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

This paper proposes a method that combines competent genetic algorithms working in discrete domains with adaptive evolution strategies working in continuous domains. We use discretization to transform solution between the two domains. The results of our experiments with the Bayesian optimization algorithm as a discrete optimizer and σ-self-adaptive mutation of evolution strategies as a continuous optimizer combined using k-means clustering suggest that the algorithm scales up well on all tested problems. The proposed method can be used to fill the gap between other optimization methods working in continuous and discrete domains and allow their hybridization.

1 INTRODUCTION

In genetic and evolutionary algorithms, the search is guided by selection and variation operators. Selection biases the search towards high-quality regions by making more copies of good solutions and less copies of the bad ones. Variation operators (such as recombination and mutation) ensure exploration of promising regions of the search space after applying selection. There are two commonly used variation operators: (1) recombination and (2) mutation. Genetic algorithms focus primarily on *recombination* that combines solutions by exchanging some of their parts. On the other hand, the dominant variation operator in evolution strategies is *mutation* that perturbs the solutions slightly. 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 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 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 *discretization*. 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 and to solve problems that contain both continuous and discrete variables.

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. Discretization is then discussed in context of genetic and evolutionary computation. 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 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 where the number of copies of each candidate solution is somehow proportional to the solution's quality. New candidate solutions are constructed by applying recombination and mutation operators to the selected solutions. The new solutions replace some of the old ones or all of them 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 is based on recombination and has been recently shown to solve boundedly difficult decomposable problems defined on fixed-length binary strings efficiently and reliably. BOA is capable of learning and exploiting a decomposition of the problem by analyzing the promising solutions. Subsequently, the section describes evolution strategies (ES) that process fixed-length vectors of real numbers and use mutation as the primary variation operator. Several methods for adapting mutation parameters are presented. The section ends by discussing discretization in context of genetic and evolutionary computation that 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

Recombination-based genetic algorithms generate new solutions by combining bits and pieces of promising solutions. The simple genetic algorithm (Goldberg, 1989) uses problem-independent crossover operators to combine promising solutions, such as uniform crossover and one-point crossover. Mutation is usually used as only a background operator 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, 2002) also try to combine important parts of the selected solutions but they approach 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. PMBGAs first estimate the distribution of the selected solutions and then use this estimate to generate new solutions. The estimated distribution can encode the interactions among the different variables in the 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* (Mühlenbein & Paaß, 1996), or *iterated density estimation algorithms* (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 (2002).

In this paper, we focus on the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1998) that uses Bayesian networks to model promising solutions and subsequently guide the exploration of the search space. The first population of strings is generated randomly with a uniform distribution. From the current population, the better strings are selected using one of the conventional selection methods such as tournament or truncation selection. A Bayesian network that fits the selected set of strings is constructed. 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.

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) and the edges defining the conditional dependencies among the variables. A directed edge relates the variables so that in the encoded distribution, the variable corresponding to the terminal node is 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 a conjunctional condition containing all its parents. To learn the network structure, a scoring metric, such as the Bayesian-Dirichlet metric or the Bayesian information criterion (BIC), can be used to discriminate competing models. A search procedure then searches the space of all potential network structures to find the one that scores the most. A greedy search procedure is often used that iteratively adds, removes, or reverses the edge that improves the score of the network the most until no more improvement is possible.

In BOA, the built Bayesian network encodes important interactions in the problem as well as its decomposition. The decomposition simplifies the problem, while the interactions allow the use of reasonably sized populations and convergence times. It has been shown that BOA is indeed capable of learning how to properly break up the problem to optimize the problems decomposable into subproblems of bounded difficulty in subquadratic or quadratic time.

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. Mutation is usually performed by adding a number generated according to a zero-mean normal distribution to the solution. This section reviews the basic principle behind using mutation as the primary variation operator. We start by discussing a simple mutation operator that mutates each variable independently and therefore the non-diagonal elements in the covariance matrix of the mutation distribution are equal to zero. Subsequently, we describe the basic idea behind some more sophisticated approaches that allow adaptation of the covariance matrix.

