CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF ELECTRICAL ENGINEERING

DEPARTMENT OF CYBERNETICS

Minimal PhD Report

Optimization of Evolutionary Algorithms

March 2003

Petr Pošík

Contents

1 Introduction to Evolutionary Algorithms

Evolutionary algorithms (EAs) are already known for a few decades and they proved to be a powerful optimization and searching tool in many research and application areas. Primarily, they are intended to be algorithms for searching the space of possible solutions of a problem at hand. In the field of artificial intelligence (AI), there exist more conventional algorithms for searching the space, e.g. local search, hill-climbing, steepest descent etc. However, the EAs differ from these iterative algorithms by one important thing – they evolve the whole population of potential solutions, rather than only one.

EAs are inspired by the processes that can be found in nature, namely by natural selection and variation. Talking about selection, it is implicitly assumed that behind the scene, there is some living environment which defines how fit each particular individual is, i.e. how likely it is that the individual survives to next generation.

Some people in evolutionary computation (EC) community divide the EAs to four groups: (1) *Evolutionary programming* (EP), which works with finite automata and numeric representations and has been developed mainly in America since 1960s, (2) *Evolutionary strategies* (ES), which work with real (rational) numbers and was invented in Germany in 1970s, (3) *Genetic algorithms* (GAs), that work with binary representations and have been known since 1970s, and (4) *Genetic programming* (GP), which works with programs represented in the tree form and is the most recent member of the EA family. In fact, there is no need to draw hard boundaries between these four types of EAs. They influence each other, they cooperate by exchanging ideas and techniques. Although there can be found some differences between them, the boundaries are very soft and exist only in our minds.

To be allowed to call an algorithm 'evolutionary', the algorithm must work with a *population* of potential solutions (*individuals*), and after *initialization*, it must operate on the basis of iterative alternating between the following two phases:

1. *Variation.* From the individuals in the population at hand, it 'somehow' creates new generation of individuals (hopefully better, on average). The word 'somehow' describes e.g. the use of genetic operators, crossover and mutation, in case of GAs, or performing small perturbations of individuals in case of ES, etc.

2. *Selection.* This operation 'somehow' prunes the population, so that worse individuals are less likely to be included in the resulting population. It acts as the survival-of-the-fittest principle in the nature.

This description is very broad – there are plenty of algorithms that could be called 'evolutionary'. In the next subsection, I describe the basic methods and principles of conventional EAs in more detail. Then the problems the EAs must face are pointed out, and the meaning of the term 'optimization of evolutionary algorithms' is made clear.

1.1 Conventional EAs

In order to solve a task by evolutionary algorithm, the designer has to think of several things: he must formulate the problem as an optimization task and choose proper evaluation function, he has to decide what representation of the solution is the most appropriate for the task at hand, he must assess the selection method, and he must set up the way how the old individuals should be recombined to breed new generation of potential solutions.

1.1.1 Fitness function

The objective of the given problem needn't be always specified as an optimization task. We have to transform it to that form by creating so called *fitness function*. Then, searching for the optimum of the fitness function should be equal to solving the original problem. The right fitness function, which is sufficiently informative and correctly defined, is a necessary condition for the success of the problem solving process. The fitness function plays the role of a living environment for the individuals – it tells us which of the individuals is better than the others.

1.1.2 Representation

Choosing suitable representation for the individuals (potential solutions) is closely related to the creation of fitness function. The fitness function must be able to evaluate the individuals in the format which we choose in this step.

The representation is also the factor that discriminates between the four types of EAs. If we choose the binary (or another discrete) representation, we will use a genetic algorithm to optimize the fitness function; in case of real number representation, we will probably use an evolutionary strategy or an evolutionary programming; if

we choose a tree representation, we possibly want to evolve programs, and we will use a genetic programming. However, there is no general rule what representation should be selected. In general, it is often recommended to use such a representation that is the most 'native' to the given problem.

1.1.3 Selection

As I already mentioned above, all EAs iteratively switch between two main parts of the algorithm: variation and selection. Let me first describe the selection phase. Selection is common to all versions of EAs and is independent of the type of variation. In other words, although there are plenty of selection methods, they can all be used across all types of evolutionary algorithms (GAs, ES, EP, GP). The basic types of selection operators are:

- *Truncation.* Just sort the individuals according to their fitness values and select first n of them.
- *Roulette wheel.* Select n individuals according to a probability distribution given by the fitness values of the individuals.
- *Rank.* Select n individuals according to a probability distribution given by the rank of individuals when they are sorted from the worst one to the best one.
- *Tournament.* Repeat n-times a tournament between two (or more) randomly selected individuals. Select the better one.

There are, of course, other types of selection operators, e.g. the Reminder Deterministic Sampling and Reminder Stochastic Sampling, which were developed to better approximate the original distribution of individuals for finite populations than the Roulette-wheel does, and many others. What they have in common is the fact, that they try to suppress the worse individuals and help better individuals to spread over the population.

1.1.4 Variation

Variation is the part of EA that is responsible for the actual search in the state space. In other words, variation samples new data points in the search space based on the information that is contained in the population of individuals. Variation itself is strongly related to the representation of individuals.

- *Binary.* Binary representation is used in genetic algorithms. Variation is carried out by successive applying two operators, crossover and mutation, with certain probabilities.

  Crossover is the primary operator and is based on the Mendel's theory of genetics. In its basic version, it takes two individuals from population and combines them somehow, trying to preserve the good features from both of the parents to at least one of the offspring. This can be done in several ways – the simplest one is to take several bits from one parent, the reminder from the second parent, and build up a new individual.

  Mutation is a supplementary operator which is used to preserve diversity of the population. It usually takes an individual from the population and performs a small change on it. It can e.g. flip a bit on a random position with a small probability. This way, the individual is moved in the search space to some point which is located in the local neighborhood of the individual itself.

- *Real.* Real representation is usually used in evolutionary strategies or evolutionary programming (in computers, real numbers are approximated by rational numbers). The primary operator is the perturbation here, which more or less resembles the mutation from GAs. In the basic versions of evolutionary strategies, there was nothing like crossover. In more recent versions, the ES were inspired by GAs and the crossover became a secondary operator.

  The perturbation is carried out in more flexible way than is the mutation in GAs. The special thing about ES is the fact that the individual is not formed only of the real vector (which is being optimized) – it contains also vector of standard deviations. The perturbation is then performed in such a way, that a random vector which came from a multidimensional gaussian distribution is added to point of the search space. The normal distribution has a diagonal covariance matrix with the standard deviations on the diagonal. The vector of standard deviations is then reduced by multiplying with factor less than 1.

  The crossover can be performed in a similar way as in the case of GAs – with the exception that it is applied on the standard deviations part of the individual too. However, we can come out with other possibilities how to cross two real representations – we can e.g. average them.

- *Permutation.* Evolutionary algorithms are often used to solve permutation problems, like the traveling salesman problem (TSP). There is no special type of EA which would be completely dedicated to permutations; they were successfully solved by GAs, ES, etc. We have to transform the permutation to the native representation of the GA or ES, i.e. find the way how to make a permutation from a bit string and vice versa, or how to make a permutation from a string of real numbers and vice versa.

  For the TSP, several specialized crossover and mutation operators were developed. They are based on preserving good subtours in individuals and on compiling them together. The TSP (and permutation representation in general) is a perfect example of one issue related to EAs – the particular components of individual are not independent. I will come back to this in next subsection.

- *Trees.* When working with trees, there exist operations which will serve as crossover and mutation. When recombining two trees, we can e.g. exchange subtrees in them. As a mutation, we can delete a subtree, insert randomly generated node, etc.

  Genetic programming uses trees to evolve computer programs. It uses a bit different operators to achieve more reasonable outcomes when recombining two programs, or modifying a program.

The designer of EA is not limited only to these representations which are mentioned here. We can evolve e.g. matrices, graphs, nets, sets of rules, images, designs of industrial facilities, etc. We can also use the basic types of variation operators mentioned above, however, it is very likely that some more advanced specialized operators will have substantial influence on the efficiency of our EA.

1.2 Problems with Conventional EAs

Evolutionary algorithms are very powerful and flexible framework for solving optimization problems. However, for each particular problem, there might exist a better specialized algorithm which would solve it more reliably. This flexibility and wide scope of EAs applicability is the strength and the weakness of EAs in the same time.

1.2.1 Instantiation of EA

Since the EA describes only the general framework how to solve problems, one must choose the right instantiation of EA which will be suitable for his problem at hand. By the word 'instantiation', I understand the setup of the EA – chosen representation, the size of population, selection method, variation operators, probabilities of applying these operators, etc. For some types of EAs, the amount of 'things to be set up' is really large, and this can be the source of frustration of many EA designers. To find really good instantiation is often very hard problem itself.

This is closely related to the so called *No-Free-Lunch Theorem*: All searching procedures based on the sampling of the search space have –on average– the same efficiency across all possible problems. Something similar holds even for the representation, and for the variation operators.

When designing the EA, we have to apply the domain specific knowledge, which will help the algorithm to 'search in the right direction', or we have to develop such an algorithm that will gain this knowledge from its own evolution. I will come back to this in next subsection.

1.2.2 Premature Convergence

During the evolution it can easily happen that one sub-optimal individual takes over the whole population, and the search almost stops. This phenomenon is called *premature convergence*. It is not problem itself, rather it is a symptom of bad instantiation of the EA. Some types of EAs are more resistant against this phenomenon – the algorithms that rely more on 'mutation' type of variational operator than on the 'crossover' type. One of the EA parameters has significant effect on the premature convergence – the population size. If it is small, there is much bigger chance that the premature convergence emerges.

