Combining the Strengths of the Bayesian Optimization Algorithm and Adaptive Evolution Strategies

Martin Pelikan
David E. Goldberg
Shigeyoshi Tsutsui

IlliGAL Report No. 2001023
June 2001

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue Urbana, IL 61801
Office: (217) 333-2346
Fax: (217) 244-5705

Combining the Strengths of the Bayesian Optimization Algorithm and Adaptive Evolution Strategies

Martin Pelikan, David E. Goldberg, and Shigeyoshi Tsutsui
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
104 S. Mathews Avenue, Urbana, IL 61801
Phone/FAX: (217) 333-2346, (217) 244-5705
{pelikan,deg,shige}@illigal.ge.uiuc.edu

Abstract

A method which combines competent genetic algorithms working in discrete domains with adaptive evolution strategies working in continuous domains is described. Discretization is used to transform solutions between the two domains. Experiments with Bayesian optimization algorithm, σ-self-adaptive mutation, and three different discretization methods are presented. The results suggest that the algorithm scales up well on all tested problems.

1 Introduction

In genetic and evolutionary algorithms, the search is guided by selection and variation operators. Selection biases the search towards high-quality regions. Variation operators (such as recombination and mutation) ensure exploration of promising regions of the search space. There are two commonly used variation operators to generate new solutions based on the information contained in the current population of promising solutions. Genetic algorithms focus primarily on *recombination* which combines solutions by exchanging some of their parts. On the other hand, the major variation operator in evolution strategies is *mutation* which performs small perturbations to solutions. There has been a lot of progress in both mutation-based and recombination-based approaches over the last decades. However, only little has been done to combine the most advanced results of these two lines of the genetic and evolutionary computation research.

The purpose of this paper is to show how some of the advanced algorithms based on the two aforementioned approaches can be combined to solve problems defined in continuous domains. In particular, the Bayesian optimization algorithm (BOA) based purely on recombination is combined with a mutation-based evolution strategy (ES) with adaptive mutation strength. However, since BOA works only on finite-alphabet strings of fixed length while ES works directly with vectors of real numbers, it is not possible to combine the two approaches without an intermediate step in between. The problem of inconsistent representations is overcome by using *discretization* as a mechanism to bridge the two seemingly incompatible approaches. The resulting approach can be seen both as the Bayesian optimization algorithm with adaptive discretization or a recombinative evolution strategy capable of linkage learning. The same method can be used to combine other competent genetic algorithms and evolution strategies with no or only minor modifications. The approach can also be used to solve problems which contain both continuous and discrete variables. We performed experiments on a few simple continuous problems. The results suggest good scale-up behavior and mixing capabilities of the algorithm.

The paper starts by introducing the Bayesian optimization algorithm and evolution strategies with adaptive mutation which are used as the basic building blocks of the proposed algorithm. Background of using discretization in genetic and evolutionary computation is then provided and discussed. Section 3 describes how a competent recombination-based genetic algorithm in a discrete domain can be combined with a mutation-based approach in a continuous domain. Section 4 provides our experimental results. Section 5 summarizes and concludes the paper.

2 Background

Genetic and evolutionary algorithms start with a randomly generated initial population of candidate solutions. In each iteration, the set of promising solutions is selected. By using recombination and mutation operators on the selected solutions, new solutions are constructed. The new solutions replace some of the old ones and the process is repeated until the termination criteria are met.

This section starts by introducing two fundamentally different approaches based on the above scheme. First, it describes the Bayesian optimization algorithm (BOA) which has been recently shown to solve difficult problems defined on fixed-length binary strings very efficiently and reliably. BOA uses a powerful recombination method based on graphical models which combines pieces of promising solutions by using statistical information from the selected set of solutions. Subsequently, it describes evolution strategies where the search is guided by mutation-based operators. Several methods for adapting mutation parameters are presented. Evolution strategies (ES) process fixed-length vectors of real numbers and were shown to be powerful local search methods. The section ends by discussing discretization in context of genetic and evolutionary computation which will later be used as a way to bridge the recombination-based discrete BOA and the mutation-based continuous ES.

2.1 Bayesian Optimization Algorithm

There are a number of ways to use the set of selected solutions in order to create new candidate solutions. Recombination-based genetic algorithms generate new solutions by combining bits and pieces of the selected solutions. A simple genetic algorithm (Goldberg, 1989) uses problem-independent crossover operators for this purpose, such as uniform crossover and n-point crossover. Mutation is usually used as only a background operator which is capable of tuning near-optimal solutions at the end of the run or introduce diversity into the population.

Probabilistic model-building genetic algorithms (PMBGAs) (Pelikan, Goldberg, & Lobo, 2000) also try to combine important parts of the selected solutions but they approach the recombination in a different way. They view the set of selected solutions as a sample from the region of the search space that we are interested in. They estimate the distribution of the selected solutions and use this estimate to generate new solutions. The estimated distribution can encode information on the interactions among different variables in a problem as well as superiority of certain combinations of values of different subsets of variables. The algorithms based on this principle are also called *estimation of distribution algorithms* (EDAs) (Mühlenbein & Paaß, 1996), or *iterated density estimation algorithms* (IDEAs) (Bosman & Thierens, 2000). It is beyond the scope of this paper to give an overview of PMBGAs. For a survey of PMBGAs, please see Pelikan, Goldberg, and Lobo (2000). In this paper, we focus on the Bayesian optimization algorithm (Pelikan, Goldberg, & Cantú-Paz, 1998).

The Bayesian optimization algorithm (BOA) (Pelikan et al., 1998) uses Bayesian networks to model promising solutions and subsequently guide the exploration of the search space. In the BOA,

the first population of strings is generated randomly with a uniform distribution. The initial population can also be biased to the regions that we are interested in. From the current population, the better strings are selected. Any selection method can be used. A Bayesian network that fits the selected set of strings is constructed. Any metric as a measure of quality of networks and any search algorithm can be used to search over the networks in order to maximize/minimize the value of the used metric. Besides the set of good solutions, prior information about the problem can be used in order to enhance the estimation and subsequently improve convergence. New strings are generated according to the joint distribution encoded by the constructed network. The new strings are added into the old population, replacing some of the old ones. The pseudo-code of the BOA is shown in Figure 1.

The Bayesian Optimization Algorithm (BOA)

(1) set $t \leftarrow 0$
randomly generate initial population $P(0)$
(2) select a set of promising strings $S(t)$ from $P(t)$
(3) construct the network B using a chosen metric and constraints
(4) generate a set of new strings $O(t)$ according to the joint distribution encoded by B
(5) create a new population $P(t+1)$ by replacing some strings from $P(t)$ with $O(t)$
set $t \leftarrow t+1$
(6) if the termination criteria are not met, go to (2)

Figure 1: The pseudo-code of the Bayesian optimization algorithm.

A Bayesian network is a directed acyclic graph with the nodes corresponding to the variables in the modeled data set (in our case, to the positions in the solution strings). Mathematically, a Bayesian network encodes a joint probability distribution given by

$$p(X) = \prod_{i=0}^{n-1} p(X_i|\Pi_{X_i}), \tag{1}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of all the variables in the problem, Π_{X_i} is the set of parents of X_i in the network (the set of nodes from which there exists an edge to X_i) and $p(X_i|\Pi_{X_i})$ is the conditional probability of X_i conditioned on the variables Π_{X_i}. A directed edge relates the variables so that in the encoded distribution, the variable corresponding to the terminal node will be conditioned on the variable corresponding to the initial node. More incoming edges into a node result in a conditional probability of the corresponding variable with conjunctional condition containing all its parents.

