Soft Computing manuscript No.
(will be inserted by the editor)

Learning behavior in abstract memory schemes for dynamic optimization problems

Hendrik Richter · Shengxiang Yang

Received: 13 August, 2008 / Accepted: 13 March 2009

Abstract Integrating memory into evolutionary algorithms is one major approach to enhance their performance in dynamic environments. An abstract memory scheme has been recently developed for evolutionary algorithms in dynamic environments, where the abstraction of good solutions is stored in the memory instead of good solutions themselves to improve future problem solving. This paper further investigates this abstract memory with a focus on understanding the relationship between learning and memory, which is an important but poorly studied issue for evolutionary algorithms in dynamic environments. The experimental study shows that the abstract memory scheme enables learning processes and hence efficiently improves the performance of evolutionary algorithms in dynamic environments.

Keywords Evolutionary Algorithm · Dynamic Optimization Problem · Learning · Memory Dynamics

1 Introduction

A main concern of evolutionary algorithms (EAs) for solving dynamic optimization problems (DOPs) is to maintain the genetic diversity of the population [6,10, 16] in order to guarantee continuing and sustainable evolutionary search for optima that change with time. Next to standard adaptive schemes that are also used in solving static optimization problems, such as self–adaption of mutation [1,3], for achieving the maintenance of diversity, two main lines have been followed in solving DOPs. One line is to preserve diversity by mainly random means, which is realized by designs such as hyper–mutation [15] and random immigrants [20]. Another line is to promote diversity by basically deterministic methods through saving individuals or groups of individuals for future re–insertion or mergence. Such ideas are implemented in memory [4,11,21] and multi–population approaches [5].

Although both of the above concepts have shown to be successful for certain dynamic environments, there are some points of criticism. One is that they do not or do not explicitly incorporate information about the dynamics and hence do not discriminate between different kinds of dynamic fitness landscapes. Another concern is the usage of past and present solutions for improving the quality of future solution finding. This aspect is not addressed by random diversity enhancement at all. In contrast, memory techniques do use previous good solutions for later reuse [4,19,22,23]. Here, it is natural to ask how and why this brings improvements in performance and an obvious answer is that by storing and reusing information some kinds of learning processes take place. However, the detailed relationships between memory and learning in dynamic optimization are poorly studied. A related example is an analysis of how self–adaptive mutation steps reflect the movement of the optima [3], which can be regarded as an implicit learning process. However, we intend to study learning in a more literal sense as for instance considered in machine learning [13,14]. Therefore, we introduce a method for evaluating the memory dynamics and the

H. Richter
HTWK Leipzig, Fachbereich Elektrotechnik
und Informationstechnik
Institut Mess–, Steuerungs– und Regelungstechnik
Postfach 30 11 66, D–04125 Leipzig, Germany
E-mail: richter@fbeit.htwk-leipzig.de

S. Yang
Department of Computer Science
University of Leicester
University Road, Leicester LE1 7RH, United Kingdom
E-mail: s.yang@mcs.le.ac.uk

learning process based on an information–theoretical quantity, the Kullback–Leibler divergence.

This paper analyzes an abstraction based memory scheme for EAs for multimodal DOPs, which was recently proposed in [18]. In this abstract memory scheme, the abstraction of good solutions (i.e., to use their approximate location in the search space to deduce a probabilistic model for the spatial distribution of good solutions) is stored in the memory instead of good solutions themselves. We show that such a memory scheme enables learning processes conceptionally and functionally similar to those considered in machine learning. It explicitly uses the past and present solutions in an abstraction process that is employed to improve future problem solving and differentiates between different kinds of dynamics of the fitness landscape. In particular, we intend to study how learning takes place in the abstract memory scheme.

The rest of this paper is outlined as below. The next section reviews the relationship between memory and learning and links it to solving DOPs with a memory enhanced EA. The abstract memory scheme is given in Section 3, where we also show how it can be described by the memory dynamics. Experiments are reported and discussed in Section 4. Section 5 concludes the paper with discussions on future work.

2 Memory and Learning

When tackling similar problems repeatedly, it is natural to credit the long–term success in problem solving to learning in both its conceptual and metaphorical meaning. By long–term success we mean that the obtained results become increasingly better over time with respect to some performance criteria. Evolutionary optimization in dynamic fitness landscapes implies the repeated solution of a multi–modal optimization problem and hence meets this description. As we are using a memory scheme in the EA to improve its performance, it makes sense to ask about the relationships between memory and learning for DOPs.

In cognitive science, learning is understood as a change of behavior as a result of experience, while memory is a record of events leading to experience [12]. So, learning emphasizes acquiring experience with the aim of extracting information from past and current events likely to be useful in the future behavior, while memory emphasizes retaining experience with the function to carry it forward in time. A computer science example that is related to this study and applies these principles is the memory design of autonomous agents in artificial life for dynamic environments [9]. In machine learning, this matter can be formalized further by defining the learning problem as finding a mapping between inputs and outputs [14]. This mapping is constructed from a training set of past and current inputs and outputs and can be used to predict future outputs using future inputs alone. The quality of the mapping is evaluated by performance measures; the quality becoming better over training time is the learning process. In this way, the experiences in the cognitive science view roughly relate to the training set of inputs and outputs and the performance evaluation of the mapping between them in machine learning. Moreover, the memory in the former functionally corresponds to the mapping in the latter, which applies almost literally in machine learning using artificial neural networks. Our aim is to employ these concepts in the design of and the numerical experiments with the abstract memory scheme.

Memory schemes that only store good solutions as themselves, known as direct memory [4,19,21], for later reuse carry out learning processes implicitly at best. Learning is something different than memorizing all previous solutions. In cases, this might be helpful. In general, every realizable memory will soon prove insufficient in a more complex context; if not by the storing capacity itself, then by a timely retrieval of the stored content for further usage. In the wider sense discussed above, learning refers to detecting the essence and meaning of a solution.

The abstract memory scheme proposed in [18] intends to address and employ these relations. Abstraction means to select, evaluate and code information before storing. A good solution is evaluated with respect to physically meaningful criteria and in the result of this evaluation, storage is undertaken but no longer as the solution itself but as the information coded with respect to the criteria. So, abstraction means a threshold for and compression of information, e.g., see [8] which proposes similar ideas for reinforcement learning. The filling of the abstract memory takes place during the run–time of the EA in the dynamic fitness function. It gradually builds a mapping between the search space elements and good solutions. This mapping is constructed via the abstract memory. So, the scheme we present is not merely concerned with anticipating the dynamics of the fitness function alone, as considered in [2], but to predict where good solutions of the DOP are likely to occur. Hence, we bring together learning and memory for evolutionary optimization in dynamic environments.

