The Bivariate Marginal Distribution Algorithm

Martin Pelikan[1], **Heinz Muehlenbein**[2]

[1]*Dept. of Mathematics, Slovak Technical University,*
81237 Bratislava, Slovakia, email: pelikan@svf.stuba.sk
[2]*GMD Forschungszentrum Informationstechnik,*
D-53754 Sankt Augustin, Germany, email: muehlenbein@gmd.de

Keywords: evolutionary algorithm, marginal distribution, dependency graph, decomposable problems

Abstract

The paper deals with the Bivariate Marginal Distribution Algorithm (BMDA). BMDA is an extension of the Univariate Marginal Distribution Algorithm (UMDA). It uses the pair gene dependencies in order to improve algorithms that use simple univariate marginal distributions. BMDA is a special case of the Factorization Distribution Algorithm, but without any problem specific knowledge in the initial stage. The dependencies are being discovered during the optimization process itself. In this paper BMDA is described in detail. BMDA is compared to different algorithms including the simple genetic algorithm with different crossover methods and UMDA. For some fitness functions the relation between problem size and the number of fitness evaluations until convergence is shown.

1. Introduction

Genetic algorithms work with populations of strings of fixed length. In this paper binary strings will be considered. From current population better strings are selected at the expense of worse ones. New strings are generated using the recombination/crossover operator and mutation. The recombination operator combines the information contained in two strings. Mutation performs a small perturbation to these strings in order to keep the population diverse and to introduce new information. The theory behind these operators is based on the schema theorem [1]. It is known that the operators often cause the disruption of schemata of large defining length. This may cause bad performance of genetic algorithms on problems where these schemata are needed to obtain the optimum. This has lead to new approaches for doing recombination. The first line of research uses different reordering methods and other methods that decrease the defining length of important schemata. In these methods, the values of bits on different positions are not the only thing that is optimized as

it was with classical genetic algorithm. The order of bits and other features are optimized as well. The methods work quite well for decomposable problems although they require some prior knowledge about the problem and they are usually very memory and time consuming. This direction has lead to the GEMGA (Gene Expression Messy Genetic Algorithm) [2].

The second line of research is based on the estimation of probability distribution. The simplest way is to estimate the distribution using univariate marginal frequencies in the set of selected parents. This is what UMDA (Univariate Marginal Distribution Algorithm) [3] is based on. This algorithm works perfect for linear problems as is shown in [3]. But it also works very well for problems that don't contain significant dependencies. Performance of UMDA can be estimated with the use of the variance decomposition [3]. In general, the greater the rate of additive variance to the sum of higher order variances, the better UMDA performs.

For problems with dependencies among different genes this approach is not sufficient. The theory of UMDA has been extended to problems where the probability model is known. In FDA (Factorization Distribution Algorithm) it is assumed that the probability can be written as some product of marginal frequencies [4]. This is an ideal schema algorithm, because no important schemata can be disrupted. A more pragmatic way to extend UMDA is to use bivariate marginal distributions.

In the MIMIC algorithm [5] the distribution is assumed to be a simple chain-like product. All bits are ordered into a chain using a simple greedy algorithm first taking as input the univariate and bivariate marginal frequencies in the set of selected parents. For each new individual, the first bit is generated using its univariate marginal frequency. All other bits are generated using the conditional probability given the previous bit. Another approach, presented in [6], uses a tree structure for the probability model. The tree is constructed to maximize the sum of the so-called mutual information of genes that are connected. The value for the bit corresponding to the root of this tree is generated using its univariate frequency. The remaining bits are generated using conditional probability given the value of their parent in the dependency tree. Both univariate and bivariate frequencies are calculated incrementally. In our approach we combine these two methods. The mutual information dependency measure is replaced by Pearson's chi-square statistics, in order to be able to identify pairs that are independent with a certain probability. These pairs are mapped into a dependency graph. The algorithm will be precisely described in the next sections. Even though the algorithm and all terms are defined for chromosomes defined as binary strings of fixed length, these can be easily reformulated for any finite alphabet. The first position in a string is referred as the 0th position in this paper. This makes the equations simpler.

