A "Futurist" approach to dynamic environments

Jano van Hemert
Leiden University
jvhemert@liacs.nl

Clarissa Van Hoyweghen
University of Antwerp
hoyweghe@ruca.ua.ac.be

Eduard Lukschandl
Ericsson & Hewlett-Packard
eduard.lukschandl@ieee.org

Katja Verbeeck
University of Brussels
kaverbee@vub.ac.be

May 27, 2001

Abstract

The optimization of dynamic environments has proved to be a difficult area for Evolutionary Algorithms. As standard haploid populations find it difficult to track a moving target, different schemes have been suggested to improve the situation. We study a novel approach by making use of a meta learner which tries to predict the next state of the environment, i.e. the next value of the goal the individuals have to achieve, by making use of the accumulated knowledge from past performance.

1 Introduction

In this study we are interested in how Evolutionary Algorithms (EAs) behave in dynamic environments. Most of the time, EAs are used in static environments, where the fitness function does not vary over time. Of course, this is not what happens in natural evolution. Most often, when standard EAs (using a haploid representation) are tested they have difficulties tracking a moving target and tend to get stuck in local optima or lag behind the moving optimum. In (Ryan and Collins, 1999) a haploid system using Polygenic Inheritance is constructed for use in dynamic environments. In this paper,we present an alternative approach based on our definition of a dynamic or non-stationary environment.

In (Lewis et al., 1998) and (Ryan and Collins, 1999) the notion of non-stationarity problems is discussed. Lewis changes the environment every 1500 generations, whereas Ryan, following (Goldberg and Smith, 1987), does so every 15 generations. We believe that the predictability and tractability of the dynamic environment are important. When we do not have any information about what is going to happen, as is the case in a chaotic environment, we should not expect the EA to adapt to new situations immediately.

Under the assumption that the environment is somehow predictable, we propose to complement the EA with a meta learner which is responsible for

estimating the next state of the environment. Therefore the meta learner analyzes the pairs of generations and best values seen so far. The basic idea is to have two evolutionary algorithms. The first EA, called the *current evolutionary algorithm* will use the given fitness function, while the second EA, called the *future evolutionary algorithm* uses a predicted fitness function which hopefully matches the fitness function in a future time step.

The rest of the paper is structured as follows. In the next section the problem solver is described in more detail. In Section 3 the two dynamic problems we use as test cases are explained. In Section 4 experiments and results are shown. And Section 5 discusses conclusions and future work.

2   Solving dynamic problems

In this section the use of a meta learner which predicts the future environment changes is motivated and techniques that can be applied by the meta learner to predict the future fitness are described.

2.1   The problem solver

The core of the problem solver consists of two steady state EAs which evolve simultaneously and a meta learner that predicts the future environment. At each generation t, the first EA tries to find good elements with respect to the current fitness function, while the second EA tries the same for the future fitness function predicted by the meta learner, e.g. the predicted fitness function at generation $t + \Delta$. The two EAs, respectively called the *current evolutionary algorithm* and the *future evolutionary algorithm*, use the same selection and reproduction operators. Every d steps, a set of individuals migrates from the future population to the current population (see Figure 1). In this way, we hope that the current population is prepared for changes that will occur in the environment Δ generations later.
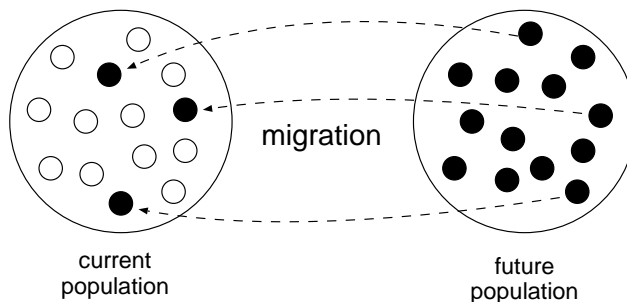

Figure 1: Migration of members of the future population into the current population

The migration step in the problem solver works as follows. First select the m best individuals from the future population and copy them to the current population. Then reevaluate the current population and resize it to its initial size by removing the m worst individuals. Figure 1 shows the pseudo code of the problem solver.

