Linkage Problem, Distribution Estimation, and Bayesian Networks

Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz

IlliGAL Report No. 98013
November 1998

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

Linkage Problem, Distribution Estimation, and Bayesian Networks

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

Abstract

In this paper, an algorithm based on the concepts of genetic algorithms that uses an estimation of the joint distribution of promising solutions in order to generate new candidate solutions is proposed. The algorithm is settled into the context of evolutionary computation and the algorithms based on the estimation of distributions. The proposed algorithm is called the Bayesian optimization algorithm (BOA). To estimate the distribution of promising solutions, the techniques for modeling multivariate data by Bayesian networks are used. The proposed algorithm identifies, reproduces and mixes building blocks up to a specified order. It is independent of the ordering of the variables in strings representing the solutions. Moreover, prior information about the problem can be incorporated into the algorithm. However, the prior information is not essential. The first experiments were done with additively decomposable problems with non-overlapping building blocks. The proposed algorithm is able to solve all tested problems in linear or close to linear time with respect to the problem size without the use of any prior knowledge about the problem.

Keywords

Linkage problem, estimation of distribution algorithm, modeling data, learning Bayesian networks, evolutionary computation.

1 Introduction

Recently, there has been a growing interest in optimization methods that explicitly model the good solutions found so far and use the constructed model to guide the further search (Baluja, 1994; Mühlenbein & Paaß, 1996; Mühlenbein, 1997; Harik et al., 1997; Pelikan & Mühlenbein, 1998). This line of research in stochastic optimization was strongly motivated by results achieved in the field of evolutionary computation. However, the connection between these two areas has sometimes been obscured. Moreover, the capabilities of model building have often been insufficiently powerful to solve hard optimization problems.

The purpose of this paper is to introduce an algorithm that uses techniques for estimating the joint distribution of multinomial data by Bayesian networks in order to generate new solutions. The proposed algorithm extends existing methods in order to solve more difficult classes of problems more efficiently and reliably. By covering interactions of higher order, the disruption of identified

*Also with the Institute of Computer Science, Faculty of Mathematics and Physics, Comenius University, Mlynska Dolina, 84215 Bratislava, Slovakia.

partial solutions is prevented. Prior information from various sources can be used. The combination of information from the set of good solutions combined with the prior information about a problem is used to estimate the distribution. The first experiments were done with additively decomposable problems with non-overlapping building blocks. The results indicate that the proposed algorithm is able to solve all tested problems in linear or close to linear time.

In Section 2, the background needed to understand the motivation and basic principles of the discussed methods is provided. The description of the algorithms based on the estimation of distributions and the connection between the methods based on the estimation of distributions and genetic algorithms are presented in Section 3. In Section 4, the Bayesian optimization algorithm (BOA) is introduced. In subsequent sections, the structure of Bayesian networks and the techniques used in the BOA to construct the network given a data set and the prior information about a problem as well as its use for the generation of new instances are described. The results of first experiments are presented in Section 8. The summary and conclusions are provided in section 9.

2 Background

Genetic algorithms (GAs) are optimization methods loosely based on the mechanics of artificial selection and genetic recombination operators. Most of the theory of genetic algorithms deals with the so-called *building blocks* (BBs) (Goldberg, 1989). By building blocks, partial solutions of a problem are meant. The genetic algorithm implicitly manipulates a large number of building blocks by mechanisms of selection and recombination. It reproduces and mixes building blocks. However, a fixed mapping from the space of solutions into the internal representation of the solutions in the algorithm and simple two-parent recombination operators soon showed to be insufficiently powerful even for problems that are composed of simpler partial subproblems. General, fixed, problem-independent recombination operators often break partial solutions what can sometimes lead to losing these and converging to a local optimum. Two crucial factors of the GA success—a proper growth and mixing of good building blocks—are often not achieved (Thierens, 1995). The problem of building block disruption is often referred to as the *linkage problem* (Harik & Goldberg, 1996). Various attempts to prevent the disruption of important partial solutions have been done and are briefly discussed in the remainder of this section. The first class of techniques is based on changing the representation of solutions in the algorithm or evolving the recombination operators among individual solutions (Goldberg et al., 1989; Harik, 1997; Kargupta, 1998). The second class of techniques is based on extracting some information from the entire set of promising solutions in order to generate new solutions (Baluja, 1994; Mühlenbein & Paaß, 1996; Mühlenbein, 1997; Harik et al., 1997; Pelikan & Mühlenbein, 1998). In this paper, we will focus on the second class of methods.

The goal of the first class of techniques based on manipulating the representation of solutions in the algorithm is to make the interacting components of partial solutions less likely to be broken by recombination operators. Various reordering and mapping operators were used. However, reordering operators are often too slow and lose the race against selection, resulting in premature convergence to low-quality solutions. Reordering is not sufficiently powerful in order to ensure a proper mixing of partial solutions before these are lost. This line of research has resulted in algorithms which evolve the representation of a problem among individual solutions. In the messy genetic algorithm (mGA) (Goldberg et al., 1989), the important building blocks are identified in the first phase. This is done by simply applying the selection operator to them. The remaining solution components are substituted from a special solution called the template. The template is updated each few generations. In the second phase, the identified building blocks are mixed using selection

and crossover operators. In the gene expression messy genetic algorithm (GEMGA) (Kargupta, 1998), the interactions in a problem are identified by manipulating individual solutions. These are used in order to improve the effects of recombination. In the linkage learning genetic algorithm (LLGA) (Harik, 1997), the variables in a problem are mapped onto a circle. Their mutual distances evolve during optimization, grouping together those with strong interactions so that recombination is less likely to disrupt them.

A different way to cope with the disruption of partial solutions is to change the basic principle of recombination. In the second class of techniques, instead of implicit reproduction of important building blocks and their mixing by selection and two-parent recombination operators, new solutions are generated by using the information extracted from the entire set of promising solutions.

One way to use the global information about the set of promising solutions is to estimate their distribution and use this estimate in order to generate new individuals. A general scheme of the algorithms based on this principle is called the estimation of distribution algorithm (EDA) (Mühlenbein & Paaß, 1996). However, estimating the distribution is not an easy task. There is a trade off between the accuracy of the estimation and its computational cost.

The simplest way to estimate the distribution of good solutions is to assume that the variables in a problem are independent. New solutions can be generated by only preserving the proportions of the values of all variables independently of the remaining solutions. This is the basic principle of the population based incremental learning (PBIL) algorithm (Baluja, 1994). In PBIL, a real vector is used instead of a population, and a simple incremental rule is used to update this vector after performing the selection on generated candidate solutions. The real vector composed of univariate frequencies of values on different positions represents the population. The update rule is known as Hebbian learning in the field of artificial neural networks (Hertz et al., 1991). It slightly shifts the vector towards the best of generated candidate solutions. The so-called compact genetic algorithm (cGA) (Harik et al., 1997) uses the same distribution estimate. The population is represented by a real vector. By using a different selection scheme and update rule than in PBIL, the connection between the cGA and the simple GA becomes more transparent. In cGA, the order-one behavior of the simple genetic algorithm with uniform crossover is approximated (Harik et al., 1997). The univariate marginal distribution algorithm (UMDA) (Mühlenbein, 1997) uses the same distribution estimate as well although it works with populations of solutions instead of a vector representing these. There is evidence that the UMDA also approximates the behavior of the simple GA with uniform crossover (Mühlenbein, 1997). The theory of UMDA based on the techniques of quantitative genetics can be found in Mühlenbein (1997). Some analyses of PBIL can be found in Kvasnicka et al. (1996).

