Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems

Jürgen Branke
Institute AIFB, University of Karlsruhe
D-76128 Karlsruhe, Germany
Phone: ++49 (721) 6086585 Fax: ++49 (721) 693717
Email: branke@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/∼jbr

Abstract- Recently, there has been increased interest in evolutionary computation applied to changing optimization problems. This paper surveys a number of approaches that extend the evolutionary algorithm with implicit or explicit memory, suggests a new benchmark problem and examines under which circumstances a memory may be helpful. From these observations we derive a new way to explore the benefits of a memory while minimizing its negative side effects.

Keywords: evolutionary algorithm, genetic algorithm, dynamic, non-stationary, time-varying, memory

1 Introduction

While most of the papers produced in the area of evolutionary computation deal with optimization in static, non-changing environments, many real-world problems are basically dynamic: new jobs have to be added to the schedule, machines may break down or wear out slowly, raw material is of changing quality etc.

Of course one could deal with this non-stationarity by regarding each change as the arrival of a new optimization problem that has to be solved from scratch (cf. [15]). However this simple approach is often impractical, e.g. because solving a problem from scratch without reusing information from the past is too time consuming.

Fortunately, unless the change in the problem is extremely strong, probably much effort could be saved and better solution quality achieved by using an optimization algorithm that is capable of continuously adapting the solution to a changing environment, reusing the information gained in the past. Evolutionary algorithms (EAs) seem to be a suitable candidate, and subsequently the interest in EAs for dynamic problems has been rising in recent years.

A number of authors have addressed the issue of transferring information from the old environment to the new environment by enhancing the EA with memory that might allow it to store good (possibly partial) solutions and reuse them later as necessary. This memory may be implicit (e.g. a diploid genome) or explicit (i.e. an extra storage space with explicit rules for storing and retrieving information).

This paper surveys the memory-based approaches published so far, suggests a new benchmark problem, examines the idea of explicit memory more closely, compares several storage strategies, and suggests a new way to use the memory.

The paper's outline is as follows: Section 2 thoroughly reviews and classifies relevant literature. The next two sections discuss a number of design decisions related to the use of a memory and motivate our approach. Then, in Section 5, a new benchmark problem is defined that is simple, but relates better to the interesting properties of real-world dynamic environments than previous benchmarks. A number of experimental results are reported in Section 6.

The paper concludes with a summary and some suggestions for future work.

2 Memory-based Evolutionary Algorithms: History

2.1 Implicit Memory

An evolutionary algorithm that uses representations containing more information than necessary to define the phenotype (i.e. redundant representations) basically has some memory where good (partial) solutions may be stored and reused later as necessary.

We call this kind of memory *implicit* because it is left to the EA to find a way to use it appropriately.

The most prominent approach to redundant representations seems to be diploidy. Goldberg and Smith [5, 19] report on experiments with using diploidy and dominance. Since it is not clear beforehand which allele value (e.g. 0 or 1) should be dominant at a particular gene position, Goldberg and Smith favor a triallelic scheme where an allele can take on one of three values "0", "recessive 1", and "dominant 1". Tested on a time-varying knapsack problem, they report better adaptive qualities than with a simple GA.

However, this approach has been reviewed critically by Ng and Wong in [14]. They argue that the triallelic scheme is biased and that the reported better performance of the triallelic scheme compared to a simple haploid representation is mainly due to the slower convergence rate of the triallelic GA which happens to be suitable in the frequent changes used for testing in [5]. In other dynamic environments, so the conclusion, the haploid GA might even outperform the triallelic scheme. Ng and Wong propose a new diploid scheme with four possible alleles (0 and 1, each dominant and recessive). To be able to adapt to changes quickly, they suggest to use a dominance change mechanism for which all allele pairs are inverted (i.e. changed from dominant to recessive and vice versa) whenever an individual‘s fitness decreases by more than 20%. In the experiments presented, their diploid scheme

outperforms the haploid as well as the triallelic scheme.

Hadad and Eick [7] use multiploidy and a dominance vector as an additional part of an individual that breaks ties whenever there is an equal amount of 0's and 1's at a specific gene location. They report on a number of experiments with varying number of gene strings per individual. Also using the time-varying knapsack problem as testbed, they observe best performance with diploid or tetraploid individuals, while tetraploidy showed a slight advantage at high change frequencies.