2. Marginal Frequencies and Pearson's statistics

Let us denote the length of chromosome as n. Let P be a set of binary strings of length n (the population). The size of P will be denoted as N. For each position $i \in \{0, \ldots, n-1\}$ and each possible value on this position $x_i \in \{0, 1\}$, we define the univariate marginal frequency $p_i(x_i)$ for set P as the frequency of strings that have x_i on ith position in set P. Similarly, for any two positions $i \neq j \in \{0, \ldots, n-1\}$ and any possible values on these positions $x_i, x_j \in \{0, 1\}$, we define the bivariate marginal frequency $p_{i,j}(x_i, x_j)$ for set P as the frequency of strings that have x_i and x_j on positions i and j, respectively. Sometimes the term of probability will be used instead of frequency. With the use of univariate and bivariate marginal frequencies, the conditional probability of appearance of the value x_i on ith position having given the value x_j on jth position can be calculated as

$$p_{i,j}(x_i|x_j) = \frac{p_{i,j}(x_i, x_j)}{p_j(x_j)} \tag{1}$$

Pearson's chi-square statistics [7] is defined by

$$X^2 = \sum \frac{(observed - expected)^2}{expected} \tag{2}$$

Here, for each pair of positions, the observed quantity is the number of possible pairs of values on these positions. If these two positions were independent, the number for each of these pairs of values could be easily calculated using the basic probability theory. This is the expected quantity. Then, in terms of univariate and bivariate frequencies and the total number of points taken into account, for positions $i \neq j$, we get [7]

$$X^2_{i,j} = \sum_{x_i, x_j} \frac{(Np_{i,j}(x_i, x_j) - Np_i(x_i)p_j(x_j))^2}{Np_i(x_i)p_j(x_j)} \tag{3}$$

If positions i and j are independent for 95%, then for Pearson's chi-square statistics following inequality holds [7]

$$X^2_{i,j} < 3.84 \tag{4}$$

3. The Construction of a Dependency Graph

In this section the construction of a dependency graph will be described. The graph will be defined by three sets, V, E, and R, i.e. $G = (V, E, R)$. V is the set of vertices, $E \subset V \times V$ is the set of edges and R is a set containing one vertex from each of the connected components of G. In a dependency graph each node corresponds to a position in a string. There is one to one correspondence between the vertices and positions in a string. Thus, we can use the set of vertices $V = \{0, \ldots, n-1\}$, where vertex i corresponds to the

ith position. As it will be clear from the construction of a dependency graph, the graph does not have be connected. That means that it does not have to form a tree. The dependency graph is always acyclic. It can be seen as the set of trees that are not mutually connected. The generation of new strings does not depend on the number of connected components of the dependency graph. When talking about frequencies in this section, we always mean the set which is used for creation of a dependency graph. In the description of the algorithm it is always said which one is meant.

Let us denote A the set of vertices that have not been processed yet. At the start of the algorithm A is set to V. Then, successively, as edges are being added into E, A gets smaller. The algorithm ends up with A equal to an empty set what means that all vertices have been processed. Another set is denoted D. It is the set of all pairs from $V \times V$ that are not independent for 95% (see Equations 3 and 4), i.e.

$$D = \{(i,j) | i \neq j \in \{0,\ldots,n-1\} \wedge X^2_{i,j} \geq 3.84\} \tag{5}$$

Then, the algorithm for the construction of a dependency tree is defined as follows

Algorithm for the Construction of a Dependency Graph

1. set $V \leftarrow \{0,\ldots,n-1\}$
   set $A \leftarrow V$
   set $E \leftarrow \emptyset$
2. $v \leftarrow$ any vertex from A
   add v into R
3. remove v from A
4. if there are no more vertices in set A, finish
5. if in D there are no more dependencies of any v and v' where $v \in A$ and $v' \in V \setminus A$, go to 2
6. set v to the vertex from A that maximizes $X^2_{v,v'}$ over all $v \in A$ and $v' \in V \setminus A$
7. add edge (v, v') into the set of edges E
8. go to 3

The basic idea of the algorithm is very simple. It is similar to the well-known algorithm for obtaining the largest spanning tree. First, an arbitrary vertex is added to the graph. Then, the vertex with the greatest dependency with some of previously added vertices and the edge corresponding to this dependency are added to the graph. The last step is repeated until there is no dependency between already added vertices and the rest. If this is the case, an arbitrary vertex is added into graph and the process repeats. The whole process repeats until all vertices are added into the graph. The effect is that an acyclic graph with a maximal sum of chi-square statistics values over the connected vertices is constructed. Resulting graph does not have to be connected, as it was already mentioned above. As the first vertex from each component (that is successively

created by adding vertices according to the dependencies) is added into the graph, it is added also into the set R (i.e., to the set of special vertices, one for each component of the resulting graph).