3 The Abstract Memory Scheme

The main idea of the abstract memory scheme is that it does not store good solutions directly but as their abstraction. The abstraction of a good solution is based on

Algorithm 1 EA with the abstract memory scheme.

1: Set the grid size ϵ and upper and lower bounds $x_{i\,min}$ and $x_{i\,max}$, $i = 1, 2, \ldots, n$
2: Set $t := 0$ and IntitializePopulation($P(0)$)
3: Define the memory matrix $\mathcal{M} \in \mathbb{R}^{h_1 \times \ldots \times h_n}$ dimensions by $h_i = \left\lceil \frac{x_{i\,max} - x_{i\,min}}{\epsilon} \right\rceil$.
4: Reset the counters $count_{\ell_1 \ell_2 \ldots \ell_n}(0) := 0, \ell_i = 1, 2, \ldots, h_i$ of the memory matrix $\mathcal{M}(0)$
5: **repeat**
6: EvaluateFitness($P(t)$)
7: *// Perform Abstract Memory Storage*
8: Select best individuals $B(t)$ from $P(t)$ for abstract storage
9: **for** each selected individual $x_j \in B(t)$ **do**
10: Calculate its partition cell indices by $\ell_i = \left\lceil \frac{x_{i\,j} - x_{i\,min}}{\epsilon} \right\rceil, i = 1, 2, \ldots, n$
11: $count_{\ell_1 \ell_2 \ldots \ell_n}(t) := count_{\ell_1 \ell_2 \ldots \ell_n}(t) + 1$
12: **end for**
13: $P_{sel}(t) :=$ Select($P(t)$)
14: $P_{rec}(t) :=$ Recombinate($P_{sel}(t)$)
15: $P'(t) :=$ Mutate($P_{rec}(t)$)
16: *// Perform Abstract Memory Retrieval*
17: **if** an environmental change detected **then**
18: Calculate the matrix $\mathcal{M}_\mu(t) := \frac{1}{\sum_{h_i} \mathcal{M}(t)} \mathcal{M}(t)$
19: Set τ *// the number of individuals to generate*
20: Calculate the distribution of individuals per partition cell using the distribution $\mathcal{M}_\mu(t)$ and ensuring $\sum \lceil \mu_{\ell_1 \ell_2 \ldots \ell_n}(t) \cdot \tau \rceil + \sum \lfloor \mu_{\ell_1 \ell_2 \ldots \ell_n}(t) \cdot \tau \rfloor = \tau$
21: Randomly fix the exact position of each individual within each partition cell
22: Merge generated individuals with $P'(t)$
23: **end if**
24: $t := t + 1$
25: **until** a termination condition holds

its approximate location in the search space. Hence, we need to partition the relevant (bounded) search space into rectangular (hyper–) cells. Each cell can be addressed by an element of a matrix. Hence, for an n–dimensional search space M we obtain an n–dimensional matrix, whose elements represent the search space sub–spaces. This matrix acts as our abstract memory, called the memory matrix, and is meant to represent the spatial distribution of good solutions.

The EA we use has a real number representation of λ individuals $x_j \in \mathbb{R}^n$, $j = 1, 2, \ldots, \lambda$, which form the population $P(t) \in \mathbb{R}^{n \times \lambda}$ at generation $t \in \mathbb{N}_0$. The pseudo–code of the EA with the abstract memory scheme is briefly outlined in Algorithm 1. The storage, retrieval, and dynamics of the abstract memory are described in the following sections respectively.

3.1 Abstract Memory Storage

The abstract memory storage process consists of two steps, a selecting process and a memorizing process. The selecting process picks the best individuals from the population $P(t)$ while the EA runs. In terms of the run–time between changes only the best over the run–time or the best over a few generations before a change occurs could be taken. We define the number of the individuals selected for memorizing as well as the number of generations where memorizing is carried out.

In the memorizing process, the selected individuals are sorted according to the partition in the search space they represent. In order to obtain this partition, we assume that the search space M is bounded in each direction by $[\, x_{i\,min}\, ,\, x_{i\,max}\,]$, $i = 1, 2, \ldots, n$. With the grid size ϵ, we obtain for every generation t the memory matrix $\mathcal{M}(t) \in \mathbb{R}^{h_1 \times h_2 \times \ldots \times h_n}$, where $h_i = \lceil \frac{x_{i\,max} - x_{i\,min}}{\epsilon} \rceil$. In $\mathcal{M}(t)$, each element $m_{\ell_1 \ell_2 \ldots \ell_n}(t)$ is a counter $count_{\ell_1 \ell_2 \ldots \ell_n}(t)$, $\ell_i = 1, 2, \ldots, h_i$, which is empty initially, i.e., $count_{\ell_1 \ell_2 \ldots \ell_n}(0) = 0$ for all ℓ_i. For each individual $x_j(t) \in P(t)$ selected to take part in the memorizing, the counter of the element representing the partition cell that the individual belongs to is increased by one. That is, we calculate the index $\ell_i = \lceil \frac{x_{i\,j} - x_{i\,min}}{\epsilon} \rceil$ for all $x_j = (x_{1j}, x_{2j}, \ldots, x_{nj})^T$ and all $1 \leq i \leq n$ and increment the corresponding $count_{\ell_1 \ell_2 \ldots \ell_n}(t)$. Note that this process might be carried out several times in a generation t if more than one individual selected belongs to the same partition. The abstraction storage process retains the abstraction of good solutions by accumulating locations where they occurred. In this way, we encode and compress the information about good solutions.

3.2 Abstract Memory Retrieval

After a change is detected, the abstract memory is retrieved as follows. First, an adjunctive memory matrix $\mathcal{M}_\mu(t)$ is calculated by dividing $\mathcal{M}(t)$ by the sum of all elements in $\mathcal{M}(t)$, i.e., $\mathcal{M}_\mu(t) = \frac{1}{\sum_{h_i} \mathcal{M}(t)} \mathcal{M}(t)$. Hence, each element in $\mathcal{M}_\mu(t)$ is an approximation of the natural measure $\mu \in [0, 1]$ of a good solution belonging to the corresponding partition cell $M_{\ell_1 \ell_2 \ldots \ell_n}$ of the search space. This natural measure can be viewed as the probability of the occurrence of a good solution within the partition over time. Hence, $\mathcal{M}_\mu(t)$ can be regarded as a mapping between a search space cell and the probability of a good solution within the cell at time $t+1$. It hence allows the dynamic prediction of good solutions which is employed in the retrieval process.