The PBIL, cGA, and UMDA algorithms work very well for problems with no significant interactions among variables (Mühlenbein, 1997; Harik et al., 1997; Pelikan & Mühlenbein, 1998). However, the partial solutions of order more than one are disrupted and therefore these algorithms experience a great difficulty to solve problems with interactions among the variables. The first attempts to solve this problem were the incremental algorithm using the so-called dependency trees in order to estimate the distribution of selected solutions (we will refer to this algorithm as Baluja '97) (Baluja & Davies, 1997) and the population-based MIMIC algorithm using a simple chain distribution with the same purpose. Another population-based attempt to solve the problem of the disruption of building blocks of order two using different techniques is the bivariate marginal distribution algorithm (BMDA) (Pelikan & Mühlenbein, 1998). The algorithms mentioned above are able to cover some pairwise interactions. The reproduction of building blocks of order one is guaranteed. Moreover, the disruption of some important building blocks of order two is prevented. Important building blocks of order two are identified using various statistical methods. Mixing of

building blocks of order one and two is guaranteed assuming the independence of the remaining groups of variables.

However, covering only pairwise interactions has been shown to be insufficient to solve problems with interactions of higher order efficiently (Pelikan & Mühlenbein, 1998). Covering pairwise interactions still does not preserve the higher order partial solutions. Moreover, interactions of higher order do not necessarily imply pairwise interactions that can be detected at the level of partial solutions of order two.

In the factorized distribution algorithm (FDA) (Mühlenbein & Rodriguez, 1998), the factorization of the distribution is used in order to estimate the distribution. FDA is capable of covering the interactions of higher order and combining important partial solutions effectively. It works very well on additively decomposable problems. The theory of UMDA can be used in order to estimate the required population size as well as the time to convergence in FDA. However, the FDA requires the prior information about the problem in the form of the problem decomposition and its factorization. As an input, this algorithm gets a complete or approximate information about the structure of a problem. With this information, the recombination in the simple GA can be modified to disrupt the partial solutions less likely (Thierens, 1995; Kargupta, 1998). However, by providing sufficient conditions for the distribution estimate that ensures a fast and reliable convergence on decomposable problems, the FDA is of great theoretical value. Moreover, for problems of which the factorization of the distribution is known, this algorithm is a very powerful optimization tool. Unfortunately, the exact factorization of the distribution is often not available without computationally expensive problem analysis. Moreover, the use of an approximate distribution according to the current state of information from the set of promising solutions can be very effective even if it is not a valid distribution factorization.

The algorithm proposed in this paper is also capable of covering higher order interactions. It uses the techniques from the field of modeling data by Bayesian networks in order to estimate the joint distribution of promising solutions. This estimate is then used to generate new candidate solutions. Besides the set of good solutions, prior information about the problem can be used in order to enhance the estimation and subsequently improve convergence. However, unlike the FDA, the prior information about a problem is not essential. In the proposed algorithm, the identified building blocks up to a specified order are reproduced and mixed very efficiently.

In this paper, the solutions will be represented by binary strings of fixed length. However, the described techniques can be extended for strings over any finite alphabet.

3 General Estimation of Distribution Algorithm

A general scheme of the algorithms based on the estimation of distributions is called the estimation of distribution algorithm (EDA) (Mühlenbein & Paaß, 1996). In EDAs the distribution of promising solutions is estimated and this estimate is used to generate new candidate solutions. The distribution estimate represents the structure of the selected solutions. The mechanics of the EDA algorithms are similar to those of the simple GA. The first population of solutions (strings) is generated at random. From the current population, the better strings are selected. The distribution of the selected strings is estimated. Using this estimate, new strings are generated. The new strings are added into the old population, replacing some of the old ones. The process is repeated until the termination criteria are met. A more precise description of the EDA algorithm can be found in Figure 1. The two main questions to consider when designing an EDA algorithm are

- How to estimate the distribution of the selected strings?
- How to use this estimate in order to generate new strings?

The Estimation of Distribution Algorithm (EDA)

(1) set $t \leftarrow 0$
randomly generate initial population $P(0)$
(2) select a set of promising strings $S(t)$ from $P(t)$
(3) estimate the distribution of the selected set $S(t)$
(4) generate a set of new strings $O(t)$ according to the estimate
(5) create a new population $P(t+1)$ by replacing some strings from $P(t)$ with $O(t)$
set $t \leftarrow t+1$
(6) if the termination criteria are not met, go to (2)

Figure 1: **The pseudocode of the estimation of distribution algorithm.**

The two questions above are strongly correlated—the distribution should be estimated so that it is accurate and the generation of new strings can be performed effectively. There is no simple and general solution to this problem that would work well in general. The more general the distribution estimate, the more time consuming it is to find it and the more it takes to generate new points. EDAs differ in the way of estimating the distribution and using this estimate for generation of new individuals, i.e. in steps (3) and (4) of the pseudocode in Figure 1.

Estimation of distribution algorithms are often compared to the simple GA or other evolutionary algorithms. But what is the connection between these two classes of methods? In simple GAs, by selecting better solutions and applying recombination operators to them, promising building blocks are reproduced and combined together. But where are the building blocks in EDAs? And how do EDAs combine these? In the following paragraphs we briefly describe a few EDA algorithms and discuss the connection between them and simple GAs. A similar view on this topic through the schema theorem is presented in Mühlenbein and Rodriguez (1998); however, in their paper the mixing of promising building blocks is not discussed. By simply reproducing promising building blocks without ensuring their proper mixing, the exploration of promising regions of the search space is not guaranteed (Thierens & Goldberg, 1993).

The univariate marginal distribution algorithm (Mühlenbein, 1997) uses a simple univariate marginal distribution, assuming that the variables are independent. The distribution used in UMDA is fixed during the whole optimization. It can be written as

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

where $X = (X_0, \ldots, X_{n-1})$ is a vector of variables and $p(X_i)$ for $i \in \{0, \ldots, n-1\}$ is the marginal frequency of X_i. A variable X_i corresponds to the ith position in strings representing the solutions. Each variable corresponds to one string position.

Assuming that all variables are independent, the partial solutions of order one are mixed together very efficiently. Combining the selection of good solutions with generation of new candidate solutions by assuming the independence of all variables results in a growth and a very effective mixing of partial solutions of length one. Partial solutions of higher order are ignored and therefore often disrupted by UMDA. It was theoretically shown that UMDA works very similarly to the simple GA with uniform crossover (Mühlenbein, 1997).

In the bivariate marginal distribution algorithm (BMDA) (Pelikan & Mühlenbein, 1998), some pairwise interactions are taken into account. The most important acyclic pairwise interactions are considered, according to the Pearson's chi-square statistics for independence. Each generation, the distribution estimate used in BMDA to generate new solutions is updated according to the selected set of solutions. A general distribution used in BMDA can be written as

$$p(X) = \prod_{j=0}^{r} p(X_{i_j}) \prod_{j=r+1}^{n-1} p(X_{i_j}|X_{e_j}), \tag{2}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of variables, i is a permutation of $\{0, \ldots, n-1\}$, $(r+1)$ is the number of mutually independent interacting subsets of variables, and $e_j \in \{i_0, \ldots, i_{j-1}\}$ for all $j \in \{r+1, \ldots, n-1\}$. By permutation i, the variables are topologically ordered so that each variable is independent or conditioned on a variable with lower order.

In BMDA, the frequencies for most strongly interacting pairs of variables are preserved by recombination. However, by using this simple model, not all pairwise interactions can be taken into account. In BMDA, similarly as in UMDA, the growth in a number of promising partial solutions of order one is ensured. Moreover, for some partial solutions of order two, their immunity of being disrupted by the recombination is guaranteed. By assuming other pairs of variables are independent, their efficient mixing is ensured. Therefore, the BMDA is processing (reproducing and mixing) the building blocks of order one and two. To decide what building blocks of order two should be taken into account, statistical methods are used. BMDA performs similarly as the simple GA with one-point crossover when the building blocks are tightly mapped onto strings and if their order is low. If the interacting variables are tightly mapped onto strings, the simple GA outperforms the BMDA as the order of the building blocks grows. On the other hand, when the variables are mapped loosely, the BMDA outperforms the simple GA with growing size of important partial solutions.