1.2.3 Linkage, Epistasis, Statistical Dependency

One of the most severe problems related to the use of EAs is the *linkage problem*. Conventional EAs treat the components of the individual independently of each other. The linkage problem (or epistasis) arises when the condition of independence doesn't hold – that is almost in all practical problems.

To describe it more concretely, imagine the following example: suppose we produce the fuel for automotive engines. We have also many additives and we want to optimize the characteristics of the resulting mixture e.g. in terms of combustion quality. We can choose a binary string as a representation – if the bit is set, the respective additive is present in the mixture, if the bit is zero, the mixture lacks the additive. It's very easy

to imagine this situation: adding substance A has no effect on the quality of combustion, the same holds for substance B, however, the quality of combustion is better, if we add substances A and B in the same time, and can be for example worse, if we added substance C to A and B, etc. There are nonlinear interactions between variables; some variables contribute to fitness value in different ways, depending on values of the other variables.

1.3 What is "Optimization of EAs"?

The summary of problems outlined in previous subsection is not complete and exhaustive at all, but it contains the most severe ones. By the term 'optimization of evolutionary algorithms' I understand the effort to improve the EAs (or to create new types of EA), which will overcome some of the above mentioned problems.

This effort can be aimed at creating algorithms which have lower number of parameters that must be set by the designer, and are more robust so that we can apply them to broader range of problems without substantial changes. The effort can be aimed at automatic control of some EA parameters so that the algorithm itself actively tries to fight the premature convergence. The effort can be aimed at detection of the relations between variables, and at incorporating this extra knowledge into the evolutionary scheme. The possibilities of 'EA optimization' are countless, however they can be classified to several groups:

1. Theory
2. Deterministic Heuristics
3. Feedback Heuristics
4. Self-Adaptation
5. Meta Optimizers
6. New Types of EAs

The following subsections review several attempts to make the EAs more perfect.

1.3.1 Theory

It would be great if we had a kind of EA model which would tell us, how the particular instantiation of EA would behave. This model could give us answers on questions like 'What is the right population size?', 'When should we expect the convergence of EA?', or 'Will the EA converge to global or local optimum?'. Such models, however, don't exist at present, and I doubt if such a general model will exist at all. The interactions between individual parts of EA can be easily described, but hardly analyzed. The lack of theoretical models is very typical for the area of evolutionary computation.

In spite of that, there are some attempts to build theoretical foundations for EAs. They are aimed at GAs, i.e. at binary representations or at strings with alphabets of higher, but finite cardinality. The deduced models are pretty accurate, but they are applicable only to small subset of EA instantiations. Some attempts to build a theoretical apparatus on the optimal population sizing can be found in [15], [16], or [21], and on optimal operator probabilities in [18], [40], or [47]. In [10], there is presented a judgement of these theoretical models: "...the complexities of evolutionary processes and characteristics of interesting problems allow theoretical analysis only after significant simplification in either the algorithm or the problem model. ...the current EA theory is not seen as a useful basis for practitioners." This is very similar to my opinion mentioned above.

1.3.2 Deterministic Heuristics

The evolution is very dynamic process. The characteristics of the population change during time, and from this point of view, it is obvious that static settings of EA components and parameters are not optimal. The EA just needs different settings for the beginning of evolution, when it has to explore the whole search space, and for the final phase, when it has to exploit the possibilities of all the candidate solutions in promising areas of the space.

From here, it is only short step to deterministic parameter control. We can for example set up a 'cooling schedule' for the mutation step (high at the beginning, low at the end). One could argue, that finding the right time schedule for some parameter must be even more hard than finding the one 'optimal' value of a static parameter. However, it turned out that even suboptimal choice of schedule often leads to better results than a near-optimal choice of static value.

Deterministic schedule for the probability of mutation can be found in [12]. [25] presents a mutation operator which has the property to search the space uniformly initially, and very locally at later stages. In [26], the penalties of solutions to constrained optimization problem are transformed by deterministic schedule resulting in dynamic evaluation function.

1.3.3 Feedback Heuristics

It is intuitively obvious that deterministic schedule for changing some parameter values can't be optimal. The moments, in which the modifications take place, are not known in advance as we do not know the actual progress made by the algorithm. Rather, the changes in various components should be triggered by some event in population. E.g. if the diversity of the population drops under some specified treshhold, the mutation can be increased (either in terms of mutation step, or mutation probability, or both). The typical feature of EAs using feedbeck heuristics is monitoring some statistics of the population (relative improvement of population, diversity of population, etc.).

In [27], mutation and crossover are used separately to create new individuals. The operator, that is more successful, is used more often. Similar principle is used for multiple crossover operators in [7] where the operators successful in creating better offsprings are rewarded. [31] presents a parallel model where each population uses different settings for mutation probability, crossover rate and population size. After certain time period, the populations are compared and the settings are shifted towards the values of the most successful subpopulation. In [45], a very complex adaptive scheme for the representation is presented. It adapts the mapping of variables (resolution, range, position) based on convergence of the genes and variance of the genes. [49] adapts the representation too by multiple restarts of GA; each epoch uses different settings for the resolution, position and range of variables. Other types of representation adaptation can be found in [41], or [17]. A co-evolutionary approach was used to adapt evaluation function e.g. in [35].

1.3.4 Self-Adaptation

The roots of self-adaptation we can find in ES. In evolutionary strategies, the mutation steps are part of individuals and they undergo the evolution together with the actual solution of the problem. This principle has spread out to other types of EAs and is widely used in the EC community.

Self-adaptation of mutation rates in GA can be found e.g. in [1]. The individuals contains not only data points from the search space, but also a mutation coefficients which are used to mutate the mutation probability itself. In [13], a self-adaptation is used to control the relative probabilities of five mutation operators for the components of a finite state machine. [46] added one extra bit to the individuals to discriminate if 2-point or uniform crossover should be used. [40] presents a self-adaptation of the number and locations of crossover points. A self-adaptive approach in the context of genetic programming is presented in [28]. Self-adaptation is used even for the representation, mainly in GAs that use diploid chromosomes. To each individual, one additional bit is connected and it determines if one should use the chromosome itself or its inversion (see e.g. [19]).

1.3.5 Meta Optimizers

Finding the best settings for EA is an optimization task itself. We can therefore use optimizers to optimize the EA which is solving (at the same time) the actual optimization problem. This procedure usually involves letting the EA run for some time with certain settings and 'measuring' its performance. Although we could use traditional searching algorithms (local search) as meta optimizers, in EC community the EAs were often used. Example can be found in [20] where a meta-GA is applied to control the parameters of another GA (including population size).

1.3.6 New Types of EA

Although many researchers reported promising results using adaptation in EAs, it is not clear if the adaptation can solve our problems completely. As stated in [10], one of the main obstacles of optimizing the EA settings is formed by the epistatic interactions between these parameters. The mutual influence of different parameters on EA behaviour is very complex, so that the basic forms of EAs, which treat the variables independently of each other, may not be successful in finding optimal settings. Furthermore, if we let the EA itself to find the right settings, we carry out the search in two separate spaces (in the space of possible solutions to our problem, and in the space of possible instantiations of the EA) which make the task much more complex. At this point another question arises: will the search in the larger space pay off? Solution of the above mentioned problems is not trivial (if it exists at all).

However, new types of evolutionary algorithms can be of great help. Such kind of algorithms that have less parameters to set (or optimize), and are capable of covering interactions between variables. Estimation of distribution algorithms belong to this class.

1.4 Roadmap

Section 2 presents a description and basic principles behind the estimation of distribution algorithms. Section 3 describes the work of mine carried out in this field. And, finally, section 4 contains my plans for the future.

2 Estimation of Distribution Algorithms

2.1 Introduction

In section 1, the basic functionality of evolutionary algorithms is shortly described. The principle they use is very simple: the EA usually starts with randomly initialized population of individuals from which more promising ones are selected to breed new individuals by means of (general or specialized) variation operators. This process repeats until some termination condition is satisfied.

However, classical variation operators do not exploit the information hidden in all individuals in the population. E.g. crossover takes two parents and combines them to produce two offsprings. It can use only the information carried by the two parents. And here the following question arises: Couldn't we produce better offsprings if we had more information? From this, it is only short step to crossovers with more than two parents.

We can for example use the Brent's method of numeric optimization: randomly choose two points, sample one additional between them, fit a quadratic function to these three points, and create one offspring as the extreme of the fitted parabola. We can also use a higher number of parents, combine them somehow, and produce arbitrary number of offsprings.

There is, however, another very general approach to realization of the variation phase. Instead of the classic *select-crossover-mutate* strategy, we can use a *select-model-sample* approach. These algorithms are called *Estimation of Distribution Algorithms* (EDAs), *Probabilistic Model Building Genetic Algorithms* (PMBGAs), or *Iterated Density Estimation Algorithms* (IDEAs), and are described in the rest of this thesis.

2.2 Basic Principles

During evolution, the estimation of distribution algorithms build an explicit probabilistic model. After each selection phase, only good-enough individuals should remain in the population. EDA builds a model which (more or less) describes the distribution of good individuals in the search space. When the model is ready, EDA produces new offsprings by sampling from this model, creating more individuals in the promising regions than elsewhere.

The overall characteristics and features of an EDA as a whole depend only on the type of employed probabilistic model. We can use very simple model which can be built up very quickly; the resulting EDA is capable of solving easy problems very fast and accurately, however it may not solve problems which involve interactions between variables (as ordinary EAs). The main advantage of EDAs is the fact, that we could use a model of arbitrary complexity. If we employ some sophisticated probabilistic model that can cover (possibly nonlinear) dependencies in the population, the resulting EDA will be able to solve very hard problems. Although EDAs will then need large populations and large number of evaluations to solve it, the conventional EAs won't be able to solve such problems at all.

