and λ is the percentage of population of the next generation, to be generated by the primary model. Their values are the same as used by the authors in their original paper [21].

For HGA, besides the crossover probability ($P_{cross.}$), there are two mutation probabilities to be specified: hypermutation

TABLE II
PARAMETER SETTINGS FOR THE ALGORITHMS.

Algorithm	Free parameter	Value
IUMDA	μ	0.2
	$P_{mut.}$	0.01
	$\sigma_{mut.}$	0.05
Tri-EDA$_G$	ρ	0.3
	λ	0.8
HGA	$P_{cross.}$	0.8
	$P_{mut.}$	0.01
	$P_{Hyp.mut.}$	0.8
	$\sigma_{mut.}$	0.05
EDA$_{\text{OGMM}}$	η	0.6
	γ	0.5
	ϵ	0.01

probability ($P_{Hyp.mut.}$), used if the environment has been changed, and $P_{mut.}$, which are used elsewhere.

One feature shared by all algorithms is the application of a mechanism to assure that the best individual of the current population is always inserted into the next population.

In all the scenarios studied, the population size is 100 and the execution time (measured by the number of changes) is 60 changes.

All the algorithms and the MPB generator were implemented using the Matlab® numerical computing environment.

The MPB generator is available for download at http://www.cs.le.ac.uk/people/syang/ECiDUE/DBG.tar.gz, in C++ and Matlab® languages, being the last one considered here. All the algorithms being considered were also implemented using Matlab®.

VI. OBTAINED RESULTS

Tables III, IV, V, VI, VII and VIII present the average offline error and standard error, for 20 independent runs, in each one of the six defined scenarios. The offline error metric is the average of the absolute error between the best solution found so far and the global optimum (known) at each time step t. The best results are those with highlighted background.

In the case of high frequency changing environments, EDA$_{\text{OGMM}}$ clearly outperforms the contender algorithms. A great advantage of the proposed method is the exploration of several peaks at the same time, thus increasing the probability of finding the global optimum, or at least better local optima. Besides that, we propagate the information from the previous to the current environment, giving to the algorithm the ability to properly adapt to a changing fitness landscape.

Estimation of distribution algorithms have a fast convergence, mainly after detecting a promising region. We may associate the behavior with the one produced by the well-known Newton's method, which at each iteration approximates the optimization surface by a quadratic function and take a step towards the minimum (or maximum) of that quadratic function. Even though the Gaussian exploration does not implement an explicit gradient descent, it take a step towards the best solutions found so far. Mixture models extrapolate this employing several Gaussian explorations simultaneously.

TABLE III
AVERAGE OFFLINE ERROR AND STANDARD ERROR FOR SCENARIO 1.

	IUMDA	Tri-EDA$_G$	HGA	EDA$_{\text{OGMM}}$
T_1	77.43(±0.75)	47.64(±1.26)	79.73(±0.88)	6.89(±0.94)
T_2	76.78(±1.13)	48.75(±1.23)	82.02(±1.01)	16.66(±2.43)
T_3	62.09(±1.30)	35.91(±1.54)	66.23(±1.36)	18.24(±1.89)
T_4	76.21(±0.23)	53.06(±0.48)	78.89(±0.08)	9.93(±1.33)
T_5	77.08(±0.50)	74.51(±0.32)	77.90(±0.91)	37.53(±1.05)
T_6	73.11(±0.83)	68.34(±0.74)	76.18(±0.58)	38.82(±1.36)
T_7	60.61(±1.42)	35.92(±2.48)	64.38(±1.74)	24.89(±2.11)

TABLE IV
AVERAGE OFFLINE ERROR AND STANDARD ERROR FOR SCENARIO 2.

