Memory Based on Abstraction for Dynamic Fitness Functions

Hendrik Richter[1] and Shengxiang Yang[2]

[1] HTWK Leipzig, Fachbereich Elektrotechnik und Informationstechnik
Institut Mess–, Steuerungs– und Regelungstechnik
Postfach 30 11 66, D–04125 Leipzig, Germany
richter@fbeit.htwk-leipzig.de
[2] Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, United Kingdom
s.yang@mcs.le.ac.uk

Abstract. This paper proposes a memory scheme based on abstraction for evolutionary algorithms to address dynamic optimization problems. In this memory scheme, the memory does not store good solutions as themselves but as their abstraction, i.e., their approximate location in the search space. When the environment changes, the stored abstraction information is extracted to generate new individuals into the population. Experiments are carried out to validate the abstraction based memory scheme. The results show the efficiency of the abstraction based memory scheme for evolutionary algorithms in dynamic environments.

1 Introduction

As a class of stochastic algorithms, evolutionary algorithms (EAs) work by maintaining and evolving a population of candidate solutions through selection and variation. New populations are generated by first selecting relatively fitter individuals from the current population and then performing variations (e.g., recombination and mutation) on them to create new off–spring. EAs have been applied to solve many stationary optimization problems with promising results. Usually, with the iteration of EAs, individuals in the population will eventually converge to the optimal solution(s) due to the pressure of selection. Convergence at a proper pace is expected for EAs to locate optimal solution(s) for stationary problems. However, many real world problems are actually dynamic optimization problems (DOPs), where changes may occur over time. For DOPs, convergence becomes a big problem for EAs because it deprives the population of genetic diversity. Consequently, when the environment changes, it is hard for EAs to escape from the optimal solution of the old environment.

For DOPs, the aim of an EA is no longer to locate a stationary optimum but to track the moving optima with time. This requires additional approaches to adapt EAs to the changing environment. In recent years, with the growing interest in studying EAs for DOPs, several approaches have been developed into EAs to address DOPs [6,9], such as diversity schemes [7,16], memory schemes [2,4,19],

M. Giacobini et al. (Eds.): EvoWorkshops 2008, LNCS 4974, pp. 597–606, 2008.

and multi–population approaches [5]. Among these approaches developed for EAs in dynamic environments, memory works by retaining and reusing relevant information, which might be as straightforward as storing good solutions.

This paper proposes a memory scheme based on abstraction. 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. Thus, abstraction means a threshold for and compression of information, see e.g. [8] which proposes similar ideas for reinforcement learning. When an environment change is detected, the stored abstraction information is extracted to generate new individuals into the population.

The rest of this paper is organized as follows. In the next section, we briefly review the usage of memory for EAs for DOPs. Sec. 3 describes the dynamic fitness landscape used as the test bed in this paper. Sec. 4 presents the abstraction memory scheme. The experimental results and their analysis are given in Sec. 5. Finally, Sec. 6 concludes this paper with discussions on future work.

2 Memory Schemes for Dynamic Environments

The principle of memory schemes for EAs in dynamic environments is to, implicitly or explicitly, store useful information from old environments and reuse it later in new environments. Implicit memory schemes use redundant encodings, e.g., diploid genotype [10,12,18], to store information for EAs to exploit during the run. In contrast, explicit memory uses precise representations but splits an extra storage space to explicitly store information from a current generation and reuse it later [4,11,17].