Ryan [17] uses additive multiploidy, where the genes determining one trait are *added* in order to determine the phenotypic trait. The phenotypic trait becomes 1 when a certain threshold b_1 is exceeded, and is 0 otherwise. In [18] this paradigm has been extended slightly such that the phenotypic trait becomes 1 when a certain threshold b_1 is exceeded, 0 if the value is below a smaller threshold b_2, and is determined at random if the value is between b_1 and b_2. The results are reported to outperform the methods by Ng and Wong ([17]) and Osmera ([18]) on several problems.

Lewis et al. [9] compared five approaches on the oscillatory knapsack problem: a simple hypermutation scheme (cf. [1]), the approach by Ng and Wong [14] with and without the mechanism of dominance change (see above) and the original approach of Ryan [17] as well as a variant extended with a dominance change mechanism. They observed that a simple dominance scheme is not sufficient to track the optimum reasonably well. If the diploid approaches are extended with a dominance change mechanism, much better results can be obtained. Still however, a simple haploid GA with a hypermutation rate similar to the number of bits flipped by a dominance change performed comparably. Experiments with an environment of two alternating states as well as a larger number of states revealed that the approach by Ng and Wong proved to be able to learn two solutions and switch between them almost instantaneously, although the best solutions achieved for each target were slightly poorer than with the other approaches. If more than two targets were used, the Ng and Wong approach failed completely, while the haploid hypermutation GA performed better than both diploid GAs with dominance scheme. It seems that the Ng and Wong approach is quite effective at maintaining memory, while the approach by Ryan, extended with a dominance change mechanism, maintains diversity similar to a hypermutation scheme.

Given the evidence available so far, it can be assumed that the multiploid representations may be useful in periodically changing environments where it is sufficient to remember a few states and where it is important to be able to return to previous states quickly. The applicability to problems without periodicity and more than a few re-occurring states is at least questionable.

A quite different redundant representation scheme that is not based on multiploidy but uses a multi-level structured gene-representation has been suggested by Dasgupta and McGregor [3]. In this representation, each level can activate or deactivate genes at the next lower level, allowing complex hierarchically structured genes and more redundant information than in the diploid scheme. In the experiments on the time-varying knapsack problem reported in [3] however, a relatively simple two-level representation was chosen, where the higher level activated exactly one of four alternative substrings, each consisting of a complete solution to the problem. Nevertheless, improvements over simple GAs were found.

This approach has additionally been tested on a simple moving parabola, results are reported in [2]. As in the previous paper, a simple two-level structure was used, this time with the higher level activating only parts of the solution. Performance found was significantly better than with a simple GA. Whether this approach has a significant advantage compared to multiploidy has yet to be determined.

2.2 Explicit memory

While redundant representations might allow the EA to implicitly store some useful information during the run, it is not clear that the algorithm actually uses this memory in an efficient way. As an alternative, the kind of approaches in this subsection use an explicit memory in which specific information is stored and reintroduced into the population at later generations.

Louis and Xu [11], for example, look at re-scheduling an open shop problem after a machine has broken down and has been replaced by a faster machine. They assume that the changes of the problem are known (i.e. it is possible to react explicitly), and they use a fixed number of generations between changes (which is a valid assumption when the time between changes is larger than the time to run the maximum number of generations for the EA). In this paper, the population's best individual is stored at regular intervals. For example, when the maximum number of generations is 300, every 50 generations the best individual is stored, resulting in a total of 6 stored individuals. After a change, the GA is seeded partially (5-10%) with individuals from the old run (i.e. the last run before the environment changed), while all other individuals are initialized randomly. The authors report significant improvements over a totally random initialization, particularly in early generations. However, when carrying over more individuals from the old run (50-100%), or for problems where the environment changes more significantly (deletion of a job), the method reportedly failed. Further experiments on the effect of the number and quality of the inserted solutions are reported in [10]. For example, the authors observe that as the similarity between the problems decreases, injecting individuals with lower fitness from the old population results in better solutions in the new run. In any case, the memory in that paper is only used to transfer individuals from one EA run to seed the initial population after a single change, the memory is not permanent.