2.3 State of the Art

EDAs can be developed for all types of representations, and the EDA approach has already been applied in discrete and continuous spaces, and even in GP for creating programs. In this subsection, I will only present an overview of EDAs in discrete and continuous spaces.

2.3.1 EDAs in Discrete Spaces

Probably the first algorithm for binary representation which uses the principles of EDAs is the population based incremental learning (PBIL) [2]. The PBIL adapts a vector of probabilities that the i-th component of the solution is zero. First, it generates a few individuals in accordance with the probability vector, then it selects the best one, and modifies the vector of probabilities by a kind of Hebbian learning rule. Similar principle uses the compact genetic algorithm (cGA) [22]. As the PBIL, the cGA uses a vector of probabilities, but it adapts it by a different learning rule which resembles a tournament selection. The univariate marginal distribution algorithm (UMDA) [33] is already a proper EDA – it selects the promising solutions and models the distribution of zeros and ones by computing a vector of probabilities.

All the above algorithms treat the variables independently of each other and their behaviour is similar. There are also models which detect the dependencies between pairs of variables. They are the first non-trivial EDAs

as they are able (to certain extent) solve the linkage problem. They differ mainly in the form of model they create; the models include a chain, a tree, or a forest. The algorithm called mutual-information-maximizing input clustering (MIMIC) [8] builds a model of bivariate dependencies in the form of a chain. The sampling is then based on the probability of the first variable, on the conditional probability of the second variable given the value of the first variable, etc. In [3], the dependency trees were used to encode the relationships between variables, while the bivariate marginal distribution algorithm (BMDA) [37] employed a forest distribution (a set of mutually independent dependency trees). These algorithms are able to identify and properly use building blocks of size two, but it is still insufficient to solve more complex problems.

EDAs with more sophisticated probabilistic models can describe even multivariate dependencies between variables. The bottleneck of these methods is the fact that they need complex algorithms to learn the distribution. It results in significant learning time overhead. Furthermore, in almost all cases we end up only with sub-optimal model of the joint distribution. The factorized distribution algorithm (FDA) [34] can contain multivariate marginal and conditional probabilities. However, the factorization of the distribution must be known *a priori*, e.g. given by an expert. The extended compact genetic algorithm (ECGA) [23] uses different principles. It divides the variables into several mutually independent groups. Each of these groups is described by a complete joint distribution. The division into groups is carried out by a greedy manner with respect to Bayesian information criterion. The most complex model for discrete variables that can be learnt is probably a general Bayesian network (the chain, tree and forest are special cases of Bayesian network). The first of EDAs that uses this complex model is the Bayesian optimization algorithm (BOA) [36]. The network for a population of good solutions is learnt every generation by a greedy algorithm which tries to minimize the Bayesian-Dirichlet metric. This network is then sampled to create new individuals. The same class of algorithm was independently introduced in [11] with the name estimation of Bayesian network algorithm (EBNA). The FDA was later improved to be able to learn the factorization, and the modified version is called learning factorized distribution algorithm (LFDA).

2.3.2 EDAs in Continuous Spaces

Similarly to the discrete spaces, the easiest EDAs in continuous spaces take into account no dependencies between variables. The joint density function is factorized by a product of univariate independent densities. E.g. the univariate marginal distribution algorithm for continuous domains (UMDA_C) [29] belongs to this type of EDAs. This algorithm performs some statistical tests in order to determine which of density functions fits the variable best. Then, it computes the maximum likelihood estimates of parameters for chosen densities. Another example of EDA using a model without dependencies is the stochastic hill-climbing with learning by vectors of normal distributions (SHCLVND) [39]. In this algorithm, the joint density is formed as a product of univariate independent normal densities where the vector of means is updated using a kind of Hebbian learning rule and the vector of variances is adapted by a reduction policy (at each iteration it is modified by a factor that is lower than 1). [43] proposed an extension of PBIL for continuous domains (PBIL_C), however their approach is very similar to an instance of evolutionary strategy (see [42]). [44] presents a progressive approach; for each dimension, an interval, in which the search takes place, is maintained, along with a probability that the respective component of the solution comes from the right half of the interval. In case that the probability gets close to 1 (or 0), the interval is updated to the right (or left) half. Finally, there are several approaches that tried to use histogram models (see e.g. [4], [48]).

There are also several attempts to model bivariate or multivariate dependencies between variables. They include probabilistic models based on bivariate normal distributions (MIMIC_C^G), on multivariate normal distributions where the complete covariance matrix is estimated (EMNA), on Gaussian networks (EGNA), etc. (for further information, see [30]). A multivariate normal distribution with full covariance matrix was used also in [5].

Sometimes the good solutions form in the search space multiple "clouds". For this type of distribution, a mixture of normal distributions is very useful model. Example of this approach can be found in [14]; the algorithm proposed there uses even adaptive mixture of multivariate gaussians which can change the number of components during evolution. Learning such complex class of models, however, requires large number of samples, and is usually carried out by some kind of iterative algorithm (e.g. EM algorithm). Another approach can be found in [5]; multivariate gaussian with relatively small variance is centered around each individual. The joint probability function is then a superposition of all those *kernels*. This model has the advantage of capturing various types of distribution; furthermore, we can think of it to be a non-parametric model (when we keep the variance of kernels constant).

3 What has been done?

In my research, I would like to aim at the estimation of distribution algorithms in continuous domain. In this section, I present some preliminary results of my research done so far. The first of the two parts presents an empirical study of continuous UMDA using various marginal probability models. The second part contains empirical comparison of an UMDA against an EDA employing clustering.

3.1 UMDA Using Various Marginal Models

This part of my research clearly builds on and extends the work of Tsutsui, Pelikan and Goldberg [48]. I have repeated several of their experiments and adopted their experimental methodology. However, I track a bit different statistics and look at the data from a bit different point of view.

All described algorithms fall into class of UMDA. The individual components of promising solutions are supposed to be statistically independent of each other. In turn, this means that the global distribution of promising solutions in the search space can be modeled by a set of univariate marginal distributions. Then, for the global distribution we may write

$$p(\mathbf{x}) = \prod_{i=1}^{n} p_i(x_i), \tag{1}$$

where the $p(\mathbf{x})$ is the global multivariate density and the $p_i(x_i)$'s are the univariate marginal densities.

It's important to stress out that the independence assumption doesn't hold for the most of real world problems. The basic UMDA is not able to cover any interactions among variables (independently of what type of marginal distribution model the UMDA uses) and that's its most severe drawback.

In following subsections, several types of marginal densities are described. I want to evaluate the behaviour of UMDA algorithm when using 4 types of marginal probability models. The first two were considered in the work of Tsutsui et al. [48], namely the *equi-width histogram* and *equi-height histogram*. This work expanded the set of models to *max-diff histogram* and *univariate mixture of gaussians*.

3.1.1 Histogram Formalization

Before I will describe particular types of histograms, let's formalize the notion of histogram model. Further subsections will describe various instances of this formalization.

From our point of view, the histogram is a parametric model of univariate probability density. The histogram consists of C bins and each i-th bin, $i \in 1..C$, is described by three parameters: b_{i-1}, b_i are the boundaries of the bin and p_i is the value of probability density function (PDF) if $b_{i-1} < x \leq b_i$.

When constructing histogram, we have a set $\{x^j\}_{j=1}^N$ of N data points. Let's describe n_i the number of data points which lies between b_{i-1} and b_i. Clearly, the values p_i must be proportional to n_i.

$$p_i = \alpha n_i \tag{2}$$

Let's describe d_i the difference $b_i - b_{i-1}$. The following condition must hold if the $p(x)$ is a PDF:

$$1 = \int_{-\infty}^{\infty} p(x)dx = \sum_{i=1}^{C} p_i d_i = \alpha \sum_{i=1}^{C} n_i d_i \tag{3}$$

From this equation we immediately get for the normalizing constant α

$$\alpha = \frac{1}{\sum_{i=1}^{C} n_i d_i} \tag{4}$$

3.1.2 Equi-Width Histogram

This type of histogram is probably the best known and the most used one, although it exhibits several unpleasant features. The domain of respective variable is divided to C equally large bins. Since the size of each bin can be written as $d_i = d = \frac{b_C - b_0}{C}$, the equation for all the values of PDF reduces to

$$p_i = \frac{n_i}{dN} \tag{5}$$

When constructing the histogram, first determine the boundaries of each bin, count the respective number of data points in each bin, and use the equation 5 to compute the values of PDF for this histogram model. Figure 1 shows what the equi-width histogram looks like.

3.1.3 Equi-Height Histogram

This type of histogram was used in the paper of Tsutsui et al. [48] too. Instead of fixing the width of bins, it fixes their heights. Each bin then contains the same number of data points. This means that in areas where the data points are sparse, the bin boundaries are far from each other, while in regions where the data points are dense, the boundaries are pretty close to each other.

Since the probability of each bin is constant, we can write

$$p = p_i = \frac{1}{b_C - b_0} \tag{6}$$

This is true only theoretically. In real cases we could hardly expect that the number of data points can be divided by the number of bins without any reminder. One simple method how to construct this type of histogram is to compute the number of data points which should each bin contain (the difference of these values across all bins is maximally 1). Then we put the boundary of consecutive bins to the half of the distance of the two neighbouring data points where the first one should come into one bin and the second point to the other bin.

It is also possible to apply some kind of interpolation so that we allow that the bins can contain also 'fractional parts' of data points. Then, the number of data points in each bin can be exactly the same (and possibly fractional) for all bins.

Figure 1 shows what the equi-width histogram looks like.

3.1.4 Max-Diff Histogram

This is the first of univariate probability density models which was not covered in Tsutsui et al. [48]. However, this type of histogram is reported to have several good features (very small errors, short construction time), see e.g. the work of Poosala et al. [38].