	IUMDA	Tri-EDA$_G$	HGA	EDA$_{\text{OGMM}}$
T_1	80.18(±1.37)	48.34(±0.49)	83.77(±1.32)	17.60(±1.57)
T_2	73.00(±0.94)	37.79(±1.34)	77.87(±0.92)	14.43(±1.48)
T_3	62.98(±1.01)	35.86(±1.27)	68.95(±0.89)	20.36(±2.12)
T_4	76.36(±0.19)	47.44(±0.57)	78.37(±0/08)	18.06(±1.70)
T_5	78.87(±0.62)	73.47(±0.49)	80.25(±0.83)	44.50(±1.14)
T_6	74.93(±0.96)	64.61(±0.46)	79.02(±0.92)	49.74(±1.63)
T_7	65.46(±1.02)	36.80(±1.57)	69.47(±1.12)	32.77(±1.66)

We can notice also that, in recurrent changes (T_5 and T_6), EDA$_{\text{OGMM}}$ is better than the contenders, but with a smaller margin. Given that the period of the recurrent change, P, is 12 (see parameter settings in [20]), i.e., at each 12 changes the optimum goes back to the same position, the information about the first position of the optimum was lost during the execution. In these situations, the use of explicit memories can be an interesting approach. But, for applications where the period is unknown, which is common in real problems, the identification of which best solution, stored in the memory, should be re-inserted into the population is a very hard task. So, in environments where the recurrent period is unknown, maybe one effective way to deal with it is to assume that the changes are random and to perform a global optimization anyway.

TABLE V
AVERAGE OFFLINE ERROR AND STANDARD ERROR FOR SCENARIO 3.

	IUMDA	Tri-EDA$_G$	HGA	EDA$_{\text{OGMM}}$
T_1	40.30(±2.84)	11.66(±0.64)	52.94(±5.13)	2.02(±0.19)
T_2	37.40(±2.27)	14.75(±0.64)	51.19(±3.07)	7.91(±0.92)
T_3	36.47(±2.26)	14.75(±1.81)	45.22(±1.70)	11.40(±1.82)
T_4	41.94(±1.12)	13.57(±0.17)	60.01(±0.51)	5.67(±0.22)
T_5	47.64(±0.55)	47.13(±0.93)	43.38(±0.01)	14.92(±0.81)
T_6	47.77(±0.61)	38.98(±0.81)	42.66(±0.36)	25.09(±0.98)
T_7	37.54(±2.24)	17.64(±1.67)	46.71(±2.73)	11.87(±1.46)

In medium frequency changing environments, EDA$_{\text{OGMM}}$ has shown better results than the contenders. Like EDA$_{\text{OGMM}}$, Tri-EDA$_G$, which has produced the second best performance, uses a Gaussian exploration too, but only one model (main model) is always active. An optional model is triggered when a new promising region, better than the one indicated by the main model, is found. So, Tri-EDA$_G$ jumps from region to

TABLE VI
AVERAGE OFFLINE ERROR AND STANDARD ERROR FOR SCENARIO 4.

	IUMDA	**Tri-EDA$_G$**	**HGA**	**EDA$_{\text{OGMM}}$**
T_1	37.57(±1.39)	12.55(±0.70)	51.04(±4.08)	3.57(±0.36)
T_2	34.68(±1.35)	10.05(±0.49)	47.74(±2.32)	5.54(±0.62)
T_3	34.85(±1.23)	17.07(±1.45)	43.10(±1.19)	13.95(±1.28)
T_4	33.23(±0.93)	11.12(±0.23)	53.10(±0.60)	4.84(±0.14)
T_5	50.21(±0.64)	40.36(±1.46)	45.63(±0.27)	13.40(±0.77)
T_6	46.55(±1.57)	40.53(±1.11)	44.54(±1.36)	35.10(±1.32)
T_7	35.73(±1.35)	20.00(±1.45)	43.80(±1.26)	15.10(±1.26)

region, looking for the best one. Based on results obtained by EDA$_{\text{OGMM}}$, Tri-EDA$_G$ would possibly get better performance if it could manipulate many active models at the same time.

TABLE VII
AVERAGE OFFLINE ERROR AND STANDARD ERROR FOR SCENARIO 5.

	IUMDA	**Tri-EDA$_G$**	**HGA**	**EDA$_{\text{OGMM}}$**
T_1	10.47(±0.85)	6.48(±0.57)	12.51(±1.15)	1.93(±0.24)
T_2	14.29(±0.97)	11.64(±0.86)	16.86(±1.22)	8.89(±0.85)
T_3	18.16(±2.09)	15.65(±2.00)	19.68(±1.94)	14.00(±1.95)
T_4	8.13(±0.11)	7.01(±0.16)	8.86(±0.08)	5.07(±0.07)
T_5	43.32(±0.01)	40.36(±1.10)	43.25(±0.01)	10.00(±0.41)
T_6	41.91(±0.65)	36.25(±0.93)	41.44(±0.56)	21.85(±0.90)
T_7	16.63(±2.37)	15.49(±2.31)	21.76(±2.79)	12.77(±2.28)

TABLE VIII
AVERAGE OFFLINE ERROR AND STANDARD ERROR FOR SCENARIO 6.