Since the set A is initialized to a finite set of vertices and in each cycle at least one vertex is removed from it, the algorithm does always finish and therefore it is well-defined. The time complexity of the described algorithm is $O(n^3)$.

4. Generation of New Individuals

To generate new individuals, a previously described dependency graph $G = (V, E, R)$ is used. For each individual the values for positions contained in R are generated by the univariate marginal frequencies. Then, if there exist a position v that is yet not generated and it is connected to some already generated position v' (according to the set of edges E), it is generated using the conditional probability (see Equation 1) for a position v having given the value on a position v'. The last step is repeated until values for all positions are generated.

In the following description of the algorithm for the generation of a new individual, one important set, among sets defining graph G, appears. It is denoted as K and it stands for the set of all positions that have been already generated. The individual is a string of length n and will be denoted by x. Its ith bit will be denoted as x_i.

Algorithm for the Generation of a New Individual

1. set $K \leftarrow V$
2. generate x_r for all $r \in R$ using univariate frequencies, i.e. set it to the value a with probability $p_r(a)$
   set $K \leftarrow K \setminus R$
3. if K is already empty, finish
4. choose k from K such that there exist k' from $V \setminus K$ connected to k in the graph G
5. generate x_k using conditional probability having given value for $x_{k'}$, i.e. set it to value a with probability $p_{k,k'}(a|x_{k'})$
6. remove k from the set K
7. go to 4

The set K is initialized as a finite set and in each cycle one vertex is removed from it. For each connected component, at least one vertex is generated first. The algorithm is therefore well-defined. The generation of one individual can be done in $O(n)$ steps. The generation of different individuals is independent. The algorithm for generation of new individuals is therefore well suited for parallelization.

5. Bivariate Marginal Distribution Algorithm

Having defined the algorithms for the construction of a dependency graph and the generation of new individuals, the bivariate marginal distribution algorithm (BMDA) can be described. In BMDA, the population is randomly generated first. From this population, the better individuals are selected. Univariate and bivariate marginal frequencies for these individuals are then calculated. Using these frequencies, a dependency graph is constructed as it is described in Section 3. Having given the dependency graph, new individuals are generated as described in Section 4. New individuals are then added into the old population from which the individuals were originally selected. They replace some of the old ones, usually the worst of them, so that the number of individuals in the population remains constant. From the new population, individuals are selected again. The process, starting off with the selection of better individuals and ending with adding the new individuals into the old population, repeats until the termination criteria are met. The termination criteria can cause the algorithm to stop if it has already found the optimum or the diversity of population is too low. The value of the optimum is usually unknown by the breeder. That is why the second condition is the more important one. When the diversity is too low almost all individuals in the population are the same. That means that there is not enough information in the population to create new individuals that would fit the problem better than already found ones.

Bivariate Marginal Distribution Algorithm

1. set $t \leftarrow 0$
   randomly generate initial population $P(0)$
2. select parents $S(t)$ from $P(t)$
   calculate univariate frequencies p_i and bivariate frequencies $p_{i,j}$ for the selected set $S(t)$
3. create dependency graph $G = (V, E, R)$ using the frequencies p_i and $p_{i,j}$
4. generate the set of new individuals $O(t)$ using dependency graph G and frequencies p_i and $p_{i,j}$
5. replace some of individuals from $P(t)$ with new individuals $O(t)$
   set $t \leftarrow t + 1$
6. if termination criteria are not met, go to 2

The termination criterion due to the lack of diversity is defined as follows: if all univariate frequencies are closer than $\epsilon > 0$ to 0 or 1, the algorithm is terminated. If this is the case, we say the algorithm ϵ-converged. In most of our experiments, we use this termination criterion.