Ramsey and Grefenstette [16] incorporate case-based reasoning into an EA. They use a knowledge base to memorize successful individuals in a permanent memory. The sys-

tem assumes that the environmental conditions can be measured. In regular intervals, the best individual is stored in the knowledge base and indexed with data characterizing the environment at that time. Whenever a new environment is encountered (the environmental variables changed), the EA is restarted. For restart, half of the population is initialized with individuals from the knowledge base that have been successful in a similar environment. Similarity of environments is calculated by a nearest neighbor analysis. Experiments proved that the knowledge base allows the EA to build upon the knowledge gained in the past. Unfortunately this approach is only applicable when the similarity of environments can be measured.

Another example is the work by Trojanowski et al. [20] in which each individual is extended with additional memory for a number of its ancestors. After a change in the environment, these older solutions are also re-evaluated and replace the current individual if they outperform it in the new environment. Since the memory is limited, this approach may be regarded as an EA with short-term memory that allows to increase variability by reintroducing individuals that have been considered good in recent generations.

A more elaborate storage strategy has been added to the Thermodynamical Genetic Algorithm (TDGA, c.f. [12, 13]). There, every generation‘s best individual is stored in the memory, and another individual is deleted from the memory depending on its age and contribution to the memory population’s diversity. The individuals from the memory then serve as additional potential candidates in the process of selecting a parent generation (in addition to the usual population).

This is perhaps the most universal approach to permanent, explicit memory and is most closely related to the following sections. However, so far it has been defined for binary representation only and has never been evaluated per se.

Together with TDGA it was tested again on the time-varying knapsack problem where it was able to recall earlier solutions when needed and to adapt to new situations much faster when they reappeared, even in high frequency changing environments.

3 General Thoughts about Memory

Why does the idea of adding a memory to an evolutionary algorithm (EA) seem to be so appealing that a larger number of authors have suggested it?

Intuitively, when the optimum reappears at a previous location, a memory could remember that location, and instantaneously move the population to the new optimum. A memory could also be useful in maintaining diversity. And it might guide evolution to promising areas after a re-initialization. But while the memory might allow to exploit knowledge gained in the past, it might as well mislead evolution and prevent it from exploring new regions and discovering new peaks.

When trying to get the most out of an added memory, a good start is to look at the possible design decisions that have to be made. First of all there is the question of implicit or explicit memory. We will restrict ourselves to explicit memory here, since its effects are easier to understand and since given the evidence so far, it seems questionable whether implicit memory is really useful for problems with more than two states.

After having decided to use an explicit memory, further questions arise:

1. when and which individuals should be stored in the memory?
2. how many individuals should be stored in the memory and which should be replaced to make space for new individuals?
3. which individuals should be retrieved when from the memory and reinserted into the population.?

Although we will not be able to answer all of these questions here, we will discuss and compare a number of alternatives and try to motivate our choices.

Intuitively, the individuals stored in the memory should be of above average fitness, not too old, and distributed across several promising areas of the search space.

Regarding the question which individuals to store in the memory, it seems quite natural to use the best individuals from time to time. Contrary to this intuition, Louis et al. [10] reported that for the case of larger changes, better results were obtained with storing inferior solutions. This might be caused by the fact that they used a simple storage scheme, storing only a very small number of individuals and keeping all of them. Storing inferior solutions then means maintaining diversity which is necessary to be able to react to larger changes. If diversity is considered explicitly when deciding which individuals to keep, we still think it is sufficient to look at the best individuals.

As replacement strategies, we consider the following alternatives:

- Compute an “importance value” for each individual as linear combination of the individual’s age, its contribution to diversity and its fitness. Since it seems very difficult to determine an optimal tradeoff, we don’t think this approach would be practicable
- Delete the individual which, when deleted, retains the maximum variance in the population, where the variance is calculated as the sum of the variances of the alleles over the population (this method with subsequently be termed *variance*).

$$V(i) = \sum_{i=1}^{m} \sum_{j \in P \setminus \{i\}} (x_{ij} - \bar{x_i})^2$$

m : length of genotype, P: population

- In order to maintain diversity, a simple crowding strategy may be used, i.e. the new individual replaces

the most similar old individual as long as it is better (termed *similar*).

- Alternatively, we can determine the two individuals with the minimum distance between each other and remove the less fit individual. The underlying idea is that one individual in a certain area should be sufficient, and if there are two close to each other, only the fitter should be retained (termed *mindist*).