Algorithm 1 Pseudo code of the problem solver

```
for t = 0 to T-1
    future_EA.do_one_generation();
    if time_to_migrate(t, delta) then migrate(m);
    current_EA.do_one_generation();
    info = meta_learner.learn();
    pred = metalearner.predict(t+delta, info);
    future_EA.change_fitness(pred);
```

2.2   Using future predictions

If the solver knows the goal we have to achieve in the future it can start creating solutions which will be useful later on. Hopefully this will improve the accuracy of our solver. The idea is that our solver will evolve two populations of candidate solutions. First, a population that optimizes the problem of the current situation. Second, a population that optimizes the problem in the future. Given that we use a steady state EA that is optimizing a function, we need for our second population a fitness function which matches the fitness function in the future. Hence we need some kind of prediction to set our goal. The idea is to create a fitness function that predicts the fitness of an individual in the future using the knowledge of the problems that we have seen so far. More accurate, by analyzing the pairs of generations and best values we hope to find a function that predicts the future best value.

2.3   Mining the past

When the steady state EA is running we record the best individual for each generation. These individuals will be used to make predictions later. There are many ways of creating a predictor, for example we can use genetic programming, linear regression, neural networks, Fourier transformations and Walsh coefficients. Here we take a different approach and test our solver using a perfect predictor and a noisy predictor. Both will be discussed in Section 4.
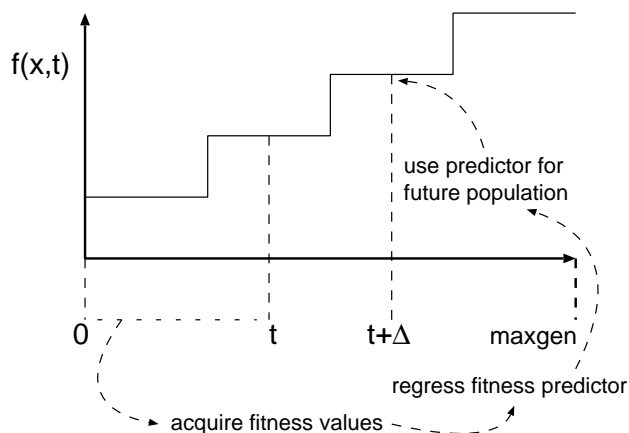
3

Figure 2: The general idea of regressing a predicting fitness function to be used in another population that should prepare for the future

3 Two dynamic problems

The problem solver is tested on two different types of dynamic problems, the 0-1 knapsack problem (Goldberg and Smith, 1987) and the Ošmera's dynamic problem (Ošmera et al., 1997). In this section we present a short description of both problems.

3.1 The knapsack problem

The aim of the knapsack problem is to fill a knapsack with objects, all having a certain value of interest and a weight. As there is a maximum weight the knapsack can hold, the problem is to choose among the available objects a subset which maximizes the value of the objects without violating the maximum weight. The problem is made dynamic by changing over time the maximum weight the knapsack can hold. In our experiments, the maximum weight is varied from 80% of the total weight of the objects to 50% of the total weight and then back to 80% and so on. This change happens every 15 generations. The objects and their associated weights and values for the problem are depicted in Table 1.

The knapsack problem can be represented by a binary string with as string length the number of available objects. If the value at string position i is one (respectively zero) the object with number i is added (respectively not added) to the knapsack. If the total weight of all items in the knapsack is more than the allowed weight, the string has fitness value zero; otherwise, the string has as fitness value the sum of the values of all items in the knapsack.

Table 1: Objects with associated weights and values for the 0-1 Knapsack problem

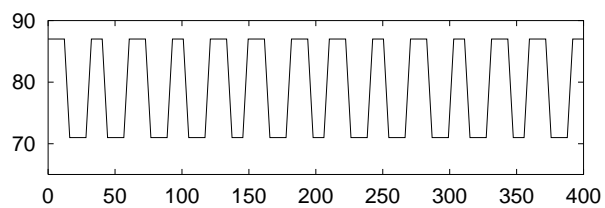| object	value	weight	object	value	weight
0	2	12	9	10	7
1	3	5	10	3	4
2	9	20	11	6	12
3	2	1	12	5	3
4	4	5	13	5	3
5	4	3	14	7	20
6	2	10	15	8	1
7	7	6	16	6	2
8	8	8			