Next, we fix the number of individuals to be generated by τ ($1 \leq \tau \leq \lambda$) and generate these individuals randomly such that their statistical distribution regarding the partition matches that stored in the memory $\mathcal{M}_\mu(t)$. This is done as follows. We first determine the number of individuals to be created for each cell by sorting $\mu_{\ell_1 \ell_2 \ldots \ell_n}(t)$ in the decreasing order and set the number $\lceil \mu_{\ell_1 \ell_2 \ldots \ell_n}(t) \cdot \tau \rceil$ of new individuals for high

values of μ and the number $\lfloor \mu_{\ell_1 \ell_2 \ldots \ell_n}(t) \cdot \tau \rfloor$ for low values respectively. The rounding needs to ensure that $\sum \lceil \mu_{\ell_1 \ell_2 \ldots \ell_n}(t) \cdot \tau \rceil + \sum \lfloor \mu_{\ell_1 \ell_2 \ldots \ell_n}(t) \cdot \tau \rfloor = \tau$. Then, we fix the positions of the new individuals uniformly randomly within each partition cell $M_{\ell_1 \ell_2 \ldots \ell_n}$. This means the τ individuals are distributed such that the number within each cell approximates the probability of the occurrence of good solutions. These individuals are inserted in the population $P(t)$ after mutation has been carried out.

This abstract retrieval process can create an arbitrary number of individuals from the abstract memory. In the implementation considered here we upper bound this creation by the number of individuals in the population. As the abstract storage can be regarded as encoding and compression of information about good solutions in the search space, the abstract retrieval becomes decoding and expansion.

3.3 Abstract Memory Dynamics

Using the scheme described above leads to a considerable reduction of the information content to be processed by the memory, which is typical for abstraction. The storing capacity needed depends on the coarseness of the partitioning, but not on the number of individuals taken to the memory. This also means that the number of individuals that take part in the memorizing and the number of individuals that are retrieved from the memory and inserted in the population are completely independent of each other. Also, in the memory matrix not the good solutions are stored but the events of occurrence of the solution at a specific location in the search space. This means a change of representation (EA uses real, memory uses integer), which is another feature of abstraction. Such a change of representation requires less storage capacity and is particularly interesting for higher–dimensional search spaces. As the memory matrix $\mathcal{M}_\mu(t)$ is filled over the run–time t, learning as discussed in Section 2 takes place and can be quantified by studying the relationship between the performance and the matrix filling.

To compare the memory dynamics to a reference, we introduce a master memory (or demon) $\mathcal{D}_\mu(t)$, which has the elements $\delta_{\ell_1 \ell_2 \ldots \ell_n}(t)$. It is a matrix of the same dimension and size as $\mathcal{M}_\mu(t)$ and is built exactly the same way with the difference being that the solution trajectory is stored in $\mathcal{D}_\mu(t)$. Hence, it is a probabilistic mapping between search space cells and the solution of the DOP. With $\mathcal{M}_\mu(t)$ and $\mathcal{D}_\mu(t)$, we have two spatial probability distributions which represent the online calculated memory and the solution. By measuring the degree of the difference between these quantities, we have a way to establish how good the memory is and to evaluate the memory dynamics. Such a difference measure is the Kullback–Leibler divergence (KLD), e.g., see [7], p. 19:

$$KLD(t) = \sum_{\substack{i \\ \ell_i}} \delta_{\ell_i}(t)\, log_2 \left(\frac{\delta_{\ell_i}(t)}{\mu_{\ell_i}(t)} \right), \tag{1}$$

where the measures $\delta_{\ell_i}(t)$ and $\mu_{\ell_i}(t)$ are the elements of $\mathcal{D}_\mu(t)$ and $\mathcal{M}_\mu(t)$, respectively. In the following, we report numerical experiments with the abstract memory scheme and study its learning behavior. Therefore, we will particularly look at the memory dynamics.

4 Experimental study

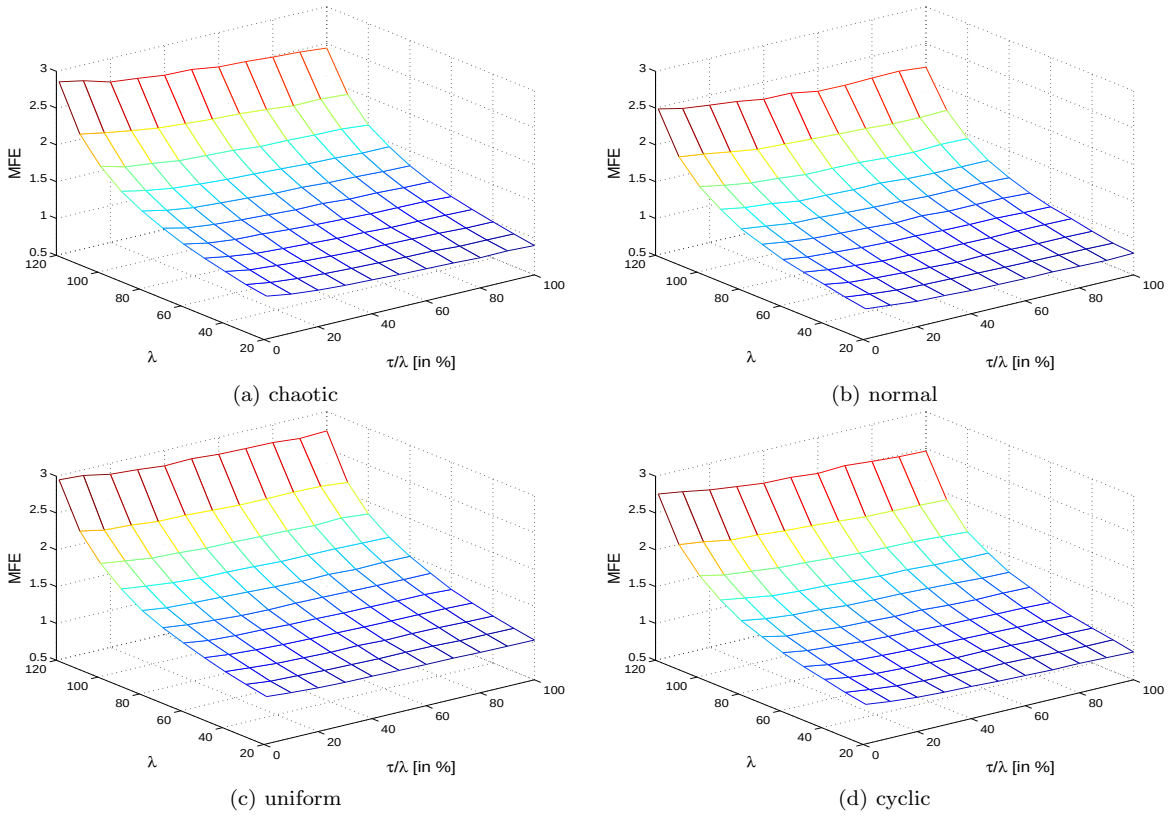
4.1 Experimental setup and performance measurement