A simple mutation that mutates each variable independently using a normally distributed random variable contains one parameter per variable. Each parameter specifies the standard deviation of the mutation for the corresponding variable. The standard deviations (mutation strengths) can be either fixed to some small constant or adapted as the search progresses. Ideally, the mutation strength should be proportional to the distance to the optimum. A fixed mutation strength results in slower convergence at either the beginning or the end of the run. Adaptive mutation dynamically updates the mutation strengths to maximize the improvement in the current stage of the algorithm. The 1/5-success rule (Rechenberg, 1973), σ-self-adaptive ES (Schwefel, 1977), and adaptive linear rule (Rechenberg, 1994) are examples of the adaptive mutation strategies.

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:

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

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 based on the fitness of the resulting solutions because individuals that lead to the best improvements are going to participate in the reproduction in the subsequent iteration. Schwefel suggests that τ should be inversely proportional to the square root of the total number of variables:

$$\tau \propto \frac{1}{\sqrt{n}}. \tag{2}$$

In the above method, mutations for different variables are independent. 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. 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 some 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. The generating set adaptation (GSA) method of Hansen, Ostermeier, and Gawelczyk (1995) is an example of a more advanced method for adapting the direction of the mutation together with its strength by adapting the covariance matrix of the mutation distribution.

Although 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 can range from two individuals to the entire parent population. For the value of each variable in the new individual, a random individual is picked from the subset and the value is copied from that individual.

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 past mutations of this variable. Since this information is associated with each variable, no major modifications are required.

ES with adaptive mutation are powerful for local search. However, without powerful recombination ES are capable of only local search. So why not combine recombination-based methods capable of learning how to recombine the solutions properly in a discrete domain and mutation-based methods capable of adapting the mutation in a continuous domain? Section 3 proposes a method that combines the two approaches using discretization to transform continuous solutions into a discrete domain and vice versa. But first, 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 reduce the complexity of a problem and make in-

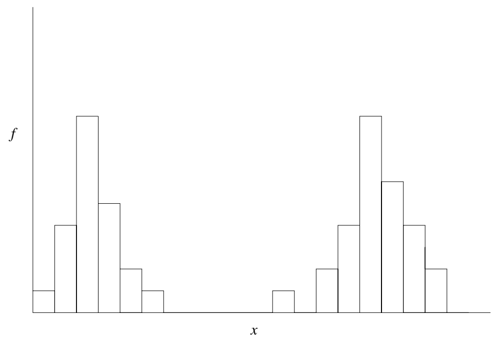


Figure 1: Fixed-width histogram.

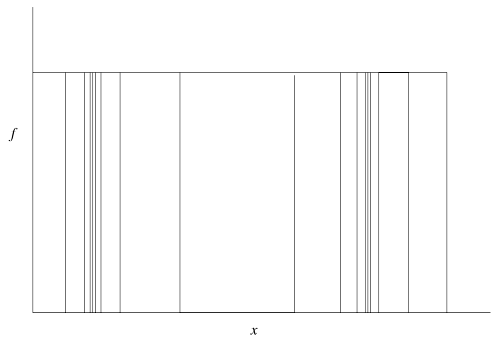


Figure 2: Fixed-height histogram.

tractable 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 the conjugate gradient.

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 at the same time. 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 distant in the discrete domain. This affects especially mutation that 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 extremely 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. For many problems only a couple of bits (say, $k = 3$ or 4) are 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.

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 points are allowed to lie within the intervals and not only on their boundaries. That can 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). Using equal-width histograms was investigated in Bosman and Thierens (2000), Tsutsui et al. (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 characteristic—they map a single continuous variable or a group of variables into a finite set of symbols. We discuss some of these methods in the following.

2.3.1 Histograms

Histograms divide the interval for each variable into k subintervals (bins). There are many ways of dividing the interval into k bins. In practice, two basic types of histograms are used: (1) fixed-width histogram and (2) fixed-height histogram.

A fixed-width histogram divides the interval into k bins of equal width. An example of a fixed-width histogram is shown in Figure 1. The disadvantage of fixing the width of each bin is that if points are concentrated in a couple of small regions, only a couple of bins will be nonempty and many bins will simply be wasted on regions with none or only a few points. Fixed-width histograms are also very sensitive to outliers and one or a few points far away from the rest can significantly decrease the accuracy.