	IUMDA	**Tri-EDA$_G$**	**HGA**	**EDA$_{\text{OGMM}}$**
T_1	14.17(±2.30)	8.62(±1.16)	13.45(±2.04)	2.61(±0.43)
T_2	12.96(±1.53)	8.93(±1.00)	13.50(±1.44)	5.93(±0.60)
T_3	16.19(±1.18)	14.10(±1.81)	16.57(±1.44)	12.33(±1.65)
T_4	7.34(±0.30)	6.11(±0.12)	6.55(±0.12)	3.37(±0.1)
T_5	45.03(±0.01)	32.54(±0.86)	45.04(±0.05)	9.39(±0.51)
T_6	39.72(±1.63)	35.79(±0.83)	39.41(±1.81)	32.85(±0.87)
T_7	17.42(±0.81)	15.58(±0.60)	18.94(±0.84)	12.97(±0.53)

For scenarios 5 and 6, the algorithm has more time to find the global optimum. If the algorithm has a fast convergence, but get stuck in a local optimum, probably it will reach poor performance in highly multimodal environments. So, the algorithm should execute a global optimization in a restricted time. Like in the other four scenarios, EDA$_{\text{OGMM}}$ has achieved the best results, followed by Tri-EDA$_G$.

In global optimization problems, the mixture model, initially, can cover several peaks and, by means of the selection operator and sampling, the mixture model finds better regions, and focuses on those promising regions. Subsequently, it applies a Gaussian exploration at each region (local optimization), to reach the local optimum.

Figures 4, 5, 6 and 7 present the median convergence behavior, over the 20 independent runs executed, of IUMDA, Tri-EDA$_G$, HGA and EDA$_{\text{OGMM}}$, respectively, when applied in scenario 2 with *random change with dimensional changes* (T_7). The performance is measure by means of the ratio between the best solution found and the global optimum. So, the ideal performance would be to stay as close as possible to the value 1.

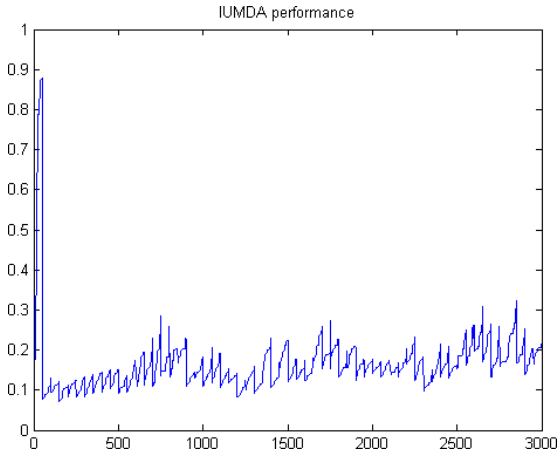

Fig. 4. Relative performance of the IUMDA algorithm.

In the first environment ($t = 1$), IUMDA finds a good local optimum (the performance achieved a value close to 0.9), but it is not able to adapt to the subsequent scenarios, reaching poor solutions in these fitness landscapes. These results showed the impact of the loss in population diversity. In $t = 1$, where the algorithm has obtained better results, the population was randomly generated, and for $t > 1$, the algorithm cannot insert sufficient diversity in the population. A possible solution is to employ the random immigrants approach at every generation or when an environment change is detected.

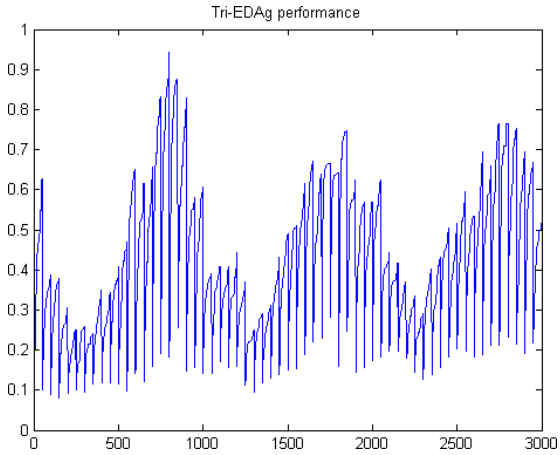

Fig. 5. Relative performance of the Tri-EDA$_G$ algorithm.