Figure 3: Optimum of the dynamic knapsack problem

3.2 Ošmera's dynamic problem domain I

Ošmera's dynamic problem is specified by the following function:

$$g_1(x,t) = 1 - e^{200(x-c(t))^2}$$

with $c(t) = 0.04(\lfloor t/20 \rfloor)$, $x \in \{0.000, \ldots 2.000\}$ and each time step $t \in \{0, \ldots 1000\}$ equal to one generation. The aim of the problem is to find the value x at time t such that $g_1(x,t)$ is minimized. Figure 4 shows the Ošmera's dynamic problem domain and the behavior of $c(t)$ over time. The problem can be represented by a binary string of length 31 which is normalized to yield a value between 0 and 2.
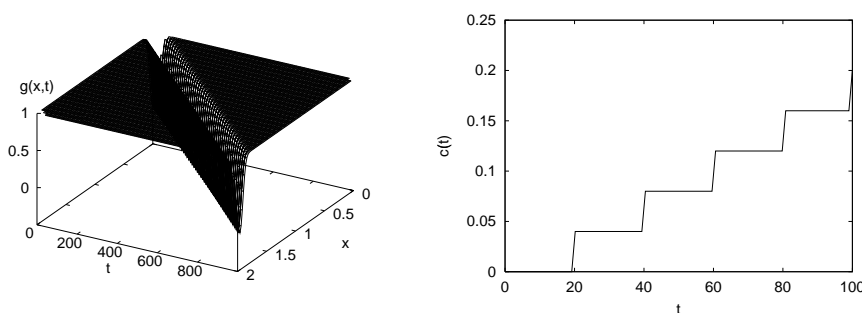

Figure 4: Ošmera's dynamic problem domain I (left) and behavior of $c(t)$ over time (right)

4 Experiments

To test our concept, the performance of our problem solver is tested in two extreme situations. First, the performance of the problem solver is checked given a perfect view of the future. In this way the idea of preparing the current population by using a future population is validated. Second, the performance of the problem solver using random or deceptive values as a predictor is compared with the performance of a normal EA without prediction. In this way we want to see whether the use of a predictor can be harmful.

The EA used in the experiments is a steady state algorithm using two-point crossover, mutation with rate $\frac{1}{l}$ and linear ranking selection (Whitley, 1989) with bias 1.5. For all experiments the population size is set to 100 and children replace older individuals in the population by using worst replacement selection.

4.1 Perfect predictor

To check the performance of the problem solver using a perfect predictor we actually do not use a predictor at all, but instead we give our solver the correct

6

optimal value at the future time step. At any time t the current population is evaluated with $f(t)$ and the future population is evaluated with $f(t + \Delta)$.

4.1.1 0-1 Knapsack Problem

Table 2 gives the results of the knapsack problem for different lookahead times and different migration numbers. The error is calculated by

$$\frac{|\text{ max} - \text{best fitness }|}{\text{max}} \cdot 100,$$

with best fitness the fitness of the best individual. In all experiment the migration step d is set to one. Figure6 shows the fitness of the best individual during the best run of the knapsack problem. The different plots vary in population size, migration number and lookahead time.

The experiments show a slightly better result using future prediction if Δ is set to 5. Unfortunately for Δ set to 15 the problem solver with perfect future predictions performs worse than a normal EA. This is an unexpected result as we imagined that $\Delta = 15$ would help the algorithm best because Δ is equivalent to the period of change in the problem. The experiments show that the settings for the parameter Δ is very important.