To construct the network given the set of selected solutions, various methods can be used. Most methods have two basic components: a scoring metric which discriminates the networks according to their quality and the search algorithm which searches over the networks to find the one with the best scoring metric value. The BOA can use any scoring metric and search algorithm. In this paper, we use decision graphs to represent local densities for all variables. A simple greedy algorithm is used which splits a particular leaf in a decision graph which improves the score the most. A Bayesian-Dirichlet metric with prior probability of networks inversely proportional to their complexity is used. For more details, please see Pelikan, Goldberg, and Sastry (2000).

BOA has been shown to optimize many difficult problems efficiently and reliably. It is capable of adapting to the problem at hand and discovering important interactions in the problem. By using this information it combines promising solutions found so far and guides the search.

2.2 Evolution Strategies

Evolution strategies (ES) (Rechenberg, 1973) use mutation as the driving force of the search and usually work on solutions represented by vectors of real numbers. Therefore, much effort has been put in developing powerful mutation methods for continuous domains. It is beyond the scope of this paper to give a complete overview of the developments in this area, and we will only present the information that is necessary for understanding this paper. For more details on adaptive mutation and other aspects of ES, the interested reader should refer to Rechenberg (1973), Schwefel (1974), Bäck (1996), Beyer (1996), and Hansen and Ostermeier (1996). Along with mutation, the use of recombination in ES has been investigated (Schwefel, 1974; Bäck, 1996; Beyer, 1995). However, in ES recombination is often seen as only a background operator that may improve the effects of mutation, while in GAs recombination is a primary source of the exploration of the search space.

ES are usually referred to by the type and parameters of selection and other operators they use. In this paper, we focus on $(\mu+\lambda)$ and (μ, λ) ES. In both $(\mu+\lambda)$ and (μ, λ) ES, the initial population of size μ is generated at random. In the simplest variant of ES, the offspring population of λ individuals is created by adding a zero-mean normally distributed random number to each variable. We describe some other mutation operators later in this section. In $(\mu+\lambda)$ ES, the offspring are first added to the original population and μ best individuals are selected to form the population in the next generation. In (μ, λ) ES, μ best individuals are selected from the offspring population to form the population in the next generation (parents do not participate in the subsequent generation other than through their offspring).

A simple example is the $(1 + 1)$ ES, where the population consists of one individual. The new individual is obtained by mutating the parent. The offspring replaces its parent in case it is better. The process is repeated until termination criteria are met.

The standard deviation of the mutation (the mutation strength) can be either fixed to some small constant or adapted as the search progresses. A fixed mutation strength results in slower convergence either at the beginning or the end of the run. Increasing the mutation strength makes the algorithm move to the optimum faster at the beginning of the run but results in slowing down the final stage of the run when the solutions are very close to the optimum by making too big steps. Decreasing the mutation strength improves the performance in the final stage of the algorithm, but it takes longer to reach the solutions close to the global optimum because of the smaller steps. This tradeoff is exploited by adaptive mutation which should ideally learn how big the mutation should be to maximize the improvement in the current stage of the algorithm. The 1/5-success rule (Rechenberg, 1973) and σ-self-adaptive ES (Schwefel, 1977) are examples of the adaptive mutation strategies.

The 1/5 success rule is designed for $(1 + 1)$ ES. It records the ratio of the number of successful mutations (mutations leading to an improvement) to the total number of mutations (mutations leading to inferior solutions). By increasing the mutation strength when the success rate is higher than 1/5 and decreasing it when the success rate is lower than 1/5, optimal performance for some simple unimodal functions can be achieved.

In σ-self-adaptive ES, a vector of standard deviations corresponding to each variable is attached to each solution. Before mutating the solution, its mutation parameters are modified by using the

following rule:

$$\sigma_i' = \sigma_i e^{\tau N(0,1)}, \tag{2}$$

where σ_i is the standard deviation corresponding to the mutation of ith variable, σ_i' is its updated value, $N(0,1)$ is a zero-mean Gaussian random variable with variance 1, and τ is the learning parameter. The above update rule assures that the mutation strength is always positive, the expected outcome of the modification without any selection pressure is neutral, and smaller modifications occur more often than the large ones (Schwefel, 1977). Good mutations are filtered by a standard selection mechanism because individuals which lead to the best improvements are going to participate in the reproduction in the subsequent iteration. ES are robust to changes in the learning parameter τ. Schwefel suggests that τ should be inversely proportional to the square root of the total number of variables:

$$\tau \propto \frac{1}{\sqrt{n}} \cdot \tag{3}$$

The above equation was further investigated by Beyer (1996) who computed an optimal τ for $(1, \lambda)$ ES as

$$\tau = \frac{c_{1,\lambda}}{\sqrt{n}}, \tag{4}$$

where $c_{1,\lambda}$ is the so-called progress coefficient. For instance, for the sphere model where the fitness is a sum of squares of all variables, we get

$c_{1,2}$	$c_{1,5}$	$c_{1,10}$	$c_{1,50}$	$c_{1,100}$
0.5642	1.1630	1.5388	2.2491	2.5076

Another example of adaptive mutation strength is a simple linear adaptive rule (Rechenberg, 1994)

$$\sigma_i' = \begin{cases} \sigma_i(1+\beta) & \text{if } u < \frac{1}{2}, \\ \sigma_i/(1+\beta) & \text{otherwise,} \end{cases} \tag{5}$$

where u is a uniform random number from (0,1), and β is a small constant from (0,0.3].

All above mutation methods change each variable independently of the changes in the remaining variables, i.e. the mutations for different variables are not correlated. This resembles the uniform crossover in genetic algorithms where each bit in the two parent strings is exchanged with a certain probability independently of the remaining bits. Even more closely, it resembles the population-based incremental learning algorithm (PBIL) (Baluja, 1994) and other similar PMBGAs which treat each variable independently of the remaining variables resulting in a very strong recombination. To reduce the disruptive effects of recombination, methods that were able to learn the structure of the problem and adapt the recombination accordingly were designed.

A similar approach can be used for adapting mutations where by extracting the information from the history of the run one can learn not only how strong the mutation for each variable should be, but also how the mutations for different variables should interact. Schwefel (1981) proposed to extend solutions by including rotation angles in addition to the original deviations (or variances) and adapt

both the variances as well as the rotation angles by the ES. There are $n(n-1)/2$ such angles where n is the total number of variables under consideration.

The generating set adaptation (GSA) method proposed by Hansen, Ostermeier, and Gawelczyk (1995) is capable of adapting an arbitrary normal mutation distribution independently of the used coordinate system. Some information about the history of mutations is stored in the solutions and this is subsequently used to compute the covariance matrix which determines the promising mutation distribution. The mutation distribution is a zero-mean multivariate normal distribution. Independent mutations can thus be seen as a special case of this algorithm when all non-diagonal elements of the covariance matrix are equal to zero. The use of covariances allows the algorithm to be invariant to the rotation and any other linear transformations of the coordinate system. Covariance matrix adaptation (CMA) (Hansen & Ostermeier, 1996) improved some of the properties of the CMA to adapt covariance matrix and a global mutation step better.

