Estimation of Fitness Landscape Contours in EAs

Petr Pošík
Czech Technical University in Prague
Dept. of Cybernetics
Technická 2, 166 27 Prague 6, Czech Republic
posik@labe.felk.cvut.cz

Vojtěch Franc
Fraunhofer-FIRST.IDA
Kekuléstrasse 7, 12489 Berlin, Germany
fravoj@first.fraunhofer.de

ABSTRACT

Evolutionary algorithms applied in real domain should profit from information about the local fitness function curvature. This paper presents an initial study of an evolutionary strategy with a novel approach for learning the covariance matrix of a Gaussian distribution. The learning method is based on estimation of the fitness landscape contour line between the selected and discarded individuals. The distribution learned this way is then used to generate new population members. The algorithm presented here is the first attempt to construct the Gaussian distribution this way and should be considered only a proof of concept; nevertheless, the empirical comparison on low-dimensional quadratic functions shows that our approach is viable and with respect to the number of evaluations needed to find a solution of certain quality, it is comparable to the state-of-the-art CMA-ES in case of sphere function and outperforms the CMA-ES in case of elliptical function.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization; G.1.2 [**Numerical Analysis**]: Approximation—*approximation of surfaces and contours, nonlinear approximation*; I.2.6 [**Artificial Intelligence**]: Learning—*concept learning, induction*

General Terms

Algorithms, Design, Experimentation, Performance, Theory

Keywords

evolutionary computation, estimation of distribution algorithms, learnable evolution model, function optimization, separating ellipsoid

1. INTRODUCTION

Many algorithms used for real-parameter black-box optimization use Gaussian distribution to sample new points. This approach started with mutative evolutionary strategies (ES) which were soon equipped with a *feedback adaptation* of the step size (Rechenberg's one fifth rule), and *self-adaptation* of the step size, coordinate-wise step sizes, and self-adaptation of the whole covariance matrix (see e.g. [12]). However, Rudolph [9] showed that self adaptive mutations can lead to premature convergence.

Other algorithms that use Gaussian distribution very often fall into the class of estimation of distribution algorithms (EDAs) [7]. They select better individuals and fit the Gaussian distribution to them—usually by means of the maximum likelihood estimation which is far from ideal (see Fig. 1(a) for an example of Estimation of Multivariate Normal distribution Algorithm (EMNA)). Without imposing limits on the minimal 'size' of the Gaussian, the variance of the distribution in the direction of the fitness function gradient quickly decreases, and the algorithm thus can get stuck even on the slope of the fitness function [3].

One way of overcoming this drawback of maximum likelihood estimation in real-valued EDAs is to estimate *the distribution of the selected mutation steps* instead of *the distribution of selected individuals* (cf. 1(a) and 1(b)). Pošík [8] applied this approach in a co-evolutionary manner. Hansen at al. [5] use similar principles in the evolutionary strategy with covariance matrix adaptation (CMA-ES) which is considered to be the state-of-the-art in real-valued black-box optimization. It adapts the step size separately from the 'directions' of the multivariate Gaussian distribution. The adaptation is based on accumulation of the previous steps that the algorithm made.

Auger et al. [1] proposed a method of improving the CMA-ES covariance matrix adaptation using a quadratic regression model of the fitness function in the local neighborhood of the current point. Their approach, however, required in D-dimensional space at least $\frac{D(D+3)}{2}+1$ data vectors to learn the quadratic function. Moreover, it assumed that each point has its fitness value, i.e. it cannot use selection schemes based on pure comparisons of two individuals.

In this paper, we propose a novel algorithm for learning the Gaussian distribution by modeling the fitness landscape contour line that lies between the selected and discarded individuals (see Fig. 1(c)). If the population does not surround a local optimum, the resulting Gaussian distribution should fit into the local neighborhood to much greater extent compared to Gaussian learned with CMA-ES-like algorithms. We present a modified perceptron algorithm that finds an elliptic decision boundary if it exists. If it does not exist, the algorithm will not stop. From this, the main limitation of our approach immediately follows: this preliminary algorithm is able to optimize only convex quadratic functions. Despite of that, it serves as a proof-of-concept and forms a strong basis for the development of more capable learning algorithm.

We provide the main principle of the method and the computational details in section 2. Section 3 describes the experiments we have carried out to assess the very basic properties of the proposed method. Section 4 presents the results of the comparison of our method against the CMA-ES algorithm. Section 5 summarizes the paper, points out the main advantages of the method and discusses the directions of future work on all the things that have to be done before the algorithm is generally applicable.

2. PRINCIPLE AND METHODS

The basic principle of the proposed method is illustrated in Fig. 1(c). After evaluation of the population, we try to model the contour line of the fitness function with an ellipse that would allow us to discriminate between the selected and discarded individuals. The decision boundary is of the form $\mathbf{x}\mathbf{A}\mathbf{x}^T + \mathbf{x}\mathbf{B} + \mathbf{C} = 0$, where $\mathbf{x}$ is a D-dimensional row vector representing a population member, $\mathbf{A}$ is a positive definite $D \times D$ matrix, $\mathbf{B}$ is a column vector with D elements, and C is a scalar.