The experimental results given here are obtained with an EA that uses the tournament selection of tournament size 2, the fitness–related intermediate sexual recombination (which is operated λ times and for each recombination two individuals are chosen randomly to produce an offspring that is the fitness–weighted arithmetic mean of both parents), a standard mutation with the mutation rate 0.1, and the proposed abstract memory (AM) scheme. The dynamic fitness landscape is an n–dimensional "field of cones on a zero plane", where N cones with coordinates $c_i(k)$, $i = 1, \cdots, N$, are moving with discrete time $k \in \mathbb{N}_0$. These cones are distributed across the landscape and have randomly chosen initial coordinates $c_i(0)$, heights h_i, and slopes s_i. So, the dynamic fitness function is given as:

$$f(x,k) = \max \left\{ 0\,,\ \max_{1 \le i \le N} \left[h_i - s_i \| x - c_i(k) \| \right] \right\}. \tag{2}$$

We study four types of dynamics regarding the coordinates $c_i(k)$ of the cones: (i.) chaotic dynamics generated by the Hénon map, see [17] for details of the generation process, (ii.) random dynamics with each $c_i(k)$ for each k being an independent realization of a normally distributed random variable, (iii.) random dynamics as in (ii.) but with a uniformly distributed random variable, and (iv.) cyclic dynamics where each $c_i(k)$ is consequently forming a circle.

We consider the dynamic fitness function (2) with dimension $n = 2$ and the number of cones $N = 7$. The upper and lower bounds of the search space are set to $x_{1\,min} = x_{2\,min} = -3$ and $x_{1\,max} = x_{2\,max} = 3$. The best three individuals of the population take part in the memorizing process for all three generations before a change in the environment occurs. Further, dynamic severity is normalized for all considered dynamics and hence has no differentiating influence. The scales t and



(a) chaotic (b) normal

(c) uniform (d) cyclic

Fig. 1 The MFE against the population size λ and the number of individuals retrieved from the memory τ, given as percentage τ/λ in %.

k are related by the change frequency $\gamma \in \mathbb{N}$ as $t = \gamma k$. The performance of the algorithms is measured by the Mean Fitness Error (MFE), defined as below:

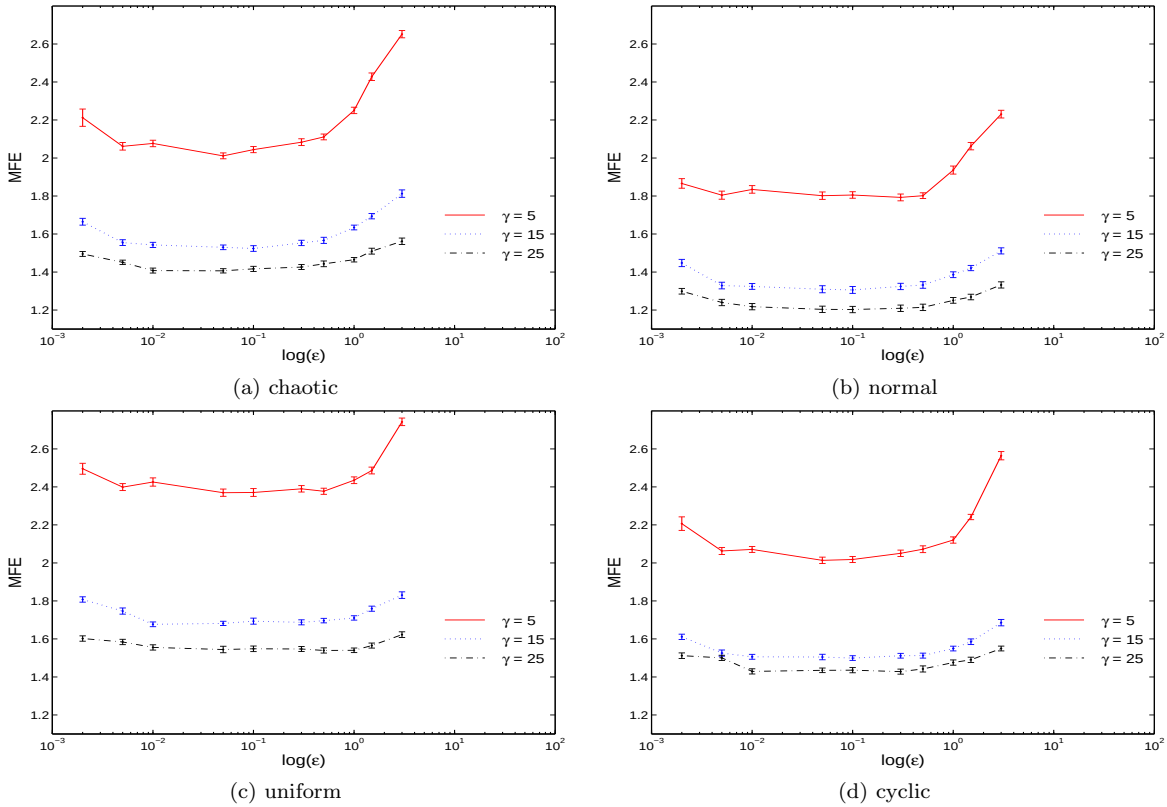
$$MFE = \frac{1}{R}\sum_{r=1}^{R}\left[\frac{1}{T}\sum_{t=1}^{T}\left(f(x_s(k),k) - \max_{x_j(t)\in P(t)} f(x_j(t),k)\right)\right]_{k=\lfloor\gamma^{-1}t\rfloor}, \quad (3)$$

where $\max_{x_j(t)\in P(t)} f\left(x_j(t), \lfloor\gamma^{-1}t\rfloor\right)$ is the fitness value of the best–in–generation individual $x_j(t) \in P(t)$ at generation t, $f\left(x_s(\lfloor\gamma^{-1}t\rfloor), \lfloor\gamma^{-1}t\rfloor\right)$ is the maximum fitness value at generation t, T is the number of generations used in the run, and R is the number of consecutive runs. We set $R = 50$ and $T = 2000$ in all experiments.