Figure 5 shows, clearly, an oscillatory performance of the Tri-EDA$_G$ algorithm. This is due to the variation in the number of dimensions of the search space. Decreasing the number of dimensions, the search space is significantly reduced and, given the same population size, the algorithm can achieve a better exploration of the search space, getting better results.

The hypermutation genetic algorithm has a slow convergence rate, which in dynamic environments is extremely undesirable. When the new position of the global optimum is very close to the old position, this approach seems interesting.

The loss of population diversity can be a bottleneck in

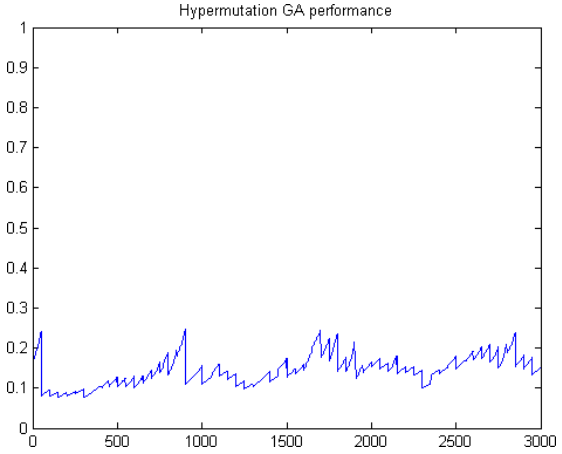

Fig. 6. Relative performance of the Hypermutation genetic algorithm.

this algorithm too. Despite the hypermutation used to produce certain diversity in the population, the exploration area is limited by the standard deviation parameter ($\sigma_{mut.}$) of the Gaussian mutation operator. An alternative to bypass this problem is to define a new standard deviation parameter or mutate the individual to any location in the search space. In this way, we establish a balance between local exploration, around the best solution found in the previous generation, and exploration of the whole search space.

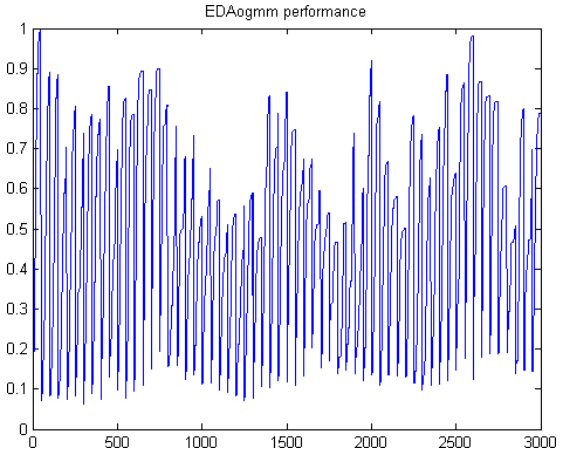

Fig. 7. Relative performance of the EDA_{OGMM} algorithm.

Oscillatory performance is not clearly evident in EDA_{OGMM} convergence behavior, different from what happens with the Tri-EDA_G, reaching consistently better results in high dimensional environments than the other three algorithms.

VII. Concluding Remarks

Dynamic environments require that the optimization algorithm adapts quickly to the new environment for tracking the global optimum. So, a continuous modeling of the fitness landscape seems an interesting alternative to keep the algorithm up-to-date.

In this paper, we have introduced an online estimation of the promising regions of the search space, by means of a Gaussian mixture model, and use this estimated model to guide an evolutionary algorithm. The estimation is made by an online version of the Expectation-Maximization algorithm, which maintains a continuously updated mixture model. Here, the dynamism of the process is captured by a vector of sufficient statistics of the best solutions, which are used by a mixture model.

The results has shown that the proposed EDA_{OGMM} outperforms all the contenders, particularly in high-frequency changing environments (Scenarios 1 and 2).