Retrieval should probably only happen after the environment changed, otherwise the continuous injection of old individuals in the population may be detrimental. We decided to acknowledge that a change has occurred whenever the fitness of at least one individual in the memory has changed. Note that although this method does not guarantee to capture all changes, it seems a reasonable indicator due to the expected variety in the memory. Whenever we decide to retrieve individuals from the memory, we merge the old population and the memory population and keep the best n individuals as new population (the memory remains unaffected).

4 The best of two worlds?

Our first experiments with memory added EAs showed that the risk to misguide evolution and prevent it from exploring new regions of the search space should not be underestimated.

The alternative, restarting evolution from scratch whenever a change in the environment occurred, will of course have a good chance to find new peaks, however, it will take a long time to reach that optimum, sudden jumps are impossible.

The question therefore was: can we have the best from both worlds, without the drawbacks?

Our suggestion is to divide the population into two, a "memory"-population and a "search"-population. The one population is memory-based and responsible for remembering good old solutions, maintaining a minimum quality and initiating jumps. The other population constantly searches for new peaks and is submitting these to the memory, but will not retrieve any information from it. In order to enforce exploration, this second population is re-initialized at random after every change in the environment (see Figure 1).

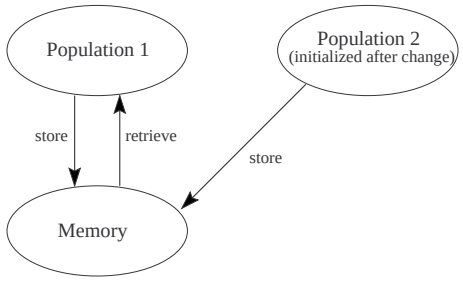

Figure 1: Memory based EA with two islands: one to exploit the memory, the other to explore new regions of the search space.

5 A New Benchmark Problem

Benchmarks should be simple, easy to describe, easy to analyze and also tunable in their parameters. On one hand they should be complex enough to allow conjectures to the real world, on the other hand they should be simple enough to allow to gain new insights into the working of the optimization algorithm.

That's why we think that scheduling ([11]) or mouse-tracking ([16]), although representative real-world problems, are not the best benchmarks for a research area in such an early stage as EAs for dynamic optimization problems currently is.

Other authors have suggested simpler problems, however they seem to be so simple that they are too far away from reality.

So far, the majority of authors tested their approach on a time varying knapsack problem [3, 5, 7, 9, 13, 17, 19] where the allowable weight limit changes over time, usually it oscillates between two predefined values. Representation is binary and invalid individuals are penalized. This problem does not seem to be typical since the environment only oscillates between two static states, in which an explicit memory would certainly outperform all implicit memory strategies. Also, the change from a higher to a lower weight limit makes all previous individuals invalid which basically eradicates the old solution and forces the EA to search for a new solution.

If a change of the problem results in a totally new, random environment with no connection to the previous environment, nothing will beat a simple restart policy, since there is just no information to transfer from one environment to the next. Thus, a reasonable benchmark problem should feature "small to medium" environmental changes.

If the environment is unimodal as in [2], the EA's task basically is to follow the peak as closely as possible. This is certainly interesting to observe and may provide insights about EA behavior in dynamic environments. However in that kind of environment, possibly a local hillclimber would be more efficient than an EA.

From the above considerations, we concluded that a suitable test environment should be multimodal and change slightly. Nevertheless, even a slight change might move the optimum to a totally different location, namely when the height of the peaks changes such that a different peak becomes the maximum peak. Although local hillclimbing might often be sufficient after a change, in these cases the EA basically has to "jump", or cross a valley, to reach the new maximum peak. A benchmark problem should simulate both sides of environmental change: sometimes it may be sufficient to adapt the current solution, and sometimes it may be necessary to switch to another, previously slightly inferior but now better solution. Therefore we here propose a new, simple benchmark problem that tries to bridge the gap between very complex, hard to understand real-world problems and all too simple toy problems. The idea is to have an artificial multi-dimensional landscape consisting of several peaks,

	X_1	X_2	X_3	X_4	X_5	W	H
peak1	8.0	64.0	67.0	55.0	4.0	0.1	50.0
peak2	50.0	13.0	76.0	15.0	7.0	0.1	50.0
peak3	9.0	19.0	27.0	67.0	24.0	0.1	50.0
peak4	66.0	87.0	65.0	19.0	43.0	0.1	50.0
peak5	76.0	32.0	43.0	54.0	65.0	0.1	50.0

Table 1: Initial parameters for all peaks

where the height, the width and the position of each peak is altered slightly every time a change in the environment occurs.