4.2 Properties of the abstract memory

The first set of experiments examines the relationships between the population size λ, the number of individuals τ retrieved from the memory, and the performance measure MFE. Fig. 1 shows the results for the fixed change frequency $\gamma = 15$ and the grid size $\epsilon = 0.1$. From Fig. 1, it can be observed that an exponential relationship exists between MFE and λ, which is typical for EAs. Along this general trend, the number of retrieved individuals, here given in percent of the total population, has only a small influence on the MFE, where in general a medium and large number gives slightly better results than a very small percentage.

Next, we look at the influence of the grid size ϵ on performance of the AM scheme, see Fig. 2. Here, the MFE is given over ϵ and different γ on a semi–logarithmic scale while we set here and subsequently $\lambda = 50$ and $\tau = 20$. For all types of dynamics and all change frequencies we obtain a kind of bath-tub curves, which indicates that an optimal grid size depends on the type of dynamics and the size of the bounded region in the search space where the memory is considered. This gives raise to the question of whether an adaptive grid size would increase the performance of the abstraction memory scheme. Also, it can be seen that a drop in performance is more significant if the grid is too large. For smaller grid the performance is not decreasing very dramatically, but the numerical effort for calculation with small grids becomes considerable. This result al-



(a) chaotic (b) normal

(c) uniform (d) cyclic

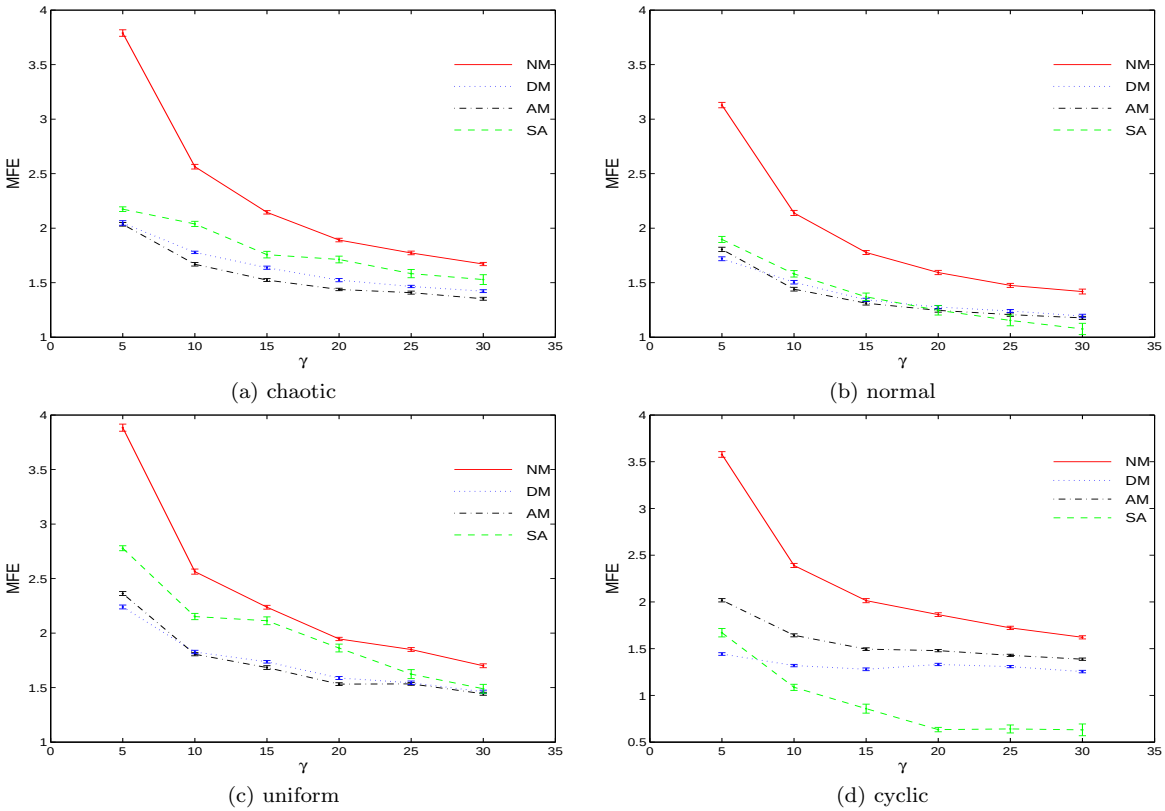
Fig. 2 Comparison of performance of abstraction memory scheme (AM) measured by the MFE for different grid size ϵ and different types of dynamics and $\gamma = 5$, $\gamma = 15$ and $\gamma = 25$.

lows us to choose an ϵ that compromises between the performance and the numerical effort. In the following experiments, we set $\epsilon = 0.1$.

In the second set of experiments, the abstract memory scheme (AM) is tested and compared with a direct memory scheme (DM) that stores good solutions and inserts them again in a retrieval process, an EA with no memory (NM) that uses hypermutation [15] with base mutation $\sim 0.1\mathcal{N}(0,1)$ and hypermutation $\sim 3\mathcal{N}(0,1)$, and an evolutionary strategy with self–adaptive mutation (SA) with 12 parents and 48 offspring candidates. Note that by these parameters, we have a comparable number of fitness function evaluations. In Fig. 3, the MFE over the change frequency γ for all four types of dynamics considered is given and the 95% confidence intervals are also given.

From Fig. 3, it can be seen that the memory schemes outperform the no memory scheme for all dynamics. This is particularly noticeable for small change frequencies γ and means that by memory the limit of γ for which the algorithm still performs reasonably can be considerably lowered. It can also be seen that the abstract memory gives better results than the direct memory for irregular dynamics, i.e., chaotic and random. For chaotic dynamics, this is even significant within the given bounds. For regular, cyclic dynamics, we find the contrary, with direct memory being better than abstract. A comparison to the self–adaptive scheme yields that memory schemes are better than self–adaption for chaotic and uniform random dynamics. For normal random dynamics, memory outperforms self–adaption for small change frequencies, while for large γ, for instance $\gamma = 25$ and $\gamma = 30$, it is the other way around. Finally, for circle dynamics, self–adaption is the best option yielding results far better than all other tested schemes.

