Linkage Problem, Distribution Estimation, and Bayesian Networks

Martin Pelikan[*]
Department of Computer Science and
Illinois Genetic Algorithms Laboratory
University of Illinois
Urbana, IL 61801, USA
pelikan@illigal.ge.uiuc.edu

David E. Goldberg
Department of General Engineering
Illinois Genetic Algorithms Laboratory
University of Illinois
Urbana, IL 61801, USA
deg@illigal.ge.uiuc.edu

Erick Cantú-Paz[†]
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
P.O. Box 808, L-551
Livermore, CA 94551, USA
cantupaz@llnl.gov

Abstract

This paper proposes an algorithm that uses an estimation of the joint distribution of promising solutions in order to generate new candidate solutions. The algorithm is settled into the context of genetic and 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 BOA 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, but it is not essential. First experiments were done with additively decomposable problems with both nonoverlapping as well as overlapping building blocks. The proposed algorithm is able to solve all but one of the tested problems in linear or close to linear time with respect to the problem size. Except for the maximal order of interactions to be covered, the algorithm does not use any prior knowledge about the problem. The BOA represents a step toward alleviating the problem of identifying and mixing building blocks correctly to obtain good solutions for problems with very limited domain information.

Keywords

Genetic and evolutionary computation, linkage learning, estimation of distribution algorithm, probabilistic modeling, learning Bayesian networks, genetic algorithm.

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

[†] Formerly with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA.

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 and Paaß, 1996; Mühlenbein, 1997; Harik et al., 1998; Pelikan and Mühlenbein, 1999; Harik, 1999). This line of research in stochastic optimization was strongly motivated by results achieved in 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 quickly, accurately, and reliably. By covering interactions of higher order, the disruption of identified partial solutions is prevented. Prior information from various sources can be used. The combination of information from the set of promising solutions and the prior information about a problem is used to estimate the distribution of the promising solutions. New candidate solutions are generated according to this estimate. The experiments were done with additively decomposable problems with both nonoverlapping as well as overlapping building blocks. The results indicate that the proposed algorithm is able to solve most of the 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. Section 3 describes the algorithms based on the estimation of distributions and draws the connection between the methods based on the estimation of distributions and genetic algorithms. In Section 4, the Bayesian Optimization Algorithm (BOA) is introduced. Section 5 provides basic information on Bayesian networks. Section 6 discusses additively decomposable functions and the difficulties with solving this class of problems using genetic algorithms. The results of experiments are presented in Section 7. Section 8 discusses the convergence theory that can be applied to the proposed algorithm and Section 9 addresses future work. The paper is summarized and concluded in Section 10.

2 Background

Genetic algorithms (GAs) are optimization methods loosely based on the mechanics of artificial selection and genetic recombination operators. By reproducing and combining promising solutions, high-quality partial solutions are combined in order to form new solutions. High-quality partial solutions are often called *building blocks* (BBs) (Holland, 1975; Goldberg, 1989). The genetic algorithm implicitly manipulates a large number of building blocks by mechanisms of selection and recombination. 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 proved to be insufficiently powerful, even for problems that are composed of simpler partial subproblems. General, fixed, problem-independent recombination operators often break partial solutions that can sometimes lead to losing these partial solutions and converging to a local optimum. Two crucial factors of the GA's success, a proper growth and mixing of good building blocks, are often not achieved (Thierens and Goldberg, 1993). The problem of building block disruption is often referred to as the *linkage problem* (Harik and Goldberg, 1996). Various attempts to prevent the disruption of important partial solutions have been made 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 and Paaß, 1996; Mühlenbein, 1997; Harik et al., 1998; Pelikan and Mühlenbein, 1999). In this paper, we will focus on the second class of methods.

2.1 Evolving Representation or Operators

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 to ensure a proper mixing of partial solutions before these are lost.

This line of research has resulted in algorithms that evolve the representation of a problem along with the individual solutions. In the Messy Genetic Algorithm (mGA) (Goldberg et al., 1989) and its more efficient descendant called the Fast Messy Genetic Algorithm (fmGA) (Kargupta, 1995), 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 the variables with strong interactions so that recombination is less likely to disrupt them. The LLGA works very well for exponentially scaled decomposable problems, but it is not very efficient on problems with uniformly scaled building blocks.

2.2 Probabilistic Modeling of Promising Solutions

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.