The probability model used by BMDA is given by

$$p(x) = \prod_{r \in R} p_r(x_r) \prod_{i \in V \setminus R} p_{i,e(i)}(x_i | x_{e(i)}) \tag{6}$$

where $e(i)$ returns the vertex connected to the vertex i but added sooner than this vertex using the algorithm for the construction of a dependency graph from Section 3. To say it in another way, $e(i)$ is the vertex that is the next one on the way from ith vertex to the $r \in R$ that corresponds to the component where the vertex i is located. This factorization is a special case of the factorization considered in [4].

6. How Does It Work and Why?

The distribution of BMDA is based on the use of conditional probabilities for pairs of positions that seem to be dependent. This information is estimated from the current population of strings. Any binary function can be decomposed as follows [4]

$$f(x) = a + \sum_{i_1} a_{i_1} x_{i_1} + \sum_{i_1 < i_2} a_{i_1,i_2} x_{i_1} x_{i_2} + \ldots + a_{0,1,\ldots,n-1} x_0 x_1 \ldots x_{n-1} \quad (7)$$

A function is decomposable of order k if all coefficients of higher order are 0. Let us talk about decomposable functions of order at most two, first. For this class of functions, the best dependency graph can be constructed fairly easily. It can be done by connecting the vertices i and j just when the coefficient $a_{i,j}$ is not equal to zero. A graph constructed this way does not have to be acyclic. If this graph is acyclic or this can be achieved by deletion of only insignificant dependencies, then it serves as the best dependency graph that can be used for the generation of new individuals. If BMDA found this dependency graph and used it for generation of new individuals, it would perform very well. It would perform as well as UMDA does with linear functions. The performance of BMDA therefore relies on whether it is able to detect these dependencies having no problem-specific knowledge in an initial stage. In the empirical part of this paper it will be shown that most of the dependencies are usually found after only a few generations. If the mentioned graph is not acyclic and this property cannot be satisfied by deletion of significantly unimportant dependencies from it, the problem becomes impossible to solve within the use of this model. This problems could be possibly solved by the use of terms of conditional independence. Not only conditional probabilities for a position having given the value for another one but for a position having given values for a set of some of other positions as well would be taken into account. The problem with this approach is, as it was already mentioned above, that BMDA does not get any problem-specific knowledge about the problem that is solved. During optimization, it is learning the structure of the problem itself. From the beginning, the information about dependencies is very unclear and almost no pair of positions seems to be independent. The approximation gradually gets more and more accurate. There is no effective method for the prediction of the more complex model. Moreover, the fitness function does not have to be decomposable. A model that is simpler than the correct model may serve well

in many cases. This can be demonstrated by the good performance of UMDA for many nonlinear problems. The right choice is somewhere between the accurate model and a simple model as univariate marginal distributions. The character of this class of algorithms does not allow very complex predictions of the model due to the amount of available information about the problem.

The problem arises when the optimized function is additively decomposable of order three or more. If this is the case, the given fitness distribution cannot be covered by the BMDA model. The best BMDA can do is to use the model that is as close as possible to the original distribution. The discovery of higher order dependencies is significantly more time and space complex and that is why it is hard to say if it was worthy to follow the way of enlarging the blocks taken into account. Moreover, there is again lack of information about the problem to predict the more complex models. If all the dependencies were known from the beginning, the best way would be to use them as in FDA [4]. Experiments show that BMDA performs well on problems of higher order dependencies as well. Higher order dependencies are usually substituted by chain-like dependency structures in our model.

7. Experiments

First experiments were done for a few different fitness functions. The choice of the fitness functions was done to make the behavior of the used algorithm more clear. Comparisons to some other methods were done. But the main purpose of this section is the explanation of the behavior of BMDA. The size of the paper does now allow us to take into account all other algorithms that could possibly compete with BMDA for solving these problems. For most of the problems the decomposition of the problem is shown too, to make things more clear.

In the experiments the so-called ordering parameter [8] is often used as a measure of convergence. The ordering parameter is defined as follows

$$\chi(p) = \frac{4}{n} \sum_{i=0}^{n-1} \left(p_i(1) - \frac{1}{2} \right)^2 \tag{8}$$

where p is the vector of univariate marginal frequencies $p_i(1)$. The closer the parameter is to one, the less diversity the population contains.

