Two Adaptive Mutation Operators for Optima Tracking in Dynamic Optimization Problems with Evolution Strategies

Claudio Rossi
Claudio.Rossi@upm.es

Antonio Barrientos
Antonio.Barrientos@upm.es

Jaime del Cerro
j.cerro@upm.es

Departamento de Automatica, Ingenieria Electronica e Informatica Industrial
Universidad Politecnica de Madrid
Madrid, 28006, Spain

ABSTRACT

The dynamic optimization problem concerns finding an optimum in a changing environment. In the tracking problem, the optimizer should be able to follow the optimum's changes over time. In this paper we present two adaptive mutation operators designed to improve the following of a time-changing optimum, under the assumption that the changes follow a non-random law. Such law can be estimated in order to improve the optimum tracking capabilities of the algorithm. For experimental assessment, a (1,λ) evolution strategy has been applied to a dynamic version of the sphere problem.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods

General Terms

Algorithms

Keywords

Evolution strategies, dynamic optimization, tracking problem, time-varying fitness function.

1. INTRODUCTION

Real-world applications must often face a highly noisy and changing environment. Evolutionary Algorithms (EAs) are known to be well suited for dealing with noisy input information, and can cope with a changing environment, since the search is performed by constantly (at each new generation) producing new candidate solutions. Thus, they inherently consider time: newly generated individuals can be subject to time-changing constraints and/or evaluated and selected according to a time-varying fitness function. Since new candidate solutions are generated using information coming from the current individuals, at each change the problem is not solved form scratch, but knowledge regarding the previous search space is actually used in order to find the new optimum. Whether such knowledge is useful depends on the nature of the change [13]. In many real-world applications such changes are smooth and respond to physical laws (consider e.g. the case of visual-based object tracking applications). In such cases, accumulating information on past can be effectively used in order do direct the search for the current optimum.

In the recent years there has been a growing interest in dynamic and time varying problems. Most of the work on such problems is aimed at studying techniques that maintain a suitable degree of population diversity, either when a change is detected or continuously throughout the run, in order to guarantee an adequate level of exploration and to avoid the algorithm to concentrate on the current optimum. As several authors have pointed out, an excessive convergence would make the algorithm miss important changes in the environment, while a spread-out population can adapt to changes more easily. The literature on dynamic optimization has focused on dealing with the changing environment by means of dynamic parameters control and specialized adaptive operators, (multi)population control and memory-based approaches.

Examples of the first approach are parameters control [2], [4], and specialized adaptive operators [11], also making use of local search techniques [19] or information theoretic based methods [1]. The latter try to get an insight of the problem identifying relevant substructures and exploit them to respond to changes in the environment. In [20] three different self-adaptive mutation operators are compared on a two-dimensional dynamic problem.

Population control methods include random migrations [12], multi-populations approaches [9] and niching [10], and are aimed at ensuring a sufficient population diversity that allow the discovery of new optima (or promising regions) in the case there is a change in the environment.

Memory-based approaches [7] are aimed at keeping track of good (partial) solutions in order to re-use them in periodically changing environments. A case-based reasoning approach has been proposed by [17] in order to recognize environments and, in case of change, reuse individuals that have proven to be successful in similar environments. Another memory-based approach is presented in [15], inspired by thermodynamical principles.

Exhaustive reviews and bibliography on dynamic optimization problems can be found in [1], [8] and [13].

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

The approach presented in this work belongs to the category of specialized adaptive operators, and is based on the consideration that in many real-world applications the changes are not random and are governed by a determined dynamic system, and can therefore be learnt. Such information can be provided to the EA in order to improve its tracking capabilities. Distinguishing characteristic of our evolution strategy is that the dynamical law is learnt explicitly by an ad-hoc external mechanism, which is then used to predict future states of the dynamic system. Predictions are incorporated into the EA by means of two mutation operators. Once predictions have sufficiently refined, individuals that are close to the estimated future position can be generated in order to help the algorithm to keep up with the moving optima, anticipating its movement. In fact, such individuals will be generated slightly ahead of the position at current time, and are thus expected to be closer to the new optimum's position. This has the further advantage to prevent the generation of individuals that are closer to the optimum, but behind it w.r.t. the direction of movement. Such individuals could in fact be optimal at current time, but sub-optimal in the near future.

Related work

A recent paper [6] theoretically discusses *time-linkage*, i.e. the fact that decisions that are made at a certain time t on the basis of maximizing a certain score, may influence the maximum score that can be obtained in the future. In order to avoid sub-optimal states caused by missing information about the future, the authors propose the use of learning techniques to predict it and help the global optimization problem, and demonstrate the goodness of their approach on two mathematical problems. In a similar way, we propose the use of a learning mechanism capable of providing predictions on the future state of a dynamic system. However, the goals pursued by the prediction in time-linkage and in optimum tracking are different. Under time-linkage the meaning of optimality is only well defined over a time-interval. Because optimization over a future time-interval is needed, prediction is required. In our case there is no time-linkage. The focus is on tracking the optimum as it moves over time, and the prediction mechanism serves to predict where the optimum is moving to. In this way the difference between the actual optimum and the one found by the EA is smaller. Therefore, the number of evaluations needed to find an good optimum for the current time is lower.