For explicit memory schemes, usually only good solutions are stored in the memory as themselves and are reused directly. When a change occurs or every several generations, the solutions in the memory can be merged with the current population [4,11]. This is called *direct memory scheme*. It is also interesting to store environmental information as well as good solutions in the memory and reuse the environmental information when a change occurs. This is called *associative memory scheme*. For example, Ramsey and Greffenstette [13] studied a genetic algorithm (GA) for a robot control problem, where good candidate solutions are stored in a memory together with information about the current environment the robot is in. When the robot incurs a new environment that is similar to a stored environment instance, the associated controller solution in the memory is re–activated. In [19], an associative memory scheme was proposed into population based incremental learning (PBIL) algorithms for DOPs, where the working probability vector is also stored and associated with the best sample it creates in the memory. When an environmental change is detected, the stored probability vector associated with the best re–evaluated memory sample is extracted to compete with the current working probability vector to become the future working probability vector for creating new samples. Similarly, an associative memory scheme was developed into GAs for DOPs in [20], where

the allele distribution statistics information of the population is taken as the representation of the current environment.

For explicit memory schemes, since the memory space is limited, we need to update information stored in the memory. A general strategy is to select one memory point to be replaced by the best individual from the population. As to which memory point should be updated, there are several strategies [4]. For example, the most–similar strategy replaces the memory point closest to the best individual from the population. Though memory schemes have been shown to be beneficial for EAs in dynamic environments, so far they are only a promising option for regular and predictable dynamics, such as cyclic or translatory, but not for irregular dynamics, such as chaotic or random dynamics. The abstraction memory scheme proposed in Sec. 4 is an attempt to overcome this shortcoming.

3 Description of the Dynamic Test Problem

As dynamic fitness landscape, we use a moving n–dimensional "field of cones on a zero plane", where N cones with coordinates c_i, $i = 1, 2, ..., N$ are initially distributed across the landscape and have randomly chosen heights h_i and slopes s_i. We introduce discrete time $k \in \mathbb{N}_0$ and consider the coordinates $c_i(k)$ to be changing with it. So, the dynamic fitness function is:

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

Hence, the DOP is $\max\limits_{x\in M} f(x,k) = \max\left\{0\,,\ \max\limits_{1\le i\le N}\left[h_i - s_i\|x - c_i(k)\|\right]\right\}, k \ge 0$ whose solution $x_s(k) = \arg\max\left\{0\,,\ \max\limits_{1\le i\le N}\left[h_i - s_i\|x - c_i(k)\|\right]\right\}$ forms a solution trajectory in the search space $M \subset \mathbb{R}^n$. This means that for every $k \ge 0$ the problem might have another solution, which we intend to find using an EA. In the experiments we report the dynamics of $c(k)$ that starts from randomly chosen $c(0)$ and is either regular (cyclic) or chaotic or random (normally and uniformly distributed).

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 $P(t+1)$. 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 γ 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. 5.

4 The Abstraction Memory Scheme

The main idea of the abstraction based 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, we obtain for an n–dimensional search space M 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. Such ideas have some similarity to anticipating the dynamics of the DOP [3].

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, selecting 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 detected by, for instance, a falling sliding mean of the best individual. 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 done.

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, which is a quantity we will examine in the numerical experiments given in Sec. 5, 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 $x_j = (x_{1j}, x_{2j}, \ldots, x_{nj})^T$ 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 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 good solutions occur. In this way, we encode and compress the information about good solutions. As the matrix $\mathcal{M}$ is filled over run–time, the memorizing process can be seen as a learning process in both its figurative and literal meaning.

After a change has been detected (for instance 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