The test function suggested here has 5 dimensions, uses real-valued parameters, and has the following form:

$$F(\vec{x},t) \quad = \quad \max_{i=1\ldots5} \frac{H_i(t)}{1+W_i(t)\sum_{j=1}^{5}(x_j - X_j(t))^2}$$

The coordinates, the height H and the width W of each peak are initialized according to Table 1. Then, every Δe generations the height and width of every peak are changed by adding a random Gaussian variable. The location of every peak is moved by a vector v of fixed length s in a random direction. Thus the parameter s allows to control the severity of a change, Δe will determine the frequency of change.

More formally, a change can be described as

$$\begin{aligned} \sigma &\in N(0,1) \\ H_i(t) &= H_i(t-1) + 7 \cdot \sigma \\ W_i(t) &= W_i(t-1) + 0.01 \cdot \sigma \\ \vec{X}(t) &= \vec{X}(t-1) + \vec{v} \end{aligned}$$

An example on how the maximum moves over time in a two-dimensional space can be seen in Figure 2.

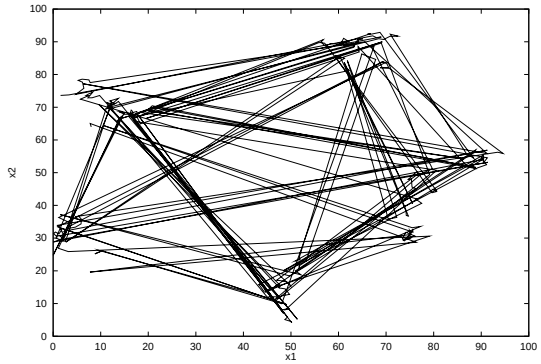

Figure 2: The movement of the maximum through a 2-dimensional subspace over 300 changes, shifting vector $\vec{v}$ has length $s = 0.9$

To allow replication, the exact test function is using a separate random number generator. The C-code can be downloaded from our web-page.

For future research, the function could also be made more complex by increasing the number of dimensions or the number of peaks and by overlaying the whole landscape with a high-frequency noise.

For this paper, we additionally used a more simple test function especially suited for memory-based EAs. This function, G, is a linear combination of two fixed 5-dimensional functions with 5 peaks each. The weight of the two functions changes over time, so G changes from g_1 to g_2 to g_1 etc. In other words, the absolute maximum of G oscillates between two points only, namely the maxima of functions g_1 and g_2. Basically this function relates to the oscillatory knapsack problem that has seen so many successful examples of memory-based EA approaches.

$$\begin{aligned} \lambda(t) &= \frac{cos(\frac{2\pi t}{100\delta}) + 1}{2} \\ g_1(\vec{x}) &= \sum_{i=1}^{5} \frac{H_i}{1+W_i\sum_{j=1}^{5}(x_j - X_j)^2} \\ g_2(\vec{x}) &= \sum_{i=6}^{10} \frac{H_i}{1+W_i\sum_{j=1}^{5}(x_j - X_j)^2} \\ G(t,\vec{x}) &= \lambda(t)g_1(\vec{x}) + (1-\lambda(t))g_2(\vec{x}) \end{aligned}$$

6 Experiments

For our experiments, we used an evolutionary algorithm with real-valued encoding, generational replacement but elite of 1, mutation rate of 0.2, crossover probability of 0.6, one population and a total population size of 100, including the memory if used, since all individuals in the memory have to be reevaluated every generation in order to detect changes of the landscape. Thus in any case, each generation involves 100 evaluations. All reported results are the averages over 20 runs with different random seed.

Since for dynamic fitness functions it is not useful to report the best solution achieved, we will here report on the offline-performance, which is the average of the best solutions at each time step, i.e. $x^*(T) = \frac{1}{T}\sum_{t=1}^{T} e_t^*$ with e_t^* being the best solution at time t (cf. [8]). If several populations are used, e_t^* is the best individual over all populations. Note that the number of values that are used for the average grows with time, thus the curves tend to get smoother and smoother.