Even though recombination has been used in evolution strategies since the early works in this area, it has been seen as only a minor operator (Beyer, 1995). For recombination in ES, a variant of uniform crossover is usually used. For each new individual, a subset of parents is chosen. The size of the selected subset determines the strength of recombination and can range from two individuals (weakest recombination) to the entire parent population (strongest recombination). For the value of each variable in the new individual, a random individual is picked from the subset and the value is copied from the picked individual.

The final effect of such recombination is very similar to the univariate marginal distribution algorithm (UMDA) (Mühlenbein & Paaß, 1996) for fixed-length discrete strings where values at each position are shuffled in the parent population. Similarly as in discrete genetic algorithms, using such strong recombination can yield to a significant decrease in performance for multimodal problems with interactions among variables. It is important to learn what pieces of solutions should be combined in order to preserve important partial solutions and combine them efficiently.

When using recombination together with adaptive mutation, one must copy not only the value of each variable but also corresponding mutation strengths or the histories of the mutations on this variable. Since this information is associated with each variable, no major modifications are required. One can also use a separate recombination on the mutation parameters (Schwefel, 1977). Using recombination and selection schemes from genetic algorithms in ES has been also investigated (Schwefel, 1995).

ES with adaptive mutation are very powerful for local search. However, simple recombination which treats all variables independently fails for problems with interacting variables. So why not combine recombination-based methods capable of linkage learning in a discrete domain and mutation-based methods capable of adapting the mutation in a continuous domain? As we will further discuss in Section 3, it is indeed possible to combine the two methods by using discretization to transform continuous solutions into a discrete domain and vice versa. The next section discusses the use of discretization in genetic and evolutionary algorithms.

2.3 Discretization

Discretization is widely used in many fields of science to transform problems from a continuous domain into a discrete one in order to simulate complex systems, solve differential equations, analyze materials, fit probability distributions, etc. Discretization often reduces the complexity of a problem and makes intractable problems tractable.

In genetic and evolutionary computation, discretization has often been used to first transform continuous solutions into binary strings and then apply the algorithm working in a discrete domain

on the resulting problem. The discrete solution can then be transformed back into the continuous domain and either taken as is or further optimized by a local searcher such as gradient and hill-climbing methods. There are several advantages and disadvantages of discretizing the solutions and solving the corresponding discrete problem (Goldberg, 1991). Discrete solutions improve the implicit parallelism of genetic algorithms and allow them to process more partial solutions with the fixed amount of resources. Moreover, the discrete space is finite and thus it is easier to guarantee that the optimal solution in this space is found and that we supply enough information for the optimization to succeed. On the other hand, the locality of the solution decreases and some phenotypically close solutions (similar solutions in a continuous domain) may become very distant in the discrete domain. This affects especially mutation which attempts to make small changes in the phenotype by making small changes in the genotype. Additionally, one must know the range of each variable to discretize it and the resulting binary strings may be very long for large problems.

A typical way of discretizing continuous solutions in genetic algorithms is to divide the range of each variable into $(2^k - 1)$ intervals of equal width (Goldberg, 1989). Boundary points of the intervals can then be encoded by k-bit binary strings. A string encoding n such continuous variables contains nk bits. Increasing k refines the discretization by factors of 2. Different variables can use different numbers of bits depending on the problem. To successfully use this simple discretization method, we must make sure that the function is smooth enough to not change much within each interval. For many problems only a couple of bits (say, $k = 3$ or 4) is sufficient to get a solution very close to the optimum. Local optimization methods can then be used to refine the final solutions to get a more accurate result. Using too few intervals may result in being unable to reach the solutions of required quality. On the other hand, a large number of bins increases computational requirements through increased number of basic building blocks and the entire strings.

There are a number of ways in which binary strings can be mapped to the boundary points discussed in the above paragraph. One can use a direct mapping where a binary number determines the number of the boundary point (binary coding) or define special representations that preserve certain properties of the resulting encoding. An example of such representation are gray codes where locality is preserved and a single bit change in each solution leads to a small change in the corresponding continuous solution. However, one must sacrifice many positive recombination effects and the overall performance often deteriorates (Rothlauf, 2001).

Since the discretization assumes that quality of solutions within one interval should not vary much, we can see the above discretization as if an equal-width histogram was used to discretize the population but only one representative solution per bin (at its beginning or end) was used. Only the representative solutions are searched by the genetic algorithm and the final solution is the best representative.

A different way of using histograms in evolutionary algorithms for continuous domains is to use histograms as a tool to estimate the distribution of promising points. The created model can then be used to generate new points. The difference between the two aforementioned ways of using histograms is that in the latter approach, points are allowed to lie within the intervals and not only on their boundaries. This may lead to further improvements of the final solutions.

Algorithms that use histograms in this fashion in order to estimate a univariate distribution where all variables are processed independently have been proposed (Bosman & Thierens, 2000; Tsutsui, Pelikan, & Goldberg, 2001; Cantú-Paz, 2001). All these algorithms share the same basic idea of discretizing each variable in a fixed number of bins. To generate each new variable, a bin is first selected with the probability proportional to the frequency of the bin. A number within the interval corresponding to the selected bin is then uniformly generated. Using equal-width histograms was investigated in Bosman and Thierens (2000), Tsutsui, Pelikan, and Goldberg (2001), and Cantú-Paz

(2001). Equal-height histograms were investigated in Tsutsui et al. (2001), and Cantú-Paz (2001). Decision trees and other supervised discretization methods were investigated in Cantú-Paz (2001).

Various discretization methods were proposed and frequently used in machine learning, statistics, and other fields. Equal-height histograms, decision trees, and clustering algorithms are examples of such methods. All these methods have the same important characteristics, they map a single continuous variable or groups of variables into a finite set of symbols. We discuss some of these methods in the following subsections.

2.3.1 Fixed-width Histograms

The fixed-width histogram divides the interval for the variable in k equal-width bins. Let us denote the entire interval by $[a, b)$. Then, the interval corresponding to the ith bin is given by

$$\left[a + \frac{i(b-a)}{k}, a + \frac{(i+1)(b-a)}{k}\right),$$

where $i \in \{0, \ldots, k-1\}$. An example of a fixed-width histogram is shown in Figure 2. A disadvantage of fixing the width of each bin is that in case points are concentrated in a couple of small regions, only a couple of bins are nonempty. The discretization thus does not give us much information on the distribution of points. Fixed-width histograms are also very sensitive to outliers and one or a few points far away from the rest can significantly decrease this discretization.

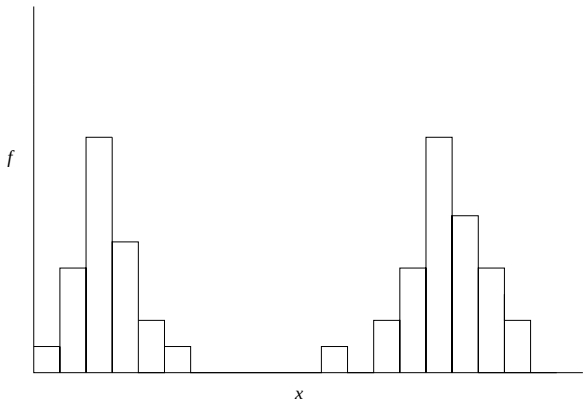


Figure 2: Fixed-width histogram.