Global information about the set of promising solutions can be used to estimate their distribution and this estimate can be used to generate new individuals. A general scheme of the algorithms based on this principle is called the Estimation of Distribution Algorithm (EDA) (Mühlenbein and 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 next subsections describe basic principles of recently proposed algorithms that use probabilistic models of promising solutions to guide the further search. For a more detailed overview, see Pelikan et al. (1999).

2.2.1 No Interactions

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 preserving only the proportions of the values of all variables independently of the remaining solutions. This is the basic principle of the Population Based Incremental Learning Algorithm (PBIL) (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 gradually shifts the vector towards the best of generated candidate solutions. The Compact Genetic Algorithm (cGA) (Harik et al., 1998) 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., 1998). The Univariate Marginal Distribution Algorithm (UMDA) (Mühlenbein, 1997) uses the same distribution estimate, although it works with populations of solutions instead of a vector representing these. There is theoretical evidence that the UMDA also approximates the behavior of the simple GA with uniform crossover (Mühlenbein, 1997). The theory of the UMDA is based on the techniques of quantitative genetics, and it can be found in Mühlenbein (1997). Some analyses of PBIL can be found in Kvasnicka et al. (1996).

The PBIL, cGA, and the UMDA algorithms work very well for problems with no significant interactions among variables (Mühlenbein, 1997; Harik et al., 1998; Pelikan and Mühlenbein, 1999). However, the partial solutions of order more than one are disrupted, and therefore these algorithms experience great difficulty solving problems with interactions among the variables.

2.2.2 Pairwise Interactions

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 and Davies, 1997)) and the population-based Mutual-Information-Maximizing Input Clustering Algorithm (MIMIC) using a simple chain distribution with the same purpose (De Bonet et al., 1997). 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 and Mühlenbein, 1999). 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 and Mühlenbein, 1999). 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.

2.2.3 Multivariate Interactions

In the Factorized Distribution Algorithm (FDA) (Mühlenbein et al., 1998), a fixed factorization of the distribution is used in order to generate new candidate solutions. The FDA is capable of covering the interactions of higher order and combining important partial solutions effectively. It works very well on uniformly-scaled additively decomposable problems. The theory of the UMDA can be used in order to estimate the time to convergence in the FDA. However, the FDA requires the prior information about a problem in the form of a problem decomposition and its factorization. As input, this algorithm gets a complete or approximate information about the structure of a problem. By providing sufficient conditions for the distribution estimate that ensure fast and reliable convergence on decomposable problems, the FDA is of great theoretical value. Moreover, for problems where 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. Using an approximate distribution according to the current state of information represented by the set of promising solutions can be very effective even if it is not a valid distribution factorization.

In Harik's Extended Compact Genetic Algorithm (ECGA) (Harik, 1999), the variables are grouped into disjoint sets so that by using the marginal distribution according to these sets the population of selected solutions can be compressed as much as possible. As a measure for quality of marginal distributions, the algorithm uses a combination of the model complexity and compressed population complexity. The used measure corresponds to a minimum description length approach in machine learning. To construct the desired distribution, a simple greedy algorithm is used. The sets can be of any size and therefore the ECGA can cover interactions of any order. The ECGA does not require the maximal number of interactions that can be covered as input. However, by using marginal distributions without any conditional probabilities, only problems with nonoverlapping building blocks can be modeled accurately. The algorithm is not able to solve problems with highly overlapping building blocks as spin-glasses without enormous computational effort.

The algorithm proposed in this paper is also capable of covering higher order interactions. It uses techniques from 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 proposed algorithm does not require any problem-specific knowledge in the initial stage. It is able to learn the model on the fly. Because of using a measure of quality of the networks that does not penalize more complex networks, only networks of bounded complexity have been used in our experiments. The models were bounded by a maximal order of probabilistic terms in the distribution estimate. We are currently investigating the effect of using other metrics to discriminate networks according to their accuracy as well as complexity.

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

In this section, we first describe a class of stochastic optimization algorithms based on the estimation of distributions. Thereafter, a brief overview of the recent algorithms based on

Table 1: A brief overview of EDAs.

Algorithm	Capabilities	Difficulties
PBIL, cGA, UMDA	Efficient on linear problems.	Higher order BBs.
MIMIC, Baluja '97, BMDA	Efficient with BBs of order 2.	Higher order BBs.
FDA	Efficient on decomp. prob.	Prior inf. is essential.
ECGA	Efficient on separable prob.	Highly overlapping BBs

this principle will be presented.

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 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. The two main questions to consider when designing an EDA are

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

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. 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.

A brief overview of the capabilities and difficulties of the discussed EDAs is provided in Table 1.