The factorized distribution algorithm (FDA) (Mühlenbein & Rodriguez, 1998) uses an exact or approximate factorization of a problem to estimate the distribution. This remains fixed during the optimization. The theory of UMDA (Mühlenbein, 1997) can be applied for a valid factorized distribution in FDA (Mühlenbein & Mahnig, 1998). The factorization of the distribution can be obtained by analyzing a prior information about the problem decomposition. A general factorized distribution can be written as

$$p(X) = \prod_{j=0}^{l-1} p(X_{i_j}|\Pi_{X_{i_j}}), \tag{3}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of variables, i is a permutation of $\{0, \ldots, n-1\}$, the set of variables $\Pi_{X_{i_j}} \subset \{X_{i_0}, \ldots, X_{i_{j-1}}\}$, and $p(X_{i_j}|\Pi_{X_{i_j}})$ is the conditional probability of X_{i_j} conditioned on the variables $\Pi_{X_{i_j}}$.

As it was correctly stated in Mühlenbein and Rodriguez (1998), in FDA the growth of interacting blocks is ensured by selecting good solutions and using the distribution factorization for generating new individuals. Moreover, as in the BMDA, their efficient mixing is guaranteed by assuming that non-interacting genes are independent, while respecting the mutual parts of interacting solutions. These properties are a consequence of the properties of the distribution factorization used in FDA and the assumptions of independence (Mühlenbein & Rodriguez, 1998).

A very brief overview of capabilities and difficulties of the discussed EDA algorithms is provided in Table 1.

Algorithm	Capabilities	Difficulties
PBIL, cGA, UMDA	Very efficient on linear problems.	Higher order BBs.
MIMIC, Baluja '97, BMDA	Very efficient with BBs of length two. Beats simple GAs for loose BBs.	Higher order BBs.
FDA	Very efficient with additively decomposable problems.	Prior information about the problem structure and its analysis are essential.

Table 1: **An overview of EDAs.**

In our approach, the techniques from modeling data by Bayesian networks are used to estimate the joint probability distribution of selected strings. The estimate is updated to match the current state of information in the selected set. Interactions up to a specified order can be covered. Moreover, various ways to incorporate the prior information about the problem can be used. In the proposed algorithm, the identified building blocks up to a specified order are reproduced and mixed.

4 The Bayesian Optimization Algorithm (BOA)

This section introduces an EDA algorithm that uses techniques from the field of modeling data by Bayesian networks to estimate the joint distribution of promising solutions. It is called the Bayesian optimization algorithm (BOA). It covers both UMDA and BMDA and extends them to cover the interactions of higher order. The order of interactions that will be taken into account is given as input to the algorithm. The proposed algorithm allows the use of prior information about the problem, both the information from the user (or a problem analysis) as well as the information about the problem acquired by the algorithm on the run. The prior information about the structure of a problem as well as the information represented by the set of high-quality solutions can be incorporated into the algorithm. The combination of the prior information and the set of promising solutions is used to estimate the distribution. Unlike in the FDA, the prior information about the problem is not essential.

In BOA, the first population of strings is generated at random. From the current population, the better strings are selected. Any selection method can be used. A Bayesian network that fits the selected set of strings is constructed. Any metric as a measure for quality of networks and any search algorithm can be used to search over the networks in order to maximize the value of the used metric. New strings are generated using the joint distribution encoded by the constructed network. The new strings are added into the old population, replacing some of the old ones. The pseudocode of the BOA can be found in Figure 2.

Constructing the network in BOA corresponds to estimating the distribution in EDA. Generating the set of new strings according to the estimated distribution is identical to the corresponding step in EDA. In the following sections, Bayesian networks and the techniques for their construction and use will be described.

5 Bayesian Networks

Bayesian networks (Howard & Matheson, 1981; Pearl, 1988) are often used for modeling multinomial data with both discrete and continuous variables. A Bayesian network encodes the relationships

The Bayesian Optimization Algorithm (BOA)

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

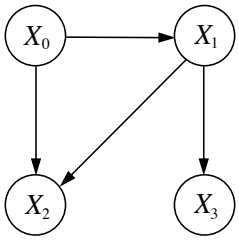
Figure 2: **The pseudocode of the Bayesian optimization algorithm.**

between the variables contained in the modeled data. It represents the structure of a problem. Bayesian networks can be used to describe the data as well as to generate new instances of the variables with similar properties as those of given data. Each node in the network corresponds to one variable. By X_i, both the variable and the node corresponding to this variable will be denoted in this text. The relationship between two variables is represented by an edge between the two corresponding nodes. The edges in Bayesian networks can be either directed or undirected. In this paper, only Bayesian networks represented by directed acyclic graphs will be considered. The modeled data sets will be defined within discrete domains.

Mathematically, an acyclic Bayesian network with directed edges encodes a joint probability distribution. This can be written as

$$p(X) = \prod_{i=0}^{n-1} p(X_i|\Pi_{X_i}), \tag{4}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of variables, Π_{X_i} is the set of parents of X_i in the network (the set of nodes from which there exist an edge to X_i) and $p(X_i|\Pi_{X_i})$ is the conditional probability of X_i conditioned on the variables Π_{X_i}. A simple example of a Bayesian network and the joint distribution encoded by this network can be found in Figure 3. This distribution can be used to generate new instances using the marginal and conditional probabilities in a modeled data set.



$$p(X) = p(X_0).p(X_1|X_0).p(X_2|X_0, X_1).p(X_3|X_1)$$

Figure 3: **A simple example of a Bayesian network and the joint distribution encoded by this network.**

The network, i.e. the relationships between the variables, can be either known or unknown. If the network is known, the joint distribution encoded by the network with the marginal and

conditional probabilities present in this joint distribution can be used as a model for data. If the network is unknown, i.e. we don't know what variable is related with which one or there is some uncertainty about that, the network has to be determined as well. The following sections address the two important questions:

- How to determine the network that would serve as a good model for a given data set?
- How to use the network to generate new instances that would best match the given data?

Note that the two questions above directly correspond to the two questions arising with the use of the EDAs. Constructing the Bayesian network for a given set of promising solutions corresponds to estimating their joint distribution. Generating new instances according to the constructed network is in fact generating new solutions with the joint distribution encoded by this network.

5.1 Learning the Network Structure

There are two basic components of the algorithms for learning the network structure (Heckerman et al., 1994). The first one is a scoring metric and the second one is a search procedure. A scoring metric is a measure of how well the network models the data. Prior knowledge about the problem can be incorporated into the metric as well. A search procedure is used to explore the space of all possible networks in order to find the one (or a set of networks) with the value of a scoring metric as high as possible. The space of networks can be reduced by constraint operators. Commonly used constraints restrict the networks to have at most k incoming edges into each node. This number directly influences the complexity of both the network construction as well as its use for generation of new instances and the order of interactions that can be covered by the class of networks restricted in this way.

In the following sections, the Bayesian Dirichlet metric and search algorithms that can be used to search over the networks are described and their complexity analyses are provided.

5.1.1 Bayesian Dirichlet Metric

The Bayesian Dirichlet (BD) metric (Heckerman et al., 1994) can be used as a measure of the quality of networks. It combines the prior knowledge about the problem and the statistical data from a given data set. In this section, the BD metric and its input parameters will be described.

The BD metric for a network B given a data set D, and the background information ξ, denoted by $p(D, B|\xi)$, is defined as

$$p(D, B|\xi) = p(B|\xi) \prod_{i=0}^{n-1} \prod_{\pi_{X_i}} \frac{m'(\pi_{X_i})!}{(m'(\pi_{X_i}) + m(\pi_{X_i}))!} \prod_{x_i} \frac{(m'(x_i, \pi_{X_i}) + m(x_i, \pi_{X_i}))!}{m'(x_i, \pi_{X_i})!}, \tag{5}$$

where $p(B|\xi)$ is the prior probability of the network B, the product over π_{X_i} runs over all instances of the parents of X_i and the product over x_i runs over all instances of X_i. By $m(\pi_{X_i})$, the number of instances in D with variables Π_{X_i} (the parents of X_i) instantiated to π_{X_i} is denoted. When the set Π_{X_i} is empty, there is one instance 0 of Π_{X_i} and the number of instances with Π_{X_i} instantiated to this instance is set to N, i.e. the size of the data set D. By $m(x_i, \pi_{X_i})$, we denote the number of instances in D that have both X_i set to x_i as well as Π_{X_i} set to π_{X_i}. It is easy to see that

$$m(\pi_{X_i}) = \sum_{x_i} m(x_i, \pi_{X_i}), \tag{6}$$

where π_{X_i} is an instance of the parents of X_i and the sum runs over all instances of X_i.