In [3], an analysis of a $(\mu/\mu, \lambda)$-ES with cumulative step length adaptation [16] is presented. The evolution strategy presented therein also relies on a mechanism for accumulating information on the search process, and to control the mutation parameter according to the accumulated information. The $(\mu/\mu, \lambda)$ evolution strategy is popular due to its proven good performances and mathematical understanding. A brief description of such strategy will be provided in Section 5. We will use this evolution strategy for comparison with our approach.

The learning technique we propose is based on the powerful mechanism of *Kalman filters* [14].

In [18] the use of a fitness-filtering technique based on an extended Kalman filter is proposed in order to improve the tracking capabilities of a Genetic Algorithm. The Kalman-extended Genetic Algorithm is based on the consideration that in a dynamic environment the fitness of an individual is affected by uncertainty, since from the time the individual has been evaluated, the function might have changed. An additional parameter, computed using a Kalman-based mechanism on the basis of past fitness evaluations, is added to each individual of the population, encoding the uncertainty associated to its fitness. Such mechanism provides a mean to acquire knowledge regarding the changing environment, and it is used in order to determine the proportion of new individuals to be generated and whether an individual should be reevaluated. The main difference with our approach is that while [18] applies filtering to the fitness of the individuals, we apply filtering to individual genes, in order to gather information on their dynamics, and try to exploit such information to deduce the underlying law ruling their changes. Then, forecasts can be made based on the deduced law in order to direct the search towards promising regions.

The rest of the paper is organized as follows: in the following section we introduce the motion estimation technique adopted. Then, Section 3 describes the mutation operators proposed that incorporate the prediction. Section 4 describes the $(\mu/\mu, \lambda)$-ES and the cumulative step length adaptation mechanism. Sections 5 and 6 describe the sphere model used to test the proposed evolutionary strategy and the experiments performed. Section 7 concludes the paper.

2. ESTIMATING MOTION

Consider an EA where candidate solutions are real-valued arrays. Such individuals, as well as the optimum solution, represent a point on the n-dimensional search space, whose position change over time. The *state* of a candidate solution at time t is the conjunction of its position and velocity, i.e. the rate at which its position changes. The state of the optimum is approximated by the state of the current best solution provided by the EA.

The state of the optimum is governed by a dynamical system. In general, the state of a system can only be partially observed. In our case, the observable part of the state is given by the EA, which provides an approximate position of the solution. The whole current state can be estimated with the current observable part and by considering the sequence of the previous states, taking into account that they can be affected by noise.

A powerful technique that performs such an estimation is the *Kalman filter* [14]. The Kalman filter is a set of mathematical equations that provides an efficient computational way to estimate the state of a dynamic system, given a sequence of noisy observations and a model of the system (dynamic law), and it is widely used in control systems engineering and tracking applications. The Kalman filter is a recursive estimator, which means that only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state. Kalman filters also provide an estimation of the *future* state of a system and a measure of the goodness of the estimation: the model is used to predict which will be the new state of the system, and then this is corrected with the observation of the real state in order to give a more accurate estimation. Although in this work we will focus on first order models (constant velocity, see Section 5), the idea can be extended to more complex motion laws.

Throughout the run, the ES provides to the Kalman filter a series of observations of the state of the system (corre-

sponding to the best individual in generation $0 \dots t-1$). Using a model of the motion law, the filter adapts its estimation of the velocities in order to match the observed positions, and after a sufficient number of iterations, is able to provide an estimation of state of the of the system at time t, i.e. an estimation of the position and velocity vectors $\hat{p}_t, \hat{v}_t$. Moreover, it provides an estimation of their precision, in terms of variances $(\hat{\sigma}_t^p)^2, (\hat{\sigma}_t^v)^2$.

We are interested in the values $\hat{p}_t$ and $\hat{\sigma}_t^p$, that give an indication of where the next optimum will most likely be, and a measure of how much confident is the filter in its prediction. In the following, we will refer to such values as the *prediction* and its *accuracy*.

2.1 The Kalman filter

In the following, a brief description of the Kalman filters is presented.