After finding the quadratic decision function, we need to turn it into the sampling Gaussian distribution. Auger et al. [1] discussed that setting the covariance matrix $\mathbf{\Sigma}$ to $\mathbf{\Sigma} = \mathbf{A}^{-1}$ is a very good (if not optimal) choice. We follow this approach since the elliptic decision boundary then corresponds to certain contour line of the Gaussian density function. The candidate members of the new population are then sampled from this distribution.

We shall not learn the elliptic function directly—we shall use a variation of the perceptron algorithm that generally finds a *linear* decision function. To learn an ellipsoid we shall map the points to a different space and then map the learned linear function back into the original space where it shall form the ellipsoid. Then, we shall turn this ellipsoid into a Gaussian distribution, and we shall also modify the Gaussian to ensure that a specified ratio of the sampled points will lie inside the ellipsoid. The following subsections introduce methods used to accomplish the process sketched above.

2.1 Quadratic Mapping

We need to learn a quadratic function which would allow us to discriminate between two classes of data points. The classifier is then given as

$$C(\mathbf{x}) = \begin{cases} 1 & \text{iff } \mathbf{x}\mathbf{A}\mathbf{x}^T + \mathbf{x}\mathbf{B} + \mathbf{C} > 0 \\ 2 & \text{iff } \mathbf{x}\mathbf{A}\mathbf{x}^T + \mathbf{x}\mathbf{B} + \mathbf{C} < 0 \end{cases}. \tag{1}$$

The decision boundary $\mathbf{x}\mathbf{A}\mathbf{x}^T + \mathbf{x}\mathbf{B} + \mathbf{C} = 0$ is required to be a hyperellipsoid which is a *special case of quadratic function*, but, as was already stated, we shall approach that problem with a method that is designed to find a *linear* decision boundary. In order to be able to do that, we have to use a coordinate transform such that if we fit the linear decision boundary in the transformed space, we can transform it back and get a quadratic function. This process is sometimes referred to as *the basis expansion* [6] or *feature space straightening* [10].

1. Transform the points $\mathbf{x}$ from the input space to points $\mathbf{z}$ in the quadratically mapped feature space using Eq. 3.
2. Find the vector $\mathbf{w}$ defining the linear decision boundary in feature space.
3. Reorder the elements of vector $\mathbf{w}$ into matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ using Eq. 4.

Figure 2: Learning quadratic decision boundary

The matrix $\mathbf{A}$ is symmetric, i.e. $a_{ij} = a_{ji},\ i,j \in \langle 1, D\rangle$. We can rewrite the decision boundary to the following form:

$$\begin{array}{lllllllll} a_{11}x_1x_1 & + & 2a_{12}x_1x_2 & + & \dots & + & 2a_{1D}x_1x_D & + & \\ & + & a_{22}x_2x_2 & + & \dots & + & 2a_{2D}x_2x_D & + & \\ & & & & & & \dots & + & \\ & & & & & + & a_{DD}x_Dx_D & + & \\ b_1x_1 & + & b_2x_2 & + & \dots & + & b_Dx_D & + & \\ & & & & & + & c & = & 0 \end{array} \tag{2}$$

This equation defines a quadratic mapping *qmap* which for each point $\mathbf{x}$ from the input space creates a new, quadratically mapped point $\mathbf{z}$, where

$$\begin{aligned} \mathbf{z} &= qmap(\mathbf{x}) = \\ &= (x_1^2,\, 2x_1x_2,\, \dots,\, 2x_1x_D,\, x_2^2,\, \dots,\, 2x_2x_d,\, \dots,\, x_D^2, \\ &\quad\ x_1,\, \dots,\, x_D,\, 1) \end{aligned} \tag{3}$$

Then, if we arrange the coefficients a_{ij}, b_i, and c into a vector $\mathbf{w}$ so that

$$\begin{aligned} \mathbf{w} &= (a_{11},\, a_{12},\, \dots,\, a_{1D},\, a_{22},\, \dots,\, a_{2D},\, \dots,\, a_{DD}, \\ &\quad\ b_1,\, \dots,\, b_D,\, c), \end{aligned} \tag{4}$$

we can write the decision boundary as $\mathbf{z}\mathbf{w}^T = 0$ and the whole classifier as

$$C(\mathbf{x}) = C(\mathbf{z}) = \begin{cases} 1 & \text{iff } \mathbf{z}\mathbf{w}^T > 0 \\ 2 & \text{iff } \mathbf{z}\mathbf{w}^T < 0 \end{cases}, \tag{5}$$

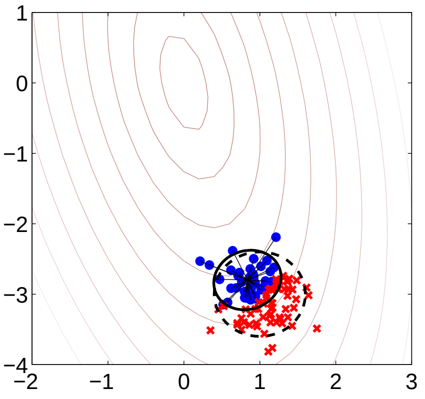
The dimensionality of the feature space is easily computed as the number of terms in Eq. 2: we have $\frac{D(D+1)}{2}$ quadratic terms, D linear terms, and 1 constant term. This gives $\frac{D(D+3)}{2} + 1$ dimensions.