2.3.2 Fixed-height Histograms

A fixed-height histogram divides the interval for the variable in k bins of equal frequencies. That means, that there are the same number of points in each bin.

An example of a fixed-height histogram is shown in Figure 3. An advantage of using fixed-height histograms is that their density is increased in dense regions. If the points are concentrated in a couple of regions, the density of bins in each of these regions is quite high and the discretization preserves more information contained in the original continuous set of points.

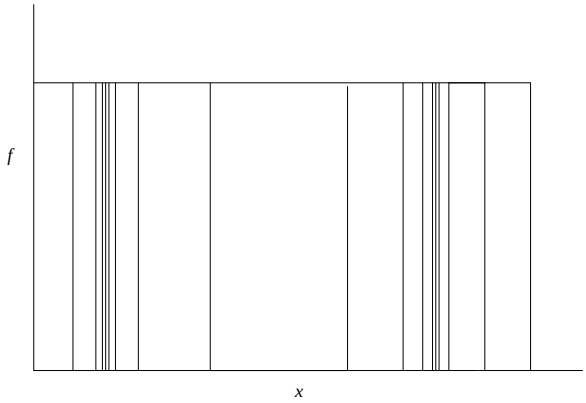


Figure 3: Fixed-height histogram.

2.3.3 k-Means Clustering

In k-means clustering, each cluster is specified by its *center*. Initially, k centers (where k is given) are generated at random. Each point is assigned to its nearest center. Subsequently, each center is recalculated to be the mean of the points assigned to this center. The points are then reassigned to the nearest center and the process of recalculating the centers and reassigning the points is repeated until no points change their location after updating the centers. The pseudo-code of the k-means clustering algorithm follows:

(1) Generate k centers c_i at random.
(2) Assign each point x_i to the nearest center.
(3) Assign each center to the centroid of the points assigned to it.
(4) If point locations have changed in step 2, go to 2.
(5) Return the cluster centers and point locations.

An example of k-means clustering in two dimensions is shown in Figurefigure-kmeans. Original (randomly generated) centers are denoted by empty circles. The final centers are denoted by filled circles.

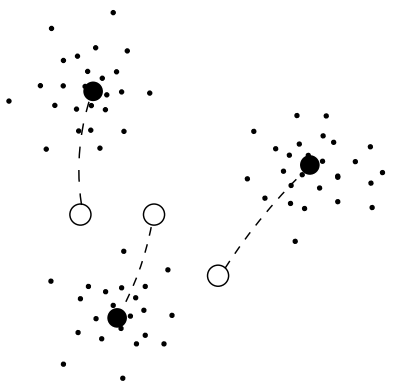


Figure 4: k-means clustering.

3 Combining Linkage Learning and Adaptive Mutation

In this section, we describe the algorithm which combines a discrete algorithm capable of linkage learning with adaptive mutation techniques from evolution strategies. The method is visualized in Figure 5.

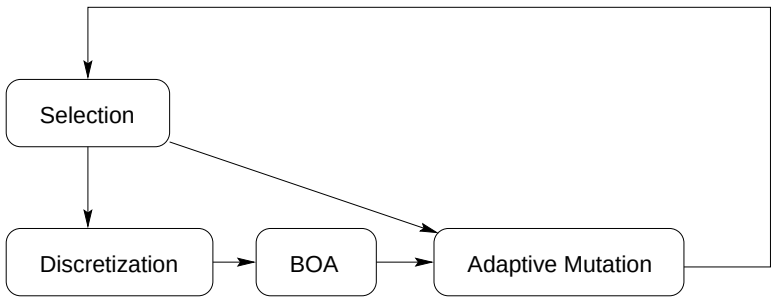


Figure 5: The algorithm.

The algorithm evolves a population of continuous solutions. The first population is generated at random. From the current population the better strings are selected. Processing of the promising solutions has three major phases:

1. Discretization of the selected promising solutions.
2. Recombination of the discrete solutions.
3. Mapping the new discrete solutions back in the continuous domain, updating the mutation parameters, and mutating the new continuous solutions.

In the first phase, the promising solutions are discretized. Each variable is independently mapped into a finite number of categories. Let us denote the number of categories (bins) for the ith variable by c_i. There are two major approaches to represent the resulting discrete population. The first approach is to use binary strings and $\lceil \log_2 c_i \rceil$ bits for each variable. The second approach is to use alphabet of a higher cardinality so that only one symbol is used to represent each variable. The ith letter in the discrete string could then obtain c_i values. Of course, there are many ways between the two extremes. Binary representation results in more possibilities to combine the strings. On the other hand, higher cardinality alphabets result in shorter solutions. Any discretization method can be used to discretize the continuous variables.

In the second phase, a discrete linkage learning algorithm, such as BOA, is applied to generate the new solutions based on the set of discrete promising solutions. The offspring discrete solutions are constructed by combining parts of the promising solutions.

In the third phase, the resulting set of discrete solutions is mapped back into the continuous domain. However, unlike in all previously proposed approaches, the new points are not generated uniformly within the boundaries of categories for each variable. Instead, original points within each category are used. Each discrete string determines a category for each variable. To "undiscretize" each variable in a particular string, a random individual in the original set of promising solutions that is consistent with the encoded category for the variable is picked. The value of the variable is obtained by mutating the value of the variable in the picked individual.

As a simple example of mapping discrete solutions back into the continuous domain, let us assume we use an equal-width histogram with only two categories for each variable. That means that a binary string determines whether the corresponding variable is in the upper or lower half of its range. Given a binary string, we look at the value of each of its bits. If the value is 0 (1), we randomly pick a solution from the set of promising continuous solutions whose value of the considered variable is in the lower (upper) half of the domain. The corresponding variable in the picked solution is copied to the newly created continuous solution. This is done for each variable separately. Finally, the created continuous solution is mutated.

Using adaptive mutation requires considering additional parameters in the continuous solutions. For σ-self-adaptive mutation, we must keep a value of the mutation strength for each variable in each string. The mutation strength is attached to the variable and therefore when we copy a value of a particular variable we also copy the corresponding mutation strength. As we copy the values of the variables and the attached parameters into the new continuous string, the mutation strengths are updated by using the rule discussed earlier in the paper (see Equation 2). The new mutation strengths are used to mutate the created solution. CMA requires storing history of the mutations.

The newly generated solutions then replace the original population or its part. Using elitism can make the runs more stable. Niching methods can be used to preserve the diversity of the population better.

Example:
Let us use an example to clarify the method for equal-width histograms with two bins and σ-self adaptive mutation. There are two continuous variables, both in range $[0, 10]$. The set of promising solutions selected from the current population with the mutation strengths consists of two individuals $i_1 = ((X_0 = 1, \sigma = 0.05), (X_1 = 7, \sigma = 0.01))$ and $i_2 = ((X_0 = 8, \sigma = 0.02), (X_1 = 6, \sigma = 0.04))$. The two equal-height bins for each variable consist of two intervals, $[0, 5]$, and $(5, 10]$. The strings are thus discretized into binary strings $i_1^* = 01$ and $i_2^* = 11$. Let us say that recombination results in the same two strings 01 and 11. The value of X_0 in the first offspring solution 01 must thus be copied from an individual that has the value of the first variable in the lower bin and the value of X_1 is picked from a solution which has the second value in the upper bin. One of the possible outcomes is $((X_0 = 1, \sigma = 0.05), (X_1 = 6, \sigma = 0.04))$, where the first variable is picked from individual i_1 and the second variable is picked from the individual i_2. Analogously, one possible outcome of undiscretizing the second new individual 11 is $((X_0 = 8, \sigma = 0.02), (X_1 = 7, \sigma = 0.01))$. The mutation strengths in the resulting solutions are updated using σ-self-adaptive rule and the mutation with the corresponding deviation is applied to both offspring solutions.