By numbers $m'(x_i, \pi_{X_i})$ and $p(B|\xi)$, prior information about the problem is incorporated into the metric. The prior probability of the network reflects how the measured network resembles the prior network. By using a prior network, the prior information about the structure of a problem is incorporated into the metric. The prior network can be set to an empty network, when there is no such information. The $m'(x_i, \pi_{X_i})$ stands for a prior information about the number of instances that have variable X_i set to x_i and the set of variables Π_{X_i} is instantiated to π_{X_i}. An analogical relation as for m in Equation 6 is satisfied for m' as well.

The prior probability of a network can be computed in a number of ways. Heckerman et al. (1994) suggest a simple assignment by the following formula:

$$p(B|\xi) = c\kappa^{\delta}, \tag{7}$$

where c is a normalization constant, $\kappa \in (0, 1]$ is a constant factor penalizing the network for each unmatched edge with the prior network, and δ is the so-called symmetric difference between B and the prior network. The symmetric difference between two networks is the number of edges where the networks differ. For $\kappa = 1$, all networks are treated equally. The smaller the κ, the stronger the networks are penalized for each missing or extra edge with respect to the prior network.

The numbers $m'(x_i, \pi_{X_i})$ can be set in various ways. They can be set according to the prior information the user has about the problem. When there is no prior information, uninformative assignments can be used. In the so-called K2 metric, for instance, the $m'(x_i, \pi_{X_i})$ coefficients are all simply set to 1 (Heckerman et al., 1994). This assignment corresponds to having no prior information about the problem. Another possibilities for setting the $m'(x_i, \pi_{X_i})$ priors include Buntine's uninformative assignment (Buntine, 1991) or a more sophisticated assignment of priors used in the so-called BDe metric (Heckerman et al., 1994). For more about the assignment of uninformative priors, see Heckerman et al. (1994) or the discussion in Bernardo and Smith (1994). In our BOA algorithm, any of the mentioned metrics can be used. In the empirical part of this paper, we will use the K2 metric and the metric used recently in BMDA (Pelikan & Mühlenbein, 1998). A simple example of a data set and the obtained score with the K2 metric for two different networks follows.

Example 5.1 (K2 Metric) Let us have two variables X_0 and X_1 and the set of their instances $D = \{00, 00, 00, 11\}$ where the first position corresponds to X_0 and the second one to X_1. First, let us compute the value of the K2 metric for a network B_{empty} with no edges. In the K2 metric, the $m'(x_i, \pi_{X_i})$ are set to 1. Thus, using an analogy of the relation from Equation 6, the $m'(\pi_{X_i})$ are set to 2. The table with values of the terms from Equation 5 follows:

i	x_i	π_{X_i}	$m(x_i, \pi_{X_i})$
0	0	0	3
0	1	0	1
1	0	0	3
1	1	0	1

The remaining terms can be computed using Equation 6. Thus,

$$p(B_{empty}, D|\xi) = \frac{2!}{(2+4)!} \cdot \frac{(1+3)!}{1!} \cdot \frac{(1+1)!}{1!} \cdot \frac{2!}{(2+4)!} \cdot \frac{(1+3)!}{1!} \cdot \frac{(1+1)!}{1!} = \frac{4}{225}$$

For a network $B_{0\to1}$ containing an edge from X_0 to X_1, the $m'(x_i, \pi_{X_i})$ are again set to 1 and $m'(\pi_{X_i})$ are set to 2. The table with values of the terms from Equation 5 follows:

i	x_i	π_{X_i}	$m(x_i, \pi_{X_i})$
0	0	0	3
0	1	0	1
1	0	0	3
1	0	1	0
1	1	0	0
1	1	1	1

The remaining terms can be again computed using Equation 6. Thus,

$$\begin{aligned} p(B_{0\to 1}, D|\xi) &= \frac{2!}{(2+4)!}\cdot\frac{(1+3)!}{1!}\cdot\frac{(1+1)!}{1!}\cdot\frac{2!}{(2+3)!}\cdot\frac{(1+3)!}{1!}\cdot\frac{(1+0)!}{1!}\cdot \\ &\cdot \frac{2!}{(2+1)!}\cdot\frac{(1+1)!}{1!}\cdot\frac{1!}{1!} = \frac{8}{225} \end{aligned}$$

We would get the same result with an edge from X_1 to X_0. Thus, the networks with connected nodes get a higher score than the network with no edges. This result is not surprising. The positions (variables) are clearly correlated in the set D. Each of them determines the value of the other one.

In the previous example, the value of the K2 metric remained the same after reversing an edge. This is not the case in general. Reversing any path among the nodes with the same parents would not affect the BDe metric. This property is called the likelihood equivalence (Heckerman et al., 1994). However, even this condition is not satisfied for other mentioned metrics (e.g., K2 metric, see Heckerman et al. (1994)). All of the mentioned metrics derived from the BD metric are consistent with the assumptions of multinomial sample, parameter independence, parameter modularity, Dirichlet assumption and the assumption of complete data. In addition to this, the BDe metric is consistent with the likelihood assumption (Heckerman et al., 1994).

Since the factorials in Equation 5 can grow to huge numbers, usually a logarithm of the scoring metric is used. Using simple logarithmic rules we get

$$\log\left(\frac{m'(\pi_{X_i})!}{(m'(\pi_{X_i})+m(\pi_{X_i}))!}\right) = -\sum_{s=m'(\pi_{X_i})+1}^{m'(\pi_{X_i})+m(\pi_{X_i})}\log s \tag{8}$$

and similarly

$$\log\left(\frac{(m'(x_i,\pi_{X_i})+m(x_i,\pi_{X_i}))!}{m'(x_i,\pi_{X_i})!}\right) = \sum_{t=m'(x_i,\pi_{X_i})+1}^{m'(x_i,\pi_{X_i})+m(x_i,\pi_{X_i})}\log t \tag{9}$$

Thus,

$$\begin{aligned} \log\left(p(D,B|\xi)\right) = \log\left(p(B|\xi)\right) + \sum_{i=0}^{n-1}\sum_{\pi_{X_i}}\Bigg(&-\sum_{s=m'(\pi_{X_i})+1}^{m'(\pi_{X_i})+m(\pi_{X_i})}\log s + \\ &+\sum_{x_i}\sum_{t=m'(x_i,\pi_{X_i})+1}^{m'(x_i,\pi_{X_i})+m(x_i,\pi_{X_i})}\log t\Bigg) \end{aligned} \tag{10}$$

With binary variables, there are maximally 2^k instances of the parents of any node. The marginal

probabilities for the variables corresponding to one node and its parents can be computed in $O(kN + 2^k)$ where N is the size of a data set (the number of given instances of all variables) and k is a maximal number of incoming edges into each node in the network. The computation of the sums from equations 8 and 9 can be done in $O(N)$ steps. Thus, the computation of all terms needed to compute the contribution of one node into the logarithm of the metric can be done in $O(kN + 2^k)$ steps. The contribution of one node to the logarithm of the metric can be then computed in $O(2^k N)$ steps. The computation of an increase of the logarithm of the value of the BD metric for an edge addition, edge reversal, or an edge removal, respectively, can be computed in time $O(2^k N)$ since the total sum contribution corresponding to the nodes of which the set of parents has not changed remains unchanged as well. Assuming that k is constant, we get a linear time complexity $O(N)$ with respect to the size of the data set of both the computation of the contribution of one node to the overall value of the metric as well as the computation of an increase in the metric for an edge addition.

BMDA uses a simple metric defined as the sum of the values of Pearson's chi-square statistic for independence for the pairs of variables that are connected in the network (Pelikan & Mühlenbein, 1998). Only the pairs that are not independent at 5% confidence level with respect to this statistic are considered. The metric used in BMDA gives preference to networks that cover pairwise interactions in a given data set. However, the interactions of higher order do not necessarily imply pairwise correlations that can be detected with probabilistic terms of length two (see Example 5.2). The Bayesian Dirichlet metric is able to give preference to networks that cover correlations of a higher order, too.