However, in our experiments with the self–adaptive scheme we observed that for a small but existing percentage of runs the EA diverged and produced invalid results. These runs were not taken into account in the performance evaluation. A possible explanation for this behavior is that a self-adaptive mutation rate evolves towards optimal values in between changes, but may become ill–posed after the change. This leads in some cases to diverging population dynamics because there is no direct feedback between the population dynamics and mutation rate. Note that such a behavior was not observed with the other three schemes. The aim here, however, is not to argue that one scheme is supe-



(a) chaotic (b) normal

(c) uniform (d) cyclic

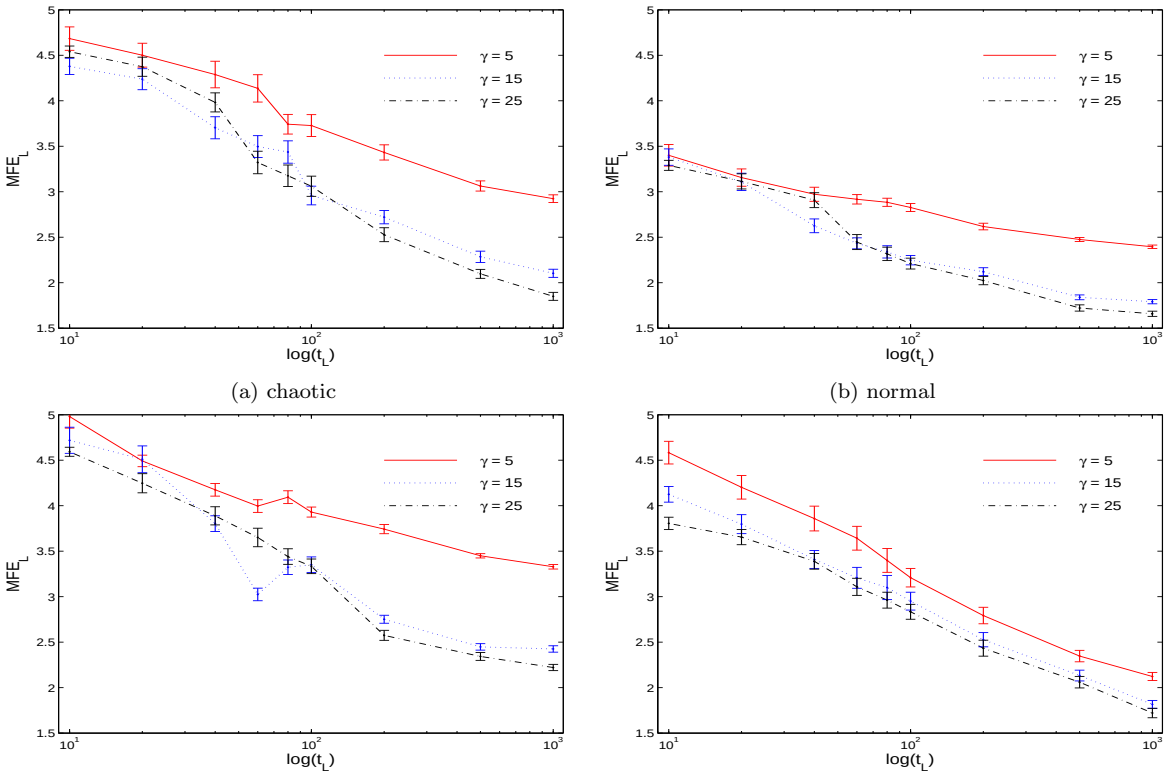
Fig. 3 Performance of the EA measured by the MFE over change frequency γ for different types of dynamics and no memory but hypermutation (NM), direct memory (DM), abstract memory (AM) and self–adaption (SA).

rior over another but to study the underlying working mechanisms and particularly the effect of learning. For self–adaption this has been done by analyzing the evolution of self–adaptive mutation steps depending on the dynamics of the fitness landscape [3], which can be regarded as an implicit learning process. Our approach to study learning is different, inspired by machine learning [14,13] and will be introduced and discussed next.

4.3 Learning behavior

To quantify learning depends on metrics for performance, which ideally shows improvement over time. For evaluating the effect of learning and obtaining the learning curve, the experiment has to enable learning for a certain time, then turn learning off and measure the performance using the learned ability [14]. Regarding the abstract memory scheme, learning takes place as long as the memory matrix $\mathcal{M}_\mu(t)$ is filled. This gives raise to the following measure for learning success. We define t_L to be the learning time. For $0 < t \leq t_L$ the matrix $\mathcal{M}_\mu(t)$ is filled as described in Section 3. For $t_L < t \leq t_L + T$ the storage process is discarded and only retrieval using the now fixed memory is carried out. We calculate the MFE in Eq. (3) for $t > t_L$ only and denote it MFE_L. It is a performance measure for the learning success, where MFE_L over t_L shows the learning curve.

Fig. 4 depicts the results for fixed $\lambda = 50$, $\tau = 20$ and several change frequencies γ on the semi–logarithmic scale. These learning curves are an experimental evaluation of the learning behavior. We see that the MFE_L gets gradually smaller with the learning time t_L becoming larger, which confirms the learning success. We find a negative linear relation between MFE and $log(t_L)$, which indicates an exponential dependency between t_L and MFE. Also, it can be seen that the learning curves are slightly steeper for larger change frequencies. An exception to this general trend is cyclic dynamics, where the learning curves are almost parallel for all γ and a large proportion of the tested t_L. A comparison of the learning success between the different kinds of landscape dynamics suggests that the uniform random movement is the most difficult to learn. The results in Fig. 4 clearly indicate the positive effect of learning on the performance of the EA.