Various algorithms can be used for discretization, linkage learning, and adapting the mutation. Due to our recent successful applications of BOA to many discrete problems, we decided to use this algorithm for linkage learning and recombination in our experiments. To adapt mutation, we used a simple σ-self-adaptive mutation where only a mutation strength of each parameters is adapted. Application of other mutation schemes such as CMA is straightforward. To discretize the solutions, we used equal-height histograms, equal-width histograms, and k-means clustering, but any method that maps real numbers into a number of categories can be used. That means that any of popular discretization, classification, and clustering techniques can be used. Using more advanced techniques should further improve the performance. For a discussion of some interesting alternatives, please see Cantú-Paz (2001).

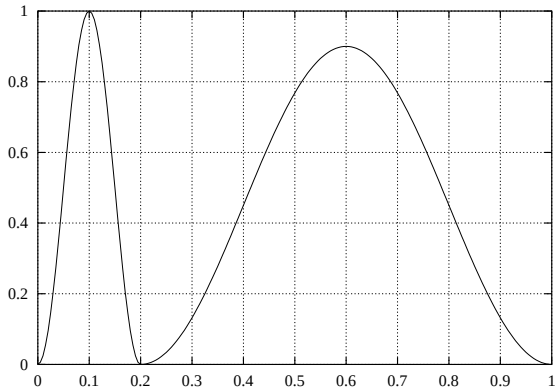


Figure 6: Two-peaks function.

4 Experiments

This section describes our experiments and presents the experimental results.

4.1 Problems

We have tested the algorithm on two test functions: (1) two-peak function and (2) deceptive function. Both test functions are created by concatenating basis functions of a small order. The contributions of all the functions are added together to determine the overall fitness. All variables in our test functions are from $[0, 1]$.
The two-peaks function is given by

$$twoPeaks(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} f_{two-peaks}(x_i).$$

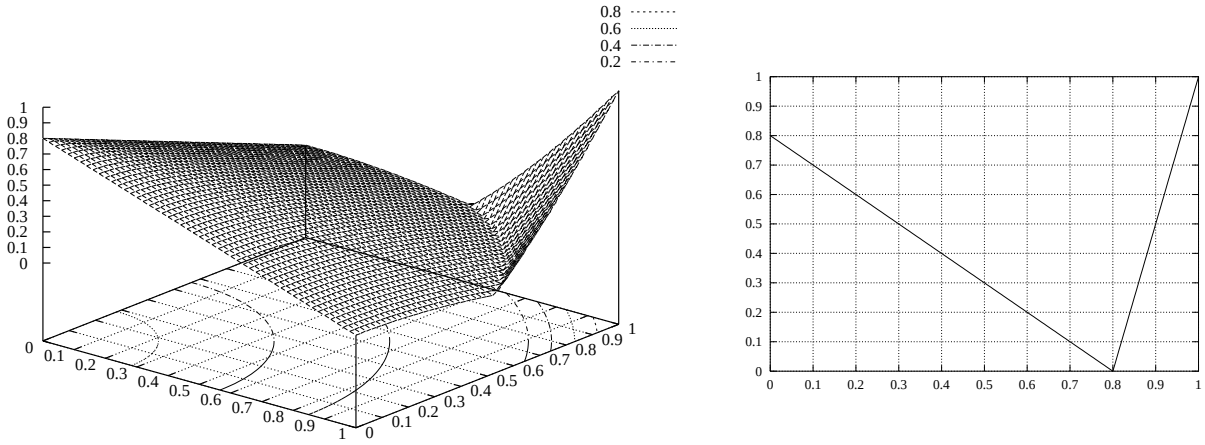
Every variable of the two-peaks function contributes to the fitness by

$$f_{two-peaks}(x) = \begin{cases} f_{peak}(x/0.1, 1) & \text{if } x < 0.2, \\ f_{peak}((x - 0.2)/0.8, 0.9) & \text{otherwise,} \end{cases}$$

where f_{peak} is a simple function for one peak, defined as

$$f_{peak}(x, h) = h \cos\left(2\pi(x - 0.5)\right).$$

Figure 6 shows a graph of the function. The function has one local and one global optimum for each variable. This yields 2^n optima for a problem of size n out of which only one optimum is global. Local optima are much wider and almost as good as the global one which makes the problem more difficult. Using mutation only does not yield good results on this problem. Recombination makes the use of the decomposability of the problem and is capable of solving the problem very efficiently and reliably by processing and combining a large number of pieces of solutions together. Simple uniform crossover is sufficient and thus any ES with recombination should work on the problem very well.



(a) Two-dimensional

(b) One-dimensional

Figure 7: Deceptive function.

The deceptive function is composed of two-dimensional deceptive functions, and is given by

$$deceptive(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{\frac{n}{2}} f_{two-peaks}(x_{2i}, x_{2i+1}).$$

Non-overlapping pairs of neighboring variables of the deceptive function contribute to the overall fitness by

$$f_{deceptive}(x, y) = f_{dec}\left(\sqrt{\frac{x^2 + y^2}{2}}\right)$$

where f_{dec} is a one-dimensional deceptive function defined on $[0, 1]$ as

$$f_{dec}(x) = \begin{cases} 0.8 - x & \text{if } x \leq 0.8, \\ \frac{1-x}{0.2} & \text{otherwise.} \end{cases}$$

Figure 7 shows the one and two-dimensional deceptive functions as defined above. The two-dimensional deceptive function requires that we learn the linkage of the two contributing variables. Each variable alone is biased to the local optimum in 0 and only when both variables are close to 1 their combination leads to an improvement. In early stages of the run there will be more points on a local attractor than the global one. If both variables are treated independently, combinations with both variables near the global attractor will vanish and the search will progress toward the local optimum. Moreover, the global optimum is very isolated and the attractor is very small. This makes it quite difficult to hit the global attractor.

4.2 Results

This section presents and discusses empirical results. We analyzed the scale-up behavior of the compared algorithms. BOA with σ-self-adaptive mutation strength with a learning parameter $\tau = 4/\sqrt{n}$. The numerator in the equation for the learning parameter τ is not the optimal one. It was chosen according to our experience with applying the algorithm to a few problem instances, but the algorithm is quite robust to changes in this constant. Fixed-width histograms, fixed-height histograms, and k-means clustering have been used as discretization methods. In all experiments, a binary tournament selection with replacement is used where to obtain each new reselected individual we first randomly select two individuals and then pick the winner. Individuals participating in the tournament can be selected in the next tournament. An elitist replacement scheme is used where the worst half of the population is replaced by the offspring.