Max-diff histogram doesn't put any 'equi-something' constraint on the model. Instead, it is defined by a very simple rule. If we construct max-diff histogram with C bins, we need to specify $C-1$ boundaries (the first and the last boundaries are equal to minimum and maximum of the respective variable domain). If we compute all the distances between all two neighbouring data points, we can select the $C-1$ largest of them and the boundaries put to the half of the respective intervals.

This procedure has a natural explanation. We try to find consistent clusters in data, and all data points in such a cluster are put to the same bin. It is just an approximation and we can argue what results this procedure gives when the data doesn't show any structure, but it turned out to be very precise and useful in many situations [38].

Figure 1 shows what the max-diff histogram looks like.

3.1.5 Univariate Mixture of Gaussians

Univariate mixture of gaussians is the last marginal probability model compared in this part. It is included in this comparison for several reasons. We can consider all the histogram models to be certain kind of mixture models too — mixture of uniforms. From this point of view, the mixture of gaussians can give interesting results when compared to histograms.

However, histograms (as mixtures of uniforms) are limited in the 'fidelity' with which they can model the actual distribution. That's why they are usually constructed with pretty large number of bins (i.e. components, in the language of mixture models). On the contrary, the mixture model is capable to describe the actual distribution with higher degree of 'fidelity', thus it allows us to drastically decrease the number of components.

To be objective, I have also to say that it requires much more effort to construct it. In mixture of gaussians, the individual components overlap each other. We can't therefore exactly decide to which component each data point belongs (all data points belong to all components with certain probability). As a result, we have to use an iterative learning scheme — the Expectation Maximization algorithm (see e.g. Moerland [32]).

Figure 1 shows what the mixture of gaussians looks like.

3.1.6 Sampling

I used the sampling method which was described in the Tsutsui paper [48] as Baker's stochastic universal sampling (SUS). It uses the same principles as the (in EC community perhaps more known) reminder stochastic sampling (RSS).

First, the expected number of data points which should be generated by each bin is computed according to the bin (or components) probabilities given by equation 2. Then, the integer part of expected number is generated by sampling from uniform (in case of histograms) or normal (in case of mixture of gaussians)

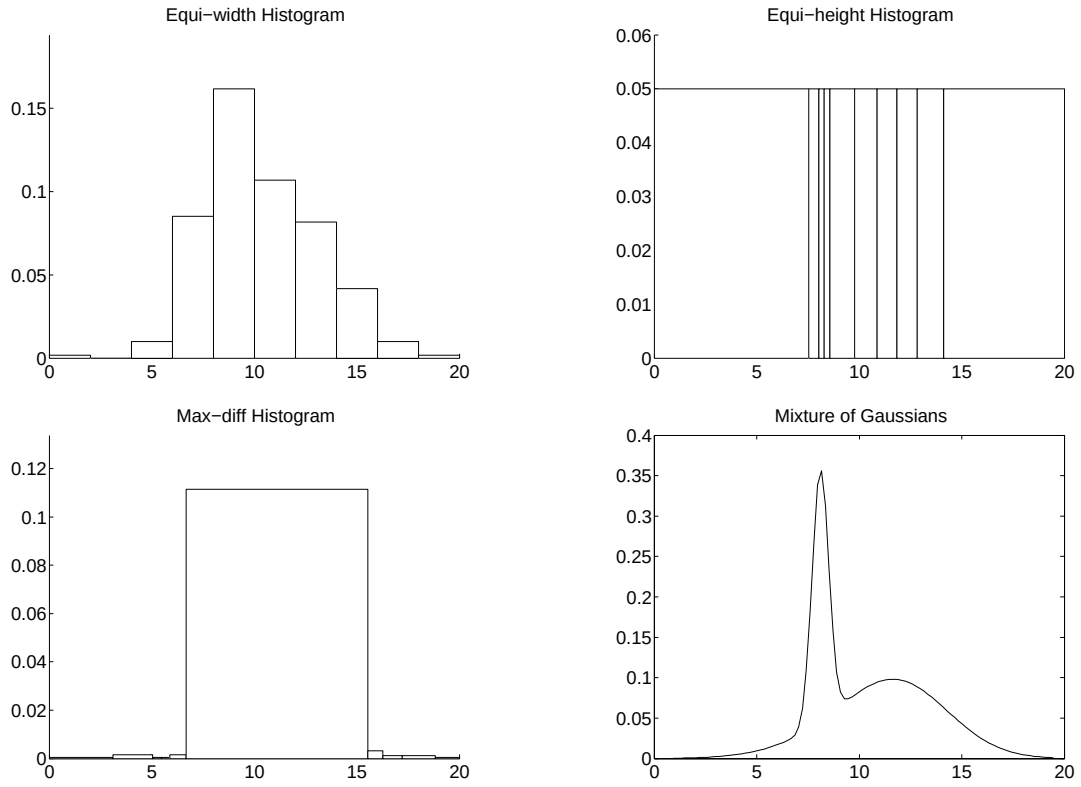

Figure 1: Equi-width, equi-height, and max-diff histograms, and mixture of gaussians

Function	Domain	Number of bins (components)
$F_{TwoPeaks} = 5n - \sum_{i=1}^{n} f_i$	$\langle 0, 12\rangle^n, n = 20$	$Low = 60(3), High = 120(6)$
$F_{Griewangk} = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} + \prod_{i=1}^{n} \cos \frac{x_i}{\sqrt{i}}$	$\langle -5, 5\rangle^n, n = 10$	$Low = 50(3), High = 100(6)$

Table 1: Test functions and related parameters. The f_i function can be described as polyline going through points $(0, 0)$, $(1, 5)$, $(2, 0)$, $(7, 4)$, $(12, 0)$.

distribution with proper parameters. The rest of points is then sampled by stochastically selecting bins (or components) with probability given by the fractional parts of expected numbers for each bin (component).

3.1.7 Evolutionary model

I used the same evolutionary model as Tsutsui et al. [48]. Let the population size be N. In each iteration, the following phases are carried out: (1) model the population with histograms (or mixtures of gaussians), (2) sample N new data points from the model, (3) join the old and new population to get data pool of size $2N$, and (4) use truncation selection to select the better half of data points (returning the population size back to N).

3.1.8 Test Suite

For the actual empirical comparison, I chose only two of the four functions used in the work of Tsutsui [48], namely the Two Peaks function and the Griewangk function. This is very small test set but this comparison is aimed on the suitability and behaviour of individual types of probabilistic models for modeling of marginal probabilities rather than comparing the efficiency of the resulting algorithms on various problems. The functions are defined in table 1.

Both test functions are considered to be easy. The Two Peaks function is separable and there exists very easy algorithm which solves separable problems very efficiently (at least with higher efficiency than here presented UMDAs). It is so called *line search* algorithm (see section 3.1.9). The form of 2-dimensional Two Peaks function shows figure 2.

The Griewangk function represents non-separable function with interactions. However, Whitley et al. [50] mentioned that the influence of interactions gets smaller when Griewangk function is applied in more dimensions.

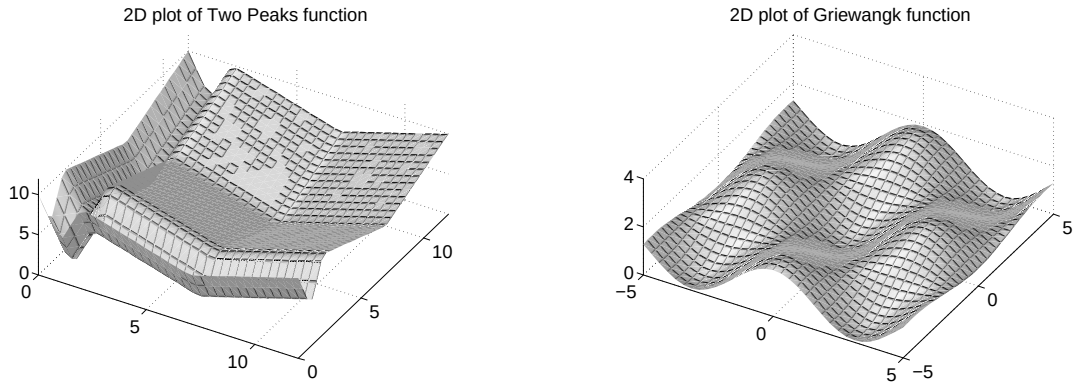

Figure 2: 2D Two Peaks function, and 2D Griewangk function.

However, this function is a bit harder to optimize than the completely separable Two Peaks.

3.1.9 Experimental Setup

For both test problems, Two Peaks and Griewangk, I compared four UMDAs — each using different model of marginal probability. The factor *model* has the following levels: (1) *equi-width histogram* (HEW, in short), (2) *equi-height histogram* (HEH), (3) *max-diff histogram* (HMD), and (4) *mixture of gaussians* (MOG).

For each of the models, the number of bins (components) was varied too. Factor *Number of Bins* has two levels, *High* and *Low*. The exact number of bins depends on several things. First, I calculated the number of bins in order to get precision 0.1 for the equi-width histogram. Specifically for the Two Peaks function, we have the size of the variable domain equal to 12. The precision should be 0.1, so the number of bins have to be $12/0.1 = 120$. This is the actual value of level *High* for histogram models and Two Peaks function. The value of level *Low* is one half of level *High*, i.e. 60 for histograms and Two Peaks function. In similar way we obtain values for the Griewangk function, i.e. $10/0.1 = 100$ for the *High* number of bins, and 50 for *Low* number of bins.

The mixture models are much more flexible and that's why we can use dramatically less components than in the case of histograms. Values for this kind of model were determined by hand after a few experiments. The levels are 6 and 3 for the *High* and *Low* number of components, respectively. The initial values for the centers of individual components of mixture were constructed using the k-means clustering method.