If the memory is used, by default the EA is started with an empty memory and writes its best individual into the memory every 10 generations, replacing another individual according to one of the four strategies described in Section 3. Unless stated otherwise, memory size is 10.

The fitness function changes every 10 generations for function G (since this is a gradual change, the maximum peak switches only every 50 generations), and every 50 generations for function F in the way described above.

6.1 Oscillating Fitness Function

As expected, the EA with memory clearly outperforms the simple EA on G after a few generations since the memory allows the EA to return to known peaks. However, when we looked at the movement of the best individual through space, we noted that the performance gain is mainly due to the *quick* change from one peak to the exact location of the other, but often the best solution is not found. In other words, after one reasonable peak has been found for each of the two states of the fitness landscape, the memory actually inhibits the search for new, better peaks.

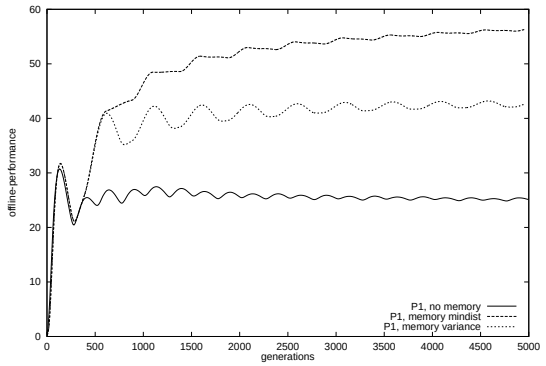

Figure 3: Offline performance with and without memory on the oscillatory function.

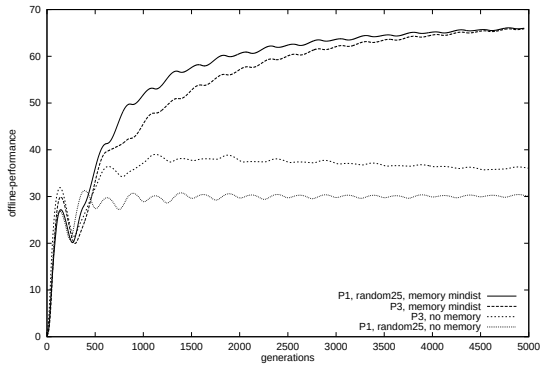

Figure 4: Offline performance with and without memory on the oscillatory function, 3 populations or 25 random immigrants.

This can be alleviated to some extend by enforcing some diversity in the population, either by using 3 independent populations (in which case we allow each population to write into the memory, while retrieving solutions from the memory is still restricted to one population), or by replacing 25 individuals in every generation by random immigrants (randomly generated individuals, this is another popular method to make EAs suitable to changing fitness functions [6]). As can be seen, performance is further increased, this time due to finding better peaks.

While the increased diversity also helps the simple EA, the positive effect of memory remains or is even increased.

As to the replacement strategies, the approach to maximize the variance in the memory (*variance*) performed significantly worse than replacing the most similar (*similar*) and replacement of the worse of the two individuals with minimum distance (*mindist*), which both showed almost identical performance. Therefore, our future experiments are restricted to the *similar* and *mindist* replacement strategies.

6.2 Changing the peak's location

In our last experiment the test function had the unrealistic property that the locations of the peaks remained the same throughout the run, only their height changed. With the next set of experiments, we want to examine the effect of varying the extent to which the location of the peaks is shifted, on the advantage of memory. This will be tested using function F.

For $s = 0$ the peaks still stay at the same place, but as opposed to function G the optimum now switches between 5 peaks, and they do not disappear completely before they rise again.

More or less, the results are similar to those obtained earlier: the memory-based EAs clearly outperforms the simple EA, the replacement strategies "most-similar" and "worse of two individuals with minimum distance" show almost identical behavior (therefore the second one is omitted in the diagram). The two-island-approach suggested in Section 4 (X2) clearly outperforms the one population approaches, and also the approach with three populations and memory (cf. Figure 5). Astonishingly, the approach that performed best on function G, namely the one that replaced 25 individuals with random immigrants, did not perform much better than the simple GA with memory on this function.