For each problem size, we performed 30 independent runs with the optimal population size which was determined empirically for each algorithm and problem size to find the optimum in all 30 runs. A number of fitness evaluations to reach solutions whose Euclidean distance from the optimum was at most 0.01 has been recorded and its average over the 30 runs is presented. Both arithmetic and log-log plots are presented. Log-log plots confirm polynomial number of fitness evaluations in all cases, since the slope does not increase with the problem size.

The two-peaks problem is very simple and could be used by using recombination with no linkage learning (i.e. traditional ES recombination based on uniform crossover). However, we present the results to show that the algorithms are capable to solve both simple as well as difficult problems. Without recombination, the ES with σ-self-adaptive mutation can not solve any of the discussed problems in less than exponential time. The deceptive problem would require exponential population sizes both if no recombination was used as well as if a traditional recombination based on uniform crossover was used. Other fixed recombination methods would also fail if the variables were not ordered according to their dependencies.

The results for fixed-width histogram on the two-peaks function are provided in Figure 8. Both arithmetic and log-log plots are presented. The algorithm was able to reach the optimum in about $O(n^{1.57})$ fitness evaluations. The population sizes ranged from $N = 700$ for a problem of size $n = 10$ to $N = 3500$ for $n = 50$. With no linkage learning, the population sizes could be significantly reduced. However, for the deceptive problem it is necessary to learn the linkage and use recombination in order to solve the problem in less than exponential time.

The results for fixed-width histograms on the deceptive function are presented in Figure 9. We performed experiments with 4 and 8 bins per variable (i.e., 2 and 3 bits per variable in the binary representation). The optimum was found in about $O(n^{2.1})$ in both cases. The population sizes for 4 bins ranged from $N = 1000$ for $n = 10$ to $N = 7400$ for $n = 40$. The population sizes for 8 bins ranged from $N = 1100$ for $n = 10$ to $N = 8750$ for $n = 40$. The results with 8 bins are slightly worse than the results with 4 bins even though we can see that the number of bins does not affect the results significantly, once there are enough bins to solve the problem.

The results for fixed-height histograms with 4 and 8 bins on the two-max function are shown in Figure 10. The optimum was found in about $O(n^{1.28})$ for 4 bins and $O(n^{1.21})$ fitness evaluations for 8 bins. The population sizes for 4 bins ranged from $N = 500$ for $n = 10$ to $N = 1400$ for $n = 50$. The population sizes for 8 bins ranged from $N = 600$ for $n = 10$ to $N = 1500$ for $n = 50$. For 8 bins, the slope in the log-log scale changes. However, for all problems larger than 20 the slopes again remains almost a constant.

The results for fixed-height histograms with 16 bins on the deceptive function are shown in Figure 11. The optimum was found in about $O(n^{2.1})$ fitness evaluations. The population sizes ranged

from $N = 3500$ for $n = 10$ to $N = 17000$ for $n = 30$. The results for the deceptive function suggest that fixed-height histograms are not a good choice for the discretization of difficult functions. The advantage of fixed-height histograms is that they increase accuracy in more dense regions. However, using adaptive mutation by itself should be sufficient to find the optimum with high accuracy, and increasing the resolution in dense regions seems to decrease the performance. For 8 and less bins per variable, the algorithm performed quite poorly.

The results for k-means clustering with 8 bins per variable on the two-peaks function are shown in Figure 12. The optimum was found in about $O(n^{1.32})$ fitness evaluations. The population sizes ranged from $N = 700$ for $n = 10$ to $N = 2100$ for $n = 50$.

The results for k-means clustering with 8 bins per variable on the deceptive function are shown in Figure 13. The optimum was found in about $O(n^{2.28})$ fitness evaluations. The population sizes ranged from $N = 900$ for $n = 10$ to $N = 8750$ for $n = 40$.

5 Summary and Conclusions

The paper overviewed some of the recent developments in areas of linkage learning and adaptive mutation. A method was proposed to combine linkage learning in discrete domains and adaptive mutation in a continuous domain. The method is based on using discretization to transform solutions between the two domains. Care must be taken to allow adaptive mutation in this scheme.

The results of the paper suggest that recombination-based methods for discrete domains and mutation-based methods for continuous domain can be combined to utilize the strengths of both methodologies. The degree to which the methods are combined can be controlled by choosing the resolution of discretization and recombination parameters. For coarse discretization, the algorithm performs very similar to the ES with recombination based on uniform crossover. Refinement of discretization yields to more possibilities for learning the linkage between different variables in the problem. However, learning linkage comes at a price of increased requirements on the population size. While ES usually require only small populations, statistical methods of BOA require quite big populations. But for most multimodal problems it is necessary to combine parts of promising solutions to avoid exponential time requirements.

There are many other alternative uses of the presented scheme. Using supervised discretization methods can yield significant improvements. Additionally, many real-world problems do not require sophisticated linkage learning procedures and simple one, two-point, or uniform crossover may suffice.

6 Acknowledgments

The authors would like to thank Kumara Sastry, Erick Cantú-Paz, Franz Rothlauf, and Hans-Georg Beyer for many useful discussions and valuable comments to the paper.

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-1-0163. Research funding for this work was also provided by the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. Martin Pelikan was partially supported by grant VEGA 1/7654/20 of the Slovak Grant Agency. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the

Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U. S. Government.

References

Bäck, T. (1996). *Evolutionary algorithms in theory and practice.* New York: Oxford University Press.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University.

Beyer, H.-G. (1995). Toward a theory of evolution strategies: On the benefit of sex – the $(\mu/\mu, \lambda)$-theory. *Evolutionary Computation*, *3*(1), 81–111.

Beyer, H.-G. (1996). Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, *3*(3), 311–347.

Bosman, P. A., & Thierens, D. (2000). Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In Wu, A. (Ed.), *Workshop proceedings of the Genetic and Evolutionary Computation Conference GECCO 2000* (pp. 197–200). San Fransisco, CA: Morgan Kaufmann.

Cantú-Paz, E. (2001). Supervised and unsupervised discretization methods for evolutionary algorithms. To appear in Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001).

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Goldberg, D. E. (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems*, *5*(2), 139–167.

Hansen, N., & Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proc. of the 1996 IEEE Int. Conf. on Evolutionary Computation* (pp. 312–317). Piscataway, NJ: IEEE Service Center.

Hansen, N., Ostermeier, A., & Gawelczyk, A. (1995). On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In Eshelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 57–64). San Francisco, CA: Morgan Kaufmann.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. In Eiben, A., Bäck, T., Shoenauer, M., & Schwefel, H. (Eds.), *Parallel Problem Solving from Nature - PPSN IV* (pp. 178–187). Berlin: Springer Verlag.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1998). *Linkage problem, distribution estimation, and Bayesian networks* (IlliGAL Report No. 98013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2000). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*. In press.

Pelikan, M., Goldberg, D. E., & Sastry, K. (2000). *Bayesian optimization algorithm, decision graphs, and Occam's razor* (IlliGAL Report No. 2000020). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog.

Rechenberg, I. (1994). *Evolutionsstrategie '94*. Stuttgart: Frommann-Holzboog Verlag.

Rothlauf, F. (2001). *Toward a theory of representations for genetic and evolutionary algorithms - with application to the analysis and design of binary and tree representations*. Master's thesis, University of Bayreuth.

