Learning in Abstract Memory Schemes for Dynamic Optimization

Hendrik Richter
HTWK Leipzig, Fachbereich Elektrotechnik und Informationstechnik,
Institut Mess–, Steuerungs– und Regelungstechnik, D–04125 Leipzig, Germany
richter@fbeit.htwk-leipzig.de

Shengxiang Yang
Department of Computer Science
University of Leicester, University Road, Leicester LE1 7RH, United Kingdom
s.yang@mcs.le.ac.uk

Abstract

We investigate an abstraction based memory scheme for evolutionary algorithms in dynamic environments. In this scheme, the abstraction of good solutions (i.e., their approximate location in the search space) is stored in the memory instead of good solutions themselves and is employed to improve future problem solving. In particular, this paper shows how learning takes place in the abstract memory scheme and how the performance in problem solving changes over time for different kinds of dynamics in the fitness landscape. The experiments show that the abstract memory enables learning processes and efficiently improves the performance of evolutionary algorithms in dynamic environments.

1 Introduction

A main concern of evolutionary algorithms (EAs) for dynamic optimization problems (DOPs) is to maintain the genetic diversity of the population [5, 7]. Only this will guarantee continuing and sustainable evolutionary search for optima that change with time. To achieve the maintenance of diversity, two main concepts have been applied. One is to preserve diversity by mainly random means, which is realized by designs such as hyper–mutation [9] and random immigrants [11]. Another is to promote diversity by basically deterministic methods through saving individuals or groups of individuals for future reinsertion or merger. Such ideas are implemented in memory [3, 8] and multi–population approaches [4].

Although both 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. A second 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. In contrast, memory techniques do use previous good solutions. This is done in implicit memory schemes by redundant representation using multiploidy and dominance [8]. Explicit memory stores good solutions (sometimes accompanied by information about environmental conditions) for later reuse [3, 12]. 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 are carried out. However, the detailed relationships between memory and learning in dynamic optimization are poorly studied.

Memory schemes that only store good solutions as themselves, known as direct memory [3], for later reinsertion carry out learning processes implicitly at best. Learning is something different than memorizing all previous solutions. In parts, this might be helpful. In general, every realizable memory will soon prove insufficient in a more complex context; if not by storing capacity itself, then by a timely retrieval of the stored content for further usage. In the more wider sense discussed above, learning refers to detecting the essence and meaning of a solution. The abstract memory scheme proposed here, which is detailed in Sec. 3, 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 coded with respect to the criteria. So, abstraction means a threshold for and compression of information, see e.g. [6] which proposes similar ideas for reinforcement learning. So, the

978-0-7695-3304-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ICNC.2008.110

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 dynamic optimization problem are likely to occur. Hence, we bring together learning and memory for evolutionary optimization in dynamic environments.

This paper presents an abstraction based memory scheme for dynamic optimization. We show that such a memory scheme enables learning processes conceptionally and functionally similar to those considered in machine learning. It explicitly uses 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 memory scheme. Therefore, we consider how the performance in problem solving changes over time for different kinds of dynamics in the fitness landscape, which are regular, chaotic, and random. The rest of this paper is outlined as below. Sec. 2 states the DOP and the EA to solve it. The abstract memory scheme is given in Sec. 3. Experiments are reported and discussed in Sec. 4. Sec. 5 concludes the paper with discussions on future work.