Furthermore, the *Population size* was varried in the experiments. Levels of this factor include populations of sizes 200, 400, 600, and 800 data points.

3.1.10 Line Search Algorithm

In order to give some indication of how complex the test problems are, I tried to optimize them also with so called *line search* (LS) method (see [50]). This method is not intended to be a universal optimization algorithm, rather it is a heuristic which is pretty effective for separable problems and thus it can give us some insight in the complexity of the evaluation function.

Its principle is very easy: start from randomly generated data point, randomly select a dimension, discretize and enumerate it, move to the best data point in the unidimensional enumerated space, and repeat for further dimension.

I have set the discretization length equal to 0.1. This means that for Two Peaks function the LS algorithm evaluates 121 data points for each dimension. If the LS gets stuck (no further improvement possible), it is restarted from another randomly generated point. Similar to UMDAs, the LS is allowed to run for 50000 evaluations in each run.

3.1.11 Monitored statistics

For each of possible factor combination, the algorithm was 20 times. Each run was allowed to continue until the maximum number of 50000 evaluations was reached. We can say, that the algorithm found the global optimum if for each variable x_i the following relation holds $\left|x_i^{best} - x_i^{opt})\right| < 0.1$ (if the difference of the best found solution x^{best} from the optimal solution x^{opt} is lower then 0.1 in each of coordinates).

In all experiments, three statistics were monitored:

1. The number of runs in which the algorithm succeeded in finding the global optimum (*NoFoundOpts*).

2. The average number of evaluations needed to find the global optimum computed from those runs in which the algorithm really found the optimum (*AveNoEvals*).

3. The average fitness of the best solution the algorithm was able to find in all 20 runs (*AveBest*).

3.1.12 Results and Discussion

The results of all experiments are presented in table 2.

Function	Model	Number of Bins (Components)								
		Low				High				
		Population Size								
		200	400	600	800	200	400	600	800	Statistics
Two Peaks	LS	20								NoFoundOpts
		2421								AveNoEvals
		0,00000								AveBest
	HEW	0	1	0	1	1	17	19	20	NoFoundOpts
			13200		47200	5600	10306	15189	19600	AveNoEvals
		5,00560	4,82110	5,05280	4,95510	3,48040	2,51180	2,50030	2,52340	AveBest
	HEH	20	20	20	20	20	20	20	20	NoFoundOpts
		6530	11080	16050	20760	7720	10620	15570	20400	AveNoEvals
		0,10710	0,02070	0,01040	0,07490	0,03740	0,00690	0,00430	0,03520	AveBest
	HMD	20	20	20	20	20	20	20	20	NoFoundOpts
		6270	14920	27960	47080	6770	10980	17490	25280	AveNoEvals
		0,00003	0,00008	0,04990	2,20380	0,00260	0,00001	0,00230	0,05740	AveBest
	MOG	7	20	20	19	16	20	20	20	NoFoundOpts
		6571	11860	17130	23032	5613	10480	15780	20720	AveNoEvals
		0,87390	0,00011	0,00930	0,13970	0,19650	0,00008	0,00660	0,06370	AveBest
Griewangk	LS	18								NoFoundOpts
		14056								AveNoEvals
		0,00250								AveBest
	HEW	13	17	16	14	1	15	18	19	NoFoundOpts
		15954	20353	25763	27886	5800	12320	18867	24926	AveNoEvals
		0,00370	0,00210	0,00230	0,00320	0,00500	0,00081	0,00095	0,00083	AveBest
	HEH	18	18	18	20	16	18	20	20	NoFoundOpts
		6667	12867	18967	25000	6650	12711	18270	23920	AveNoEvals
		0,00074	0,00074	0,00074	0,00000	0,00150	0,00074	0,00000	0,00000	AveBest
	HMD	17	17	18	16	18	17	19	20	NoFoundOpts
		6482	12376	19200	25850	6456	12235	18221	23720	AveNoEvals
		0,00110	0,00110	0,00074	0,00140	0,00074	0,00110	0,00037	0,00000	AveBest
	MOG	15	15	18	19	19	19	17	18	NoFoundOpts
		5693	10613	16100	21726	5894	12084	17682	23644	AveNoEvals
		0,00190	0,00180	0,00074	0,00037	0,00037	0,00037	0,00110	0,00074	AveBest

Table 2: Results of carried out experiments

The results of the line search heuristic indicate that the Two Peaks function could be solved much more effectively by other types of algorithms. Using EDAs to solve it can be compared to killing a midge by a hammer. Nevertheless, the EDAs should be able to solve this easy type of problem. The Griewangk function is more complex and the line search heuristic was not able to solve it in all experiments. This suggests that the test function doesn't lost all its complexity when used in more dimensional form (as mentioned in section 3.1.8).

Let's first consider the results for the HEW model. In accordance with the results of Tsutsui [48], it was confirmed that the HEW model is the least flexible one. Furthermore, it can be characterized by one important feature: if the initial population doesn't contain values from all the intervals for a particular variable, there is no way how a value from this interval could be generated along the course of evolution. In other words, if the population is initialized in such a way that the resulting probabilistic density will be equal to zero somewhere in the search space, the algorithm has no possibilities to overcome it and to start searching in these 'zero areas'. This is also documented by the very weak results obtained by the UMDA with HEW model. If the 'resolution' (the number of bins) of the histogram is lower than the precision required, the algorithm becomes very unprecise. If the 'resolution' is sufficient, the efficiency of the algorithm is very dependent on the population size. This can be seen especially in the case of the Two Peaks evaluation function.

The other three models, HEH, HMD, and MOG, seem to be much more robust with respect to the parameters. All of them also don't exhibit the unpleasant feature of HEW model ('zero areas') described in previous

paragraph. The models are constructed in such a way that they do not allow for a zero probability density (during the evolution, the value of PDF can become 'very small' in some areas, but is never zero, theoretically).

HEH and HMD models showed very similar behaviour, although on these two test functions the HEH model seems to work better. However, the differences are very insignificant. Both are very successful in the number of runs they succeed to find the global optima. The average number of evaluations they need to find the optima is comparable for both models. However, looking at the results table, we can see that the achieved average fitness scores of best individuals are still highly dependent on the size of population.

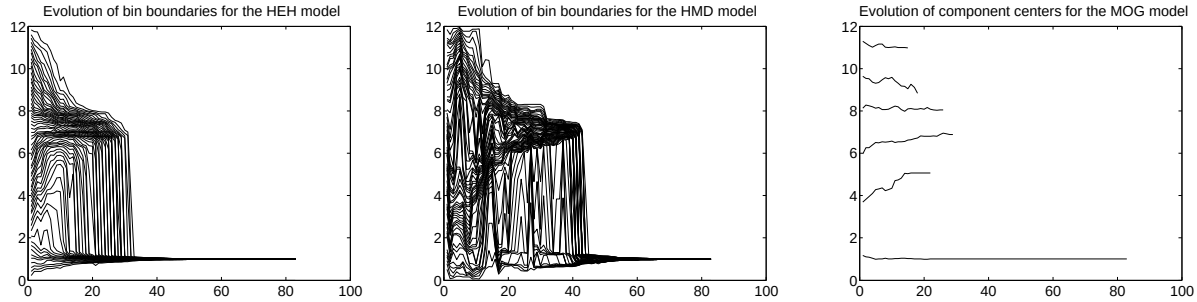

Figure 3: Two Peaks function — Evolution of bin boundaries for HEH and HMD models and evolution of component centers for MOG model.

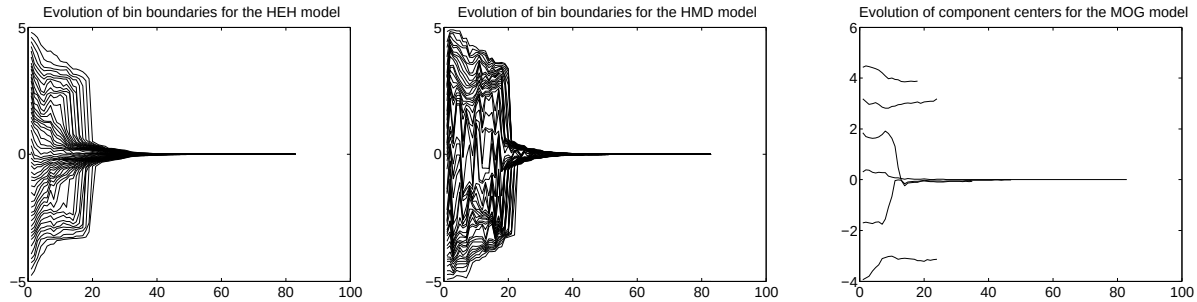

Figure 4: Griewangk function — Evolution of bin boundaries for HEH and HMD models and evolution of component centers for MOG model.

In the figures 3 and 4, the typical 'tracks' of evolution of bin boundaries (for histogram models) and components centers (for mixture of gaussians model) are presented. These *typical* representats show cases in which all the models converged to the right solution (for this particuar dimension) for both evaluation functions. However, to find the global optimum the algorithm had to converge to the right solution in all the dimensions and that's often not the case for the MOG model (as can be seen from the results table).

For the MOG model, we can see that several components of the mixture simply stopped evolving. The reason for this is very simple — their mixture coefficients dropped to zero and the components simply vanished. Generally, this should be a good feature, however, I have experienced several runs in which a particular component vanished before it could take over the whole population although it discovered the right solution (for particular dimension). This can be thought of as an analogy of the 'premature convergence' in genetic algorithms.