The Kalman filter estimates a system by using a feedback control: the filter estimates the process state, and then obtains feedback in the form of (noisy) measurements. The kind of dynamical system considered by Kalman filters is the following;

$$\begin{aligned} x_t &= Ax_{t-1} + Cu_{t-1} + w_{t-1}, \\ z_t &= Hx_t + v_t. \end{aligned} \tag{1}$$

where $x_t \in \mathbb{R}^n$ is the state of the system at time t, A is the state transition model ($n \times n$ matrix) and C is an optional control input model. The value $z_t \in \mathbb{R}^m$ is the observation of the system (measurement) at time t and $H \in \mathbb{R}^{m \times n}$ is the model thay relates the measurement to the state, since not necessarily all variables of the state are being observed: in our case we only measure position, while the state also contains velocities. The quantities w and v are random variables that represent the process and measurement noise, and are assumed to have a Gaussian distribution with covariance matrices $Q \in \mathbb{R}^n$ and $R \in \mathbb{R}^m$, respectively ($p(w) \sim N(0, Q), p(v) \sim N(0, R)$).

The model provides an *a priori* estimation of the new state x_t^-. The new state is computed correcting the a priori estimation with the real observation z_t through the *Kalman gain* matrix $K_t \in \mathbb{R}^{n \times m}$ according to the formula:

$$\hat{x}_t = x_t^- + K_t(z_t - Hx_t^-)$$

The gain matrix is computed recursively and balances the two terms x_t^- and z_t according to their respective covariance.

To begin to work, the filter needs an initial state x_0, an initial error covariance matrix P_0, usually set to a default value, and the matrices Q and R, usually measured during the early testing of the system.

Then, the filter loops through a series of steps. The first step is to generate the a priori estimate using the model and to calculate the a priori error covariance P_t^-. The a priori estimation and the a priori covariance provide the values we refer to as the *prediction* and its *accuracy* throughout the paper.

Then, the measurement update phase takes place, first computing the Kalman gain K_t. The matrix K_t is computed in such a way that minimizes the a posteriori error covariance P_t. The next step is to actually measure the process (obtain the observation z_t, in our case corresponding to the best individual at time t), and then to generate an a posteriori state estimate x_t by incorporating the measurement, which is the estimation usually employed in practical applications. The final step is to obtain an a posteriori error covariance estimate P_t. Note how the corrected values for x_t and P_t will be used in the next time step to predict the new state and its error, hence the model actually incorporates the information coming from the measurement to correct its predictions and adapt to the measurements.

In our case, $x_t = [p_t, v_t]^T$, and a first order model can be written as

$$x_t = Ax_{t-1}, \qquad A = \begin{bmatrix} I_n & \Delta t \cdot I_n \\ 0 & I_n \end{bmatrix} \tag{2}$$

I_n being the $n \times n$ identity matrix, $\Delta t = 1$ and there is no control input, i.e. $C = [0]$.

The computational cost of the Kalman predictor is essentially due to matrix algebra and a matrix inversion, which has a polynomial computational cost.

The main disadvantage of filters is the need of off-line setting of initial noise (co)variances for optimal performances. In this work we used fixed pre-calculated values, estimated during early testing of the algorithm.

3. INCORPORATING MOTION INFORMATION TO THE EA

The information provided by the estimator can be used in order to bias the exploration, directing it towards the region where the new optimum will most likely be, according to the prediction. The technique proposed in this work consists in the design two new mutation operators.

A mutation operator modifies one or more genes of a selected individual. Such modification can be done by ignoring the previous value (*new random gene* value), or performing some operation on the original value of the gene (*perturbation*), the simplest operation being adding or subtracting a small random quantity to the gene. Instead of using random quantities, the predicted values can be used to mutate existing individuals. In the first case, the new random gene will be generated directly in the region indicated by the prediction, while in the second the perturbation will be biased in the direction of the prediction (cf. Fig. 1). In this way, the mutation operator adapts to the changing environment thanks to the hints provided by the motion estimation mechanism.

3.1 The Kmut-N and Kmut-P operators

Let s be the chromosome selected for mutation and s' the chromosome after mutation. Given that all the genes of the optimum can change at each generation, the proposed operators act on all the genes $i = 1 \dots n$, n being the number of genes (chromosome length).

Let $\hat{p}$ and $\hat{\sigma}$ be the predicted position and its accuracy.