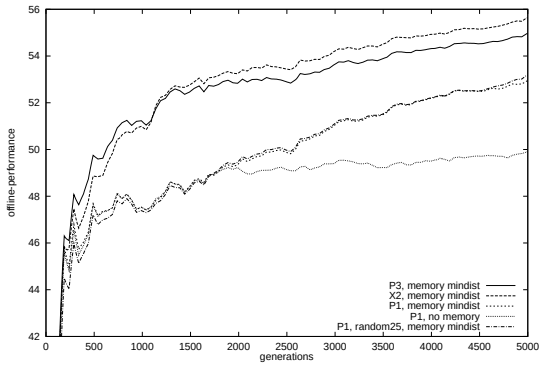

Figure 5: Offline-Performance of several approaches, no shift of the peak locations.

If the length s of the shift vector is increased, the main difference is that the three-island approach becomes the best one (cf. Fig. 6). This may be explained by its possibility to maintain island populations on up to three different peaks, following all three peaks simultaneously when they move, while all other tested approaches can only follow one peak at a time.

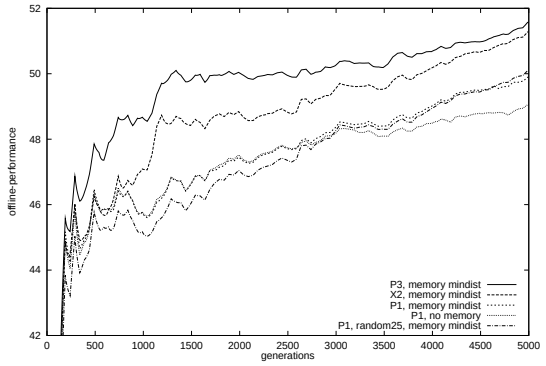

Figure 6: Offline-Performance of several approaches, peak locations shifting by $s = 0.6$.

In general, when the length s of the shift vector is increased, the performance of all approaches decreases (Figure 7). In particular, the difference between the tested approaches becomes smaller. Nevertheless, since the standard deviation of these values is usually very low (around 0.25), the difference remains significant. Figure 7 shows the offline-performance after 5000 generations (100 change intervals) over a range of values for s.

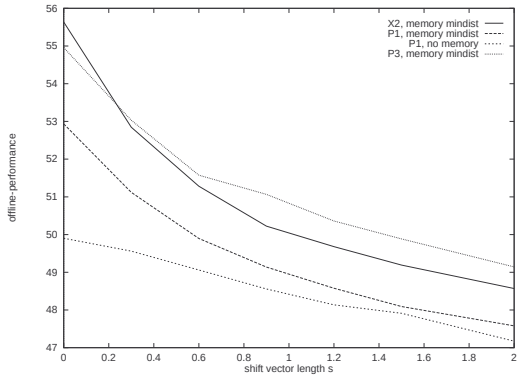

Figure 7: Offline-Performance of several approaches after 5000 generations, varying shift length s.

We also tried to gain even better performance by using a larger memory (20 individuals instead of 10) and by introducing 25 random immigrants in every generation. However, no significant improvements could be achieved.

7 Conclusion

This paper makes three contributions: First, it surveys previous approaches using implicit or explicit memory in evolutionary algorithms applied to dynamic fitness functions. Second, it critically observes previous test problems and suggests a new benchmark problem, aimed at bridging the gap between complex real-world applications and all too simple toy problems. The suggested benchmark is not limited to memory-based approaches, but might become a standard benchmark for evolutionary algorithms for dynamic fitness landscapes. And finally, the idea of explicit memory is examined more thoroughly, and a new memory based approach, using a "memory-population" and a "search-population" has been shown to be at least competitive with standard approaches.

From our experiments we draw the following conclusions:

- If the optimum repeatedly returns to exact previous locations, that's perfect for memory-based EAs since it allows them to switch instantaneously. Given the right memorization strategy, a large number or re-occurring peaks may be stored in an explicit memory.
- However, the advantage of a memory quickly diminishes when the location of the optimum changes even slightly, therefore the range of problems where memory-based approaches promise noteworthy improvements is probably quite small.
- If one wants to retrieve good individuals from the memory, they first have to be stored. In other words, the basic evolutionary algorithm needs to be able to "switch peaks" or to maintain diversity if we want to have memory-individuals on several peaks.
- Using several independent subpopulations allows to follow several peaks simultaneously.
- As replacement strategy for the memory, replacing the most similar or replacing the worse of the two individuals with minimum distance worked better than a variance maximization scheme.
- Problems where a change significantly decreases the fitness of the old solution are actually easier for EAs, since this enforces the search for a new peak

Summarizing our experiences, the application of memory-based EAs seems to be restricted to a small set of problems where the optimum repeatedly returns to previous locations, in other cases, diversity-based methods seem to be preferable. In any case, some diversity-keeping method should be used in conjunction with the memory.