2 Solving the DOP

Learning behavior in abstract memory schemes occurs in solving a DOP, which in this study is posed as follows. We employ as dynamic fitness landscape an n–dimensional "field of cones on a zero plane", where N cones with coordinates $c_i(k)$, $i = 1, 2, ..., N$ are moving with discrete time $k \in \mathbb{N}_0$. These cones are distributed across the landscape and have randomly chosen initial coordinates $c(0)$, heights h_i and slopes s_i. So, we will employ the dynamic fitness function

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

The EA we use has a real number representation and λ individuals $x_j \in \mathbb{R}^n$, $j = 1, 2, \ldots, \lambda$, which build the population $P \in \mathbb{R}^{n\times\lambda}$. Its dynamics is described by the generation transition function $\psi : \mathbb{R}^{n\times\lambda} \to \mathbb{R}^{n\times\lambda}$, see e.g. [1], p.64–65. It can be interpreted as a nonlinear probabilistic dynamical system that maps $P(t)$ onto $P(t+1)$ by the standard genetic operators selection, recombination and mutation as well as by using an abstract memory, which is described in detail in Sec. 3. It hence transforms a population at generation $t \in \mathbb{N}_0$ into a population at generation $t+1$, $P(t+1) = \psi\left(P(t)\right), t \ge 0$. Starting from an initial population $P(0)$, the population sequence $P(0)$, $P(1)$, $P(2)$, ... describes the temporal movement of the population in the search space. Both the time scales t and k are related by the change frequency $\gamma \in \mathbb{N}$ as

$$t = \gamma k. \tag{2}$$

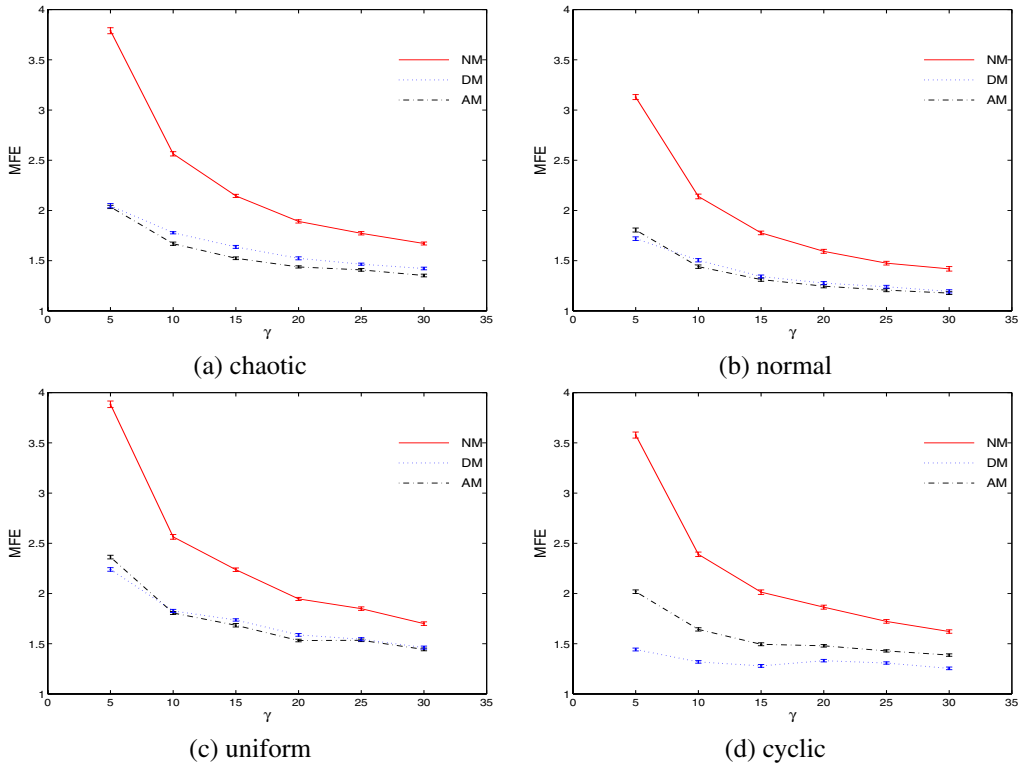
For $\gamma = 1$, apparently, the dynamic fitness function is changing every generation. For $\gamma > 1$, the fitness function changes every γ generations. The change frequency is an important quantity in dynamic optimization and will be the subject of the experimental studies reported in Sec. 4.