Example 5.2 (Pairwise vs. Higher Order Interactions) A simple example of a data set where the interactions of higher order do not imply correlations identifiable with the probabilistic terms of order two might be the set $D = \{000, 011, 101, 110\}$. In this set, each pair of positions seems to be independent. However, the strings are not uniformly distributed with proportions of 50% of ones on each position and therefore the variables (corresponding to the positions in strings) are not mutually independent. In spite of this, the metric used in BMDA gives the same score 0 to all networks for D.

5.1.2 Searching for a Good Network

In this section, the basic principles of algorithms that can be used for searching over the networks in order to maximize the value of a scoring metric are described. Only the classes of networks with restricted number of incoming edges denoted by k will be considered.

a) $k = 0$

This case is trivial. An empty network is the best one (and the only one possible).

b) $k = 1$

To compute an increase in the score of a network with at most one incoming edge into each node only a constant time is required. All components of the BD metric are composed of univariate and bivariate frequencies and these can be precomputed in $O(n^2 N)$ steps. The metric can be written as the sum of the contributions of all edges. Each edge can be weighted by the increase in the value of the metric in case of its addition (Heckerman et al., 1994). To find the best network with $k = 1$ reduces then to a special case of the so-called maximal branching problem. To solve this problem, there exists a polynomial algorithm (Edmonds, 1967). The same algorithm can be used for the metric used in BMDA.

c) $k > 1$

For $k > 1$ the problem gets much more complicated. Although for $k = 1$ there exists a polynomial algorithm for finding the best network, for $k > 1$ the problem of determining the best network with respect to a given score metric is NP-complete for most Bayesian and non-Bayesian metrics (Heckerman et al., 1994; Chickering et al., 1994).

Various algorithms can be used in order to find a good network, from a total enumeration to a blind random search. Usually, due to their effectiveness in this context, simple local search based methods are used (Heckerman et al., 1994). A simple greedy algorithm, local hill-climbing, or simulated annealing can be used. Simple operations that can be performed on a network include an edge addition, an edge reversal and an edge removal. In each iteration of a simple greedy search, the edge which increases the score the most is added. Edge reversal or removal operations can be allowed as well. Only operations that keep the network acyclic and with at most k incoming edges into each of the nodes are allowed (i.e., the operations that do not violate the constraints). The algorithms can start with an empty network, the best network with one incoming edge into each node at maximum, or a randomly generated network.

In our implementation, we have used a simple greedy algorithm with only an edge addition allowed for $k > 1$. The algorithm starts with an empty network. The time complexity of this algorithm can be computed using the time complexity of a simple edge addition and the number of edges that have to be processed at most. First, the univariate and bivariate frequencies have to be calculated and used to compute the increase in the score of a metric for all edge additions into an empty graph. As in the algorithm for $k = 1$, this requires $O(n^2 N)$ steps. Picking the best edge to add can be done in $O(n^2)$ steps since there are maximally n^2 edges to add. To recompute the gains for the edge additions after performing a particular edge addition, the computational time depends on the used metric. For the metric used in BMDA nothing has to be recomputed. For the BD metric, the increases can be recomputed in $O(2^k nN)$. Adding an edge into the network and updating the information needed to keep the network acyclic and consistent with the constraint on the maximal number of incoming edges can be done in $O(n^2)$ steps. With k incoming edges into each node at maximum, at most $(k.n)$ edges can be added into any network. The overall time to construct the network using the described greedy algorithm is then $O(n^2 N + kn^3)$ with the metric used in BMDA and $O(k2^k n^2 N + kn^3)$ with BD metric. Assuming that k is constant we get the overall time complexity $O(n^2 N + n^3)$ with both the metric from BMDA as well as with the BD metric, i.e. the resulting time requirements are qualitatively the same with respect to the size of the problem and the size of the selected set of solutions.

A similar algorithm can be used for constructing the network under different constraints. Only operations that do not violate the constraints can be allowed and the time complexity can be computed in the same way as with the maximal incoming edges constraint. Allowing other operations (e.g., an edge reversal or removal) would make the analyses somewhat more complicated.

With the metric used in BMDA, based only on the pairwise statistics for independence, we have used a more sophisticated algorithm for a network construction. Since the Pearson's chi-square statistics is symmetric, this metric is independent of the orientation of edges. Therefore, after adding an edge, the edges can be redirected to satisfy the constraints and to reduce the chance of future constraint violations. In the algorithm, the best edge is added first, even if it violates the constraints. Then, a path is chosen beginning in the ending node of this edge such that, if the path were reversed, it would reduce the constraint violation as much as possible. If, after reversing this path, the constraints are not violated, the chosen path is reversed. If there is no path to reverse that would relax the network to satisfy the constraints, the new edge is removed and the network remains unchanged. With the constraint on the maximal number of edges into each node,

The algorithm for the generation of a new instance

(1) mark all variables as unprocessed
(2) pick up an unprocessed variable X_i with all parents processed already
(3) set X_i to x_i with probability $p(X_i = x_i | \Pi_{X_i} = \pi_{X_i})$
(4) mark X_i as already processed
(5) if there are unprocessed variables left, go to (2)

Figure 4: **The pseudocode of the algorithm for generating a new instance given the network and the set of conditional probabilities from the corresponding joint distribution.**

a path that minimizes the maximal number of incoming edges in the network if it is reversed, can be chosen. Each edge is processed exactly once. Once an edge is rejected, it is not considered to be added anymore. One edge addition in the algorithm can be performed in polynomial time since there is maximally n^2 edges and each edge can be examined at most once. However, this algorithm cannot be used with the BD metric since the BD metric is sensitive to edge orientation. For metrics that are consistent with the likelihood assumption, a path among nodes with the same set of parents can be reversed though.

5.2 Generating New Instances

In this section, the generation of new instances using a network B and the marginal frequencies for few sets of variables in the modeled data set will be described. This corresponds to the step (4) in the pseudocode of BOA (see Figure 2).
The joint distribution encoded by the network B can be written as

$$p(X) = \prod_{i=0}^{n-1} p(X_i | \Pi_{X_i}), \tag{11}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of random variables, Π_{X_i} is the set of parents of X_i in the network B, and $p(X_i|\Pi_{X_i})$ is a conditional probability of X_i given (conditioned on) the set of variables Π_{X_i}, i.e.

$$p(X_i | \Pi_{X_i}) = \frac{p(X_i, \Pi_{X_i})}{p(\Pi_{X_i})} \tag{12}$$

Since the network is acyclic, it is easy to generate a new instance. First, the conditional probabilities of each possible instance of each variable given all possible instances of its parents in a given data set are computed. The conditional probabilities are used to generate each new instance. Each iteration, the nodes whose parents are already fixed are generated using the corresponding conditional probabilities. This is repeated until the values for all variables are generated. Since the network is acyclic, it is easy to see that the algorithm is defined well. It is similar to the so-called forward simulation in Bayesian networks (Henrion, 1988). A more detailed description of the algorithm for generating a new instance can be found in Figure 4.

The time complexity of generating the value for each variable is $O(k)$ where k is a maximal

number of incoming edges in the network. The time complexity of generating an instance of all variables is then bounded by $O(kn)$ where n is the number of variables. Assuming the k is constant, the overall time complexity is $O(n)$, i.e. the complexity of generating each new instance grows linearly with the problem size.

6 Additively Decomposable Functions, the Interactions, and What is Hard for Simple GAs

In this section, the class of additively decomposable functions is defined. The problems defined by this class of functions can be decomposed into smaller subproblems. Although by combining partial solutions it is possible to get a global solution, the simple GAs experience a great difficulty to solve some decomposable problems. This topic is addressed in this section as well. The relation between BOA and the class of additively decomposable problems is discussed in the next section.