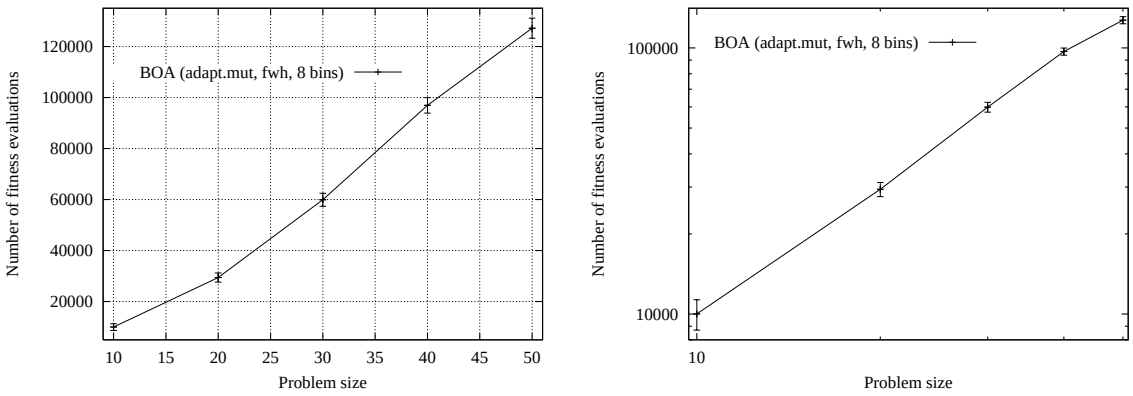
Schwefel, H.-P. (1974, July). *Adaptive Mechanismen in der biologischen Evolution und ihr Einfluss auf die Evolutionsgeschwindigkeit* (Technical Report of the Working Group of Bionics and Evolution Techniques at the Institute for Measurement and Control Technology Re 215/3). Technical University of Berlin.

Schwefel, H.-P. (1977). Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. *Interdisciplinary Systems Research*, *26*.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. New York, New York: John Wiley and Sons.

Schwefel, H.-P. (1995). Contemporary evolution strategies. Berlin, Germany: Springer Verlag.

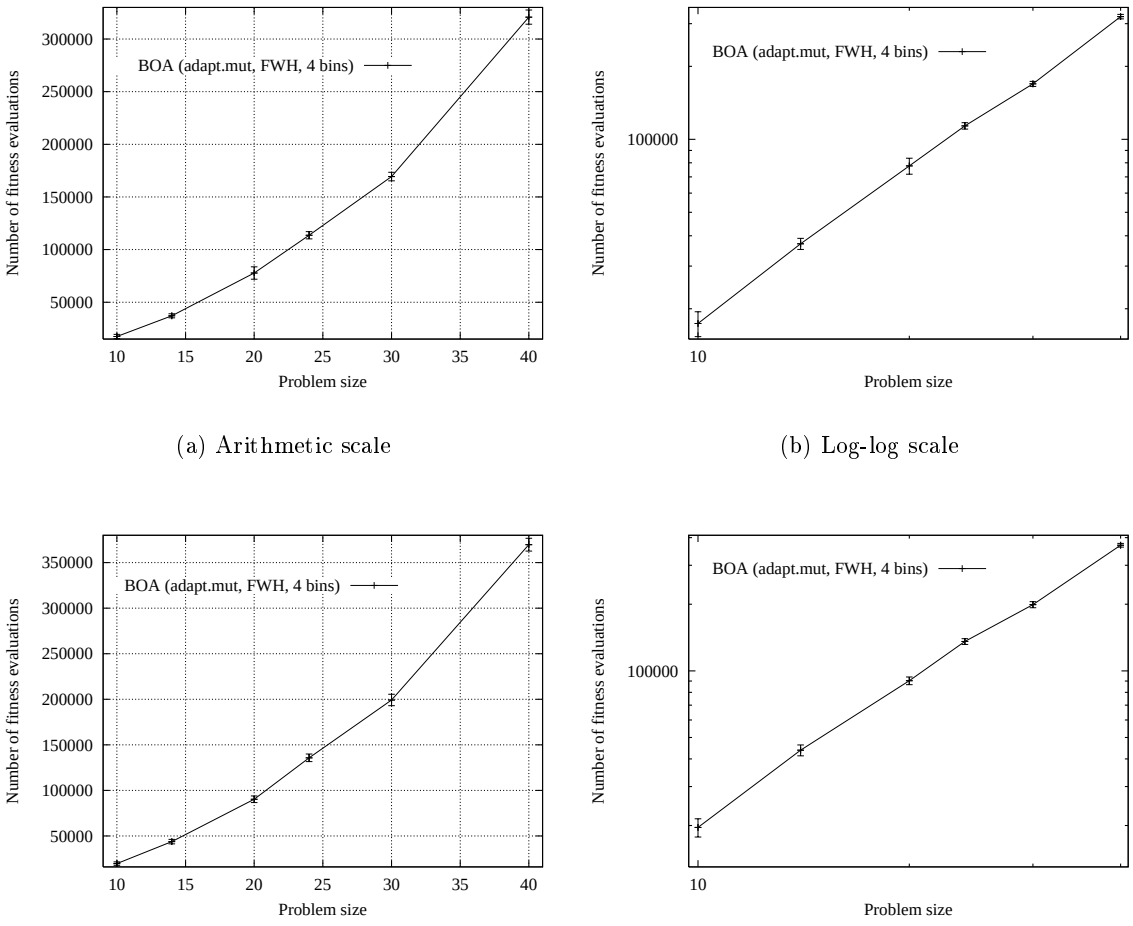
Tsutsui, S., Pelikan, M., & Goldberg, D. E. (2001). *Evolutionary algorithm using marginal histogram models in continuous domain* (IlliGAL Report No. 2001019). Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.



(a) Arithmetic scale

(b) Log-log scale

Figure 8: Results of BOA with fixed-width histograms (8 bins per variable) and σ-self-adaptive mutation on the two-peaks function.