For now, many questions remain open worth for future study: first of all, some other memory-based approaches already known from the literature, like considering age for replacement or storing not the best but inferior individuals should be examined under the same framework. For the new 2-population approach, the distribution of individuals to "search"-population, "memory"-population, and the actual memory could probably be optimized, maybe even made adaptive, dependent on success history. Currently, we are also investigating the possibility to use more than one memory population. To study the effect of changing the number of peaks or the change frequency is another interesting topic. And finally, other EA approaches designed for use in dynamic environments should be tested and compared on the benchmark problem presented.

Acknowledgements: Thanks to Steffen Lämmermeier

and Eberhard Münz for interesting discussions and providing some of the code, and to the anonymous reviewers for their valuable comments.

Bibliography

[1] H. G. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuouis, time-dependent nonstationary environments. Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA, 1990.

[2] D. Dasgupta. Incorporating redudancy and gene activation mechanisms in genetic search. In L. Chambers, editor, *Practical Handbook of Genetic Algorithms*, volume 2, pages 303–316. CRC Press, 1995.

[3] D. Dasgupta and D. R. McGregor. Nonstationary function optimization using the structured genetic algorithm. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature*, pages 145–154. Elsevier Science Publisher, 1992.

[4] A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors. *Parallel Problem Solving from Nature*, number 1498 in LNCS. Springer, 1998.

[5] D. E. Goldberg and R. E. Smith. Nonstationary function optimization using genetic algorithms with dominance and diploidy. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 59–68. Lawrence Erlbaum Associates, 1987.

[6] J. J. Grefenstette. Genetic algorithms for changing environments. In R. Maenner and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 137–144. North Holland, 1992.

[7] B. S. Hadad and C. F. Eick. Supporting polyploidy in genetic algorithms using dominance vectors.

[8] K. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor MI, 1975.

[9] J. Lewis, E. Hart, and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In Eiben et al. [4], pages 139–148.

[10] S. J. Louis and J. Johnson. Solving similar problems using genetic algorithms and case-based memory. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 283–290. Morgan Kaufmann, 1997.

[11] S. J. Louis and Z. Xu. Genetic algorithms for open shop scheduling and re-scheduling. In M. E. Cohen and D. L. Hudson, editors, *ISCA Eleventh International Conference on Computers and their Applications*, pages 99–102, 1996.

[12] N. Mori, S. Imanishi, H. Kita, and Y. Nishikawa. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In T. Bäck, editor, *International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 1997.

[13] N. Mori, H. Kita, and Y. Nishikawa. Adaptation to a changing environment by means of the thermodynamical genetic algorithm. volume 1141 of *LNCS*, pages 513–522. Springer Verlag Berlin, 1996.

[14] K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimization. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 159–166. Morgan Kaufmann, 1995.

[15] N. Raman and F. B. Talbot. The job shop tardiness problem: a decomposition approach. *European Journal of Operational Research*, 69:187–199, 1993.

[16] C. L. Ramsey and J. J. Grefenstette. Case-based initialization of genetic algorithms. In S. Forrest, editor, *Fifth International Conference on Genetic Algorithms*, pages 84–91. Morgan Kaufmann, 1993.

[17] C. Ryan. Diploidy without dominance. In J. T. Alander, editor, *Third Nordic Workshop on Genetic Algorithms*, pages 63–70, 1997.

[18] C. Ryan and J. J. Collins. Polygenic inheritance - a haploid scheme that can outperform diploidy. In Eiben et al. [4], pages 178–187.

[19] R. E. Smith. Diploid genetic algorithms for search in time varying environments. In *Proceedings of the Annual Southeast Regional Conference of the ACM*, pages 175–179, New York, 1987.

[20] K. Trojanowski, Z. Michalewicz, and Jing Xiao. Adding memory to the evolutionary planner/navigator. In *IEEE Intl. Conference on Evolutionary Computation*, pages 483–487, 1997.