4 The Bayesian Optimization Algorithm (BOA)

This section introduces an EDA that uses techniques from modeling data by Bayesian networks to estimate the joint distribution of promising solutions. It is called the Bayesian Optimization Algorithm (BOA). It covers both the UMDA and BMDA and extends them to cover the interactions of higher order. The BOA is designed to solve problems that can be decomposed into subproblems of bounded order.

The BOA uses the identical class of distributions as the FDA. However, unlike the FDA, the algorithm does not require a valid distribution factorization as input. It is able to learn the distribution on the fly without the use of any problem-specific information. Moreover, information about the problem can be incorporated. Prior information about the structure of a problem, as well as the information represented by the set of high-quality solutions, can be used. The ratio between the amount of prior information and the information acquired by the algorithm during the run can be controlled. The proposed algorithm fills the gap between the fully-informed FDA and totally uninformed black-box optimization methods.

```
The Bayesian Optimization Algorithm (BOA)

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

Figure 1: The pseudocode of the Bayesian optimization algorithm.

In the 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 1.

Constructing the network in the BOA corresponds to estimating the distribution in EDAs. Generating the set of new strings according to the distribution encoded by the constructed network is identical to the corresponding step in EDAs. The following section presents details on the algorithms for Bayesian network construction and its use for generation of new instances (steps (3) and (4) in the pseudocode of the BOA).

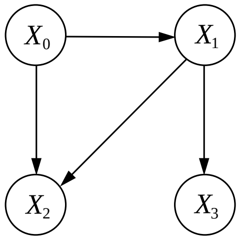
5 Bayesian Networks Basics

This section describes basic techniques for learning Bayesian networks that have been used in the experiments performed with the BOA in this paper and gives the pointers to other works that discuss the topic in a more detailed way. For a more complete overview and recent advances in modeling data by Bayesian networks, consult the cited papers.

The section starts with describing the structure of Bayesian networks and providing a simple example of a Bayesian network and its semantics. Thereafter, some of the techniques for learning Bayesian networks and the use of the constructed network for generating new instances of modeled data are described.

5.1 General Description

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



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

Figure 2: A simple example of a Bayesian network and the encoded joint distribution.

the probabilistic relationships 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. Both the variable and the node corresponding to this variable will be denoted in this text by X_i. 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

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

where $X = (X_0, \ldots, X_{n-1})$ is a vector of all the variables in the problem, Π_{X_i} is the set of parents of X_i in the network (the set of nodes from which there exists an edge to X_i), and $p(X_i|\Pi_{X_i})$ is the conditional probability of X_i conditioned on the variables Π_{X_i}. A directed edge relates the variables so that in the encoded distribution, the variable corresponding to its ending node will be conditioned on the variable corresponding to its starting node. More incoming edges into a node result in a conditional probability of the corresponding variable with conjunctional condition containing all its parents. A simple example of a Bayesian network and the joint distribution encoded by this network can be found in Figure 2.

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 can be used to generate new candidate solutions throughout whole optimization. If the network is unknown, i.e., we don't know which variables are correlated 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 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 according to the joint distribution encoded by this network.

5.2 Learning the Network Structure

There are two basic components of the algorithms for learning the network structure (Heckerman et al., 1994). The first is a scoring metric and the second 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 highest possible value of the scoring metric. The number of considered networks can be reduced by imposing constraints on the network structure. 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 and its use for generation of new instances. It corresponds to a maximal order of probabilistic terms that can be used in a considered class of distributions or to a maximal order of interactions that can be covered. However, it is not an easy task to determine the value of k that would be sufficient to model given data well and yet be as small as possible to make the construction of a network and generation of new solutions more efficient. Nevertheless, by preferring simpler networks, the use of this constraint can be evaded without a significant increase in a model complexity. We are currently investigating various ways to eliminate a fixed k without increasing time-to-convergence on decomposable problems.