The Kmut-N mutation operator

The Kmut-N operator sets the value of the genes ignoring their previous values. If the prediction was fully trusted, one could let $s'_i = \hat{p}_i$, and the new value of the i^{th} gene would equal to the predicted one. However, since predictions are not perfect, a better strategy is to generate a value in the neighborhood of the prediction. Since the accuracy value provide a measure of the confidence of the prediction, it can

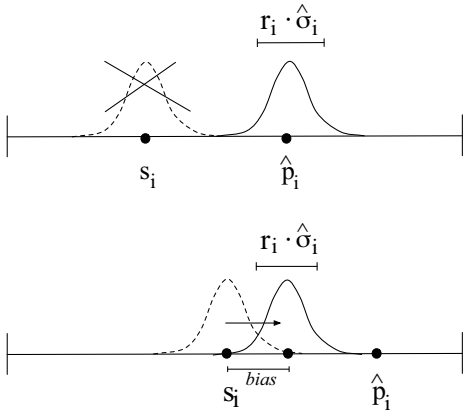

Figure 1: New random gene and perturbation. The index i refers to the $i-th$ gene of the individual, and the time subscripts are dropped.

be used to compute the range of such neighborhood (see Figure 1, top):

$$s'_i = N(\hat{p}_i, r_i \cdot \hat{\sigma}_i) \tag{3}$$

$r_i > 0$ being a parameter that acts on the size of the range, and that can be either fixed at design time or controlled at run time according to some heuristic rule. In this way, the more accurate is the prediction, the closer to it the values will be generated.

The Kmut-P mutation operator

The Kmut-P mutation is a perturbation operator. A common perturbation strategy is the Gaussian perturbation centered in 0 and with a certain standard deviation r_i, $N(0, r_i)$. In our case, the perturbation is desired to generate values that take s'_i close to the estimation $\hat{p}_i$, depending on the accuracy $\hat{\sigma}_i$, and that are in a range also depending of the accuracy. The perturbation strategy that we consider in this work is

$$s'_i = s_i + pert(r_i, \hat{p}_i, \hat{\sigma}_i) \tag{4}$$

$$pert(r_i, \hat{p}_i, \hat{\sigma}_i) = \left[\frac{1}{1+\hat{\sigma}_i} \cdot (\hat{p}_i - s_i)\right] + N(0, r_i \cdot \hat{\sigma}_i)$$

where the first term is a *direction bias* (Fig. 1, bottom), that gives the amount of shift we want to give to the perturbation, in the direction of the prediction. Bad prediction accuracy produces a low bias: if the prediction is considered not good, it must not be taken too much into consideration. The second term determines the perturbation range, making it bigger or smaller according to the accuracy of the prediction. As for the Kmut-N operator, the parameter r_i controls the range of the perturbation.

3.2 Discussion

In principle, when the optimum has been found by the Evolutionary Algorithm, and a sufficient number of tracking steps have been performed to let the estimator get enough information on the dynamics, one could let the estimator follow it, and stop the EA. In fact, if the current optimum and the motion estimation were perfectly determined, one could simply *compute* the new optimum. However, the uncertainty regarding both the current and the sequence of the previous optima makes such option unfeasible.

In general, we do not know when the EA found a satisfactory optimum and tracking should begin. Moreover, the estimator needs time to adjust, and when it can be trusted is also an issue that must be dealt with. The selection mechanism of the EA can be used to cope with these issues. The estimator can be started at the same time as the EA, providing almost useless predictions, and the individuals generated using them will most likely be sub-optimal and be discarded by the selection mechanism of the EA. While the EA evolves, the sequence of optima becomes more coherent, reflecting the fact that the algorithm got close to the optimum and began to follow it. Then, the predictions of the estimator will refine, and individuals incorporating them will take advantage of the information they carry.

Noise can heavily affect the outcome of the EA, making observations unreliable. Although estimators like Kalman filter are very good in dealing with noisy information, we do not want to fully rely on them. Moreover, the model used by the filter can be approximate. Thus, the estimator constantly needs real observations in order to adapt to the real motion laws. Also, without constant update the estimator would not detect changes in the trajectory of the moving optima: the estimator needs new observations to keep refining.

Finally, better solutions must be constantly looked for, as the algorithm could be following optimum. Even in the tracking phase, there could be better solutions than the current EA's best individual, close to it or in unexplored regions. Then, the algorithm has to be given the capacity to reach them, and not just focus on the current optimum. In other words, while the estimator is helping in the *exploitation* task, a suitable degree of *exploration* must be ensured. This can be only provided by the EA, that is also indispensable in the case the moving is lost. In fact, the EA can get stuck and not successfully follow the optimum, because of noise in the input data, or to a change of motion of the optimum not captured by the estimator's model. If this happens, the algorithm must be able to recover tracking, and this can be achieved only by the EA's exploration capabilities.