Table 2: Error percentages for the 0-1 Knapsack problem over 50 runs with a bitstring representation

predictor	Δ	m	error	stdev	best run
none	$\times$	$\times$	16.6%	3.52	8.96%
perfect	5	10	11.9%	3.77	4.85%
perfect	15	10	20.3%	4.26	11.7%
perfect	5	50	11.8%	3.70	5.97%
perfect	15	50	21.4%	6.06	12.0%
noisy	5	10	12.6%	4.12	6.77%
noisy	15	10	13.0%	3.74	7.50%
noisy	5	50	12.7%	4.02	6.47%
noisy	15	50	12.8%	4.07	4.99%

4.1.2 Ošmera's Function

Table 3 shows the results of the dynamic Ošmera Problem for different lookahead times Δ and different migration numbers m. The error is calculated by

$$|\text{ min} - \text{best fitness }| \cdot 100 = |\text{ 0} - \text{best fitness }| \cdot 100,$$

with best fitness the fitness of the best individual. Figure 8 shows the average checking error over all generations for the Ošmera function.

If we compare the performance of the problem solver using a perfect predictor with the performance of a normal EA (Figure 5 right), we see (as probably could be expected) that for all tested settings the problem solver performs much better.

Table 3: Error percentage for the Ošmera problem for 50 runs

predictor	Δ	m	error	stdev	best run
none	×	×	63.8%	10.3	41.2%
perfect	5	10	0.261%	0.153	0.0751%
perfect	5	50	0.168%	0.148	0.0266%
perfect	10	10	0.241%	0.220	0.0680%
perfect	10	50	0.203%	0.099	0.0698%
noisy	5	10	0.241%	0.186	0.0488%
noisy	5	50	0.144%	0.122	0.0358%
noisy	10	10	0.241%	0.186	0.0488%
noisy	10	50	0.168%	0.148	0.0266%

4.2   Noisy predictor

To check how harmful the use of a predictor can be, the performance of the problem solver using noisy and deceitful values as a predictor is compared with the performance of a normal EA without prediction.

For the knapsack problem the maximum size of the sack at the wrong point in the future is used as noisy prediction, which makes it a deceptive goal. Instead of supplying the correct value we take the other maximum size. For the Ošmera function all individuals in the future population are assigned a random fitness value between 0 and 1. Figure 7 shows the fitness of the best individual during the best run of the knapsack problem. Figure 9 shows the average checking error over the best run of the Ošmera function. The experiments show that the use of a noisy predictor does not harm the search process. Even more the migration of bad predicted individuals increases the diversity of the current population which makes it easier for the population to adapt to the changing environment. In case of the Ošmera function the error percentages for noisy prediction are equally good as those for perfect prediction. In case of the knapsack problem the error percentages for the noisy predictor are even better than those for the perfect predictor. It seems that in these cases prediction is not really necessary. However, adding more diversity to the population, e.g. by migrating some individuals from an other population, helps the population to adapt to changes of the dynamic environment.

8

5   Discussion

Our results show that in case of the the Ošmera problem setting, perfect future prediction as well as noisy future prediction helps the current population to adapt to changes of the environment. However for the knapsack problem the results are not so promising. More experiments on different problems should be done to evaluate the concept of future prediction. It would be interesting to study theoretically the symbiosis of the current and the future population. Hereby we think on the importance and the influence of the parameter Δ and m in the migration step and the rate of change of the environment.