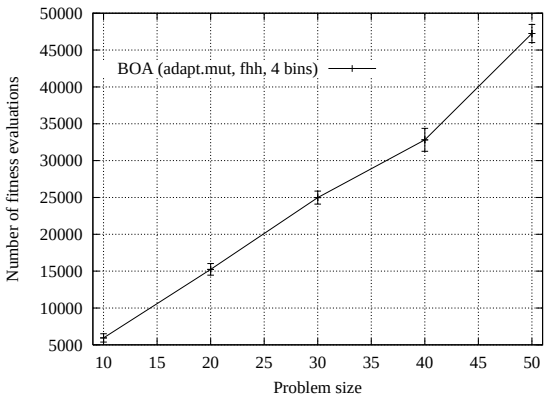
(a) Arithmetic scale

(b) Log-log scale

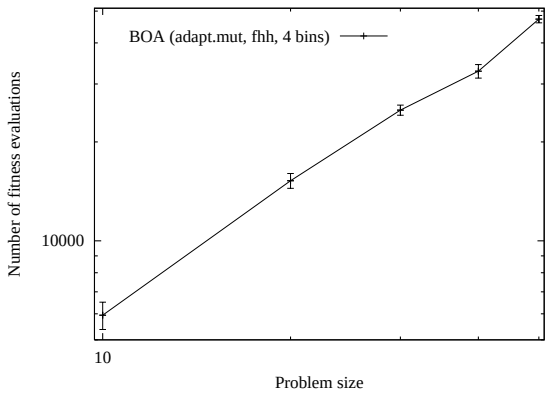
(c) Arithmetic scale

(d) Log-log scale

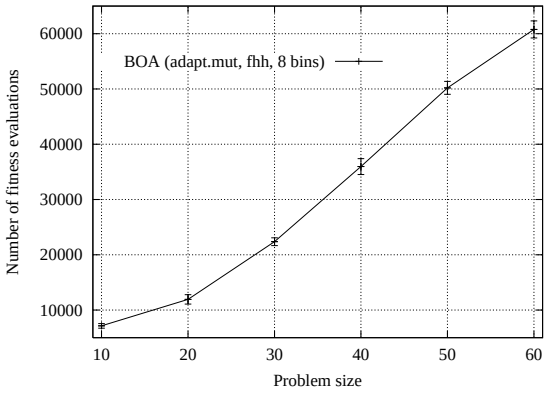
Figure 9: Results of BOA with fixed-width histograms (4 and 8 bins) and σ-self-adaptive mutation on the deceptive function.



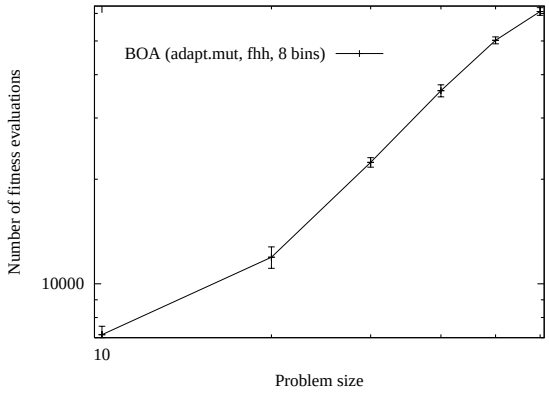
(a) Arithmetic scale, 4 bins per variable



(b) Log-log scale, 4 bins per variable

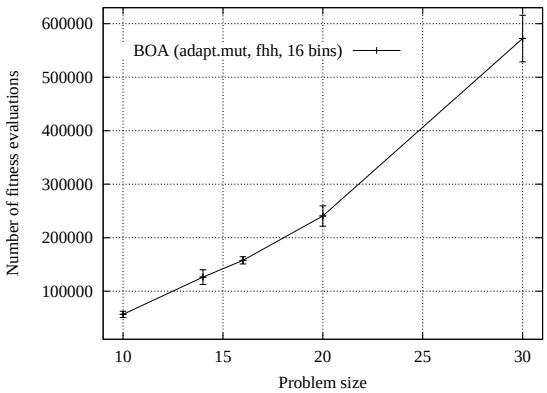


(c) Arithmetic scale, 8 bins per variable

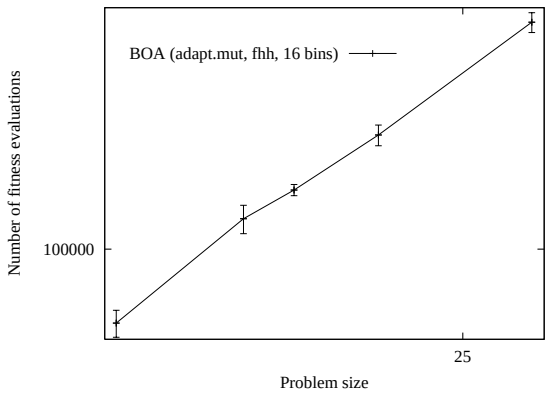


(d) Log-log scale, 8 bins per variable

Figure 10: Results of BOA with fixed-height histograms and σ-self-adaptive mutation on the two-peaks function.

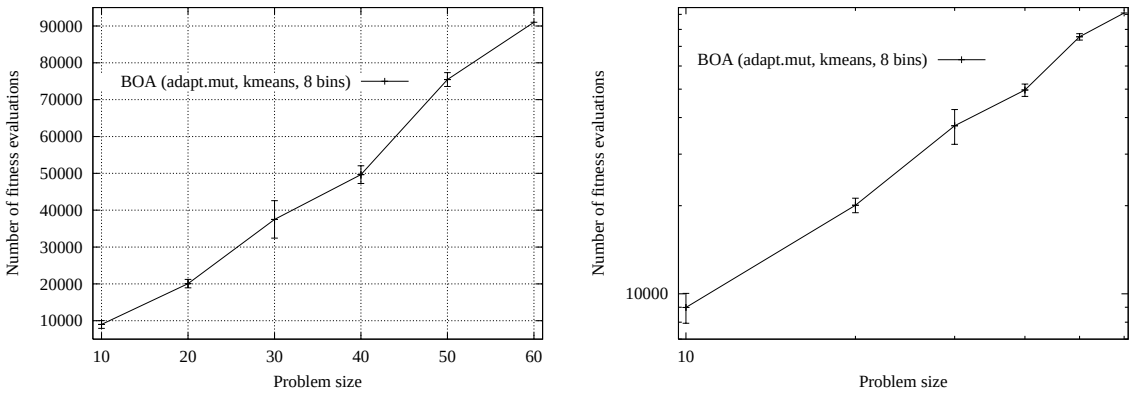


(a) Arithmetic scale



(b) Log-log scale

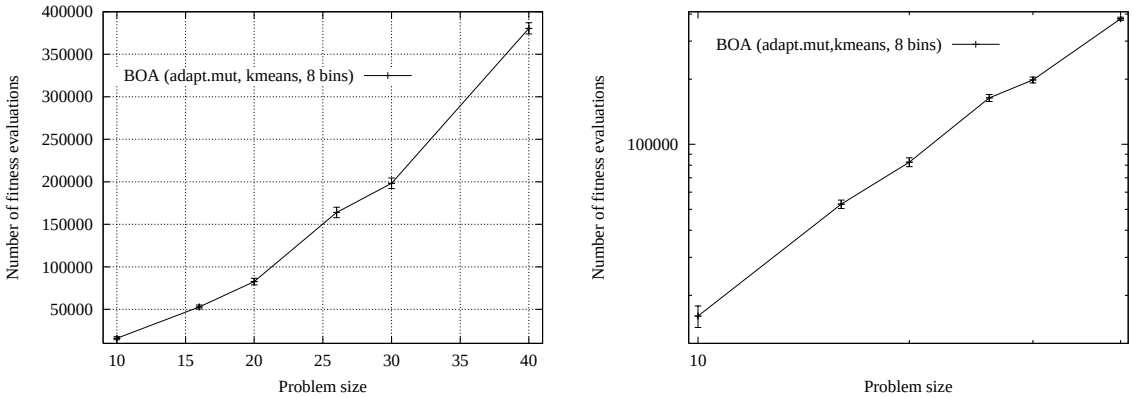
Figure 11: Results of BOA with fixed-height histograms (16 bins per variable) and σ-self-adaptive mutation on the deceptive function.



(a) Arithmetic scale

(b) Log-log scale

Figure 12: Results of BOA with k-means (8 clusters per variable) and σ-self-adaptive mutation on the two-peaks function.



(a) Arithmetic scale

(b) Log-log scale

Figure 13: Results of BOA with k-means (8 cluster per variable) and σ-self-adaptive mutation on the deceptive function.