Evolutionary Approaches to Dynamic Optimization Problems - Updated Survey -

Jürgen Branke
Institute AIFB, University of Karlsruhe
D-76128 Karlsruhe, Germany
Email: branke@aifb.uni-karlsruhe.de

Abstract

If the optimization problem is dynamic, the goal is no longer to find the extrema, but to track their progression through the space as closely as possible. This paper surveys a number of techniques that have been published in the literature in order to make evolutionary algorithms suitable for changing optimization problems.

1 Introduction

Most research in evolutionary computation focuses on optimization of static, non-changing problems. Many real world optimization problems however are actually dynamic, and optimization methods capable of continuously adapting the solution to a changing environment are needed.

The main problem with standard evolutionary algorithms used for dynamic optimization problems appears to be that EAs eventually converge to an optimum and thereby loose their diversity necessary for efficiently exploring the search space and consequently also their ability to adapt to a change in the environment when such a change occurs.

Over the past few years, a number of authors have addressed this problem in many different ways, most of those could be grouped into one of the following categories:

1. the EA is run in standard fashion, but as soon as a change in the environment has been detected, explicit actions are taken to increase diversity and thus to facilitate the shift to the new optimum.
2. convergence is avoided all the time and it is hoped that a spread-out population can adapt to changes more easily.
3. the EA is supplied with a memory to be able to recall useful information from past generations, which seems especially useful when the optimum repeatedly returns to previous locations.
4. multiple subpopulations are used, some to track known local optima, some to search for new optima.

The following sections will present typical examples for each of the above mentioned categories. Due to the tight space restrictions however, this survey is necessarily incomplete and restricted to few important aspects.

2 Increasind Diversity After a Change

A simple restart of the EA after a change in the environment has been detected would of course be the simplest option to deal with changes. However, if one assumes that the changes of the problem are relatively small, it is likely that the new optimum will be in some sense related to the old one. In that case one should be able to do better than simple restart by transferring knowledge from the old population to the new initial population, e.g. by transferring individuals.

Hypermutation [8], for example, keeps the whole population after a change, but increases population diversity by drastically increasing the mutation rate for some number of generations.

A variant thereof, called *Variable Local Search* (VLS)[30], increases mutation gradually after a change in the environment has been detected. In [31] a learning strategy to adapt the range of mutation is suggested.

If the dynamics of the optimization problem affect the genetic representation, it is not possible to simply keep old individuals, the individuals have to be adapted.

For example in job shop scheduling, when new additional jobs arrive, they have to be represented in the genotype. However the adaptation of the individuals is usually rather straightforward and introduces additional variance that automatically stimulates exploration. Overall, significant improvements in convergence speed and solution quality have been found when the altered (old) individuals are reused (see e.g. [3, 4, 17, 25]).

3 Maintaining Diversity Throughout the Run

Grefenstette [14] introduced the method of *random immigrants* where in every generation, the population is partly replaced by randomly generated individuals. As opposed to strong mutations, random immigrants only affects part of the population. Thus it introduces diversity without disrupting the ongoing search process.

Andersen [1] examines the effect of genotypic and phenotypic sharing on the GA's ability to track moving optima. The idea is that, since these methods try to spread out the population over multiple peaks, they should maintain diversity in the population. And indeed Andersen concludes that sharing remarkably enhances the GA's ability to track optima in slowly changing environments.

Cedeno and Vemuri [7] use a crowding-like replacement scheme, called "Worst among Most Similar", together with a selection scheme that chooses the second parent with respect to similarity to the first parent. The authors show that this approach is capable of maintaining a number of different solutions and to adapt to new peaks appearing in the landscape.

In [12], it is suggested to modify the fitness function by taking the individual's age into account, i.e. $f_{mod} = g(f_{old}, age)$. Interestingly, g is chosen in a way that middle-aged individuals are favoured. As the authors show, this approach also helps maintaining diversity.