A fixed-height histogram divides the interval for the variable in k bins of equal frequencies (each bin contains the same number of points). An example of a fixed-height histogram is shown in Figure 2. The advantage of using fixed-height histograms is that the density of bins is increased in regions with many points. The regions that seem interesting (those that contain many points) are modeled with high accuracy, while the bins with only few points are merged together to decrease the accuracy where it is not needed. A fixed-height histogram can therefore preserve more information contained in the original set of points.

2.3.2 k-means Clustering

In k-means clustering, each cluster (category) 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 next section describes how to use a particular discretization or clustering method to combine BOA (or other discrete optimizer) with ES (or other continuous optimizer).

3 COMBINING LINKAGE LEARNING AND ADAPTIVE MUTATION

This section describes an algorithm that combines a discrete recombination-based algorithm (such as BOA) with adaptive mutation techniques of ES.

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

1. Discretize the selected promising solutions.
2. Recombine the discrete solutions.
3. Map the new discrete solutions back in the continuous domain, update the mutation parameters, and mutate the new continuous solutions.

In the first phase, the promising solutions are discretized. Each variable is independently mapped into a finite number of categories (bins, clusters). Any discretization, clustering, or classification method can be used to discretize the continuous variables. Let us denote the resulting number of categories 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 an 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, alphabets of higher cardinality result in shorter representation.

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 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 the 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 chosen. The value of the variable is obtained by mutating the value of the variable in the chosen individual.

As a simple example, let us assume we use an equal-width histogram with only two categories for each variable. Each candidate solution is represented in the discrete domain by a binary string with one bit per variable determining whether the variable is in the upper or lower half of its range. To decode a binary string, we look at the value of each of its bits. If the value is 0 (1), we randomly choose a solution from the original 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 chosen 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 attach the mutation

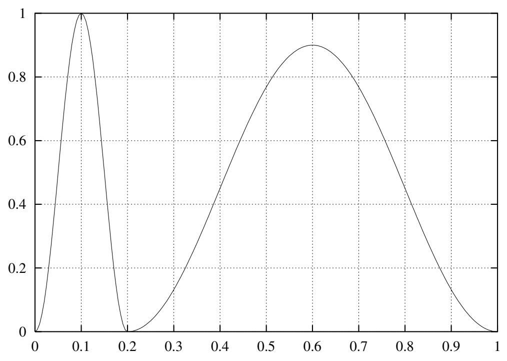


Figure 3: Two-peaks function.

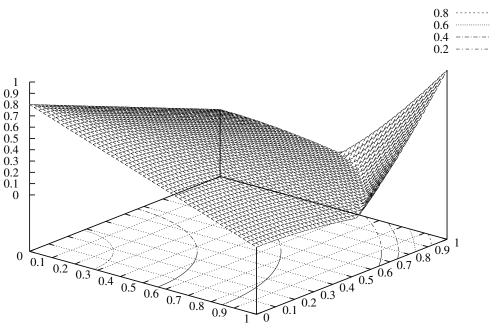


Figure 4: Two-dimensional deceptive function.

strength to each variable in each string. When we copy a value of a particular variable we must 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 1). The new mutation strengths are used to mutate the created solution. GSA and its successors require copying and updating a history of the mutations or other parameters as well.

The newly generated solutions then replace the original population or its part.

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 GSA is straightforward. To discretize the solutions, we used equal-height histograms, equal-width histograms, and k-means clustering, but any other 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, see Cantú-Paz (2001).

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 and the goal is to maximize the functions. 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

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

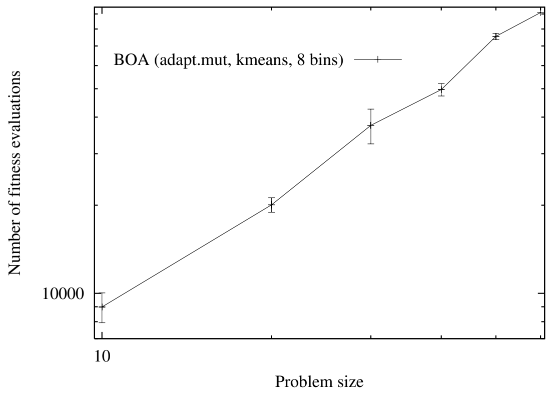
where f_{peak} is a simple function for one peak, defined as