6   Future work

Although we have not that many results yet, we hope that our idea of predicting the future using a meta learner will prove to be useful. Despite the fact that we did not perform well on the dynamic knapsack problem our results on the Ošmera's function have given us the feeling that the concept might actually work.
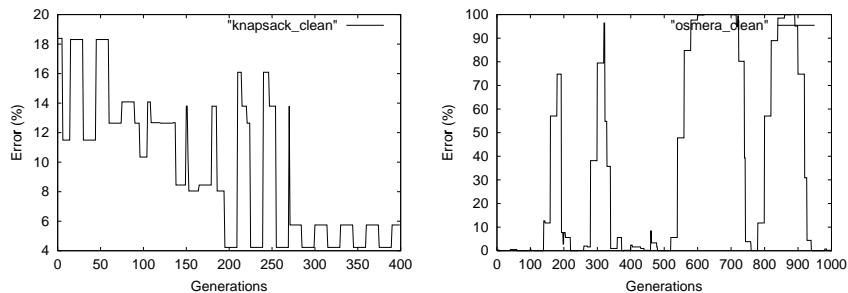
Figure 5: Error percentage of the best individual during the best run of the knapsack problem and the Ošmera problem without prediction
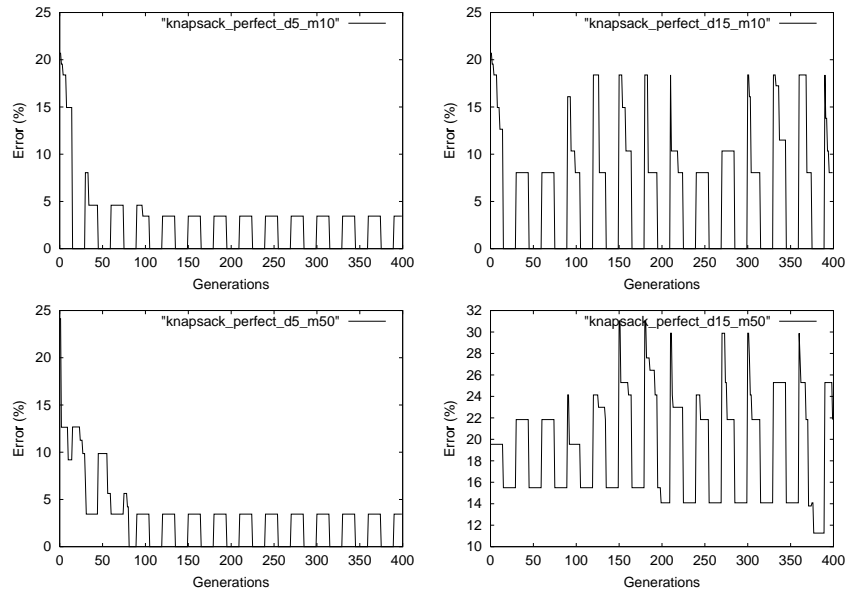
Acknowledgments

9

Figure 6: Error percentage of the best individual during the best run of the knapsack problem with perfect prediction
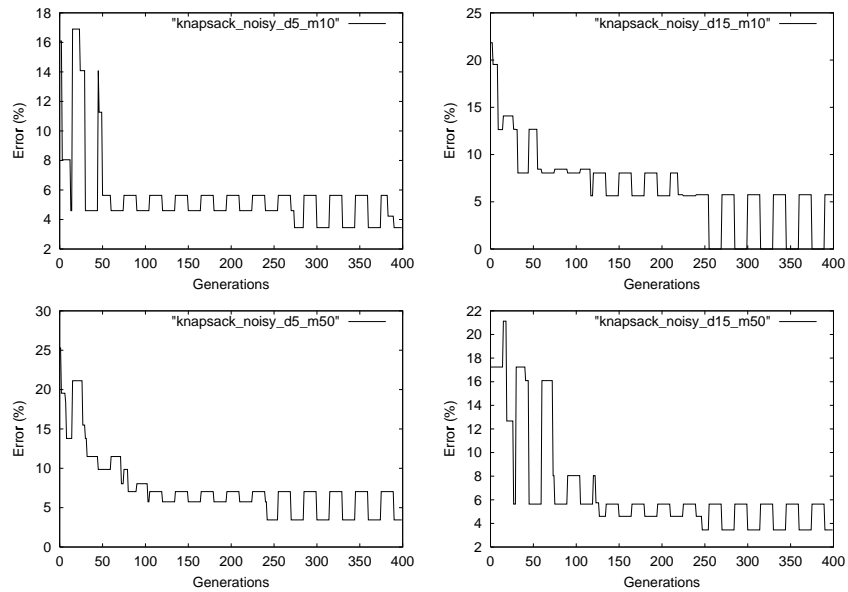