The basic idea behind the Thermodynamical Genetic Algorithm (TDGA) [21], is to control the diversity in the population explicitly by controlling a measure of so called "free energy" F. For a minimization problem, this term is calculated as:

$$F = \langle E \rangle - TH$$

where $\langle E \rangle$ stands for the average population fitness and TH is a measure for the diversity in the population. The temperature T is a parameter of the algorithm and reflects the emphasis on diversity (the problem of adjusting the parameter T, especially in dynamic environments, has been addressed in [22]). In every generation, the best individual is preserved as an elite, then the n individuals are paired to produce n offsprings. Next, mutation is applied to all parents and offspring. From the resulting pool of $2n+1$ individuals, individuals are selected one by one for the next generation. For this selection, in each step the energy F of the slowly forming new population is calculated assuming that individual i ($i = 1, ..., 2n+1$) would be added to the new population. The individual that minimizes F is then actually added and the process is started anew until the new population consists of n members.

4 Memory-based approaches

Supplying the EA with some sort of memory might allow it to store good (partial) solutions and reuse them later as necessary. Obviously, strategies with a memory may be especially beneficial in periodically changing environments, when there are repeated occurrences of a small set of situations. Additionally, redundant representations may slow down convergence and favor diversity.

Memory may be provided in two general ways: *implicitly* by using redundant representations, or *explicitly* by introducing an extra memory and formulating strategies to store in and retrieve solutions from it.

The most prominent approach to redundant representations seems to be multiploidy, with different implementations of the dominance mechanism (see e.g. [13, 15, 23, 27]).

Ryan [26] 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, 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.

An interesting comparative study on multiploidy has been performed by Lewis et al. [16]. 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 (reversing the dominance relation after a change), 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 diploid

approach is able to learn two solutions and switch between them almost instantaneously. If more than two targets were used however, the approach failed completely.

A quite different redundant representation scheme using a multi-level structured gene-representation has been suggested by Dasgupta and McGregor [10]. 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. However, in the experiments on the time-varying knapsack problem [10] and a moving parabola [9], relatively simple representations were chosen. Nevertheless, improvements over simple GAs were found.

While redundant representations 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 following approaches use an explicit memory in which specific information is stored and reintroduced into the population at later generations.

Louis and Xu [19], for example, look at re-scheduling an open shop problem after a machine has broken down and has been replaced by a faster machine. The memory is used to transfer individuals from one EA run to seed the initial population after a single change. A fixed number of generations between changes is assumed, and 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, 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 [18].

Ramsey and Grefenstette [24] incorporate case-based reasoning into an EA. They use a knowledge base to memorize successful individuals in a permanent memory. The system 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. 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.

Yet another storage strategy has been added to the Thermodynamical Genetic Algorithm (TDGA, c.f. [20, 21]). 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 (measured as variance over bit positions). 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). However, so far it has been defined for binary representation only and has never been evaluated per se.

In a similar setting, Branke [5] compares a number of replacement strategies for inserting new individuals into the memory. A simple replacement of the most similar individual performed almost equivalent to a strategy that replaces the worse of the two individuals in the memory closest to each other. Both strategies performed significantly better than a variance maximation scheme. Also in that paper, the importance of diversity for memory-based approaches is stressed. To allow a simple balance between exploitation and exploration, the paper suggests to divide the population into two, a "memory-based" population and a "search" population. The first uses the memory and is responsible for remembering good old solutions and maintaining a minimum quality. 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 force exploration, this second population is re-initialized after every change. The reported experiments also indicate that the advantage of a memory vanishes when the optimum does not repeatedly return to the exact previous location but to a slightly different one.

5 Multi-Population Approaches

A general problem with memory is that the stored information, like the location of peaks found, becomes obsolete as the environment changes. One possibility to reduce this problem is to maintain small subpopulations in several promising areas of the serach space which can track the peaks as they move and change, thus acting as a self-adaptive memory.

In the Self-Organizing-Scouts approach [6, ?], whenever a new promising peak has been detected, a small scout subpopulation is assigned exclusively to the area around that peak, i.e. the search space is explicitly divided into sub-regions similar to the way used in the Forking GA [28] for static problems. The scout individuals are continuously redistributed to the most promising areas. A larger base population is kept off the already discovered scout population areas, its task being to constantly search for new peaks.

A similar idea is pursued in the Multinational GAs as presented in [29], except that subpopulations are formed based on a so called "hill valley detection procedure" in which the fitness landscape is specifically probed at several locations between the best individuals of two subpopulations in order to determine whether they may be separated by a valley and thus should form different subpopulations.

The approach by Ronnewinkel and Martinez, presented in the subsequent chapter of the workshop proceedings, also falls into the category "Multi-Population Approaches".

6 Conclusion

This paper surveyed and categorized a number of different approaches to adapt standard evolutionary algorithms to handle dynamic optimization problems.