The learning of a quadratic decision boundary can be carried out by the process sketched up in Fig. 2.

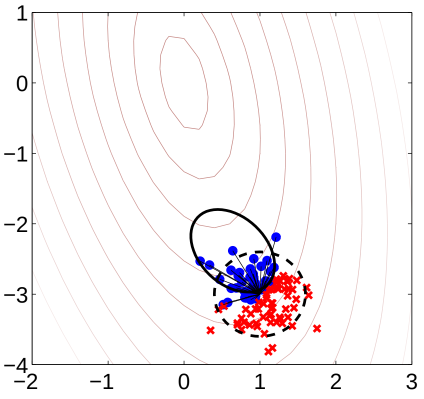
2.2 Separating Hyperplane

There are many ways to learn a separating hyperplane. In this paper, the well-known perceptron algorithm is used. The reason for this decision is the fact that in case of the perceptron algorithm we found a relatively easy way to ensure that the learned linear function will correspond to a quadratic function with positive definite matrix $\mathbf{A}$ in the original space (see Sec. 2.3).

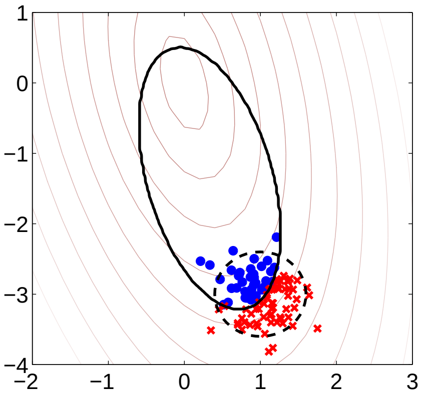
The perceptron algorithm can be stated as follows. We have training vectors $\mathbf{z}_i \in \mathbf{Z}$ of the form $\mathbf{z}_i = (z_{i1}, \dots, z_{iD}, 1)$, each of them is classified into one of the two possible classes, $C(\mathbf{z}_i) \in \{1, 2\}$. We search for a $(D+1)$-dimensional weight vector $\mathbf{w}$ so that $\mathbf{z}_i\mathbf{w}^T > 0$ iff $C(\mathbf{z}_i) = 1$ and $\mathbf{z}_i\mathbf{w}^T < 0$ iff $C(\mathbf{z}_i) = 2$. In other words, we search for a hyperplane that separates the two classes and contains the coordinate origin. The algorithm is presented in Fig. 3.



(a) EMNA estimates the Gaussian from *selected points* using maximum likelihood method. It is highly prone to premature convergence even on the slope of the fitness function.



(b) CMA-ES-like algorithm estimates the Gaussian from *selected mutation steps* using maximum likelihood method. It fights the premature convergence very well, but the learned Gaussian generally does not fit to the fitness landscape contour lines.



(c) Elliptic classifier learns the Gaussian by estimating the fitness landscape contour line directly. Generally, the fit is much closer.

Figure 1: An example of learning the covariance matrix using various methods. Contour lines illustrate a 2D quadratic function. Generating Gaussian (---) is used to sample new points which are then divided to selected (•) and discarded (×) points which are in turn used to learn the generating Gaussian for the next generation (—).

1. Initialize the weight vector: $\mathbf{w} = \mathbf{0}$
2. Invert points in class 2: $\mathbf{z}_i = -\mathbf{z}_i$ for all i where $C(\mathbf{z}_i) = 2$
3. Find the training vector with minimal projection onto the weight vector $\mathbf{w}$: $\mathbf{z}^* = \arg\min_{\mathbf{z}\in\mathbf{Z}}(\mathbf{zw})$.
4. If the minimal projection is positive, $\mathbf{z}^*\mathbf{w}^T > 0$, the separating hyperplane is found and the algorithm finishes.
5. If the minimal projection is not positive, $\mathbf{z}^*\mathbf{w}^T \leq 0$, the separating hyperplane was not found yet and the point $\mathbf{z}^*$ is the one with the greatest error. Adapt the weight vector using this point: $\mathbf{w} = \mathbf{w} + \mathbf{z}^*$.
6. Go to step 3.

Figure 3: The perceptron algorithm

Of course, the algorithm will not stop if the two classes of *qmap*-ed vectors are not linearly separable, i.e. if the original vectors are not separable by a quadratic decision boundary.

2.3 Ensuring Ellipticity

Previous sections showed how to learn a quadratic decision boundary by mapping the training vectors into the quadratic space, finding a linear decision boundary, and rearranging the elements of the weight vector $\mathbf{w}$ into matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. However, this quadratic decision function might not be elliptic, i.e. the matrix $\mathbf{A}$ might not be positive definite.

The perceptron algorithm described in Fig. 3 is basically an algorithm for the satisfaction of constraints given in the form of linear inequalities. The usual set of constraints that must be satisfied is $\mathbf{z}_i\mathbf{w}^T > 0$ for all i. If we found a way to describe the requirement of positive definiteness of the matrix $\mathbf{A}$ in the form of similar inequalities, and if we were able to find vectors that violate these inequalities, we could use only slightly modified perceptron algorithm to learn an elliptic decision boundary. Such a way exists and is described in the following paragraphs.