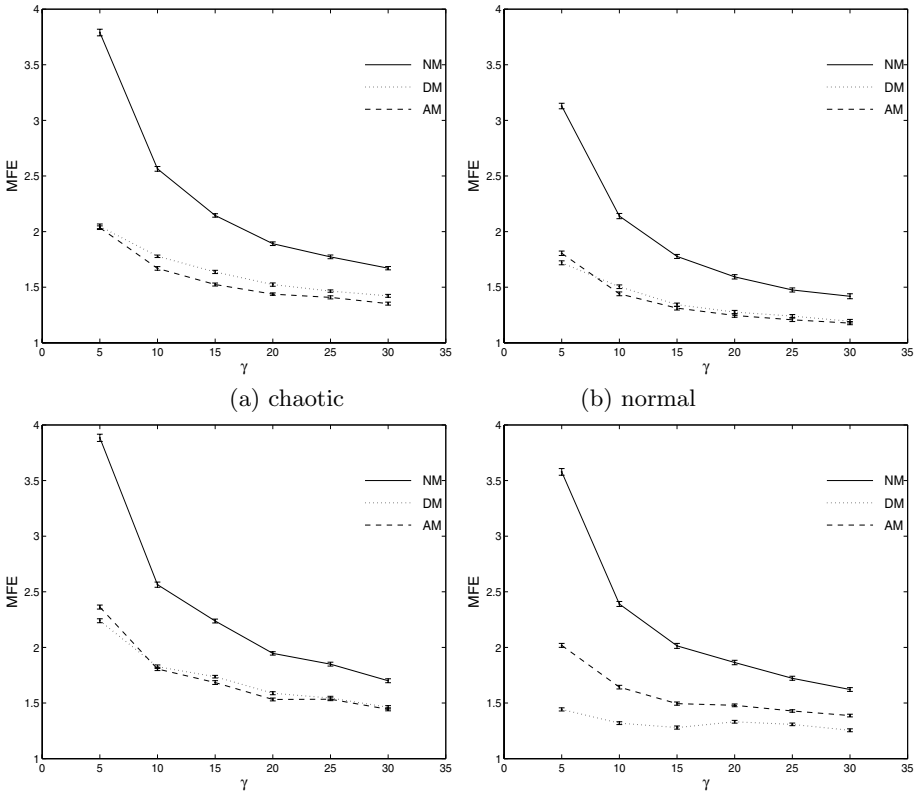
all matrix elements, that is $\mathcal{M}_\mu(t) = \frac{1}{\sum_{h_i} \mathcal{M}(t)} \mathcal{M}(t)$. Hence, the sum of all elements $\mu_{\ell_1 \ell_2 \ldots \ell_n}(t)$ in $\mathcal{M}_\mu(t)$ adds up to one. Each element in $\mathcal{M}_\mu(t)$ contains an approximation of the natural measure $\mu \in [0, 1]$ 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. Secondly, we fix a 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 their number for each cell by sorting the $\mu_{\ell_1 \ell_2 \ldots \ell_n}(t)$ according to their magnitude and producing 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 $M_{\ell_1 \ell_2 \ldots \ell_n}$. That means the τ individuals are distributed such that the number within each cell approximates the expected value for 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 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 compressing of information about good solutions in the search space, abstract retrieval becomes decoding and expansion.

An advantage of such a memory scheme is that it leads to a reduction of the information content, which is typical for abstraction. Naturally, this depends on the coarseness of the partitioning, which will be an important quantity to study. This also means that the number of individuals that take part in the memorizing and the number of individuals that come out of the memory and are inserted in the population are completely independent of each other. So, in contrast to explicit schemes, memory space and memory updating are not topics that need to be addressed. Another attribute of the proposed memory scheme is that in the memory matrix not the good solutions are stored but the event of occurrence of the solution at a specific location in the search space.

5 Experimental Results

We now report numerical experiments 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. The performance of the algorithms is measured by the Mean Fitness Error (MFE), defined as:

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

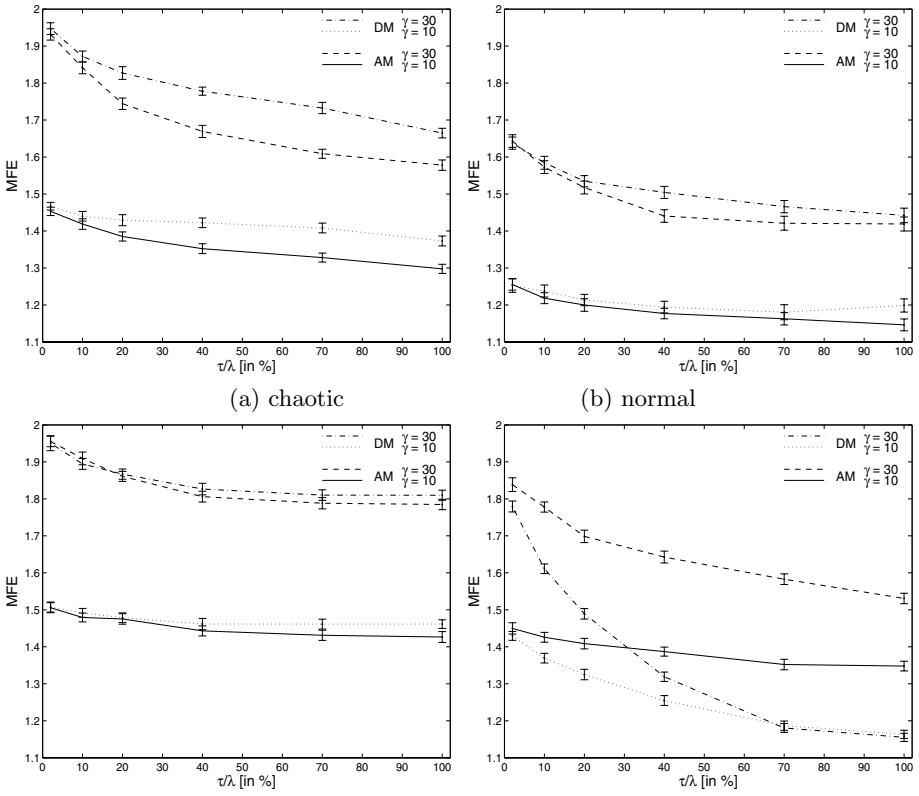


(a) chaotic (b) normal

(c) uniform (d) cyclic

Fig. 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 scheme (DM) and abstraction memory scheme (AM)

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, $f\left(x_s, \lfloor \gamma^{-1} t \rfloor\right)$ 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 $\max_{x_j(t)\in P(t)} f\left(x_i(t), \lfloor \gamma^{-1} \rfloor t\right)$ change every γ generations according to Eq. (2). The parameters we use 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 [14,15] 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 dynamic severity is an important factor in dynamic optimization, for