For some fitness functions a permutation of the variables will be allowed. The permutations will be denoted as π_k where $k \in \{1, \ldots, n\}$. The permutation π_k is well-defined for any k such that n can be divided by k. It is defined as follows

$$\pi_k(i) = \left\lfloor \frac{n(i \bmod k) + i}{k} \right\rfloor \tag{9}$$

π_1 is the identity. For π_2 and $n = 12$ we get π_2 =(0, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9, 11). Permutations will be used to change the order of the positions of a

string to show the behavior of different algorithms with respect to the used permutation.

For all experiments a fixed selection method (Truncation selection [3] with $\tau = 50\%$) was used. The worse half of the old population was replaced by the new individuals.

7.1 Onemax fitness function

This fitness function is actually a simple linear function over the single bits with all coefficients equal to 1. That means it is just the sum of all bits in a string, i.e.

$$f_{onemax}(x) = \sum_{i=0}^{n-1} x_i \tag{10}$$

where x_i is the value on the ith position in string x. Onemax fitness function does not have a permutation as an input parameter because its value is the same for any permutation of bits in an input string. This function is decomposable of order one, as we can see from its basic form already. That means that BMDA should give good results for this function. As it was already shown in [3], UMDA works very well for this class of problems. For UMDA a very small population size is sufficient for its convergence into the optimum. BMDA uses a more complex distribution and to make this correct the dependencies that are present in the problem have to be discovered well. That is why the population size has to be enlarged. For each algorithm, the set of parameters is chosen to make it converge to the optimum in 100% of totally 30 independent runs. For all algorithms the ϵ-convergence termination criterion was used with $\epsilon = 0.05$.

As it is shown in Figures 1 and 2, BMDA converges slower than UMDA and the simple genetic algorithm with uniform crossover. This is caused by the use of statistical methods to discover dependencies that require to enlarge the population. UMDA takes the bits as independent from the beginning, so it uses the information BMDA has to discover! If BMDA had this information, it would perform exactly the same as UMDA. GA with uniform crossover and a great crossover rate performs similar to UMDA [3], i.e. it performs very well for linear problems. Genetic algorithm with onepoint crossover keeps the dependencies among neighboring bits so it has the same disadvantage with linear problems as BMDA. In Figure 1, an average number of fitness evaluations until convergence for 30 independent runs is shown. In Figure 2 the evolution of ordering parameter in a randomly chosen run is shown.

7.2 Quadratic Function

The quadratic fitness function used for comparisons in this section is defined as

$$f_{quadratic}(x, \pi) = \sum_{i=0}^{\frac{n}{2}-1} f_2(x_{\pi(2i)}, x_{\pi(2i+1)}) \tag{11}$$

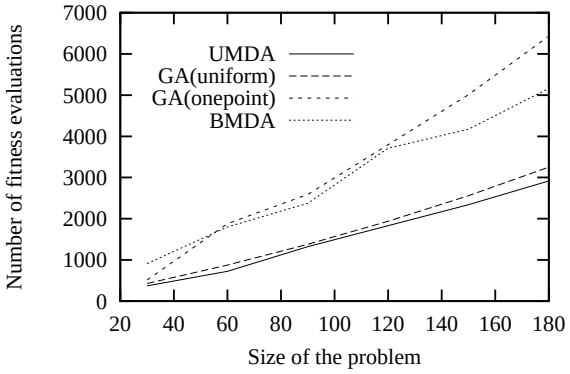

Figure 1. Number of fitness evaluations for f_{onemax}. The ranges of the used population size (for $n = 30$ to $n = 180$) were $50 - 170$ for UMDA, $32 - 100$ for GA (uniform), $32 - 160$ for GA (onepoint), and $120 - 260$ for BMDA. The truncation selection with $\tau = 50\%$ was used.

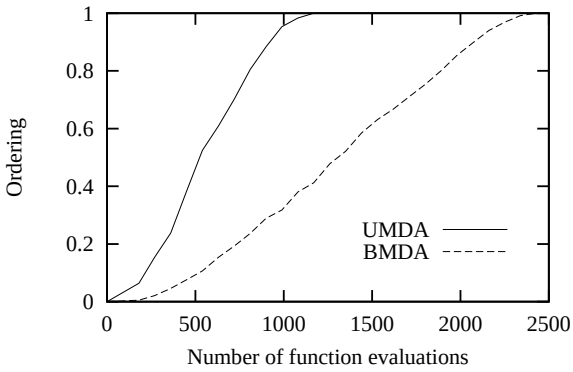

Figure 2. Ordering parameter for f_{onemax}, $n = 90$. The population sizes were 110 for UMDA and 180 for BMDA. The truncation selection with $\tau = 50\%$ was used.