Let us start with a simple definition of an order-k decomposable function (problem). We say that a function f from the vector of n variables to real numbers is order-k (additively) decomposable if there exists a set of l functions f_i over the subsets of variables S_i for $i = 0, \ldots, l-1$, each of the size at most k, for which the following equation is satisfied over the domain of f:

$$f(X) = \sum_{i=0}^{l-1} f_i(S_i), \tag{13}$$

where X is the vector of variables.

In other words, the function is order-k (additively) decomposable, if we can write it as the sum of simpler functions, each over at most k variables from the original domain of the function f.

First, let us consider the functions which can be decomposed by using only non-overlapping sets of variables S_i, i.e. the functions for which there exists a set of disjoint sets S_i, each of at most k variables, and a function f_i for each S_i, so that Equation 13 is satisfied. With discrete variables with finite domain, these functions can be optimized in time linearly proportional to the size of a problem (a total number of variables). Since the subsets S_i are non-overlapping, each subfunction can be considered independently. The optimum can be obtained by simply optimizing each of the subfunctions f_i by enumeration. The time complexity of the enumeration of one of the subfunctions is $O(2^k)$. For disjoint sets S_i, the number of subfunctions is upper-bounded by n. Thus, the overall time complexity is $O(2^k n)$. Assuming that k is constant, we get linear overall time complexity $O(n)$. Therefore, if the structure of this function is known, it is numerically very easy to optimize it.

The subfunctions f_i can be constructed so that the problem is hard for the simple GA. This can be done by using the so-called fully deceptive subfunctions (Deb & Goldberg, 1994) for which all the schemata of order less than k mislead the algorithm away from the global optimum into a local one. Important building blocks have tendency to vanish if they are disrupted. The performance of the simple GA for these problems crucially depends on the mapping of variables onto the strings. For a mapping that puts the variables from the same subset close to each other on the string, the deceptiveness of the building blocks does not mean a significant increase in the problem difficulty in terms of population sizing or the number of function evaluations until convergence in order to achieve the solution of a particular quality (Thierens, 1995). The relation between the number of building blocks and the population size remains qualitatively the same. When the mapping spreads the variables from the same set throughout the whole string, the population sizes for the simple

GA have to grow exponentially with the problem size and so does the required number of fitness evaluations in order to obtain the solution of a particular quality (Thierens, 1995).

For the functions that can be decomposed into subfunctions of order at most k but the corresponding domains overlap, the problem becomes much more complicated even if the structure of the problem is known. We cannot consider each subfunction independently, optimize it and get the global optimum by combining the partial solutions together as it was with non-overlapping sets. The global optima of two overlapping sets of variables can differ in some of the values from their intersection. Therefore these sets cannot be taken independently. A possible solution to this problem is the factorized distribution algorithm (Mühlenbein & Mahnig, 1998), which is able to combine important building blocks together even if they are overlapping. However, the FDA needs as an input a valid or an approximate factorization of a problem and without this information it is not applicable. The simple GA experiences a great difficulty optimizing the problems with subfunctions that are both deceptive and overlapping, e.g. spin-glass systems (Mühlenbein & Rodriguez, 1998).

To make the mentioned problems easy for the simple GA with one-point crossover, one would have to ensure that the interacting variables are located close to each other. The longer the distances between interacting variables are, the worse the simple GA with one-point crossover performs for certain functions. On the other hand, for uniform crossover, which is independent of the ordering of variables in a string, to obtain the solution of a particular quality for deceptive problems, populations have to grow exponentially with the size of the problem (Thierens, 1995). The performance of the simple GA can be improved with problem-specific recombination operators that respect the building blocks according to the solved problem.

There are other sources of difficulty for simple GAs. These include functions with noise and the functions composed of subfunctions that are scaled in some way. However, in this paper, we focus on solving the difficulty arising from gene interactions only.

7 BOA and Additively Decomposable Functions

In this section, the relation between the constraint on the number of incoming edges, denoted by k, in the BOA and the order of function decomposition is briefly discussed.

For $k = 0$, no interactions in the problem can be covered. The algorithm is equivalent to UMDA. Therefore, it works very well for linear problems and problems with no significant interactions (Mühlenbein, 1997; Harik et al., 1997; Pelikan & Mühlenbein, 1998).

For $k = 1$, the algorithm is able to cover some pairwise interactions. However, it is not able to cover pairwise interactions that create cycles in underlying undirected graph or interactions of higher order completely. It works very well for linear and quadratic problems (Pelikan & Mühlenbein, 1998). For problems with significant gene interactions, it works much better than UMDA and for loose gene orderings with deceptive problems (spreading the deceptive building blocks across the string) it outperforms the simple GA as well (Pelikan & Mühlenbein, 1998).

For $k > 1$, the interactions of order $(k+1)$ can be covered. This actually does not mean that all interactions in a problem that is order-$(k+1)$ decomposable can be covered. Let us, for instance, have a 2D spin-glass problem instance with adjacent neighbors (Mühlenbein & Rodriguez, 1998). This problem is decomposable of order two. However, in this problem the building blocks are highly overlapping. It is easy to see that the interactions cannot be covered using only univariate and bivariate frequencies, i.e. the probabilistic terms of order two. Therefore, not only is the order of function decomposition necessary to determine the sufficient number of incoming edges to cover the interactions. Graph theory deals with the mentioned problem. Identifying the cliques

in the independence graph and investigating the clique-separators can give the minimal number of a parameter k so that the network could model the data accurately (Mühlenbein & Rodriguez, 1998). However, using a complete model is often not necessary. The set of promising solutions can be used in order to determine which of their properties are important according to the current state of information and which ones are not.

8 Experiments

First experiments were done for decomposable problems composed of functions of unitation. These include the linear function as well as few deceptive functions. The experiments were designed in order to show the behavior of the proposed algorithm only on non-overlapping decomposable problems with deceptive building blocks. For all problems, the scalability of the proposed algorithms is investigated. In the following sections, the functions of unitation used in experiments will be described and the results of the experiments will be presented.

8.1 Functions of Unitation

A function of unitation is a function whose value depends only on the number of ones in an input string. The function values for the strings with the same number of ones are equal.

Several functions of unitation can be additively composed in order to form a more complex function. Let us have a function of unitation f_k defined for strings of length k. Then, the function additively composed of l functions f_k is defined as

$$f(X) = \sum_{i=0}^{l-1} f_k(S_i), \tag{14}$$

where X is the set of n variables and S_i for $i \in \{0, \ldots, l-1\}$ are subsets of k variables from X. Sets S_i can be either overlapping or non-overlapping and they can be mapped onto a string (the inner representation of a solution) so that the variables from one set are either mapped close to each other or spread throughout the whole string. A function composed in this fashion is clearly additively decomposable of order of the subfunctions it was composed with. Each variable will be required to contribute to the function through some of the subfunction, i.e.

$$X = \bigcup_{i=0}^{l-1} S_i \tag{15}$$

A simple function called $OneMax$ is defined for a single bit as its value, i.e.

$$f_{onemax}(u) = u, \tag{16}$$

where u is the number of ones in an input string (the value of a single bit in this case). By concatenating more single-bit $OneMax$ functions we get a simple linear function that returns the number of ones in an input string.

A deceptive function of order 3, denoted by 3-*deceptive*, is defined as

$$f_{3deceptive}(u) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases} \tag{17}$$

where u is the number of ones in an input string.

A trap function of order 5, denoted by *trap*-5, is defined as

$$f_{trap5}(u) = \begin{cases} 5 - u & \text{if } u < 5 \\ 5 & \text{otherwise} \end{cases} \tag{18}$$

where u is the number of ones in an input string.

A bipolar deceptive function of order 6, denoted by 6-*bipolar*, is defined with the use of the 3-*deceptive* function as follows

$$f_{6bipolar}(u) = f_{3deceptive}(|3 - u|) \tag{19}$$

where u is the number of ones in an input string.