Since only very few comparisons between different approaches have been published so far, it is difficult to draw conclusions about the superiority of one approach over the other.

References

[1] H. C. Andersen. An investigation into genetic algorithms, and the relationship between speciation and the tracking of optima in dynamic functions. Honours thesis, Queensland University of Technology, Brisbane, Australia, Nov. 1991.

[2] T. Bäck, editor. *Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.

[3] C. Bierwirth and H. Kopfer. Dynamic task scheduling with genetic algorithms in manufacturing systems. Technical report, Department of Economics, University of Bremen, Germany, 1994.

[4] C. Bierwirth and D. C. Mattfeld. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, 7(1):1–18, 1999.

[5] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Congress on Evolutionary Computation CEC99*, 1999.

[6] J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck. A multi-population approach to dynamic optimization problems. In *Adaptive Computing in Design and Manufacturing 2000*. Springer, 2000.

[7] W. Cedeno and V. R. Vemuri. On the use of niching for dynamic landscapes. In *Intl. Conf. on Evolutionary Computation*. IEEE, 1997.

[8] 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.

[9] 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.

[10] 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.

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

[12] A. Ghosh, S. Tstutsui, and H. Tanaka. Function optimization in nonstationary environment using steady state genetic algorithms with aging of individuals. In *IEEE Intl. Conf. on Evolutionary Computation*, pages 666–671, 1998.

[13] D. E. Goldberg and R. E. Smith. Nonstationary function optimization using genetic algorithms with dominance and diploidy. In J. J. Grefenstette, editor, *2nd Intl. Conf. on Genetic Algorithms*, pages 59–68. Lawrence Erlbaum Associates, 1987.

[14] 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.

[15] B. S. Hadad and C. F. Eick. Supporting polyploidy in genetic algorithms using dominance vectors. In P. J. A. et al., editor, *6th Intl. Conf. on Evolutionary Programming*, volume 1213 of *LNCS*, pages 223–234. Springer, 1997.

[16] J. Lewis, E. Hart, and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In Eiben et al. [11], pages 139–148.

[17] S.-C. Lin, E. D. Goodman, and W. F. Punch. A genetic algorithm approach to dynamic job shop scheduling problems. In Bäck [2], pages 481–488.

[18] S. J. Louis and J. Johnson. Solving similar problems using genetic algorithms and case-based memory. In Bäck [2], pages 283–290.

[19] 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 11th Intl. Conf. on Computers and their Applications*, pages 99–102, 1996.

[20] N. Mori, S. Imanishi, H. Kita, and Y. Nishikawa. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In Bäck [2], pages 299–306.

[21] 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, 1996.

[22] N. Mori, H. Kita, and Y. Nishikawa. Adaptation to a changing environment by means of the feedback thermodynamical genetic algorithm. In Eiben et al. [11], pages 149–158.

[23] K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimization. In *6th Intl. Conf. on Genetic Algorithms*, pages 159–166. Morgan Kaufmann, 1995.

[24] C. L. Ramsey and J. J. Grefenstette. Case-based initialization of genetic algorithms. In S. Forrest, editor, *5th Intl. Conf. on Genetic Algorithms*, pages 84–91. Morgan Kaufmann, 1993.

[25] C. Reeves and H. Karatza. Dynamic sequencing of a multiprocessor system: a genetic algorithm approach. In R. F. Albrecht, C. R. Reeves, and N. C. Steele, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 491–495. Springer, 1993.

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

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

[28] S. Tsutsui, Y. Fujimoto, and A. Ghosh. Forking genetic algorithms: GAs with search space division schemes. *Evolutionary Computation*, 5(1):61–80, 1997.

[29] R. K. Ursem. Mutinational GAsptimization techniques in dynamic environments. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Genetic and Evolutionary Computation Conference*, pages 19–26. Morgan Kaufmann, 2000.

[30] F. Vavak, K. Jukes, and T. C. Fogarty. Adaptive combustion balancing in multiple burner boiler using a genetic algorithm with variabe range of local search. In Bäck [2], pages 719–726.

[31] F. Vavak, K. Jukes, and T. C. Fogarty. Learning the local search range for genetic optimisation in nonstationary environments. In *IEEE Intl. Conf. on Evolutionary Computation ICEC'97*, pages 355–360. IEEE Publishing, 1997.