where f_2 is defined as

$$f_2(u, v) = 0.9 - 0.9(u + v) + 1.9uv \tag{12}$$

With both arguments equal to 0 we get $f_2(0, 0) = 0.9$. With different arguments we get $f_2(0, 1) = f_2(1, 0) = 0$. With both arguments equal to 1 we get $f_2(1, 1) = 1$. The optimum is clearly in the string with 1's on all positions. Criteria for convergence as well as the requirement for 100% convergence in 30 independent runs and the choice of optimal parameters for all algorithms are the same as in the last section. UMDA was not used for comparisons since

it performed much worse than any of other methods. An average number of function evaluations until convergence over 30 runs is taken into account.

The function is not deceptive and that is why it should be not so hard to find the optimum for simple GA. It is an ideal function for the BMDA because it is decomposable of order 2 (see its definition) and the dependencies do not form cycles. First, the $f_{quadratic}$ with permutation π_1 (i.e., identity) will be discussed. Experiments have shown that although this function is not a big problem to solve by simple GA, BMDA performs much better. For GA with onepoint crossover the number of fitness evaluations seems to grow much faster than for BMDA with increasing size of a problem (see Figure 3). GA with uniform crossover performs significantly worse than GA with onepoint crossover (see Figure 4, notice that there is used a log-scaling for the number of fitness evaluations in this figure). The bad performance of GA is caused by the similar fitness values for both leading schemata that are opposite to each other and by the fact that their disruption significantly decreases the fitness. In both figures 4 an average number of fitness evaluations needed for convergence in 30 independent runs is shown.

Another permutation that was used for comparisons was π_2 that spreads pairs of positions $(2i)$ and $(2i+1)$ so that the distance between them is half of the size of a string. This reordering of bits has different effect on different algorithms. For BMDA (as well as it would be for UMDA), it actually does not affect anything because both algorithms are independent of positioning of bits. For simple GA with uniform crossover the situation is analogical. The problem arises for simple GA with onepoint crossover. In this case, this algorithm performs very poorly. Already for a problem of size $n = 30$, GA with onepoint crossover requires about 10 times more fitness evaluations than with π_1. This gap enlarges with the size of a problem. Results for BMDA and simple GA with uniform crossover are just slightly different from results with permutation π_1.

7.3 Deceptive function of order 3

Deceptive function is often used for comparisons of different optimization methods for its being deceptive. With deceptive problems, the average fitness of low order schemata present in optimum is lower than the average fitness of alternative ones. This property makes this class of functions hard to solve by the simple genetic algorithm as well as UMDA and many other evolutionary algorithms because these algorithms are based on the superior position of low order schemata that are matched by optimum. The fitness function is then defined as

$$f_{3deceptive}(x,\pi) = \sum_{i=0}^{\frac{n}{3}-1} f_3(x_{\pi(3i)} + x_{\pi(3i+1)} + x_{\pi(3i+2)}) \qquad (13)$$

where x is a bit string, π is any permutation of order n, and f_3 is defined as

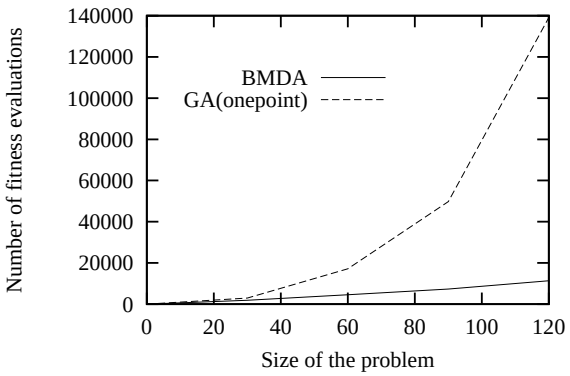

Figure 3. Number of fitness evaluations for $f_{quadratic}$. The ranges of the used population size (for $n = 30$ to $n = 120$) were $200 - 570$ for BMDA and $260 - 2500$ for GA (onepoint). The truncation selection with $\tau = 50\%$ was used.

$$f_3(u) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases} \tag{14}$$

Two different ordering permutations π_1 and π_3 (see Equation 9) are used. The π_3 operator mixes up the positions of the bits in a string so that the three neighboring bits $(3i)$, $(3i+1)$, and $(3i+2)$ are positioned so that the distance between each two of them is at least one third of the length of chromosome. For instance, for $n = 12$ we get $\pi_3 = (0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11)$. The problem is actually the same for any permutation although for π_3 the dependencies are distributed more widely so that important schemata are of a greater defining length than originally.