EDA_{OGMM} has a fast convergence because it can explore several peaks simultaneously. Each peak is represented by one component in the mixture model. However, comparing the results at high-frequency changing environments (Scenarios 1 and 2) with the ones obtained with the low frequency changing environments (Scenarios 5 and 6), we can detect a less prominent performance in low frequency scenarios, indicating that, once converged, the EDA_{OGMM} reduces its exploration power. So, a continued control to avoid premature convergence is desirable.

Next steps of the research are: to incorporate a continued convergence control mechanism, to compare EDA_{OGMM} with other algorithms designed to deal with dynamic environments, to increment the experimental tests aiming at investigating scalability and other aspects related to the relative performance of the proposed algorithm, and to perform a parameter sensitivity analisys.

Acknowledgment

This work was supported by grants from the So Paulo Research Foundation – FAPESP, process number 2009/06757-0, and by the Brazilian National Research Council – CNPq.

References

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

[2] R. Tinós and S. Yang, "A self-organizing random immigrants genetic algorithm for dynamic optimization problems," *Genetic Programming and Evolvable Machines*, vol. 8, no. 3, pp. 255–286, 2007.

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

[4] R. Mendes, J. Kennedy, and J. Neves, "The fully informed particle swarm: simpler, maybe better," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 3, pp. 204–210, June 2004.

[5] X. Li, J. Branke, and T. Blackwell, "Particle swarm with speciation and adaptation in dynamic environments," in *Proceedings of the 8th Genetic and Evolutionary Computation Conference*, ser. GECCO '06. New York, USA: ACM, 2006, pp. 51–58.

[6] F. de França, F. von Zuben, and L. de Castro, "An artificial immune network for multimodal function optimization on dynamic environments," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM, 2005, p. 296.

[7] F. de França and F. Von Zuben, "A dynamic artificial immune algorithm applied to challenging benchmarking problems," in *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, ser. CEC'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 423–430.

[8] S. Yang and X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," *Soft Comp.*, vol. 9, no. 0, pp. 815–834, 2005.

[9] X. Liu, Y. Wu, and J. Ye, "An Improved Estimation of Distribution Algorithm in Dynamic Environments," in *Fourth International Conference on Natural Computation*. IEEE Computer Society, 2008, pp. 269–272.
[10] A. Gonçalves and F. Von Zuben, "Hybrid evolutionary algorithm guided by a fast adaptive gaussian mixture model applied to dynamic optimization problems," in *III Workshop on Computational Intelligence - Joint Conference*, 2010, pp. 553–558.
[11] P. Larrañaga and J. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Netherlands, 2002.
[12] P. Larrañaga, "A Review of Estimation of Distribution Algorithms," in *Estimation of Distribution Algorithms: A new tool for Evolutionary Computation*, P. Larrañaga, J.A. Lozano, Ed. Kluwer Academic Publishers, 2001.
[13] C. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. Springer, October 2007.
[14] R. Duda, P. Hart, and D. Stork, *Pattern classification*, 2nd ed. Wiley, November 2001.
[15] A. Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.
[16] S. Nowlan, "Soft competitive adaptation: neural network learning algorithms based on fitting statistical mixtures," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1991.
[17] R. Neal and G. Hinton, "A view of the EM algorithm that justifies incremental, sparse and other variants," in *Learning in Graphical Models*. Kluwer Academic Publishers, 1998, pp. 355–368.
[18] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Congress on Evolutionary Computation CEC99*, vol. 3, 1999, pp. 1875–1882.
[19] C. Li and S. Yang, "A generalized approach to construct benchmark problems for dynamic optimization," in *Proc. of the 7th Int. Conf. on Simulated Evolution and Learning*, 2008.
[20] C. Li, S. Yang, T. Nguyen, E. Yu, X. Yao, Y. Jin, H. Beyer, and P. Suganthan, "Benchmark Generator for CEC'2009 Competition on Dynamic Optimization," University of Leicester, Tech. Rep., 2008.
[21] B. Yuan, M. Orlowska, and S. Sadiq, "Extending a class of continuous estimation of distribution algorithms to dynamic problems," *Optimization Letters*, vol. 2, no. 3, pp. 433–443, 2008.
[22] H. Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," Naval Research Laboratory, Tech. Rep., 1990.