4. THE $(\mu/\mu, \lambda)$-ES WITH CSLA

The $(\mu/\mu, \lambda)$ evolution strategy consists in repeatedly updating a search point $x \in \mathbb{R}$ that is the centroid of the population of candidate solutions, using the following steps:

1. Generate λ offspring candidate solutions

   $$y_i = x + \sigma z_i, i = 1, \ldots, \lambda$$

   where the quantity $\sigma > 0$ is referred to as the *mutation strength* that determines the step length, and the z_i are vectors consisting of n independent, standard normally distributed components, referred to as the *mutation vectors*;

2. Determine the objective function values of the offspring candidate solutions and compute the average $\langle z \rangle$ of the vectors z_i corresponding to the μ best offspring. Vector $\langle z \rangle$ is referred to as the *progress vector*;

3. Update the search point according to

$$x \leftarrow x + \sigma\langle z \rangle.$$

The notation μ/μ indicates that all μ parents participate in the in the creation of every offspring candidate solution (see [5] for a formal definition). The *mutation strength* is adapted at each step using the cumulative step length adaptation algorithm (CSLA) [16]:

1. Update s according to

$$s \leftarrow (1-c)s + \mu c(2-c)\langle z \rangle;$$

2. Update the mutation strength according to

$$\sigma \leftarrow \sigma \cdot exp\left(\frac{||s||^2 - n}{2Dn}\right).$$

Vector s, initially set to 0, accumulates information on the search process, and is referred to as the *search path* or *accumulated progress vector*. The *cumulation parameter* c determines the length of the memory of the accumulation process, and D is a damping constant. Following recommendations from the literature, the cumulation parameter c and damping constant D were set in the experiments to $1/\sqrt{n}$ and $\sqrt{n}$, respectively.

5. EXPERIMENTAL SETUP

In order to perform an experimental assessment of the operators described above, we have applied them to a function known as the *dynamic sphere model*, used by several authors ([3], [4] amongst others) in order to analyze their strategies.

The dynamic sphere model

The sphere model is the set of all functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$ with

$$f(x) = g(||x^* - x||)$$

where $x^* \in \mathbb{R}^n$ is the (moving) optimum, and $g : \mathbb{R} \rightarrow \mathbb{R}$ is a strictly monotonic function of the Euclidean distance $d = ||x^* - x||$ of a candidate solution x from the target solution x^*. In our experiments, the function g is the identity function (i.e. the fitness of an individual is the euclidean distance from the target).

The motion model used for the experiments is the following [3]:

$$x^{*(t+1)} = x^{*(t)} + \delta v \tag{5}$$

where vector $v \in \mathbb{R}^n$ has unit length and represents the direction of the movement, and δ is a constant scalar and referred to as the *target speed*. In Equation (5) the vector v is independent of time, i.e. is constant throughout the run. Then, such model is a linear (first-order) motion model. In general, v can be changing in time.

The Evolutionary Algorithm

The algorithm we used to test the proposed mutation operators is a (1,λ) evolution strategy. Such choice is motivated by the nature of the problem, which is continuous and real-valued, and by the fact that since the optimum moves we do not want to include the original individual in the future generation as it is likely to become sub-optimal. Moreover, it would need to be re-evaluated as the fitness landscape has changed since it was evaluated. Three evolution strategies have been compared, two adopting a (1,λ) evolution strategy with the Kmut-N and Kmut-P mutation operators, and one adopting a $(1/1, \lambda)$ evolution strategy with cumulative step length adaptation algorithm.

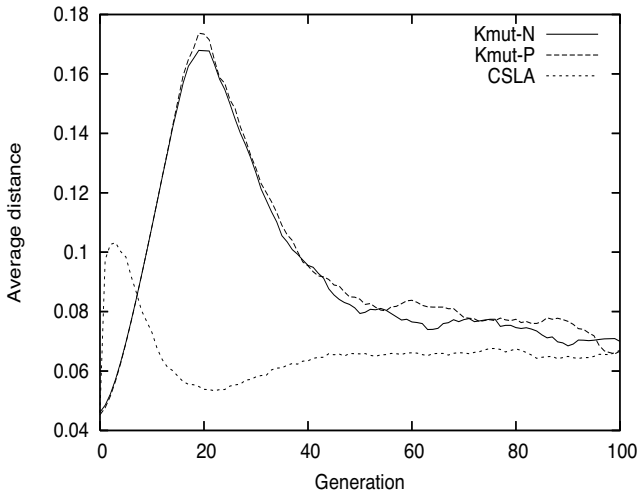

Figure 2: Comparison of the three evolution strategies: average distance from the optimum during the run (first 100 generations, n=10, λ=10, speed=0.025).

For simplicity, the Kalman filter used in the experiments is one-dimensional. In fact, since all genes of the target optimum are modified by multiplying the same value (target speed) by the corresponding component of the vector v, in this case it is sufficient to estimate the target speed by considering the motion of a single gene. Then, provided that vector v is known, all the other motion laws can be calculated. In general, if v was unknown, or if each gene is be moving independently from the others, a n-dimensional Kalman filter should be used.

The values of the array r of Eq. (3) and (4) have been set to 1 ($r_i = 1, i = 1 \ldots n$), and the process, measurement noise and error variances of the Kalman filter were set respectively to 10^{-5}, 1 and 1 (scalar values).