As shown in Sec. 2.1, the quadratic form can be written using a linear function:

$$\mathbf{xAx}^T + \mathbf{xB} + \mathbf{C} = \mathbf{zw}^T\,. \tag{6}$$

Matrix $\mathbf{A}$ is positive definite iff the condition $\mathbf{xAx}^T > 0$ holds for all non-zero vectors $\mathbf{x} \in \mathcal{R}^{1\times D}$. In order to write the condition of positive definiteness $\mathbf{xAx}^T > 0$ in terms of the weight vector $\mathbf{w}$, we define a 'pure' quadratic mapping *pqmap* for the vectors $\mathbf{x}$ where only the quadratic elements are present while the D linear elements and 1 absolute element are substituted with zero:

$$\mathbf{q} = pqmap(\mathbf{x}), \tag{7}$$

where

$$q_i = \begin{cases} z_i & \text{iff} \quad i \in \langle 1, \frac{D(D+1)}{2} \rangle \\ 0 & \text{iff} \quad i \in \langle \frac{D(D+1)}{2} + 1, \frac{D(D+3)}{2} + 1 \rangle \end{cases}, \tag{8}$$

and

$$\mathbf{z} = qmap(\mathbf{x}). \tag{9}$$

Using any D-dimensional vector $\mathbf{x}$ and its transformed variant $\mathbf{q} = pqmap(\mathbf{x})$, the following two conditions are equivalent:

$$\mathbf{x}\mathbf{A}\mathbf{x}^T > 0 \iff \mathbf{q}\mathbf{w}^T > 0 \tag{10}$$

Furthermore, all eigenvalues of any positive definite matrix are positive. If we perform the eigendecomposition of matrix $\mathbf{A}$ and get negative eigenvalues, then the related eigenvectors $\mathbf{v}$ violate the condition for positive definiteness, i.e. $\mathbf{v}\mathbf{A}\mathbf{v}^T = \mathbf{q}\mathbf{w}^T \leq 0$, where $\mathbf{q} = pqmap(\mathbf{v})$. These *pqmap*-ed eigenvectors can thus be used to adapt the weight vector $\mathbf{w}$ in the same way as ordinary *qmap*-ed data vectors. A modified version of the perceptron algorithm that ensures the positive definiteness of the resulting matrix $\mathbf{A}$ is shown in Fig. 4.

1. Initialize the weight vector: $\mathbf{w} = \mathbf{0}$
2. Invert points in class 2: $\mathbf{z}_i = -\mathbf{z}_i$ for all i where $C(\mathbf{z}_i) = 2$
3. Find the training vector with minimal projection onto the weight vector $\mathbf{w}$: $\mathbf{z}^* = \arg\min_{\mathbf{z}\in\mathbf{Z}}(\mathbf{z}\mathbf{w})$.
4. Arrange the first $\frac{D(D+1)}{2}$ elements of vector $\mathbf{w}$ into a matrix A and find its minimal eigenvalue λ^* and corresponding eigenvector $\mathbf{v}^*$.
5. If both the minimal projection and the minimal eigenvalue are positive, $\mathbf{z}^*\mathbf{w}^T > 0$ and $\lambda^* > 0$, the separating hyperplane is found and the algorithm finishes.
6. If the minimal projection is lower than the minimal eigenvalue, $\mathbf{z}^*\mathbf{w}^T < \lambda^*$, adapt the weight vector using the vector with greatest error: $\mathbf{w} = \mathbf{w} + \mathbf{z}^*$.
7. If the minimal eigenvalue is lower or equal to the minimal projection, $\mathbf{z}^*\mathbf{w}^T \geq \lambda^*$, adapt the weight vector using the *pqmap*-ed eigenvector for the sake of ellipticity: $\mathbf{w} = \mathbf{w} + pqmap(\mathbf{v}^*)$.
8. Go to step 3.

Figure 4: The modified perceptron algorithm

Again, the algorithm will not stop if the original vectors are not separable by an elliptic decision boundary.

2.4 From Quadratic Function to Gaussian Distribution

The quadratic function $\mathbf{x}\mathbf{A}\mathbf{x}^T + \mathbf{x}\mathbf{B} + \mathbf{C}$ learned by the modified perceptron algorithm is not defined uniquely; all functions of the type $k(\mathbf{x}\mathbf{A}\mathbf{x}^T + \mathbf{x}\mathbf{B} + \mathbf{C})$, $k \neq 0$, have the same decision boundary (see Fig. 5 for an illustration in 1D case).

One reasonable way of the standardization (assuming matrix $\mathbf{A}$ is positive definite) is to fix the function value at the minimum of the function. The minimum lies in the point

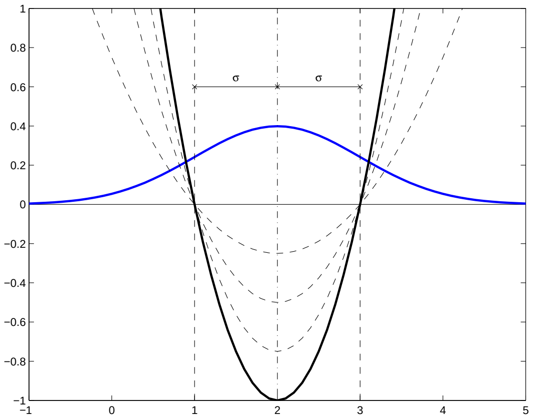


Figure 5: Non-uniqueness of the quadratic decision function. All of them define the same decision boundary.