Figure 7: Error percentage of the best individual during the best run of the knapsack problem with noisy prediction
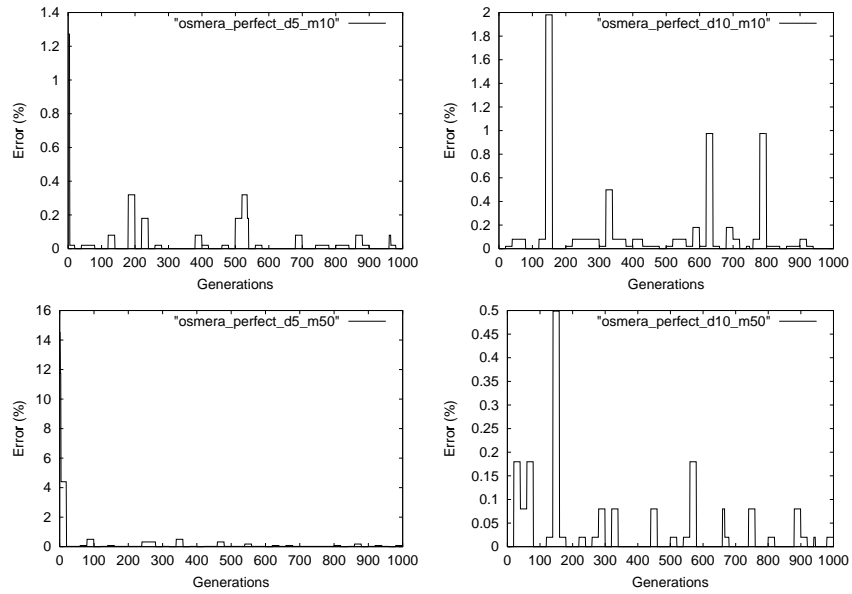
10

Figure 8: Error percentage of the best individual during the best run of the Ošmera problem with perfect prediction
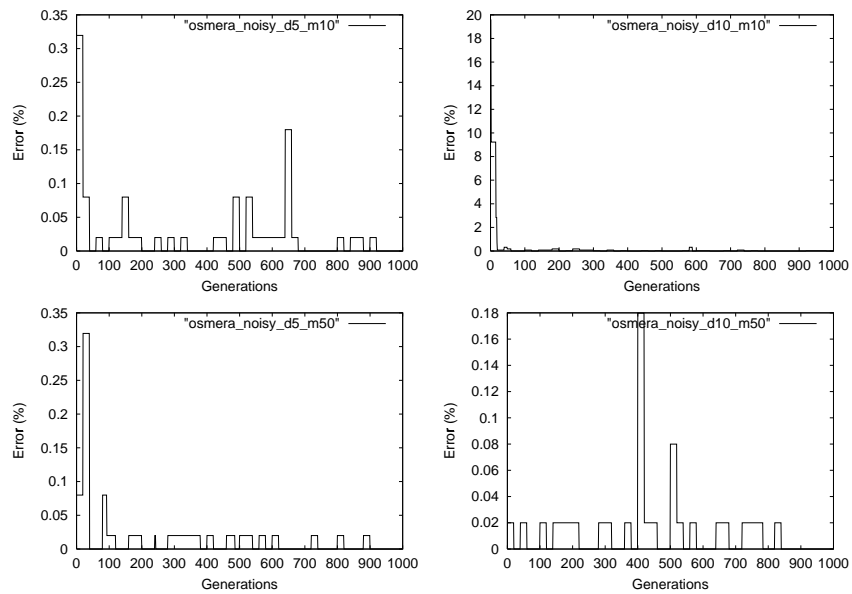

Figure 9: Error percentage of the best individual during the best run of the Ošmera problem with noisy prediction

References

Goldberg, D. and Smith, R. (1987). Nonstationary function optimization with dominance and diploidy. In Grefenstette, J., editor, *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*, pages 59–68. Lawrence Erlbaum Associates.

Lewis, J., Hart, E., and Ritchie, G. (1998). A comparison of dominance mechanisms and simple mutation on nonstationary problems. In Eiben, A. E., Back, T., Schoenauer, M., and Schwefel, H.-P., editors, *Proceedings of Parallel Problem Solving from Nature*, number 1498 in Lecture Notes in Computer Science, pages 139–148. Springer, Berlin.

Ošmera, P., Kvasnicka, V., and Pospichal, J. (1997). Genetic algorithms with diploid chromosomes. In *Proceedings of Mendel '97*, pages 111–116.

Ryan, C. and Collins, J. (1999). Non-stationary function optimization using polygenic inheritance. In *Submitted to IEEE Transactions on Evolutionary Computation*.

Whitley, D. (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 116–123, San Mateo, California. Morgan Kaufmann Publishers, Inc.