(a) chaotic (b) normal

(c) uniform (d) cyclic

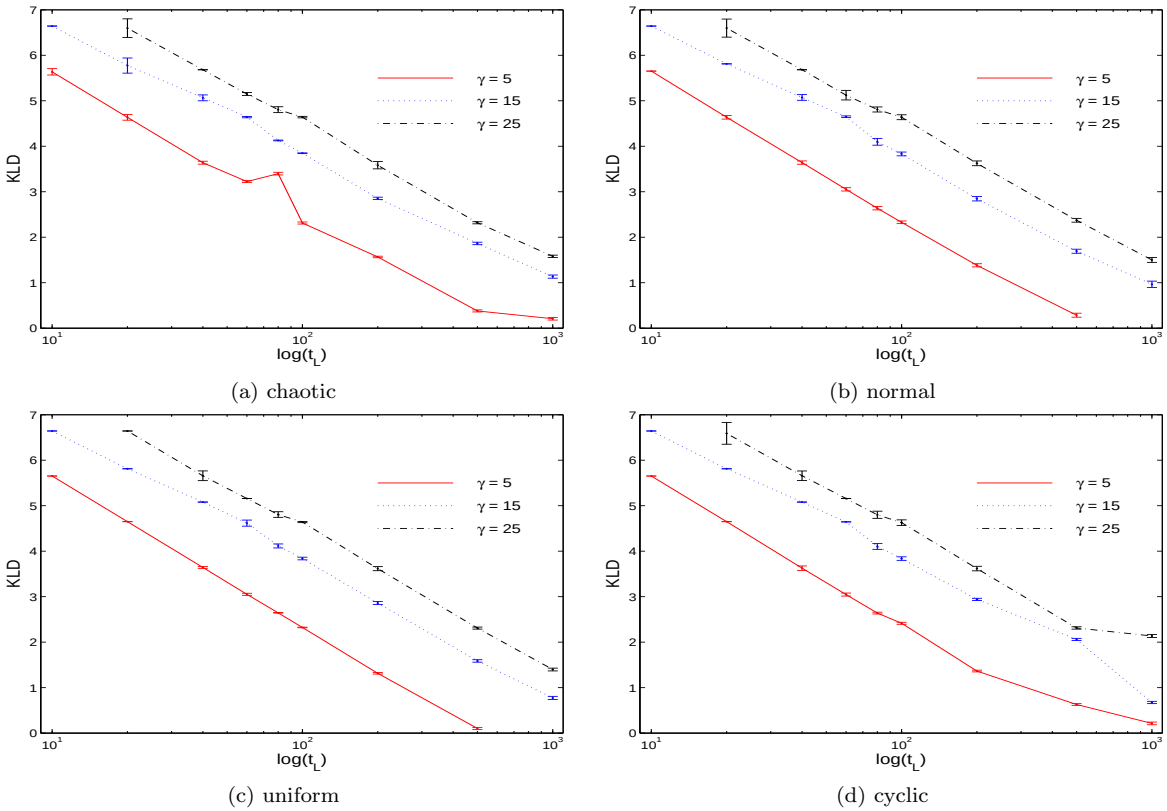
Fig. 4 Learning curves for the abstract memory scheme showing the learning success measured by MFE_L over the learning time t_L.

Next, we are interested in how the memory reflects the learning process. We consider the memory dynamics which can be quantified by the KLD in Eq. (1). The KLD for the learning time t_L, that is, $KLD = KLD(t_L)$, over t_L on the semi–logarithmic scale is plotted in Fig. 5. As the KLD may differ in every run, we record the mean over $R = 50$ runs and the 95% confidence intervals. The KLD measures the difference between the spatial probability distribution stored in the memory $\mathcal{M}_\mu(t_L)$ compared to the reference of the master memory (or demon) $\mathcal{D}_\mu(t_L)$ that stores the solution of the DOP for the learning time t_L. The KLD is a measure of the degree of similarity between the "true" distribution in $\mathcal{D}_\mu(t_L)$ and the "estimated" distribution in $\mathcal{M}_\mu(t_L)$; $KLD = 0$ defines that both distributions are equal. The results in Fig. 5 show that the memory gets gradually better with the learning time becoming larger, following similar characteristics as the learning curves. For a small γ, the KLD goes near zero, indicating that the distribution in the memory almost fits the distribution obtained for the solution of the DOP. The reason for this result lies most likely in that for a smaller γ, a larger variety of the landscape's dynamics is feeded to the memory for a constant learning time compared to a larger γ, which causes it to become better.

We finally relate the memory dynamics to the learning success. In Fig. 6, the relationship between the learning success MFE_L and the memory dynamics KLD is shown. Note that as both quantities are the result of numerical experiments, the means over $R = 50$ runs are recorded and we get vertical as well as horizontal confidence intervals. The general trend is that both quantities are directly proportional for a constant γ, which implies that a good memory results in a good performance of the EA. The most striking detail is that KLD is the smallest for the smallest change frequency, while this is not accompanied by the MFE_L being the smallest, too. One explanation is that the change frequency affects the performance much stronger than the quality of memory. In other words, a good memory does not guarantee for high performance if the EA does not have a certain run time between changes in the landscape.

5 Conclusions

This paper investigates an abstract memory scheme for EAs in dynamic environments, where memory is used to store the abstraction of good solutions (i.e., to use their approximate location in the search space to deduce a



(a) chaotic (b) normal

(c) uniform (d) cyclic

Fig. 5 Memory dynamics measured by the KLD over the learning time t_L.

probabilistic model for the spatial distribution of good solutions) instead of good solutions themselves. This abstraction is employed to generate solutions to improve future problem solving. In order to understand the relationship between memory and learning in dynamic environments, experiments were carried out to study how learning takes place in the abstract memory and how the performance changes over time for different kinds of dynamics in the fitness landscape. The experimental study revealed several results on the dynamic test environments. First, the abstraction based memory scheme enables learning processes, which efficiently improves the performance of EAs in dynamic environments. Second, the effect of the abstract memory on the performance of the EA depends on the learning time and the frequency of environmental changes.