8.2 The Results of Experiments

For all problems, the average number of fitness evaluations until convergence in 30 independent runs is shown. For *OneMax*, 3-*deceptive*, and *trap*-5 functions, the population is said to have converged when the proportion of some value on each position reaches 95%. This criterion of convergence is applicable only for problems with at most one global optimum and selection schemes that do not force the algorithm to preserve the diversity in population (e.g. niching methods). For 6-*bipolar* function, the population is said to have converged when there is over a half of optimal solutions there. For all algorithms, the population size for all problems and all problem sizes was determined empirically as a minimal size so that the algorithms converged to the optimum in all of 30 independent runs. In all runs, the truncation selection with $\tau = 50\%$ was used (the better half of individuals was selected). Offspring replace the worse half of the old population. The crossover and mutation rate for the simple GA were empirically determined for each problem with one problem instance. In the simple GA, the best results were achieved with the probability of crossover 100%. The probability of flipping a single bit by mutation was set to 1%. In BOA, no prior information was incorporated into the algorithm. All networks were treated equally (the κ parameter from Equation 7 was set to 1).

In Figure 5, the results for *OneMax* function are shown. Since for $k = 0$ the BOA uses identical distribution estimate as UMDA and for $k = 1$ and the metric used in BMDA it uses the same distribution estimate as BMDA, the corresponding results were adopted from Pelikan and Mühlenbein (1998). The results for the simple GA with both types of crossover are obtained from the same source. In addition to this, the BOA with $k = 2$ and the metric from BMDA is examined. Since the *OneMax* is linear and therefore there are no interactions among genes, the BOA with $k = 0$ performs the best in terms of the number of functions evaluations until successful convergence. The simple GA with uniform crossover performs slightly worse than BOA with $k = 0$. BOA with $k = 1$ performs similarly as the simple GA with one-point crossover. Both algorithms perform worse than BOA with $k = 0$ and the simple GA with uniform crossover. This is caused by slower mixing. For $k = 2$, the BOA performs worst of all compared algorithms, since it covers

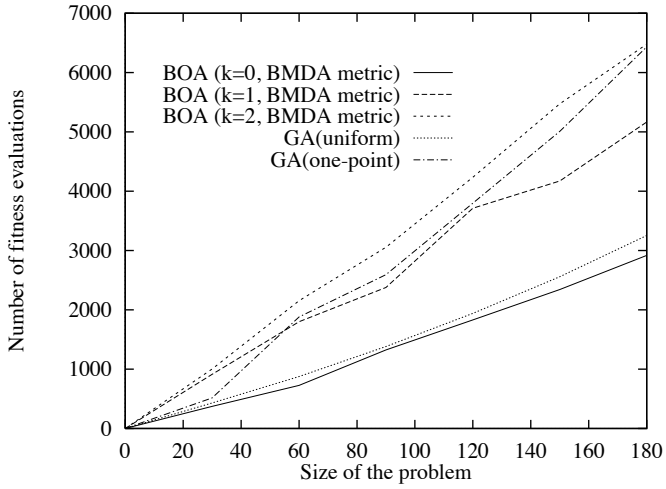


Figure 5: **Results for *OneMax* function.** The population sizes for BOA with $k = 0$ ranged from $N = 50$ for $n = 30$ to $N = 170$ for $n = 180$, for BOA with $k = 1$ it was $N = 120$ for $n = 30$ to $N = 260$ for $n = 180$, and finally for BOA with $k = 2$, it was $N = 200$ for $n = 30$ to $N = 380$ for $n = 180$. The population sizes with the simple GA with uniform crossover ranged from $N = 32$ for $n = 30$ to $N = 160$ for $n = 180$. The population sizes for the simple GA with one-point crossover ranged from $N = 32$ for $n = 30$ to $N = 100$ for $n = 180$.

the interactions of order three and the mixing is slowed down until the algorithm detects the genes are independent. However, the differences between the performance of all compared algorithms are not significant. All algorithms converge in linear time.

In Figure 6, the results for 3-*deceptive* function are presented. In this function, the deceptive building blocks are of order 3. The building blocks are non-overlapping and mapped tightly onto strings. Therefore, one-point crossover is less likely to disrupt them. The looser the building blocks would be, the worse the simple GA would perform. Since the building blocks are deceptive, the computational requirements of the simple GA with uniform crossover and the BOA with $k = 0$ (i.e., the UMDA) grow exponentially and therefore we do not present the results for these algorithms. Some results for BMDA (i.e., the BOA with $k = 1$ and the metric from BMDA) can be found in Pelikan and Mühlenbein (1998). BOA with $k = 2$ and the K2 metric performs the best of the compared algorithms in terms of the number of functions evaluations till successful convergence. It converges to the global optimum in linear time with respect to the size of the problem. It performs just little bit better than the BOA with $k = 2$ and the metric from BMDA. The results of BOA with both types of metric are very similar. In this problem, the important correlations are attended by pairwise interactions. That is why the metric used in BMDA is sufficient. The simple GA with one-point crossover performs worse than the BOA with $k = 2$ as the problem size grows. For loose building blocks, the simple GA with one-point crossover would require the number of fitness evaluations growing exponentially with the size of a problem (Thierens, 1995). On the other hand, the BOA (with any configuration) would perform the same since it is independent of the variable ordering in a string.

In Figure 7, the results for *trap*-5 function are presented. The building blocks are non-overlapping and they are again mapped tightly onto a string. Therefore, the interacting variables are located close to each other in a string and the building blocks are less likely to be disrupted by one-point crossover. The results for this function are similar as the results for 3-*deceptive* function,

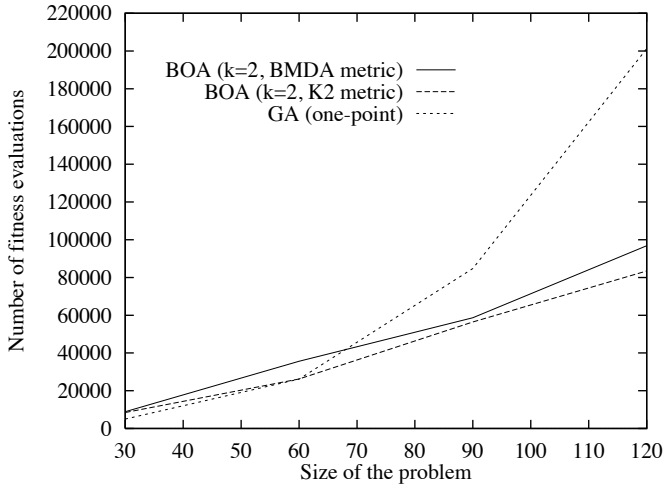


Figure 6: **Results for 3-*deceptive* function.** The population sizes for the simple GA ranged from $N = 200$ for $n = 15$ to $N = 3800$ for $n = 120$. The population sizes for BOA with BMDA metric ranged from $N = 400$ for $n = 30$ to $N = 5500$ for $n = 120$. For BOA with the K2 metric, the population sizes ranged from $N = 900$ for $n = 30$ to $N = 4900$ for $n = 120$.

although since the size of deceptive building blocks is scaled, so are the results. BOA converges to the global optimum in almost linear time with respect to the problem size.

In Figure 8, the results for a 6-*bipolar* function are presented. The BOA algorithm with the metric used in BMDA was not able to discover the interactions well and it performed significantly worse than other compared algorithms—the simple GA with one-point crossover and BOA with $k = 5$ and the K2 metric. This is caused by the fact that with 6-*bipolar* function, the local optima are distributed so that the pairwise interactions are not easy to identify on the level of probabilistic terms of length two. For smaller problems, the simple GA with one-point crossover performs better than the BOA with $k = 5$. As the problem size grows, the BOA outperforms the simple GA. BOA converges to the global optima in time linearly proportional to the problem size. In addition to the faster convergence with BOA for larger problems, the BOA algorithm discovers a number of solutions out of totally $2^{\frac{n}{6}}$ global optima of 6-*bipolar* function instead of converging into a single solution. This effect could be further magnified by using niching methods.