Results for the problem of size $n = 30$ are shown in Table 1.

Table 1. Number of fitness evaluations for $f_{3deceptive}$. The population sizes for a permutation π_1 were 400 for GA (onepoint) and 1300 for BMDA. For permutation π_3 the population size for BMDA was 1300. GA (uniform) for both used permutations and GA (onepoint) for permutation π_3 did not achieve 100% convergence even for populations larger than 15000 (there is given a lower bound for the number of fitness evaluations in the table). The truncation selection with $\tau = 50\%$ was used.

	fitness eval. for π_1	fitness eval. for π_3
BMDA	$17,550$	$17,420$
GA (onepoint)	$4,977$	$> 230,000$
GA (uniform)	$> 650,000$	$> 650,000$

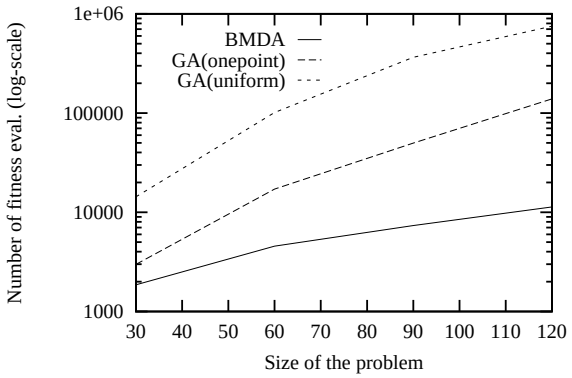

Figure 4. Number of fitness evaluations for $f_{quadratic}$ (log-scaling). The ranges of the used population size (for $n = 30$ to $n = 120$) were $200-570$ for BMDA, $260-2500$ for GA (onepoint), and $680 - 7000$ for GA (uniform). The truncation selection with $\tau = 50\%$ was used.

7.4 NK fitness function

There are two important input parameters for the so-called NK fitness function [10], the length of an input string n, the number of neighbors to take into account k. When the function is initialized, for each position of a string there are chosen k other positions at random. In this fashion we get n groups of positions, each of length $k+1$. Each position is contained in at least one of these groups. Let us denote $\{p_{i,0}, \ldots, p_{i,k}\}$ to be the ith group. For each group there is randomly generated value for each of possible combinations of values on corresponding positions. That means that for ith group, there are generated 2^{k+1} values. Let us denote by f_i the function that returns the generated value for combination of values $\{x_{p_{i,0}}, \ldots, x_{p_{i,k}}\}$ on positions from ith group. Then for given matrix of positions $P = (p_{i,j})_{i,j=0}^{n-1,k}$ and vector of functions $F = (f_i)_{i=0}^{n-1}$ (both generated in the initial stage) the fitness function is defined as

$$f_{NK}(x) = \sum_{i=0}^{n-1} f_i(x_{p_{i,0}}, \ldots, x_{p_{i,k}}) \tag{15}$$

Since the neighbors are picked at random it makes no sense to use any permutation to reorder the positions. NK fitness function is clearly decomposable of order at most $k + 1$. Random generation of both the neighbors and of the values can reduce the order, of course. However, the chance that this is the case is very small.

For $n = 50$ two different k were used, $k = 2$ and $k = 3$. For each of them, the matrix P and vector of functions F was generated first. Only one generated set of these parameters was used for each k. Results for various algorithms are presented in Table 2.

Since the optimum is not known for any of possible k, the best found number ever was taken as optimal and all algorithms were required to converge to this number. The problem space was explored by all algorithms with populations that should be large enough to find the real optimum. Even if this is not the case, i.e. the optimum was not found well, the best number ever found is taken into account, and even if the optimum were different from this number, this would not change anything on comparisons and their results.