6. EXPERIMENTAL RESULTS

Three series of tests have been performed, for a dimension of the space (chromosome length) of $n = 10, n = 50$ and $n = 100$ and a constant $\delta \in \{0.005, 0.01, 0.025, 0.05, 0.1, 0.15, 0.2\}$. The three series involved a value of λ of 10 and each run terminated after 500 generations. For each parameters setting, one hundred independent runs of the $(1, \lambda)$-Kmut-N $(1, \lambda)$-Kmut-P and $(1/1, \lambda)$-CSLA strategies have been performed. In each run, the initial individual was generated in the neighborhood of the initial position of the target x^*, and both the target and v vectors were generated randomly with the genes in the domain $[0, 1]$. Note that this domain only refers to initial values. During the run the genes could reach higher or smaller values. For instance, supposing $n = 10, v_i = 1/n, i = 1 \ldots n, \delta = 0.1$, an increment of $\delta \cdot v_i$ for 500 steps means that the final value of each element of x^* would be incremented of 5. Assuming $x^{*(0)} = 0$, $x^{*(500)}$ will be $\sqrt{5^2 \cdot n} = 15.8$ units far from its initial position.

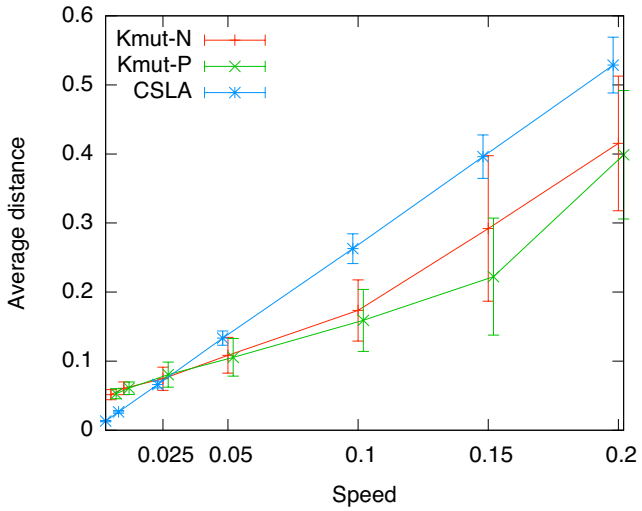

Figure 3: Comparison of the different strategies: average distance from the optimum (n=10, λ=10). The values on the x axis have been slightly shifted to avoid overlapping of standard deviation bars.

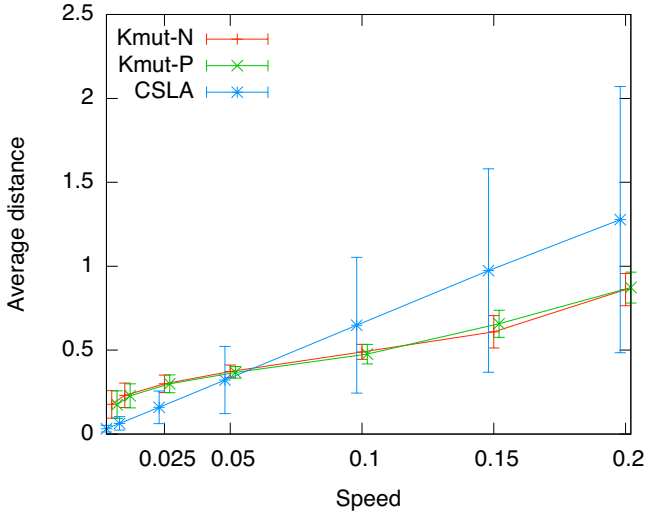

Figure 4: Comparison of the different strategies: average distance from the optimum (n=50, λ=10). The values on the x axis have been slightly shifted to avoid overlapping of standard deviation bars.

The three evolution strategies have been compared using the average distance of the best individual from the optimum, an index similar to the *offline performance* $f_{offline}$ proposed in [13] for dynamic environments. In our case the average is computed using the total number of generations, while $f_{offline}$ is computed using the total number of evaluations. Since at each generation a constant number of individuals is generated and evaluated, the two indexes are qualitatively equivalent.

Figure 2 shows an example of the evolution of the three different strategies. Note how in the beginning of the run the Kmut operators have poor performances, due to the fact that the prediction mechanism has not refined sufficiently. After approximately 50 generations the predictions start to be accurate, and the Kmut operators have better performances. In all the three strategies, after a transient period, needed by the respective mechanisms to adapt, the optimum value of the evolution strategy tends to stabilize to a constant value.