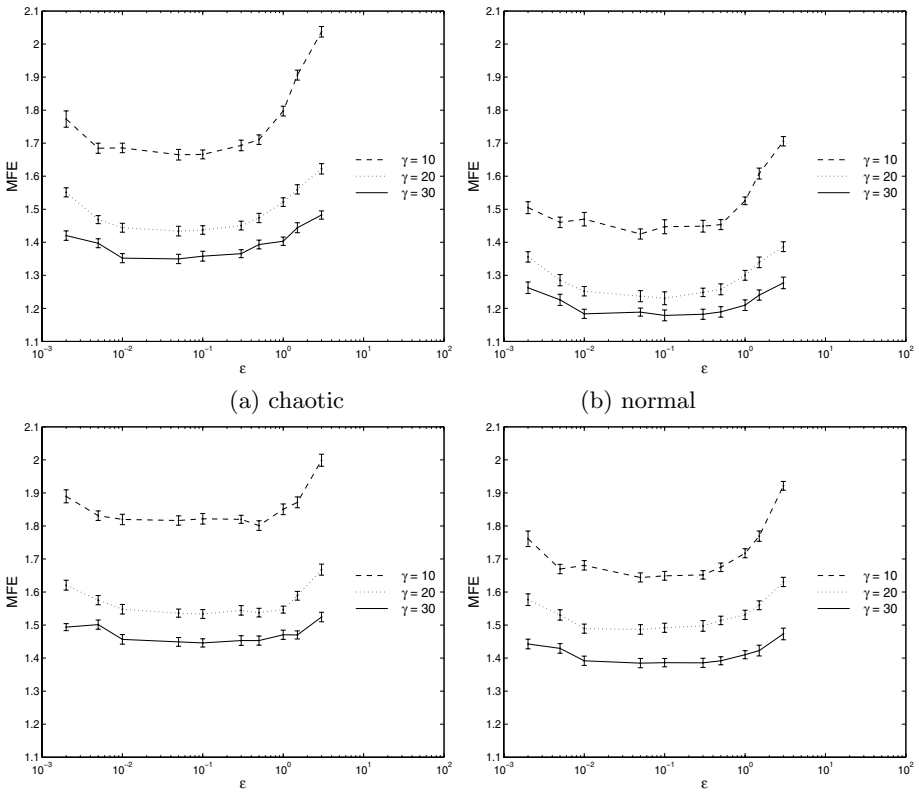


(a) chaotic (b) normal

(c) uniform (d) cyclic

Fig. 2. Comparison of performance of direct memory scheme (DM) and abstraction memory scheme (AM) measured by the MFE over the percentage of individual inserted from the memory for different types of dynamics and $\gamma = 10$ and $\gamma = 30$

all considered dynamics, severity is normalized and hence has no differentiating influence. In a first set of experiments, the abstraction memory scheme (AM) is tested and compared with a direct memory scheme (DM), which stores the good solutions as themselves 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 Fig. 1. Here, as well as in the other experiments, we fixed the upper and lower bounds of the search space at $x_{1\,min} = x_{2\,min} = -3$ 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, 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 Fig. 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



(a) chaotic (b) normal

(c) uniform (d) cyclic

Fig. 3. Comparison of performance of abstraction memory scheme (AM) measured by the MFE for different grid size ϵ and different types of dynamics and $\gamma = 10$, $\gamma = 20$ and $\gamma = 30$

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 AM gives better results than the DM 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 DM being better than AM. In a second set of experiments, we examine the effect of different amount τ of individuals inserted from the abstract memory, see Fig. 2, where the MFE is given for the four types of dynamics and the change frequencies $\gamma = 10$ and $\gamma = 30$, over the percentage of individuals τ retrieved and inserted in the population of size $\lambda = 50$, that is $\frac{\tau}{\lambda}$ in %. It can be seen that an increase in the number of inserted individuals leads to a better performance, but also that a certain saturation sets in, particularly for random dynamics with normal and uniform distribution. Further, it can be observed that the AM is a better option than the DM for chaotic and random, but not for cyclic dynamics. Here, we find that a

large number of individuals inserted results in an even better performance, which can be attributed to the fact that for a large insertion the best solution is most likely among the retrieved. On the other hand, it is extremely unlikely to retrieve a solution from a DM for chaotic and random dynamics which appears to be one reason for the better performance of the AM. Finally, we look at the influence of the grid size ϵ on performance of the AM scheme, see Fig. 3. Here, the MFE is given over ϵ and different γ on a semi–logarithmic scale while we again set $\lambda = 50$ and $\tau = 20$. For all types of dynamics and all change frequencies we obtain a kind of bathtub curve, which indicates that an optimal grid size depends on the type of dynamics and the size of the bounded region in search space which the memory considers. This gives raise to the question 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.

6 Conclusions and Future Work

In this paper an abstraction based memory scheme is proposed for EAs to address dynamic environments. In this scheme, memory is used to store the abstraction (i.e., the spatial distribution) of good solutions instead of good solutions themselves. When an environment change is detected, the stored spatial distribution is used to generate solutions into the population. A series of experiments were carried out to investigate the performance of the abstraction based memory scheme against a traditional direct memory scheme for EAs in dynamic environments. Experimental results show that the abstraction based memory scheme efficiently improves the performance of EAs for dynamic environments, especially for irregular chaotic and random dynamic environments. For regular cyclic dynamic environments, a traditional direct memory scheme seems a better choice than the abstraction based memory scheme. As there is a strong link between the proposed scheme and learning, a high–diversity scheme should be used besides the memory, in particular in the beginning of learning, that is, in the filling of the memory matrix. In the experiments, the learning curves should be studied. Also, it could be tested if after the learning, that is, the matrix filling, has reached a certain degree of maturity, memory should replace high–diversity to obtain the best results. These topics fall in our future research.