I hypothesize that this phenomenon is partially caused by the fact that the individuals are not independent during the evolution. When we create a new population (by sampling from a model), the new individuals are very dependent on the members of the former population (from which the model was created). This way the search is very biased and something like 'premature convergence' can easily occur. That's why the MOG model (which features the most 'fidelity' among the compared models when described in terms of likelihood) exhibits worse behaviour then the HEH and HMD models, because it is usually more biased.

To be objective, a few words on the running times of the algorithms must be said. I observed that the evolution of MOG model took approximately two times more time than the evolution of histogram models. However, this needn't be due to the complexity of the models, rather it can be caused by the complexity of sampling (a normal random generator is usualy much more slower then a uniform random generator). This feature deserves further investigation.

3.2 Influence of Clustering

In the second part of my preliminary research I wanted to evaluate the influence of clustering on the efficiency of the search process. This study compares two algorithms: the UMDA with global equi-height histogram model, and a clustering evolutionary algorithm which uses the same model (HEH) for modeling distribution in each cluster separately. A description of the first type of probabilistic model can be found in 3.1.3. Now, I give a short description of the latter algorithm.

3.2.1 Clustering Estimation of Distribution Algorithm with Histograms

The algorithm is conceptually very easy. During one generation step, the following phases are performed: (1) the data in population are clustered by the k-means algorithm in the D-dimensional space, (2) for each cluster, a HEH model is created, (3) new N points are sampled – proportionaly to sizes of individual clusters, (4) old and new populations are merged, and (5) the truncation selection is carried out to come back to population size N.

The clustering EDA can be seen as an alternative to a fine-grained model of an ordinary parallel EA. We can imagine, there are several subpopulations (clusters) which are evolved separately, however the boundaries between them are somewhat fuzzy (an individual can be generated by one subpopulation, but in the next generation it can be assigned to another subpopulation).

3.2.2 Performing the Clustering

As already mentioned, the k-means clustering with euclidean metric was used. I didn't used a fixed number of clusters. Instead, I let the algorithm itself to decide the number of clusters. The Calinski-Harabasz index [6] was used. It is defined as follows:

$$CH(k) = \frac{\frac{BSS(k)}{k-1}}{\frac{WSS(k)}{n-k}}, \tag{7}$$

where $BSS(k)$ is between-group sum of squares, $WSS(k)$ is within-group sum of squares, k is the number of clusters, and n is the number of data points.

I set the maximum number of clusters to 20. Before each run of the algorithm, the Calinski-Harabasz index was computed for all 20 possible clusterings, and that one with the highest value of $CH(k)$ was selected. During the evolution, the number of clusters was allowed to move only in a neighborhood of its last value (local search in the neighborhood of size 3 was performed).

Usage of the Calinski-Harabasz index has one drawback which can be very serious in some situations: the value of CH index is not defined for $k = 1$; this procedure can't tell us if we should use 1 or more clusters. This means that in this algorithm we will have always at least two clusters.

3.2.3 Test Suite

I used different test functions than in previous subsection. The first of them is the *Michalewicz* function. It is defined as follows:

$$F_{Michalewicz} = -\sin(x)\sin^{20}(\frac{x^2}{\pi}) - \sin(y)\sin^{20}(\frac{2y^2}{\pi}) \tag{8}$$

This base function was 10 times added, so that the number of dimensions was 20 ($\langle 0; \pi \rangle^D$, $D = 20$). For the histogram, the value of 60 was used for the number of bins. The Michalewicz function should be approximately of the same difficulty, as the Two Peaks function described in 3.1.8.

The second function is called *Shekel's Fox Holes*.

$$F_{FoxHoles} = \frac{1}{0.002 + \sum_{i=1}^{25} \frac{1}{i + \sum_{j=1}^{2} (x_j - a_{ji})^6}} \tag{9}$$

where

$$a = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & \dots & 16 & 32 & -32 & -16 & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \dots & 16 & 16 & 32 & 32 & 32 & 32 & 32 \end{pmatrix} \tag{10}$$

Also in this case the base was 10 times added to form a 20 dimensional function ($\langle -65,536; 65,536\rangle^D$, $D = 20$). This function is pretty hard to solve by ordinary EA, and EDAs have great difficulties to solve it too.

The last function, *Blobs*, is a bit more complex. It is defined in D-dimensional inetrval, ($\langle 0; 20\rangle^D$, $D = 20$). In this hypercube, $D+1$ local minima exist, let's denote them $\{m_i\}_{i=0}^{D}$. Let's further denote the global optimum as m_0, and set it to $(5, 5, \ldots, 5)$. The other D local minima can be defined as follows: all coordinates of m_i are set to 15, except the i-th coordinate that is set to 5. Then, the *Blobs* function is defined as follows:

$$F_{Blobs}(x) = D \min(dist^2(x, m_0), \min_{y \in \{m_i\}_{i=1}^{D}} (dist^2(x, y)) + 0.5) \tag{11}$$

where $dist(x, y)$ is euclidean distance between x and y. The Blobs function creates the fitness landscape in such a way, that the global optimum m_0 is in one corner of the hypercube, and the remaning D local minima are located around the opposite corner of hypercube. Around each of local minima, a quadratic bowl is placed.

The fitness landscapes of 2D versions of the above defined functions are showed in figure 5

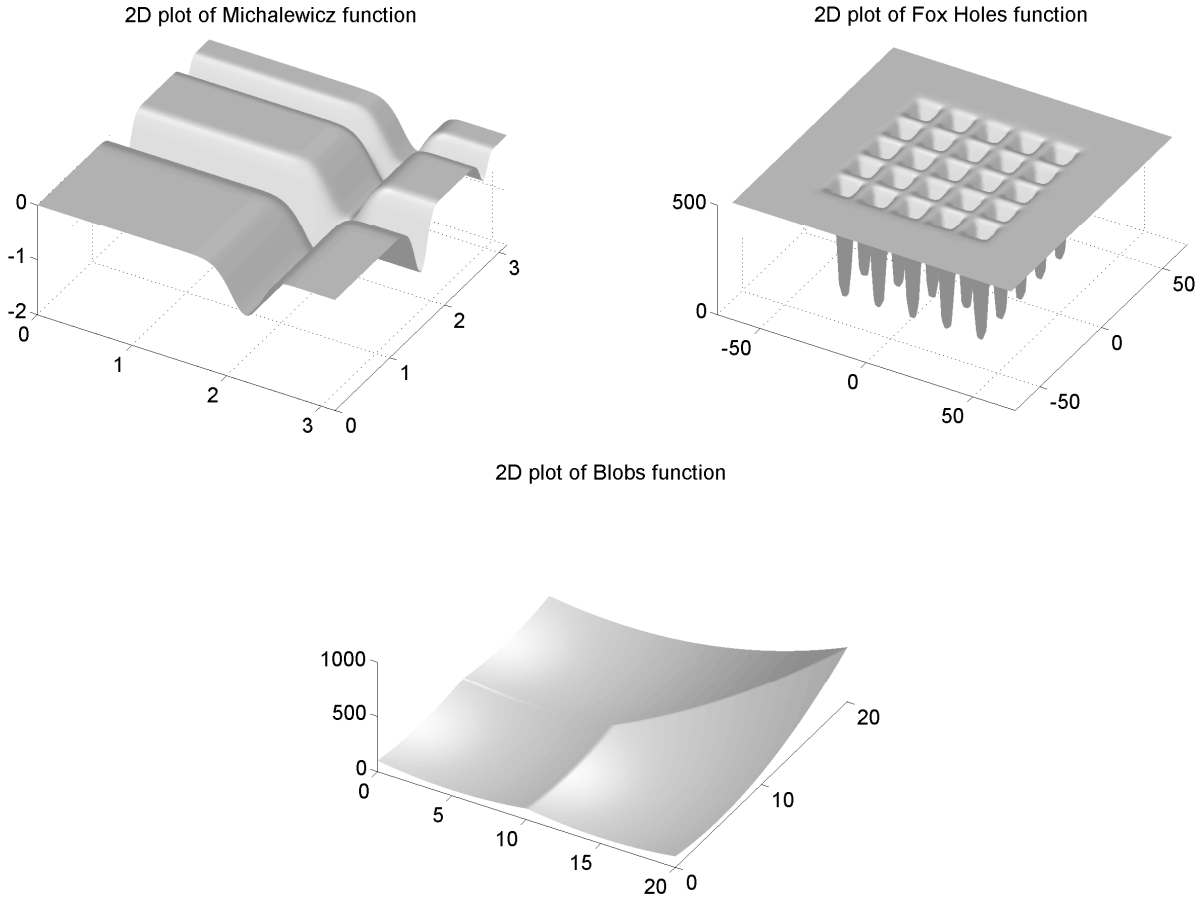

Figure 5: 2D fitness landscapes of the Michalewicz, Fox Holes, and Blobs functions.

3.2.4 Experimental Setup

For all three problems, Michalewicz, Fox Holes, and Blobs, I compared two types of EDA. The first one is the UMDA with HEH model (marked as HEH), the latter one is the Clustering EDA described in 3.2.1 (marked as CLH). Furthermore, the population size was varried in the experiments — the algorithms were tested with populations of sizes 200, and 800 data points.

The required precission of the algorithm was set to 0.05 for the Michalewicz function, 1 for the Fox Holes function, and 0.1 for the Blobs function (the 'required precission' is the size of the global optimum neighborhood).

The same statistics were monitored: number of runs in which algorithm found the global optimum of the problems (NoFoundOpts), number of evaluations needed to find the optimum averaged across the runs in which the optimum was really found (AveNoEvals), average best fitness after 50,000 evaluations (AveBest).

3.2.5 Results and Discussion

The results of all experiments are presented in figure 6.