Figure 3 shows the comparison of the three strategies w.r.t. speed for the test $n = 10, \lambda = 10$ (all values are average over 100 runs). For low speeds, the strategy adopting the CSLA operator outperforms the ones adopting the Kmut operators, while for higher speeds the Kmut operators appear to behave better than the CSLA. However, the CSLA-based strategy has better performance as far as standard deviation is concerned. Figure 4 shows the comparison of the four variants w.r.t. speed for the test $n = 50, \lambda = 10$. Again, for the lowest speeds the strategy adopting the CSLA operator outperforms the ones adopting the Kmut operators, while Kmut-based strategies have better performances at high speeds. Figure 5 shows similar results for the case $n = 100$. Figures 4 and 5 also show that increasing space dimensionality appears to affect more the CSLA-based strategy that the Kmut-based ones, both in terms of average distance from the optimum and standard deviation.

Table 1 reports on the running time of the algorithms. Note how the running time only depends of n and λ. At each generation a constant number of individual is generated, and the adaptation mechanisms need a computational effort that is independent from the speed δ. Hence, the computational cost is independent form speed. Table 1 shows that the CSLA-based strategy has better performances as far as running time is concerned, due to the fact that its

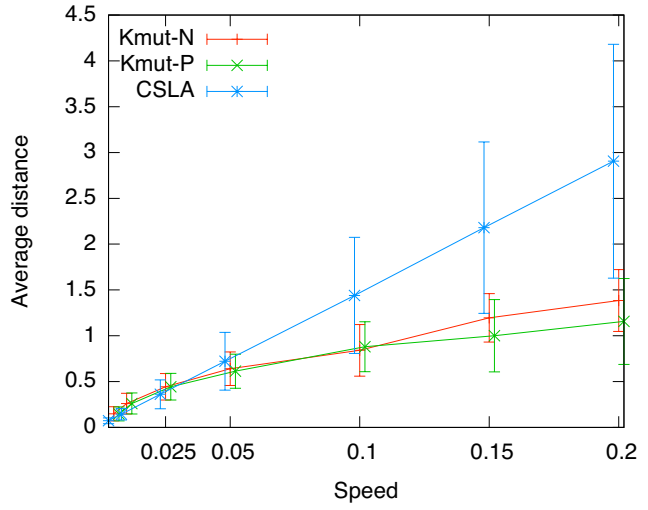

Figure 5: Comparison of the different strategies: average distance from the optimum (n=100, λ=10). The values on the x axis have been slightly shifted to avoid overlapping of standard deviation bars.

Table 1: Average running time (seconds) for a run of 500 generations, $\lambda = 10$.

n	Kmut-N	Kmut-P	CSLA
10	0.04	0.04	0.01
50	0.07	0.07	0.03
100	0.11	0.11	0.07

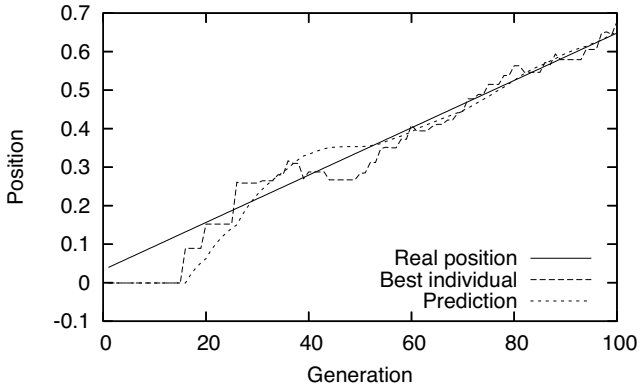

Figure 6: Evolution of a gene during the first 100 generations (Kmut-P strategy, speed=0.025, n=100, $\lambda = 10$).

adaptation mechanism has a lower computational complexity than the Kalman filter.

Finally, Figure 6 depicts a detail of the evolution of a gene using the Kmut-P strategy, showing the real value, the corresponding gene of the current best individual and the value predicted by the Kalman filter. The best solution evolves in steps (each step a good mutation) keeping up with the real moving value, while the predicted value moves smoothly and, after refining, (fifty generations approximately) very close to the real value.

7. CONCLUSION

We have presented two mutation operators designed for tracking a moving optimum in dynamical optimization problems making use of information provided by a prediction mechanism. The prediction mechanism is based on the assumption that in real world applications changes are not random and can be learnt. Such information can be provided to the EA in order to improve its tracking capabilities. The idea underlying the two variants of the proposed mutation operator is to generate individuals that are close to the estimated future position, in order to help the algorithm to keep up with the moving optima, anticipating its movement.

The learning and prediction mechanism we adopted is based on Kalman filters.

The experiments conducted indicate that the predictions actually help to improve the tracking of the moving optimum, and have shown performances comparable with other adaptation mechanisms. For high speeds the proposed operators outperformed the cumulative step length adaptation mechanism, although the latter has shown smaller execution times, due to a lower computational cost.