3 The Abstract Memory Scheme

The main idea of the abstract memory scheme is that it does not store good solutions as themselves but as their abstraction. We define an abstraction of a good solution to be its approximate location in the search space. Hence, we need to partition the search space. This can be obtained by partitioning the relevant (bounded) search space into rectangular (hyper–) cells. Every cell can be addressed by an element of a matrix. So, for an n–dimensional search space M we obtain an n–dimensional matrix whose elements represent search space sub–spaces. This matrix acts as our abstract memory and will be called memory matrix $\mathcal{M}$. It is meant to represent the spatial distribution of good solutions.

The abstract storage process consists of two steps, a selecting process and a memorizing process. The selecting process picks good individuals from the population $P(t)$ while the EA runs. In general, selection has to be done in terms of (i.) the amount and choice of considered individuals (ideally sorted according to their fitness) from the population and (ii.) points in the run–time of the EA (ideally sorted according to changes in the environment). For the individuals, either only the best or a few best from the population could be used. In terms of the run–time between changes only the best over 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 their partition in the search space which they represent. In order to obtain this partition, we assume that the search space M is bounded and in every direction there are lower and upper bounds, $x_{i\,min}$ and $x_{i\,max}$, $i = 1, 2, \ldots, n$. 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 the memory $\mathcal{M}(t)$ the entry of 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 for initialization, that is, $count_{\ell_1\ell_2\ldots\ell_n}(0) = 0$ 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 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 $x_j = (x_{1j}, x_{2j}, \ldots, x_{nj})^T$



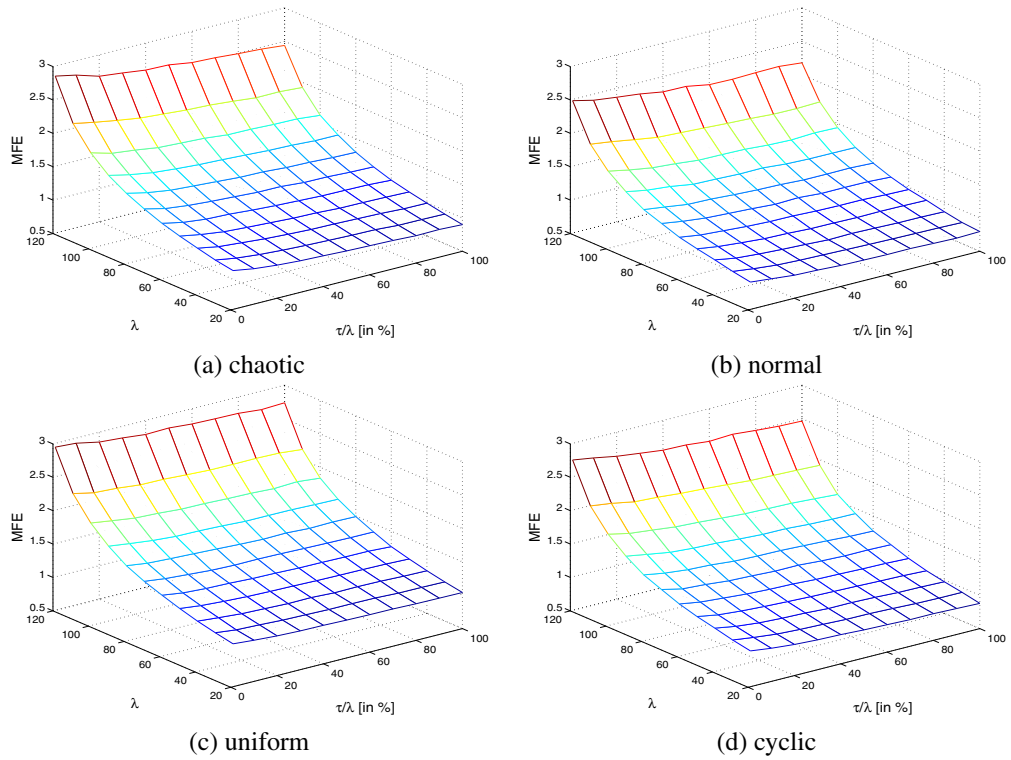
(a) chaotic (b) normal

(c) uniform (d) cyclic

Figure 1. 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) and abstract memory (AM).