Function	Algorithm	Population Size		
		200	800	Statistics
Michalewicz	HEH	20 5220 -18,01270	20 17880 -18,01260	NoFoundOpts AveNoEvals AveBest
	CLH	20 4810 -18,01300	20 18280 -18,01300	NoFoundOpts AveNoEvals AveBest
Fox Holes	HEH	0 15,99500	0 22,09430	NoFoundOpts AveNoEvals AveBest
	CLH	0 29,38050	0 10,86000	NoFoundOpts AveNoEvals AveBest
Blobs	HEH	0 10,00470	0 15,79300	NoFoundOpts AveNoEvals AveBest
	CLH	17 9553 0,55990	20 37600 0,00180	NoFoundOpts AveNoEvals AveBest

Figure 6: The effect of clustering on optimization of 3 test functions

Let's first discuss the results for the Michalewicz function. The function is very easy, it is separable and probably can be solved by other algorithms more efficiently (e.g. the line search algorithm). Both algorithms found the global optimum in all 20 runs, independently of the population size. However, the CLH outperformed the HEH for small population size (200). It could be due to the fact that in the CLH, the population is devided into two or more clusters, i.e. there are two or more small subpopulations. The one of them located in the area of global optimum converges more quickly, than a global population in HEH. Furthermore, individual clusters can aim at different parts of the search space and solve the problem in this areas more precisely. On the other hand, if the large population (800) is used, the clustering slows down the information interchange between clusters, so that the population needs longer time to converge.

The Fox Holes function turned out to be very difficult for both algorithms, HEH and CLH. None of these algorithms was able to find the global optimum in any of the 20 runs. In terms of fitness value of the best individual found in 50,000 evaluations, the HEH has better performance than CLH, where smaller clusters converge faster to worse results; the HEH has more time to evolve before convergence occurs, because it evolves larger population. The results for large population size (800) are a bit misleading; it seems that for this population size the CLH algorithm outperforms the HEH. However, the truth is a bit different: the CLH only converges more quickly (same, as for the small population size) so that it was able to find better solution in 50,000 evaluations than the HEH. The HEH, however, has not fully converged yet, and it is very likely that it would find better solution than CLH if it was allowed to evolve longer.

The third evaluation function, Blobs, was specificaly designed to show, that there are such functions which could be solved much more reliably by an algorithm that takes advantage of clustering. The HEH algorithm was not able to solve this function at all. The function has some characteristics resembling the deceptive functions for GAs; the population in HEH algorithm converges very often to the deceptive attractor, while the CLH is able to preserve a cluster in the area of global optimum. As the result, the CLH algorithm clearly outperformes the HEH algorithm (for this test function).

4 Future Work

This section presents a description of ideas I would like to 'make real' in my future work. My plans could be divided into two groups: basic research, and applications.

4.1 Basic Research

In next paragraphs, I shortly describe some techniques which have potential to improve the behavior of estimation of distribution algorithms. However, to all of them some additional issues are related (computational overhead, need of larger data sets, etc.). It must be explored not only if they are able to make the EDAs more efficient, but also if it all is worthy.

4.1.1 Competing Models

The results presented in this work indicate many things that deserve further exploration. Conceptually very simple UMDA can be used for easy problems, however, it can give good results even for problems which are not considered to be easy. On the other hand, more complex algorithms can outperform the easy ones on more difficult problems, but they may not be able to solve easy problems at all (or with large amount of computational overhead). Different algorithms are suitable for different problems. During evolution, the EDA itself should be able to dynamicly choose proper probabilistic model which will give best results. This can be done via a set of models which would compete against each other.

4.1.2 Transformations of Coordinates

As already mentioned in 1.2.3, one of the problems that prevents the EAs from being more efficient is the gene linkage. However, there exist methods that can reduce the amount of statistical dependency in the population. The principal component analysis (PCA) [32] is linear transformation which will de-correlate the data so that if they follow the normal distribution, they are independent. It offers similar possibilities of encoding the multivariate normal distribution with full covariance matrix as the gaussian network does, but in a different way. The independent component analysis [24] is another type of linear transformation which will make the data as independent of each other as possible. Using these transformations can be of great help in the evolution. This is probably a new, original idea, because to the best of author's knowledge, nobody has investigated this possibilities in the field of EAs yet.

4.1.3 Non-linear Transformations of Coordinates

A direct extension of the concept described in previous paragraph is the use of non-linear techniques for coordinates transformation. The precision of the model describing the data should be much higher than the precision of linear models. Non-linear tranformations can be classified into two groups:

- *Globally non-linear.* All transformations which work with the search space as a whole belong to this group. As an example, I can present the kernel PCA [32] which uses the same trick that is used in the support vector machines to make the transformation non-linear. The principal curves and surfaces [9] are another example of non-linearization.
- *Locally linear.* The mixture models [32] form this class of non-linear models. Each local component is linear, but when mixed together, they present non-linear model.

4.1.4 Clustering

If we look back to clustering EDA used in this work, we can say that the clustering method used here was not very powerfull — k-means algorithm with euclidean metric forms clusters which have spherical contours in the search space. The precission of such a clustering is very limited. Some clusters are elongated, some of them are rotated with respect to the search space axes. If we used different metric (e.g. Mahalanobis), we could get much better clusterings.

There are also possibilities to perform so called soft-clustering. The individual data points are not asigned to certain clusters (as they are in hard-clustering), rather a probability of belonging to each cluster is computed for each data point. However, this soft-clustering is only another name for the mixture models. There are very natural methods, how to perform a soft-clustering with e.g. Mahalanobis distance (fitting a mixture of principal component analysers).

4.2 Application Areas

Experiments with developed algorithms on real-world problems present one of the goals of my PhD thesis. Optimization of real function in real domain is very general task which can be applied in many practical situations. Except of the actual minimization of some (economic) lost function, we can identify many general application areas. For the artificial intelligence, the following two cases belong to the most important ones:

- *General multivariate regression* or *model fitting* is an example of the most straightforward possible applications of EDAs. Given a set of multidimensional data points, and a kind of parametrized model, we can use EDA to evolve the parameters of the (possibly non-linear) model at hand so that the model minimizes some error criterion measured over the data set (e.g. the mean squared error). To solve this problem, one can use conventional iterative methods, however, these methods generally fall within the class of local search or steepest descent, and thus are highly dependent on the initialization. Evolutionary algorithms can overcome this problem.

- *Training of classifiers and predictors.* Many classifiers and predictors are controlled by a set of real parameters. They can be trained by an evolutionary algorithm. Furthermore, the evolutionary algorithms have the ability to evolve not only the parameters, but also the structure of a classifier or predictor. In literature, we can find methods for training the neural networks by means of evolutionary algorithms, or methods for evolving the weights of individual dimensions for computing distances in k-nn classifier [30].

These algorithms can be used in several projects which has been running at our department. As examples, I can present the following:

- *Optimal cutting.* This problem is usually presented in 1-dimensional or 2-dimensional form. For the 1D optimal cutting, we have several bars of equal length, and we should cut these bars to pieces of given lengths so that we waste as little material as possible. The 2D optimal cutting problem is similar. We should cut given pieces (of various shapes) from rectangular sheets with as little garbage as possible. This task has direct practical impact — the resulting algorithm can be used directly in many industrial companies.

- *Signal processing.* In the analysis of eye movements, a problem of optimal signal processing emerges. It was already successfully solved by genetic algorithms, and I expect the EDA to have even better performance then the GA.

- *Hidden Markov models* are used e.g. in speech recognition, or in time signals modeling. Their simpler form, hidden Markov chains, are used to recognize separate words very often, and for their training, the EAs have been already applied with very promising results. However, these EAs used domain specific knowledge when performing the variation operations. Incorporating this domain specific knowledge into EDAs would be very interesting too.

- *Hough transformation* is a simple technique used in image processing for segmentation. It is usually applied on black-and-white images and the goal is to identify and find some geometric primitives (lines, circles) in the picture. The actual Hough transformation uses exhaustive search in the space of possible primitives. This task can be substituted by an EDA approach.

- *Inverse kinematic problem* is solved in robotics and can be described as follows. We have a manipulator arm which consists of several links, and we want to set them in such a way that the manipulator 'hand' would have a defined position and orientation. This task is not always solvable analytically, and it need not have solution at all.

I am sure that further possible applications will emerge in the future. In my PhD thesis, I would like to present the results of performance tests of EDAs on several of the practical problems mentioned above.

Acknowledgements

The project was supported by the Grant agency of the Czech Republic with the grant No. GACR 102/02/0132 entitled "Use of Genetic Principles in Evolutionary Algorithms", and was supervised by Doc. Ing. Jiří Lažanský, CSc.

References

[1] Bäck, T. *Self-Adaptation in Genetic Algorithms.* In Varela, F.J., Bourgine, P. (Eds.), Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life, pages 263-271, MIT Press, 1992.

[2] Baluja, S. *Population Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning.* Technical Report No. CMU-CS-94-163. Pittsburgh, PA: Carnegie Mellon University, 1994.

[3] Baluja, S., Davies, S. *Using Optimal Dependancy-Trees for Combinatorial Optimization: Learning the Structure of the Search Space.* In Proc. of the International Conference on Machine Learning, 30-38.

[4] Bosman, P.A.N., Thierens, D. *An Algorithmic Framework for Density Estimation Based Evolutionary Algorithms.* Technical Report UU-CS-1999-46, Utrecht University, 1999.

[5] Bosman, P.A., Thierens, D. *Continuous Iterated Density Estimation Evolutionary Algorithms Within the IDEA Framework.* Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), pages 197-200, 2000.

[6] Calinski, R.B., Harabasz, J. *A Dendrite Method for Cluster Analysis.* Communications in Statistics, 3:1-27, 1974.

[7] Davis, L. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[8] De Bonet, J.S., Isbell, C.L., Viola, P. *MIMIC: Finding Optima by Estimating Probability Densities.* Advances in Neural Information Processing Systems (NIPS-97), 9:424-431, 1997.