Table 2. Number of fitness evaluations for f_{NK}. The population sizes for $n = 50$ and $k = 2$ were 210 for UMDA, 220 for GA (uniform), 340 for BMDA and 500 for GA (onepoint). The population sizes for $n = 50$ and $k = 3$ were 300 for GA (uniform), 700 for GA (onepoint), 2500 for BMDA and 8000 for UMDA. The truncation selection with $\tau = 50\%$ was used.

	$(n, k) = (50, 2)$	$(n, k) = (50, 3)$
BMDA	5,480	51,625
GA (onepoint)	11,010	17,440
GA (uniform)	4,288	7,826
UMDA	3,398	259.600

8. Conclusions

For linear and quadratic problems, BMDA works well what can be explained by the use of probabilistic distribution based on pair dependencies for the generation of new individuals. For linear problems UMDA and GA with uniform crossover perform better and GA with onepoint crossover performs similarly or worse than BMDA. The convergence of BMDA is slowed down by the need to discover the probabilistic model during optimization first. For quadratic problem, BMDA performs best among all compared algorithms. The GA with onepoint crossover performs best of all other algorithms but it fails when the defining length of important schemata is enlarged by different ordering of bits in a string. Experiments have shown that the gap between BMDA and other algorithms is for a quadratic function very large. The difference between this and other algorithms with linear function is insignificant in this context.

For BMDA, the problem arises with problems with dependencies of a higher order than two. For this class of problems, the used model of the search space is not sufficient. It is approximated somehow but this is often not sufficient. For deceptive fitness function of order 3, BMDA converged slower than GA with onepoint crossover. The GA with uniform crossover converged much worse than BMDA and UMDA was mislead in most of the cases even for huge population sizes. What is important though is that when the length of schemata is enlarged, GA with onepoint crossover performs very poor even for huge populations sizes and the number of fitness evaluations but the performance of BMDA remains almost the same. This, as well as the results for quadratic function, gives an evidence that GA with onepoint crossover does not effectively discover and use the schemata of a large defining length. It works very

well for short building blocks. Uniform crossover is independent of the length of schemata but it disrupts dependencies very often. UMDA disrupts dependencies too. BMDA takes into account dependencies of order at most two and this causes that although it is independent of length of the schemata and it does not disrupt dependencies of order two, it might disrupt the dependencies of a higher order. The dependencies of a higher order are substituted by chain-like structures composed of the dependencies of order two but this is often not sufficient. The solution to this problem might be the use of Factorization Distribution Algorithm although this requires a problem specific knowledge in the initial stage. This is not required by any of UMDA, BMDA, or GA. If this were overcome, FDA would perform very well for all problems that are decomposable.

Acknowledgements

Authors would like to thank The Department of Mathematics of Slovak Technical University and The German National Center for Information Technology (GMD) for a technical support of the project. Martin Pelikan's stay at GMD was supported by Catholic Academic Program for Foreigners (KAAD, Bonn). Special thanks to Vladimir Kvasnicka and Jiri Pospichal for useful comments and help with the preparation as well as the completion of the paper.

References

1. Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA

2. Kargupta H 1996 The Gene Expression Messy Genetic Algorithm. In: *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation.* Nagoya, pp 631–636

3. Muehlenbein H 1998 The Equation for Response to Selection and its Use for Prediction. *Evolutionary Computation* 5: 303–346

4. Muehlenbein H, Rodriguez A O 1998 *Schemata, Distributions and Graphical Models in Evolutionary Optimization*, submitted for publication

5. De Bonet J S, Isbell Ch L, Viola P 1997 MIMIC: Finding Optima by Estimating Probability Densities. In: Mozer M, Jordan M, Petsche Th (Eds) 1997 *Advances in Neural Information Processing Systems 9.* MIT Press, Cambridge

6. Baluja S, Davies S 1997 Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space. Report Number CMU-CS-97-107, Carnegie Mellon University, Pittsburgh, PA

7. Marascuilo L A, McSweeney M 1977 *Nonparametric and Distribution-Free Methods for the Social Sciences.* Brooks/Cole Publishing Company, CA

8. Kvasnicka V, Pelikan M, Pospichal J 1996 Hill Climbing with Learning (An Abstraction of Genetic Algorithm). *Neural Network World* 5: 773-796

9. Baluja S 1994 Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Report Number CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA

10. Kauffman S A 1993 *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, Inc., NY