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 the good solution occurred. In this way, we encode and compress the information about good solutions. As the matrix $\mathcal{M}$ is filled over run–time, the memorizing incorporates a learning process. After a change has been detected (usually if the sliding mean of the best individual is falling), the abstract retrieval process is carried out. It consists of two steps. First, a matrix $\mathcal{M}_\mu(t)$ is calculated by dividing the matrix $\mathcal{M}(t)$ by the sum of all matrix elements, that is $\mathcal{M}_\mu(t) = \frac{1}{\sum_{h_i}\mathcal{M}(t)}\mathcal{M}(t)$. Hence, the sum of all elements $\mu_{\ell_1\ell_2\ldots\ell_n}(t)$ in $\mathcal{M}_\mu(t)$ adds up to one. Each element in $\mathcal{M}_\mu(t)$ contains an approximation of the natural measure $\mu_{\ell_i} \in [0,1]$ belonging to the corresponding partition cell $M_{\ell_1\ell_2\ldots\ell_n}$ of the search space M. This natural measure can be viewed as the probability of the occurrence of a good solution within the partition over time of the dynamic environment. Hence, it is a probabilistic mapping between search space cells and the expected value for a good solution within each cell.

Next, we fix the number of individuals to be created by τ, $1 \leq \tau \leq \lambda$ and create these individuals randomly such that their statistical distribution regarding the partition matches that stored in the memory $\mathcal{M}_\mu(t)$. Therefore, we first determine the number of individuals to be created for each cell by sorting the $\mu_{\ell_1\ell_2\ldots\ell_n}(t)$ according to their magnitude 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 by taking realizations of a random variable uniformly distributed 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, while the exact position within partition cells is random. 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



(a) chaotic (b) normal

(c) uniform (d) cyclic

Figure 2. The MFE depending on the population size λ and the number of individuals retrieved from the memory τ, given as percentage τ/λ in $\%$.

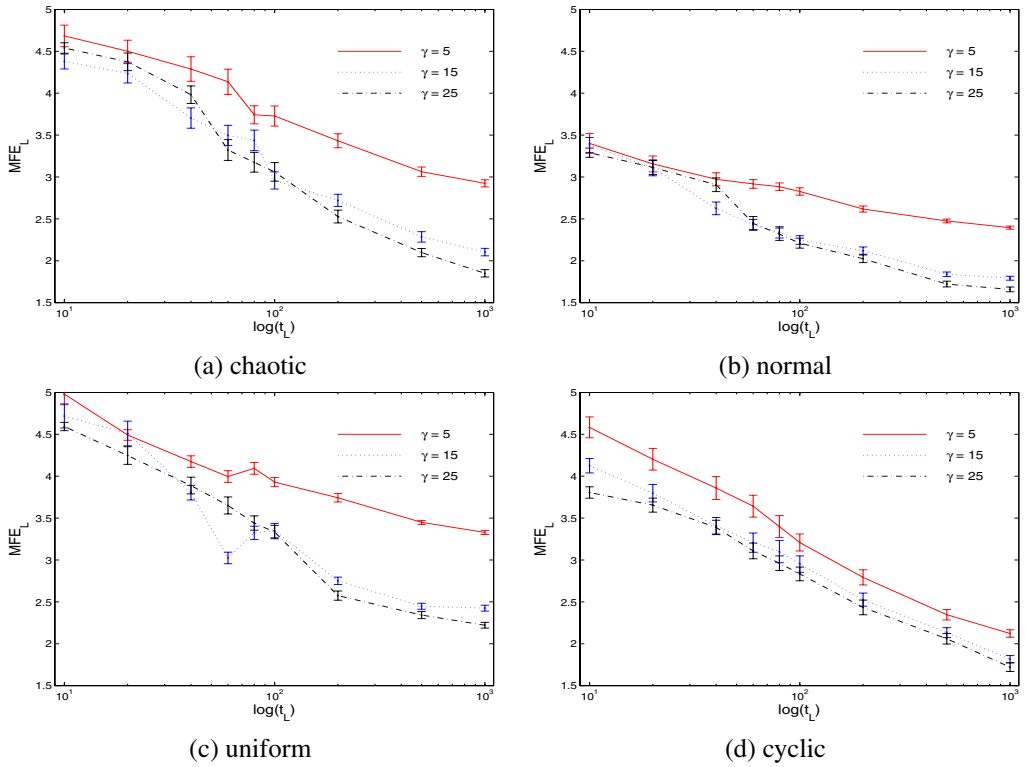
memory. In the implementation considered here we upper bound this creation by the number of individuals in the population. As abstract storage can be regarded as encoding and compression of information about good solutions in search space, abstract retrieval becomes decoding and expansion.