The computational cost of the Kalman predictor is essentially due to matrix algebra and a matrix inversion, which has a polynomial computational cost of the order of n^3. However, the added cost of the adaptation mechanism can be acceptable, depending on the application, assuming that the computational costs of the optimization process are dominated by the costs of evaluating the objective function.

The main disadvantage of the proposed approach is that the kind of prediction mechanism we adopted needs an off-line tuning prior to its application, in order to estimate initial parameters (noise covariance). Future work will be aimed at computing such parameters on-line on the basis of the actual behavior of the algorithm.

The focus of this work was on tracking, focusing the search around a region containing the estimated future position of the optimum. However, concentrating the search around the current optimum in a dynamic environment could let the algorithm miss important changes in different regions of the search space. Coupling predictions with some diversity-control technique can be a promising strategy for taking advantage of the improved exploitation capabilities provided by the predictor, while maintaining a suitable degree diversity needed for exploration.

Acknowledgments

The work of the first author has been carried out under a "Ramon y Cajal" research fellowship from the Ministerio de Ciencia y Tecnología of Spain.

The authors would also like to thank the anonymous reviewers for their useful comments that allowed a significant improvement of this paper.

8. REFERENCES

[1] H. A. Abbass, K. Sastry, and D. E. Goldberg. Oiling the wheels of change: The role of adaptive automatic problem decomposition in non-stationary environments. Technical Report IlliGAL Report No. 2004029, Illinois Genetic Algorithms Laboratory, 2004.

[2] P. J. Angeline. Tracking extrema in dynamic environments. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, editors, *International Conference on Evolutionary Programming*, pages 335–345. Springer Verlag, 1997.

[3] D. V. Arnold and H.-G. Beyer. Optimum tracking with evolution strategies. *Evolutionary Computation*, 14(3):291–308, 2006.

[4] T. Bäck. On the behavior of evolutionary algorithms in dynamic environments. In D. B. Fogel, H.-P. Schwefel, T. Back, and X. Yao, editors, *IEEE International Conference on Evolutionary Computation*, pages 446–451. IEEE Press, 1998.

[5] H. G. Beyer. *The Theory of Evolutionary Strategies*. Natural Computing Series. Springer-Verlag, Heidelberg, 2001.

[6] P. A. N. Bosman. Learning, anticipation and timedeception in evolutionary online dynamic optimization. In *Fourth Workshop on Evolutionary Algorithms for Dynamic Optimization Problems (EvoDOP-2005)*, pages 39–47, 2005.

[7] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Congress on Evolutionary Computation*, volume 3, pages 1875–1882. IEEE Press, 6-9 1999.

[8] J. Branke. Evolutionary approaches to dynamic optimization problems. In J. Branke and T. Bäck, editors, *Evolutionary Algorithms for Dynamic Optimization Problems*, pages 27–30, 7 2001.

[9] J. Branke, T. Kaußler, l Schmidt, and H. Schmeck. A multi-population approach to dynamic optimization

problems. In *Adaptive Computing in Design and Manufacturing 2000*, pages 299–308, 2000.

[10] W. Cedeno and V. R. Vemuri. On the use of niching for dynamic landscapes. In *Congress on Evolutionary Computation*, pages 361–366. IEEE Press, 1997.

[11] H. G. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report AIC-90-001, Navy Center for Applied Research in Artificial Intelligence, Washington, DC, 1990.

[12] J. Grefenstette. Genetic algorithms for changing environments. In R.Männer and B.Manderick, editors, *Parallel Problem Solving from Nature*, pages 137–144. Elsevier Science Publisher, 1992.

[13] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments - a survey. *IEEE Transactions on evolutionary computation*, 9(3):303–317, 2005.

[14] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[15] 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*, pages 299–306. IEEE Press, 1997.

[16] A. Ostermeier, A. Gawelczyk, and N. Hansen. Step-size adaptation based on non-local use of selection information. In Y. Davidor, H.-P. Schwefel, and R. Männerm, editors, *Parallel Problem Solving from Nature PPSN III*, pages 189–198. Springer-Verlag, 1994.

[17] C. L. Ramsey and J. J. Grenfenstette. Case-based initialization of genetic algorithms. In S. Forrest, editor, *International Conference on Genetic Algorithms*, pages 84–91. IEEE Press, 1993.

[18] P. D. Stroud. Kalman-extended genetic algorithm for search in nonstationary environments with noisy fitness evaluations. *IEEE Transactions on Evolutionary Computation*, 5(1):66–77, 2001.

[19] F. Vavak, T. C. Fogarty, and K. Jukes. A genetic algorithm with variable range of local search for tracking changing environments. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, pages 376–385. Springer, 1996.

[20] K. Weicker and N. Weicker. On evolution strategy optimization in dynamic environments. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Congress on Evolutionary Computation*, volume 3, pages 2039–2046. IEEE Press, 1999.