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

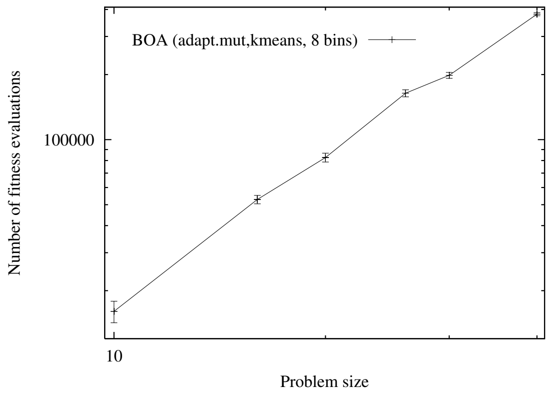
Figure 3 shows the two-peak function. The function has one local and one global optimum for each variable. This yields 2^n optima for a problem of size n out of which only one optimum is global. Local optima are much wider and almost as high as the global one. That makes the problem more difficult. Using mutation only does not yield good results on this problem. Recombination makes uses decomposability of the problem and is capable of solving the problem very efficiently and reliably. Simple uniform crossover is sufficient and thus any ES with recombination should work well.

The deceptive function is composed of two-dimensional deceptive functions:

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



(a) Two-peaks function.



(b) Deceptive function.

Figure 5: Results of BOA with k-means (8 clusters per variable).

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

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

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 4 shows the two-dimensional deceptive function. The two-dimensional deceptive function requires that we learn the linkage of the contributing pairs of 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 are more points on the local attractor than the global one. If both variables are treated independently, combinations with both variables near the global attractor vanish and the search progresses toward the local optimum. Moreover, the global optimum is almost isolated and the attractor is small. This makes it quite difficult to hit the global attractor.

4.2 RESULTS

This section presents and discusses empirical results that primarily focus on the scalability. BOA with σ-self-adaptive mutation strength with a learning parameter $\tau = 4/\sqrt{n}$ is used. Due to the limited size of the paper, we only present the results of using k-means clustering for discretization. However, the results of other discretization methods are comparable. In all experiments, a binary tournament selection with replacement is used where to select each new individual, a tournament among two randomly selected individuals is performed and the winner of the tournament is added to the mating pool. An elitist replacement scheme is used that replaces the worst half of the population by the offspring.

For each problem size, we performed 30 independent runs with the optimal population size that was determined empirically for each algorithm and problem size so that the optimum is found in all 30 runs. The average number of fitness evaluations to reach solutions whose Euclidean distance from the optimum is at most 0.01 is provided.

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 algorithm is capable to solve both simple and difficult problems. Without recombination, the ES with σ-self-adaptive mutation can not solve any of the discussed problems efficiently. 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.

Figure 5 shows the results of the proposed algorithm with k-means clustering on the two-peaks and deceptive functions. For the two-peaks function, the population sizes ranged from $N = 700$ for $n = 10$ to $N = 2100$

for $n = 50$, and the required number of evaluations is approximately $O(n^{1.32})$. For the deceptive function, the population sizes ranged from $N = 900$ for $n = 10$ to $N = 8750$ for $n = 40$ and the required number of evaluations grows approximately with $O(n^{2.28})$. Therefore, the performance in both cases can be estimated by a low-order polynomial.

5 CONCLUSIONS

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.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. The US 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, or the U.S. Government.

References

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.

Bosman, P. A., & Thierens, D. (2000). Continuous iterated density estimation evolutionary algorithms within the IDEA framework. *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 197–200.

Cantú-Paz, E. (2001). Supervised and unsupervised discretization methods for evolutionary algorithms. *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 213–216.

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., & Gawelczyk, A. (1995). On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. *Proceedings of the International Conference on Genetic Algorithms (ICGA-95)*, 57–64.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, 178–187.

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. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*(1), 5–20. Also IlliGAL Report No. 99018.

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

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

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*, Volume 26 of *Interdisciplinary Systems Research.* Basle, Switzerland: Birkhäuser.

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

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