[9] Chang, K.-Y., Ghosh, J. *A Unified Model for Probabilistic Principal Surfaces.* IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 23, No. 1, 2001.

[10] Eiben, A.E., Hinterding, R., Michalewicz, Z. *Parameter Control in Evolutionary Algorithms.* IEEE Trans. on Evolutionary Computation, Vol. 3, 2:124–141, 1999.

[11] Etxeberria, R., Larrañaga, P. *Global Optimization Using Bayesian Networks.* In Rodriguez, A.A.O., Ortiz, M.R.S., Hermida, R.S. (Eds.), Second Symposium on Artificial Intelligence (CIMAF-99), pages 332-339, Habana, Cuba, 1999.

[12] Fogarty, T. *Varying the Probability of Mutation in Genetic Algorithms.* In Schaffer J.D. (Ed.), Proc. of the 3rd International Conference on Genetic Algorithms, pages 104-109, Morgan Kaufmann, 1989.

[13] Fogel, L.J., Angeline, P.J., Fogel, D.B. *An Evolutionary Programming Approach to Self-Adaptation on Finite State Machines.* In McDonnel, J.R., Reynolds, R.G, Fogel, D.B. (Eds.), Proc. of the 4th Annual Conference on Evolutionary Programming, pages 355-365, MIT Press, 1995.

[14] Gallagher, M.R., Frean, M., Downs, T. *Real-Valued Evolutionary Optimization Using a Flexible Probability Density Estimator.* In Proceedings of Genetic and Evolutionary Computation Conference, pages 840-864. Morgan Kaufmann, 1999.

[15] Goldberg, D.E. *Genetic algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[16] Goldberg, D.E., Deb, K., Clark, J.H. *Genetic Algorithms, Noise and the Sizing of Populations.* Complex Systems, 6:333-362, 1992.

[17] Goldberg, D.E., Deb, K., Korb, B. *Don't Worry, Be Messy.* In Belew, R.K., Booker, L.B. (Eds.), Proc. of the 4th International Conference on Genetic Algorithms, pages 24-30, Morgan Kaufmann, 1991.

[18] Goldberg, D.E, Deb, K., Thierens, D. *Toward a Better Understanding of Mixing in Genetic Algorithms.* In Belew, R.K., Booker, L.B. (Eds.), Proc. of the 4th International Conference on Genetic Algorithms, pages 190-195, Morgan Kaufmann, 1991.

[19] Goldberg, D.E., Smith, R.E. *Nonstationary Function Optimization Using Genetic Algorithms with Dominance and Diploidy.* In Grefenstette, J.J. (Ed.), Proc. of the 2nd International Conference on Genetic Algorithms and Their Applications, pages 59-68, Lawrence Erlbaum Associates, 1987.

[20] Grefenstette, J.J. *Optimization of Control Parameters for Genetic Algorithms.* IEEE Trans. on System, Man, and Cybernetics, 16(1):122-128, 1986.

[21] Harik, G., Cantu-Paz, E., Goldberg, D.E., Miller, B.L. *The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations.* In Proc. of the 4th IEEE Conference on Evolutionary Computation, pages 7-12, 1997.

[22] Harik, G.R., Lobo, F.G., Goldberg, D.E. *The Compact Genetic Algorithm.* Technical Report (IlliGAL Report No. 97006). University of Illinois at Urbana-Champaign, 1997.

[23] Harik, G.R. *Linkage Learning via Probabilistic Modeling in the ECGA.* Technical Report (IlliGAL Report No. 99010). University of Illinois at Urbana-Champaign, 1999.

[24] Hyvärinen, A. *Survey on Independent Component Analysis.* Neural Computing Surveys, 2:94-128, 1999.

[25] Janikow, C., Michalewicz, Z. *An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms.* In Belew, R.K., Booker, L.B. (eds.), Proc. of the 4th International Conference on Genetic Algorithms, pages 151-157, Morgan Kaufmann, 1991.

[26] Joines, J.A., Houck, C.R. *On the Use of Non-stationary Penalty Functions To Solve Nonlinear Constrained Optimization Problems with GAs.* In Proc. of the 1st IEEE Conference on Evolutionary Computation, pages 579-584, IEEE Press, 1994.

[27] Julstrom, B.A. *What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm.* In Eshelman, L. (Ed.), Proc. of the 6th International Conference on Genetic Algorithms, pages 81-87, Morgan Kaufmann, 1995.

[28] Kantschik, W., Brameier, P.D.M., Banzhaf, W. *Empirical Analysis of Different Levels of Meta-Evolution.* Technical Report, Department of Computer Science, Dortmund University, 1999.

[29] Larranaga, P., Etxeberria, R., Lozano, J.A., Sierra, B., Inza, I., Pena, J.M. *A Review of the Cooperation Between Evolutionary Computation and Probabilistic Graphical Models.* In Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99, pages 314-324. La Habana, 1999.

[30] Larranaga, P., Lozano, J.A., Bengoetxea, E. *Estimation of Distribution Algorithms Based on Multivariate Normal Distributions and Gaussian Networks.* Technical Report KZZA-IK-1-01, Dept. of Computer Science and Artificial Intelligence, University of Basque Country, 2001.

[31] Lis, J., Lis, M. *Self-Adapting Parallel Genetic Algorithm with Dynamic Mutation Probability, Crossover Rate and Population Size.* In Arabas, J. (Ed.), Proceedings of the 1st Polish National Conference on Evolutionary Computation, pages 324-329, Oficina Wydawnica Politechniki Warszawskiej, 1996.

[32] Moerland, P. *Mixture Models for Unsupervised and Supervised Learning.* PhD Thesis, Computer Science Department, Swiss Federal Institute of Technology, Lausanne, 2000.

[33] Mühlenbein, H., Paass, G. *From Recombination of Genes to the Estimation of Distributions I. Binary Parameters.* Parallel Problem Solving from Nature, 178-187.

[34] Mühlenbein, H., Mahnig, T., Rodriguez, A.O. *Schemata, Distributions, and Graphical Models in Evolutionary Optimization.* Journal of Heuristics, 5:215-247, 1999.

[35] Paredis, J. *Coevolutionary Computation.* Artificial Life, 2(4):355-375, 1995.

[36] Pelikan, M., Goldberg, D.E., Cantú-Paz, E. *Linkage Problem, Distribution Estimation, and Bayesian Networks.* Technical Report (IlliGAL Report No. 98013). University of Illinois at Urbana-Champaign, 1998.

[37] Pelikan, M., Mühlenbein, H. *The bivariate marginal distribution algorithm.* Advances in Soft Computing – Engineering Design and Manufacturing, pages 521-535, 1999.

[38] Poosala, V., Ioannidis, Y.E., Haas, P. J., Shekita, E.J. *Improved histograms for selectivity estimation of range predicates.* In Proc. 1996 ACM SIGMOD Intl. Conf. Managment of Data, pages 294–305. ACM Press, 1996.

[39] Rudlof, S., Koeppen, M. *Stochastic Hill Climbing with Learning by Vectors of Normal Distributions.* In Proceedings of the First Online Workshop on Soft Computing (WSC1). Nagoya, Japan, 1996.

[40] Schaffer, J.D., Morishima, A. *An Adaptive crossover Distribution Mechanism for Genetic Algorithms.* In Grefenstette, J.J. (ed.), Proc. of the 2nd International Conference on Genetic Algorithms and Their Applications, pages 36-40, Lawrence Erlbaum Associates, 1987.

[41] Schraudolph, N., Belew, R. *Dynamic Parameter Encoding for Genetic Algorithms.* Machine Learning, vol. 9(1):9-21, 1992.

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

[43] Sebag, M., Ducoulombier, A. *Extending Population Based Incremental Learning to Continuous Search Spaces.* In Parallel Problem Solving From Nature - PPSN V, pages 418-427. Springer Verlag, Berlin, 1998.

[44] Servet, I., Trave-Massuyes, L., Stern, D. *Telephone Network Traffic Overloading Diagnosis and Evolutionary Techniques.* In Proceedings of the Third European Conference on Artificial Evolution, (AE 97), pages 137-144, 1997.

[45] Shaefer, C.G. *The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique.* In Grefenstette, J.J. (Ed.), Proc. of the 2nd International Conference on Genetic Algorithms and Their Applications, pages 50-55, Lawrence Erlbaum Associates, 1987.

[46] Spears, W.M. *Adapting Crossover in Evolutionary Algorithms.* In McDonnel, J.R., Reynolds, R.G, Fogel, D.B. (Eds.), Proc. of the 4th Annual Conference on Evolutionary Programming, pages 367-384, MIT Press, 1995.

[47] Thierens, D., Goldberg, D.E. *Mixing in Genetic Algorithms.* In Belew, R.K., Booker, L.B. (eds.), Proc. of the 4th International Conference on Genetic Algorithms, pages 31-37, Morgan Kaufmann, 1991.

[48] Tsutsui, S., Pelikan, M., Goldberg, D.E. *Evolutionary Algorithm Using Marginal Histogram Models in Continuous Domain.* Technical Report (IlliGAL Report No. 2001019). University of Illinois at Urbana-Champaign. March, 2001.

[49] Whitley, D., Mathias, K., Fitzhorn, P. *Delta Coding: An Iterative Strategy for Genetic Algorithms.* In Belew, R.K., Booker, L.B. (Eds.), Proc. of the 4th International Conference on Genetic Algorithms, pages 77-84, Morgan Kaufmann, 1991.

[50] Whitley, D., Mathias, K., Rana, S., Dzubera, J. *Evaluating Evolutionary Algorithms.* Artificial Intelligence Volume 85, pp. 245-2761, 1996.