4 Numerical Experiments

The numerical results given here are obtained with an EA that uses tournament selection of tournament size 2, fitness–related intermediate sexual recombination, a mutation operator with base mutation rate 0.1 and the proposed abstraction memory scheme. We measure performance by the Mean Fitness Error $MFE = \frac{1}{R}\sum_{r=1}^{R}\left[\frac{1}{T}\sum_{t=1}^{T}\left(f_{best}(k) - f_{max}(k)\right)\right]_{k=\lfloor\gamma^{-1}t\rfloor}$ where $f_{best}(k) = \max_{x_j(t)\in P(t)} f\left(x_j(t), \lfloor\gamma^{-1}t\rfloor\right)$ is the fitness value of the best–in–generation individual $x_j(t) \in P(t)$ at generation t, $f_{max}(k) = f\left(x_s(\lfloor\gamma^{-1}t\rfloor), \lfloor\gamma^{-1}t\rfloor\right)$ 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. Note that $f\left(x_s, \lfloor\gamma^{-1}t\rfloor\right)$ and $\max_{x_j(t)\in P(t)} f\left(x_j(t), \lfloor\gamma^{-1}t\rfloor\right)$ change every γ generations according to Eq. (2).

The parameters in all experiments are $R = 50$ and $T = 2000$. We consider the dynamic fitness function (1) with dimension $n = 2$ and the number of cones $N = 7$. We study four types of dynamics of the coordinates $c(k)$ of the cones; (i.) chaotic dynamics generated by the Hénon map, see [10] for details of the generation process, (ii.) random dynamics with $c(k)$ being realizations of a normally distributed random variable, (iii.) random dynamics with $c(k)$ being realizations of a uniformly distributed random variable, and (iv.) cyclic dynamics where $c(k)$ are consequently forming a circle. As the dynamic severity is an important factor in dynamic optimization, severity is normalized for all considered dynamics and hence has no differentiating influence.

In a first set of experiments, the abstract memory scheme (AM) is tested and compared with a direct memory scheme (DM), which stores the good solutions and inserts them again in a retrieval process, and with an EA with no memory (NM), that uses hypermutation with the hypermutation rate set to 30, see Figure 1. Here, as well as in the



(a) chaotic (b) normal

(c) uniform (d) cyclic

Figure 3. Learning curves for the abstract memory scheme showing the learning success measured by MFE_L over learning time t_L.