$\mu = -\frac{1}{2}(\mathbf{A}^{-1}\mathbf{B})^T$. We deliberately chose the function value at the minimum to be -1, i.e. the following equation must hold:

$$k(\mu\mathbf{A}\mu^T + \mu\mathbf{B} + \mathbf{C}) = -1 \tag{11}$$

$$k = -\frac{1}{\mu\mathbf{A}\mu^T + \mu\mathbf{B} + \mathbf{C}} \tag{12}$$

The matrices defining the standardized quadratic function are then given as $\mathbf{A}_S = k\mathbf{A}$, $\mathbf{B}_S = k\mathbf{B}$, and $\mathbf{C}_S = k\mathbf{C}$.

The multivariate Gaussian distribution $N(\mu, \mathbf{\Sigma})$ which will be used to sample new points is then given by $\mu = -\frac{1}{2}(\mathbf{A}^{-1}\mathbf{B})^T$ and $\mathbf{\Sigma} = \mathbf{A}_S^{-1}$.

2.5 Sampling from Gaussian Distribution

Sampling from the Gaussian distribution with the center μ and covariance matrix $\mathbf{\Sigma}$ is rather a standard task. The distribution, however, suffers from the curse of dimensionality in such a way that the proportion of generated vectors that lie inside the separating ellipsoid varies (drops with increasing dimensionality).

Suppose we have a set of vectors generated from D-dimensional standardized Gaussian distribution. Each of the coordinates has unidimensional standardized Gaussian distribution and their sum of squares has a χ^2 distribution with D degrees of freedom, χ^2_D. Thus, if we wanted to specify the percentage p, $p \in (0, 1)$, of vectors lying inside the separating ellipsoid we can employ the inverse cumulative distribution function of the χ^2 distribution, $CDF^{-1}_{\chi^2_D}$, in a way that is described in step 2 of the sampling algorithm in Fig. 6.

This modification of the sampling algorithm can be considered a counterpart to the step size adaptation mechanism in the CMA-ES. Although it is based on different basis, *it modifies the size of the Gaussian.* Furthermore, we should note that by fixing the percentage of points lying inside the separating hyperellipsoid, the search becomes more local with increasing dimensionality.

2.6 Relations to Other Algorithms

Support Vector Machine (SVM) [11] is very successful and popular method for solving classification and regression tasks. They combine two techniques:

1. Eigendecompose the covarinace matrix so that $\mathbf{\Sigma} = \mathbf{R}\mathbf{\Lambda}^2\mathbf{R}^T$, where $\mathbf{R}$ is a rotational matrix of eigenvectors and $\mathbf{\Lambda}^2$ is a diagonal matrix of the eigenvalues, i.e. $\mathbf{\Lambda}$ is a diagonal matrix of standard deviations in individual principal axes.

2. Modify the standard deviations using the critical value of the χ^2 distribution,
$$\mathbf{\Lambda} = \frac{\mathbf{\Lambda}}{\sqrt{CDF^{-1}_{\chi^2_D}(p)}}. \tag{13}$$

3. Generate the desired number of vectors $\mathbf{x}_S$ from the standardized multivariate Gaussian distribution $N(\mathbf{0}, \mathbf{I})$.

4. Rescale them using the standard deviations $\mathbf{\Lambda}$ and rotate them using $\mathbf{R}$, i.e. $\mathbf{x}_C = \mathbf{x}_S\mathbf{\Lambda}\mathbf{R}$.

5. Decenter the vectors $\mathbf{x}_C$ using the center μ, i.e. $\mathbf{x} = \mathbf{x}_C + \mu$.

Figure 6: Sampling algorithm

1. *Maximum margin separating hyperplane.* When trying to find a linear function discriminating between two classes of observations, it is advantageous to take into account only those observations that lie close to the boundary of the two classes, and to maximize the distance of the separating hyperplane from these points (to maximize the margin) [13]. This task was formulated in the form of quadratic programming (is thus solvable by standard quadratic programming solvers) and has a unique solution. Later in [4], this method was extended to handle even the non-separable case.

2. *Kernel trick.* The maximum margin separating hyperplane is still only a linear function. When trying to make it non-linear, one could transform the observations from the original space to a non-linearly mapped high dimensional feature space (e.g. to the quadratic one, as it is done in this paper), find a linear decision boundary in that space, and then map it back to the original space, which gives a non-linear decision boundary (e.g. quadratic, or elliptic, as it is done in this paper). The kernel trick [2] is a way of doing the same without having to map the points from the original space to the high dimensional feature space. Any linear algorithm based on the dot products of the observations can be made non-linear just by replacing the dot product with a kernel. Kernel is a function which takes two observations, and produces a single number. If the kernel function fulfills certain conditions, it can be shown that it actually computes a dot product of the observations in a non-linearly mapped high-dimensional feature space.