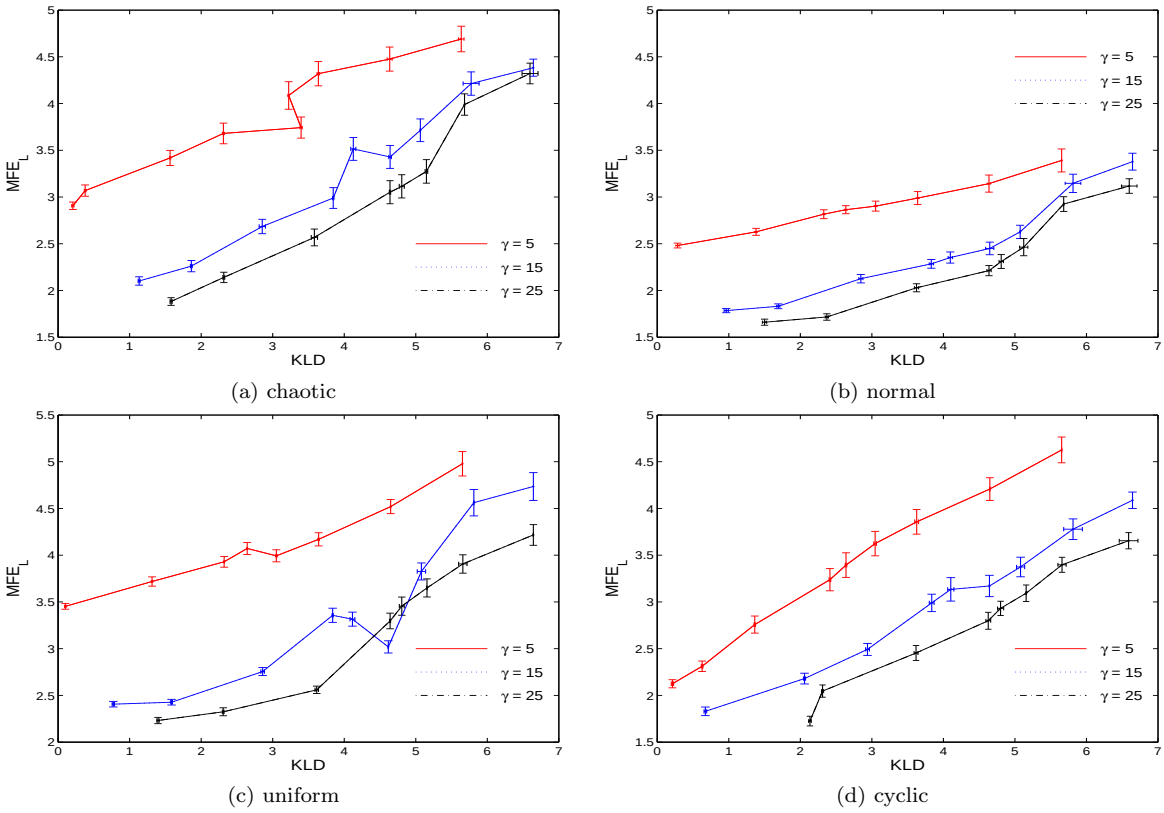
We studied the relationship between learning and the abstract memory in dynamic environments. For the future work, it is valuable to compare and combine the abstract memory scheme with other approaches developed for EAs in dynamic environments. Also, if the dynamics is non–stationary in a strict statistical sense, that is, the statistical properties change fast over the algorithm's run–time, as for instance in the translatory movements, then forecasting the movements requires other schemes, for instance prediction by a linear estimator. However, if the changes of statistical properties are rather slow, it might be helpful if the memory matrix has some evaporation to prevent unlimited accumulation of its elements. This would mean that in the storage process a third step is needed to add: an amnesia (or forgetting) process.

Acknowledgments

The work by S. Yang was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/1.

References

1. D. V. Arnold and H. G. Beyer. Optimum tracking with evolution strategies. *Evol. Comput.*, **14**(3): 291–308, 2006.
2. P. A. N. Bosman. Learning and anticipation in online dynamic optimization. In: S. Yang, Y. S. Ong, and Y. Jin (eds.), *Evolutionary Computation in Dynamic and Uncertain Environments*, Chapter 6, pp. 129–152, Springer-Verlag Berlin Heidelberg, 2007.



(a) chaotic (b) normal

(c) uniform (d) cyclic

Fig. 6 Relationship between learning success MFE_L and memory dynamics KLD.

3. A. M. Boumaza: Learning environment dynamics from self-adaptation. GECCO Workshops 2005: pp. 48–54, 2005.
4. J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In: *Proc. of the 1999 IEEE Congress on Evolutionary Computation*, pp. 1875–1882, 1999.
5. J. Branke, T. Kaußler, C. Schmidt and H. Schmeck. A multi-population approach to dynamic optimization problems. *Proc. of the 4th Int. Conf. on Adaptive Computing in Design and Manufacturing*, pp. 299–308, 2000.
6. J. Branke. *Evolutionary Optimization in Dynamic Environments*, Kluwer Academic Publishers, 2002.
7. T. M. Cover and J. A. Thomas. *Elements of Information Theory*, Wiley, Hoboken, NJ, 2006.
8. R. Fitch, B. Hengst, D. Suc, G. Calbert, and J. Scholz. Structural abstraction experiments in reinforcement learning. In: *AI 2005: Advances in Artificial Intelligence*, pp. 164–175, 2005.
9. W. C. Ho, C. Nehaniv, and K. Dautenhahn. Autobiographic agents in dynamic virtual environments - performance comparison for different memory control architectures. In: *Proc. of the 2005 IEEE Congress on Evol. Comput.*, pp. 573–580, 2005.
10. Y. Jin and J. Branke. Evolutionary optimization in uncertain environments - a survey. *IEEE Trans. on Evol. Comput.*, **9**(3): 303–317, 2005.
11. E. H. J. Lewis and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In: *Parallel Problem Solving from Nature–PPSN V*, pp. 139–148, 1998.
12. D. A. Lieberman. *Learning and Memory: An Integrative Approach*, Wadsworth, Belmont, CA, 2004.
13. R. S. Michalski. Learnable evolution model: Evolutionary processes guided by machine learning. *Machine Learning*, **38**(1): 9–40, 2000.
14. T. M. Mitchell. *Machine Learning*, McGraw–Hill, New York, 1997.
15. R. W. Morrison and K. A. De Jong. Triggered hypermutation revisited. In: *Proc. of the 2000 IEEE Congress on Evol. Comput.*, pp. 1025–1032, 2000.
16. R. W. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*, Springer, Berlin, 2004.
17. H. Richter. A study of dynamic severity in chaotic fitness landscapes. In: *Proc. of the 2005 IEEE Congress on Evol. Comput.*, pp. 2824–2831, 2005.
18. H. Richter and S. Yang. Memory based on abstraction for dynamic fitness functions. In: *EvoWorkshops 2008: Applications of Evolutionary Computing*, LNCS 4974, pp. 597–606, 2008.
19. A. Simões and E. Costa. Variable-size memory evolutionary algorithm to deal with dynamic environments. In: *EvoWorkshops 2007: Applications of Evolutionary Computing*, LNCS 4448, pp. 617–626, 2007.
20. R. Tinós and S. Yang. A self-organizing random immigrants genetic algorithm for dynamic optimization problems. *Genetic Programming and Evolvable Machines*, **8**(3): 255–286, 2007.
21. S. Yang. Population-based incremental learning with memory scheme for changing environments. *Proc. of the 2005 Genetic and Evol. Comput. Conf.*, vol. 1, pp. 711–718, 2005.
22. S. Yang. Associative memory scheme for genetic algorithms in dynamic environments. In: *EvoWorkshops 2006: Applications of Evolutionary Computing*, LNCS 3907, pp. 788–799, 2006.

23. S. Yang and X. Yao. Population-based incremental learning with associative memory for dynamic environments. *IEEE Trans. on Evol. Comput.*, **12**(5): 542–561, October 2008.