9 Summary and Conclusions

In this paper, the Bayesian optimization algorithm (BOA) was proposed. The proposed algorithm belongs to the class of the estimation of distribution algorithms (EDAs), the algorithms based on the estimation of the distribution of promising solutions and the generation of new candidate solutions according to this estimate. The joint distribution of promising points is estimated by means of techniques from the field of modeling data by Bayesian networks. The algorithm can cover interactions up to a specified order. In BOA, the structure of a problem is being discovered during optimization. The prior information about the problem from various sources can be incorporated into the algorithm although that has not been investigated in this paper. The distribution is estimated in order to match the prior information about the problem as well as the set of promising solutions. However, the prior information is not essential.

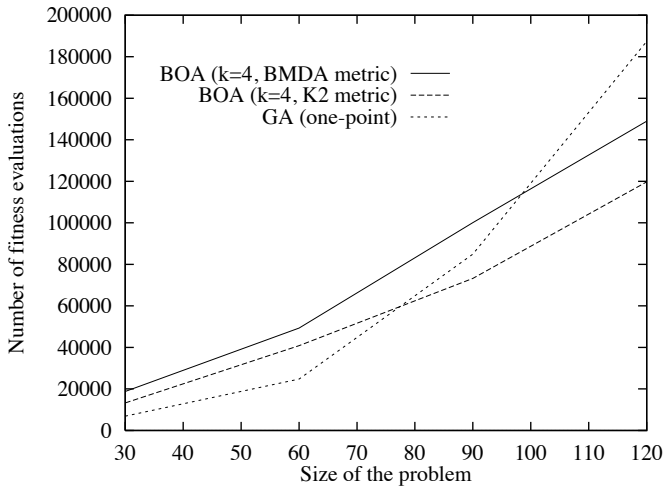


Figure 7: **Results for *trap*-5 function.** The population sizes for the simple GA ranged from $N = 600$ for $n = 30$ to $N = 4400$ for $n = 120$. The population sizes for BOA with BMDA metric ranged from $N = 2100$ for $n = 30$ to $N = 9300$ for $n = 120$. For BOA with the K2 metric, the population sizes ranged from $N = 1300$ for $n = 30$ to $N = 7400$ for $n = 120$.

The experiments have shown that the proposed algorithm outperforms the simple GA even on decomposable problems with tight building blocks as the problem size grows. The gap between the proposed algorithm and the simple GA would significantly enlarge for large problems with loose building blocks. For loose mapping the time requirements of the simple GA grow exponentially with the problem size. On the other hand, the BOA is independent of the ordering of the variables in a string and therefore changing this would not affect the performance of the algorithm. With the K2 metric, the BOA was able to solve efficiently even problems where the pairwise interactions are not transparent, although there are higher order interactions present in the problem (e.g. bipolar deceptive functions). The proposed algorithm works very well also for other problems with highly overlapping building blocks, e.g. spin-glasses, that are not discussed in this paper.

In future work, we will examine the effect of using various sources of prior information in the BOA. In this paper, all networks were treated equally. Combining the proposed techniques for estimating the distribution of promising solutions along with the use of good prior information about the problem interactions could result in a very powerful optimization tool. The information about locally optimal solutions obtained on the run can be incorporated into the metric as well. Moreover, the effects of niching to the performance of the algorithm will be investigated. We will also closely examine the proposed algorithm on other problem domains, including the decomposable functions with overlapping building blocks, fitness functions with noise as well as non-uniformly scaled decomposable functions. These problems can be used as boundary problems of a reasonably large class of problems the proposed stochastic optimization algorithm BOA could be able to solve.

10 Acknowledgements

The authors would like to thank Heinz Mühlenbein, David Heckerman, and Ole J. Mengshoel for valuable discussions and useful comments. Martin Pelikan was supported by grants number 1/4209/97 and 1/5229/98 of the Scientific Grant Agency of Slovak Republic.

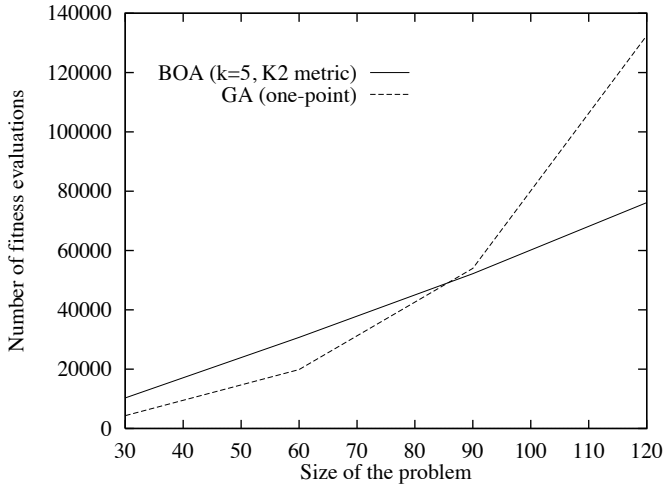


Figure 8: **Results for 6-*bipolar* function.** The population sizes for the simple GA ranged from $N = 360$ for $n = 30$ to $N = 2800$ for $n = 120$. The population sizes for BOA ranged from $N = 900$ for $n = 30$ to $N = 3500$ for $n = 120$.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0050. Research funding for this project was also provided by a grant from the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies and endorsements, either expressed or implied, of the Air Force of Scientific Research or the U.S. Government.

References

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

Baluja, S., & Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proceedings of the 14th International Conference on Machine Learning* (pp. 30–38). Morgan Kaufmann.

Bernardo, J. M., & Smith, A. F. M. (1994). *Bayesian theory.* John Wiley & Sons, Inc.

Buntine, W. L. (1991). Theory refinement of Bayesian networks. In D'Ambrosio, B. D., Smets, P., & Bonissone, P. P. (Eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Seventh Conference* (pp. 52–60). Los Angeles, CA: Morgan Kaufmann Publishers.

Chickering, D. M., Geiger, D., & Heckerman, D. (1994). *Learning Bayesian networks is NP-hard* (Technical Report MSR-TR-94-17). Redmond, WA: Microsoft Research.

Deb, K., & Goldberg, D. E. (1994). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, *10*, 385–408.

Edmonds, J. (1967). Optimum branching. *J. Res. NBS*, *71B*, 233–240.

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

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3*(5), 493–530. (Also TCGA Report 89003).

Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor. Also IlliGAL Report No. 97005.

Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. *Foundations of Genetic Algorithms 4*, 247–262.

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1997). *The compact genetic algorithm* (IlliGAL Report No. 97006). Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Heckerman, D., Geiger, D., & Chickering, M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data* (Technical Report MSR-TR-94-09). Redmond, WA: Microsoft Research.

Henrion, M. (1988). Propagation of uncertainty in Bayesian networks by logic sampling. In Lemmer, J. F., & Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence 2* (pp. 149–163). Amsterdam, London, New York: Elsevier/North-Holland.

Hertz, J. A., Krogh, A. S., & Palmer, R. G. (1991). *Introduction to the theory of neural computation.* Reading, MA: Addison-Wesley.

Howard, R. A., & Matheson, J. E. (1981). Influence diagrams. In Howard, R. A., & Matheson, J. E. (Eds.), *Readings on the Principles and Applications of Decision Analysis*, Volume II (pp. 721–762). Menlo Park, CA: Strategic Decisions Group.

Kargupta, H. (1998). Revisiting the GEMGA: Scalable evolutonary optimization through linkage learning. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation* (pp. 603–608). IEEE Press.

Kvasnicka, V., Pelikan, M., & Pospichal, J. (1996). Hill climbing with learning (An abstraction of genetic algorithm). *Neural Network World*, *6*, 773–796.

Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. *Evolutionary Computation*, *5*(3), 303–346.

Mühlenbein, H., & Mahnig, T. (1998). *Convergence theory and applications of the factorized distribution algorithm.* To be published.

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

Mühlenbein, H., & Rodriguez, A. O. (1998). *Schemata, distributions and graphical models in evolutionary optimization.* Submitted for publication.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* San Mateo, California: Morgan Kaufmann.

Pelikan, M., & Mühlenbein, H. (1998). The bivariate marginal distribution algorithm. London: Springer-Verlag. In printing.

Thierens, D. (1995). *Analysis and design of genetic algorithms.* Leuven, Belgium: Katholieke Universiteit Leuven.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.