Combining these two techniques, we get the SVM. In fact, SVM with a quadratic kernel function can learn much better elliptic boundary than our modified perceptron algorithm (see Fig. 7 for comparison) without the need to map the points to the quadratic space. The reason why we do not use it instead of the modified perceptron algorithm is that this algorithm produces general *quadratic* decision function, while we need a special case of it—quadratic function with positive definite matrix $\mathbf{A}$ in order to be able to invert it.

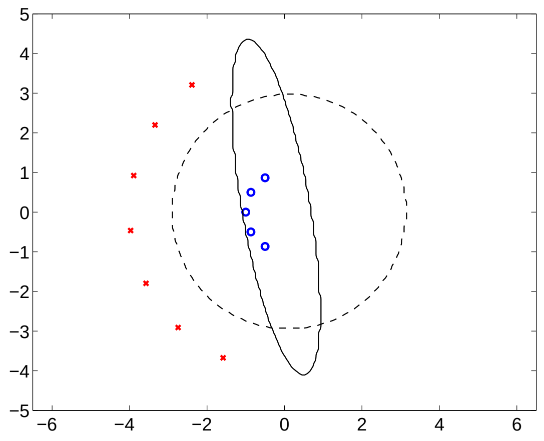


Figure 7: The difference in decision boundaries found by the modified perceptron algorithm (solid line) and the support vector machine (dashed line).

Learnable Evolution Model (LEM) The underlying principle of the method presented in this paper (modeling the contour line of the fitness function landscape that discriminates the selected and discarded individuals, and to turn the part of model describing the promising individuals into a probabilistic model from which new points are sampled) is our original idea. Nevertheless, later it turned out that it can be described as a special case of the learnable evolution model [15]. Current known implementations of LEM, however, use almost exclusively the AQ21 classification rules [14].

3. EMPIRICAL EVALUATION

To assess the basic characteristics of the algorithm, we chose two quadratic fitness functions, *spherical* and *ellipsoidal*:

$$f_{\text{sphere}} = \sum_{d=1}^{D} x_d^2 \tag{14}$$
$$f_{\text{elli}} = \sum_{d=1}^{D} (10^6)^{\frac{d-1}{D-1}} x_d^2 \tag{15}$$

3.1 Evolutionary Model

The evolutionary algorithm used in the experiments is the following:

1. Initialize the population of size N and evaluate it

2. Assign the discarded individuals (the worse half of the population) to class 1, the selected individuals (the better half of population) to class 2.

3. Map the population to quadratic feature space, use the modified perceptron algorithm to find a linear decision boundary given by vector $\mathbf{w}$, and rearrange its coefficients into matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$.

4. Turn the quadratic model into Gaussian distribution, and modify its eigenvalues so that the ellipsoid contains the desired proportion of new individuals.

Table 1: **Best population sizes for both test problems**

Dimension	2	4	6	8
CMA-ES	6	8	9	10
Our method, Sphere	9	8	7	6
Our method, Ellipsoidal	11	10	8	6

5. Sample $N-1$ new individuals from learned Gaussian and evaluate them.

6. Join the old and new population using elitism and throw away some individuals so that the population is of size N again.

7. If termination criteria are no met, go to step 2.

The population is initialized in area $\langle -10,-5\rangle^D$ in order to test not only the ability to focus the search when it resides in the area of the optimum, but also to test the ability to efficiently shift the population toward the optimum.

Elitism and sampling $N-1$ new individuals together ensure, that the new population will contain at least 1 old individual and 1 new individual. This feature greately prevents the stagnation in situations when no new individual is better than the worst old individual.

The algorithm was stopped if the best fitness value in the population dropped under 10^{-8}. The results reported are taken from 20 independent runs for each algorithm and configuration.

3.2 Where to Place the Gaussian?

After finding the decision ellipsoid and turning it into a Gaussian distribution, we can decide where we want to place it. There are basically two possible decisions:

1. Place it in the center of the learnt quadratic function. This placement is reminiscent of the process done in conventional EDAs (and is actually depicted in Fig. 1(c)). Also, if the ellipsoid were fit precisely the algorithm could jump to the area of the global optimum in a few iterations.

2. Place it around the best individual of the population. Such an approach is similar to mutative ES and the search is more local.

Since we plan to compare our algorithm to CMA-ES which uses the second option, we use it as well and center the learned Gaussian distribution around the best individual in the population.

3.3 Population Sizes

The CMA-ES uses a population sizing equation of the form $N = 4 + \lfloor 3log(D)\rfloor$. We do not have any population sizing model yet. However, we want to evaluate the potential hidden in our method, and thus we decided to tune the population size for individual test problems and individual dimensionalities.[1] The best settings found for our algorithm along with the population size used by CMA-ES are presented in Table 1.

[1]This is not a good practice for production systems but in this early stage of the research such a tuning is acceptable for discovering the potential of the proposed method.