other experiments we fixed the upper and lower bounds of the search space at $x_{1\,min} = x_{2\,min} = -3$ and at $x_{1\,max} = x_{2\,max} = 3$. 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, we set the grid size to $\epsilon = 0.1$. We used a fixed population size of $\lambda = 50$ and inserted $\tau = 20$ individuals in the retrieval process for one generation after the change. In Figure 1 we give the MFE over the change frequency γ for all four types of dynamics considered. Also, the 95% confidence intervals are given. We observe 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. Also, it can be seen that the abstract memory gives better results than the direct memory for irregular dynamics, that is, 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 second set of experiments examines the relationship between the populations size λ, the number of retrieved individuals τ and the performance measure MFE, see Figure 2, which shows results for the fixed change frequency $\gamma = 15$. We see an exponential relationship 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.

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. Regarding the abstract memory scheme considered here, 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 Sec. 3. For $t_L < t < T$ the storage process is discarded and only retrieval from the now fixed memory is carried out. We calculate the MFE 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. Figure 3 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 MFE and t_L. 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. To summarize, the results in Figure 3 clearly indicate the positive effect of learning on the performance of the EA.

5 Conclusions

This paper investigates an abstract memory scheme for evolutionary algorithms in dynamic environments, where memory is used to store the abstraction of good solutions (i.e., their approximate location in the search space) instead of good solutions themselves. This abstraction is employed to generate solutions to improve future problem solving. In order to understand the relationships between memory and learning in dynamic environments, experiments were carried out to investigate how learning takes place in the abstract memory scheme and how the performance changes over time for different kinds of dynamics in the fitness landscape. The experimental study reveals several results on the dynamic test environments. First, it is usually beneficial to integrate memory schemes into EAs for DOPs and the abstract memory performs better than direct memory in chaotic and random dynamics. Second, the abstraction based memory scheme enables learning processes, which efficiently improves the performance of EAs in dynamic environments. Third, the effect of the abstract memory on the performance of the EA depends on the learning time and the frequency of environmental changes. This paper studied the relations between learning and the abstract memory in dynamic environments. For the future work, it is valuable to further investigate this important but poorly studied issue, the relations between learning and other memory schemes for DOPs. Another interesting research is to compare and combine the abstract memory scheme with other approaches developed for EAs in dynamic environments.

References

[1] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press, NY, 1996.

[2] P. A. N. Bosman. Learning and anticipation in online dynamic optimization. *Evolutionary Computation in Dynamic and Uncertain Environments*, Springer–Verlag, Berlin, pp. 129–152, 2007.

[3] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. *Proc. of the 1999 Congress on Evol. Comput.*, pp. 1875–1882, 1999.

[4] J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck. A multi-population approach to dynamic optimization problems. *Proc. of the Adaptive Computing in Design and Manufacturing*, pp. 299–308, 2000.

[5] J. Branke. *Evolutionary Optimization in Dynamic Environments*, Kluwer Academic Publishers, 2002.

[6] R. Fitch, B. Hengst, D. Suc, G. Calbert, and J. Scholz. Structural abstraction experiments in reinforcement learning. *AI 2005: Advances in Artificial Intelligence*, Springer–Verlag, Berlin, pp. 164–175, 2005.

[7] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments – a survey. *IEEE Trans. on Evol. Comput.*, **9**(3): 303–317, 2005.

[8] E. H. J. Lewis and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. *Parallel Problem Solving from Nature–PPSN V*, Springer–Verlag, Berlin, pp. 139–148, 1998.

[9] R. W. Morrison and K. A. De Jong. Triggered Hypermutation Revisited. *Proc. of the 2000 Congress on Evolutionary Computation*, pp. 1025–1032, 2000.

[10] H. Richter. A study of dynamic severity in chaotic fitness landscapes. *Proc. of the 2005 IEEE Congress on Evolutionary Computation*, pp. 2824–2831, 2005.

[11] 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.

[12] S. Yang. Associative memory scheme for genetic algorithms in dynamic environments. *Applications of Evolutionary Computing*, Springer–Verlag, Berlin, pp. 788–799, 2006.