Acknowledgement

The work by Shengxiang Yang was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant No. EP/E060722/1.

References

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

2. Bendtsen, C.N., Krink, T.: Dynamic memory model for non–stationary optimization. In: Proc. of the 2002 IEEE Congress on Evol. Comput., pp. 145–150 (2002)
3. Bosman, P.A.N.: Learning and anticipation in online dynamic optimization. In: Yang, S., Ong, Y.-S., Jin, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments, pp. 129–152 (2007)
4. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: Proc. of the 1999 Congr. on Evol. Comput., pp. 1875–1882 (1999)
5. Branke, J., Kauß, T., Schmidt, C., Schmeck, H.: A multi–population approach to dynamic optimization problems. In: Proc. of the 4th Int. Conf. on Adaptive Computing in Design and Manufacturing, pp. 299–308 (2000)
6. Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer, Dordrecht (2002)
7. Cobb, H.G., Grefenstette, J.J.: Genetic algorithms for tracking changing environments. In: Proc. of the 5th Int. Conf. on Genetic Algorithms, pp. 523–530 (1993)
8. Fitch, R., Hengst, B., Suc, D., Calbert, G., Scholz, J.: Structural abstraction experiments in reinforcement learning. In: Zhang, S., Jarvis, R. (eds.) AI 2005. LNCS (LNAI), vol. 3809, pp. 164–175. Springer, Heidelberg (2005)
9. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments – a survey. IEEE Trans. on Evol. Comput. 9, 303–317 (2005)
10. Lewis, E.H.J., Ritchie, G.: A comparison of dominance mechanisms and simple mutation on non–stationary problems. In: Parallel Problem Solving from Nature V, pp. 139–148 (1998)
11. Mori, N., Kita, H., Nishikawa, Y.: Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In: Proc. of the 7th Int. Conf. on Genetic Algorithms, pp. 299–306 (1997)
12. Ng, K.P., Wong, K.C.: A new diploid scheme and dominance change mechanism for non–stationary function optimisation. In: Proc. of the 6th Int. Conf. on Genetic Algorithms, pp. 159–166 (1995)
13. Ramsey, C.L., Greffenstette, J.J.: Case–based initialization of genetic algorithms. In: Proc. of the 5th Int. Conf. on Genetic Algorithms, pp. 84–91 (1993)
14. Richter, H.: Behavior of evolutionary algorithms in chaotically changing fitness landscapes. In: Parallel Problem Solving from Nature VIII, pp. 111–120 (2004)
15. Richter, H.: A study of dynamic severity in chaotic fitness landscapes. In: Proc. of the 2005 IEEE Congress on Evolut. Comput., vol. 3, pp. 2824–2831 (2005)
16. Tinos, R., Yang, S.: A self–organizing random immigrants genetic algorithm for dynamic optimization problems. Genetic Programming and Evolvable Machines 286, 255–286 (2007)
17. Trojanowski, T., Michalewicz, Z.: Searching for optima in non–stationary environments. In: Proc. of the 1999 Congress on Evol. Comput., pp. 1843–1850 (1999)
18. Uyar, A.Ş., Harmanci, A.E.: A new population based adaptive dominance change mechanism for diploid genetic algorithms in dynamic environments. Soft Computing 9, 803–815 (2005)
19. Yang, S.: Population–based incremental learning with memory scheme for changing environments. In: Proc. of the 2005 Genetic and Evol. Comput. Conference, vol. 1, pp. 711–718 (2005)
20. Yang, S.: Associative memory scheme for genetic algorithms in dynamic environments. In: Applications of Evolutionary Computing: EvoWorkshops 2006, pp. 788–799 (2006)