4. RESULTS AND DISCUSSION

The comparison of the algorithms is depicted in Fig. 9. As can be seen, for the sphere function our approach is slightly better for dimensions 2 and 4, and slightly worse for dimensions 6 and 8. For the ellipsoid function, our algorithm clearly outperforms the CMA-ES for all tested dimensions (2, 4, 6, 8). But again, the difference between our method and CMA-ES gets lower with increasing dimensionality. This could suggest that the efficiency of the algorithm does not scale up well and drops with increasing dimensionality of the problem.

However, another reason of the diminishing performance of the algorithm can be hidden in the perceptron algorithm we used. The algorithm finds *any* elliptical decision boundary, not the optimal one (compare the boundaries found by our algorithm and by the SVM in Fig. 7). With increasing dimensionality, there is a higher chance that the resulting separating ellipsoid will significantly differ from the optimal one.

Coming back to Table 1, we can observe very interesting phenomenon. The 'optimal' population size drops with increasing dimensionality which is something nobody of us expected. We do not have any sound explanation for that. We can only hypothesize that it is again due to the learning algorithm, the modified perceptron. It may be profitable to use smaller populations which would impose less constraints on the ellipsoid that would be in turn less deformed.

Another interesting observation is that the rate of convergence is constant during the whole evolution (which can be seen in Fig. 9(b)). The CMA-ES has to adapt the Gaussian to all dimensions (slow progress phase) before it can aim for the global optimum (fast progress phase); it resembles the steepest descent algorithm. Our method, on the contrary, exhibits a constant progress which suggests better fit of the Gaussian distribution to the local neighborhood; it resembles second-order quazi-newton methods.

5. SUMMARY AND FUTURE WORK

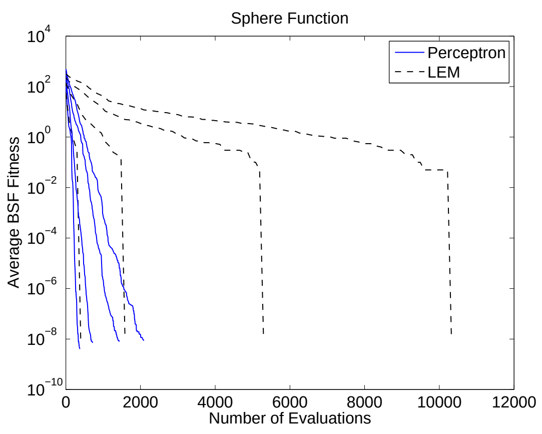
In this paper, a new and original way of learning the parameters of Gaussian distribution in the context of EDAs is presented. The learning algorithm is based on modeling the fitness landscape contour line. The whole process comprises of transforming the original data points into quadratically mapped feature space, finding a linear decision function in that space using our modified perceptron algorithm, transforming the parameters of the decision function back into the original space where they form a quadratic decision function, and computing the parameters of the Gaussian distribution using the parameters of the quadratic decision function.

We expect this concept to have a number of advantages over the conventional evolutionary algorithms. The most important are:

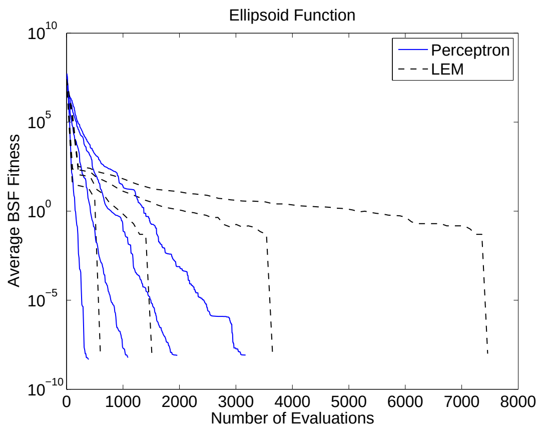
1. the Gaussian distribution should fit the local neighborhood much better than in case of EMNA or CMA-ES,

2. it works with individuals marked only with *select/discard* labels, it does not need the fitness values for each of them (as is the case in [1]),

3. it estimates a 'reasonable' Gaussian even from a small number of individuals (much less than $D(D+3)/2+1$ that are needed by [1]).

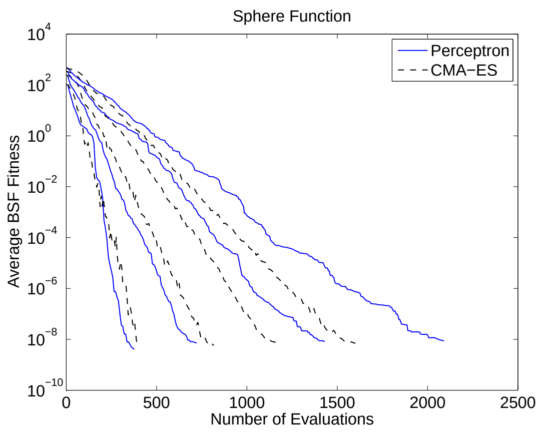


(a) Sphere function.

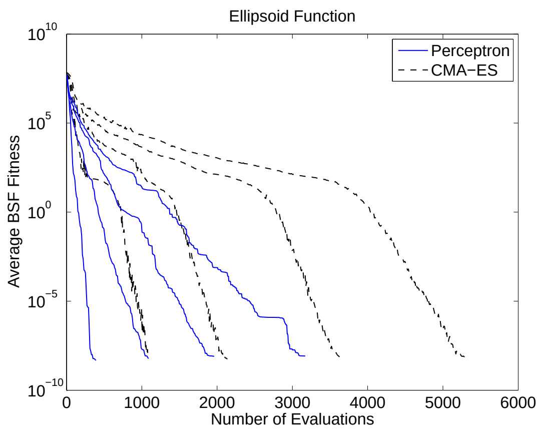


(b) Eliptic function.

Figure 8: Comparison of average evolution traces for the proposed algorithm (—) and the LEM3 system (---). The individual lines belong to 2, 4, 6, and 8 dimensional versions, respectively, from left to right.



(a) Sphere function.



(b) Eliptic function.

Figure 9: Comparison of average evolution traces for the proposed algorithm (—) and the CMA-ES (---). The individual lines belong to 2, 4, 6, and 8 dimensional versions, respectively, from left to right.

The presented algorithm is an initial attempt to proof the concept of modeling the fitness landscape contour line. As such it has strong assumptions which must be relaxed before the algorithm is generally applicable. At present state it is able to optimize only convex quadratic functions.

A number of promising topics for future work remain to be addressed:

- We work on a way of modifying the maximum margin hyperplane algorithm (mentioned in Sec. 2.6) to produce elliptic decision boundary which would allow us to construct optimal separating ellipsoid (and not just any separating ellipsoid). We expect that such an algorithm could take advantage of larger populations and would preserve its superior performance with increasing dimensionality.
- Further extension on our list is to employ the soft margin which will allow us to apply the algorithm even on the non-convex functions, i.e. in cases when the selected and discarded points are not separable with an ellipsoid.
- We have not pursued the possibility to use the learned center of the quadratic function as the center of the Gaussian. This feature must be also studied and we expect it to further increase the efficiency of the algorithm.
- With a learning algorithm that finds an elliptic decision boundary even for non-separable cases, there are possibilities to create a learning algorithm for a whole mixture of Gaussians which would allow us to successfully apply the algorithm on multimodal functions. Moreover, such an algorithm can be able to automaticly select the number of Gaussian components. It would be more time demanding, however, it is a generalization worth trying.

On the other hand, we have thought of several other possibilities of extending this algorithm in which we do not see much promise:

- Could we use the quadratic kernel function in the maximum margin hyperplane algorithm instead of the explicit quadratic mapping of the data points from the original space? This seems to be a reasonable suggestion, however, based on our preliminary modifications of the maximum margin hyperplane algorithm this would make the requirement of positive definite matrix **A** significantly more difficult to achieve.

- If we used a different mapping (not quadratic), could we estimate the distribution of individuals using a different probabilistic model (not Gaussian)? Although this is theoretically possible, we do not see it as a promising research direction. We could for sure learn a different type of decision function this way, however, the critical step is the transformation of the decision function to the probabilistic distribution. The transformation used in this work (quadratic function $\rightarrow$ Gaussian distribution) is (more or less) exception since for this case the transformation is rather straightforward.

To conclude, the experiments carried out suggest there is a big potential in this method if our objective is to find a solution of certain quality using the least possible number of fitness function evaluations. For high-dimensional fitness functions the process of learning the separating ellipsoid may be higly time demanding so that this method should be mainly applicable in cases when the fitness function evaluation is expensive or takes a long time to compute. Nevertheless, we believe that this method can play a significant role in the future development of the real parameter optimization field.

6. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

7. REFERENCES

[1] A. Auger, M. Schoenauer, and N. Vanhaecke. LS-CMA-ES: A second-order algorithm for covariance matrix adaptation. In X. Y. et al., editor, *Parallel Problem Solving from Nature VIII*, number 3242 in LNCS, pages 182–191. Springer Verlag, 2004.

[2] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

[3] P. A. Bosman and J. Grahl. Matching inductive bias and problem structure in continuous estimation-of-distribution algorithms. *European Journal of Operational Research*, 2006.

[4] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[5] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[6] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer series in statistics. Springer Verlag, 2001.

[7] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms*. GENA. Kluwer Academic Publishers, 2002.

[8] P. Pošík. Real-parameter optimization using the mutation step co-evolution. In *IEEE Congress on Evolutionary Computation*, pages 872–879. IEEE, 2005. ISBN 0-7803-9364-3.

[9] G. Rudolph. Self-adaptive mutations may lead to premature convergence. *IEEE Trans. on Evolutionary Computation*, 5(4):410–413, August 2001.

[10] M. I. Schlesinger and V. Hlaváč. *Ten Lectures on Statistical and Structural Pattern Recognition*. Kluwer Academic Publishers, Dodrecht, The Netherlands, 2002.

[11] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, Massachusetts, 2002.

[12] H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[13] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.

[14] J. Wojtusiak. Aq21 user's guide. Reports of the Machine Learning and Inference Laboratory MLI 04-5, George Mason University, 2004. http://www.mli.gmu.edu/papers/2003-2004/04-5.pdf.

[15] J. Wojtusiak and R. S. Michalski. The LEM3 system for non-darwinian evolutionary computation and its application to complex function optimization. Reports of the Machine Learning and Inference Laboratory MLI 04-1, George Mason University, Fairfax, VA, February 2006.