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.

5.2.1 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.

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

$$p(D,B|\xi) = p(B|\xi) \prod_{i=0}^{n-1} \prod_{\pi_{X_i}} \frac{\Gamma\left(m'(\pi_{X_i})\right)}{\Gamma\left(m'(\pi_{X_i}) + m(\pi_{X_i})\right)} \prod_{x_i} \frac{\Gamma\left(m'(x_i,\pi_{X_i}) + m(x_i,\pi_{X_i})\right)}{\Gamma\left(m'(x_i,\pi_{X_i})\right)}, \tag{2}$$

where $p(B|\xi)$ is the prior probability of the network B, Γ function is defined as $\Gamma(a) = (a-1)!$, 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. $m(\pi_{X_i})$ denotes the number of instances in D with variables Π_{X_i} (the parents of X_i) instantiated to π_{X_i}. 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. We denote by $m(x_i,\pi_{X_i})$ 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 for all instances π_{X_i} of the parents of X_i,

$$m(\pi_{X_i}) = \sum_{x_i} m(x_i, \pi_{X_i}), \tag{3}$$

where 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 3 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:

$$p(B|\xi) = c\kappa^{\delta}, \tag{4}$$

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 symmetric difference between B and the prior network. The symmetric difference between two networks is the number of edges where the networks differ. The bias of the metric to the prior network can be controlled by parameter κ. 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. In order to favor networks of lower complexity by the BD metric, an empty prior network can be used (Heckerman et al., 1994), or the prior probability of the network can be set to two to the power of the defining length of the network (i.e., a maximal number that can be encoded using a number of bits needed to encode the network (see Friedman and Goldszmidt (1999)).

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 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. Other 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 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, any of the mentioned assignments of priors can be used. In the empirical part of this paper, we will use the K2 metric, and all networks will be treated equally. The reason for this is that before investigating the numerous possibilities arising with various sources of prior information, a general principle of the algorithm should be tested.

A simple example of a data set and the obtained score with the K2 metric and $p(B|\xi) = 1$ for all networks B (all networks are treated equally) 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 3, the $m'(\pi_{X_i})$ are set to 2. The table with values of the terms from Equation 2 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 3. Thus,

$$\begin{aligned} p(B_{empty}, D|\xi) &= \frac{(2-1)!}{(2+4-1)!} \cdot \frac{(1+3-1)!}{(1-1)!} \cdot \frac{(1+1-1)!}{(1-1)!} \cdot \frac{(2-1)!}{(2+4-1)!} \cdot \\ &\cdot \frac{(1+3-1)!}{(1-1)!} \cdot \frac{(1+1-1)!}{(1-1)!} = \frac{1}{400}. \end{aligned}$$

For a network $B_{0\to 1}$ 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 2 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 computed again using Equation 3. Thus,

$$\begin{aligned} p(B_{0\to 1}, D|\xi) &= \frac{(2-1)!}{(2+4-1)!} \cdot \frac{(1+3-1)!}{(1-1)!} \cdot \frac{(1+1-1)!}{(1-1)!} \cdot \frac{(2-1)!}{(2+3-1)!} \cdot \frac{(1+3-1)!}{(1-1)!} \cdot \\ &\cdot \frac{(1+0-1)!}{(1-1)!} \cdot \frac{(2-1)!}{(2+1-1)!} \cdot \frac{(1+1-1)!}{(1-1)!} \cdot \frac{(1-1)!}{(1-1)!} = \frac{1}{160}. \end{aligned}$$

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, the BDe metric is consistent with the likelihood equivalence assumption.

Since the factorials in Equation 2 can grow to huge numbers, usually a logarithm of the scoring metric is used. Using simple logarithmic rules we get

$$\log\left(\frac{(m'(\pi_{X_i})-1)!}{(m'(\pi_{X_i})+m(\pi_{X_i})-1)!}\right) = -\sum_{s=m'(\pi_{X_i})}^{m'(\pi_{X_i})+m(\pi_{X_i})-1} \log s, \tag{5}$$

and, similarly,

$$\log\left(\frac{(m'(x_i,\pi_{X_i})+m(x_i,\pi_{X_i})-1)!}{(m'(x_i,\pi_{X_i})-1)!}\right) = \sum_{t=m'(x_i,\pi_{X_i})}^{m'(x_i,\pi_{X_i})+m(x_i,\pi_{X_i})-1} \log t. \tag{6}$$

Thus,

$$\begin{aligned}\log\left(p(D,B|\xi)\right) = \log\left(p(B|\xi)\right) + \sum_{i=0}^{n-1}\sum_{\pi_{X_i}} \Bigg(& -\sum_{s=m'(\pi_{X_i})}^{m'(\pi_{X_i})+m(\pi_{X_i})-1} \log s + \\ & + \sum_{x_i} \sum_{t=m'(x_i,\pi_{X_i})}^{m'(x_i,\pi_{X_i})+m(x_i,\pi_{X_i})-1} \log t \Bigg).\end{aligned} \tag{7}$$

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 $O(kN + 2^k)$ steps, 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 5 and 6 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 then be computed in $O(2^k N)$ steps. To compute an increase of the logarithm of the value of the BD metric for an edge addition, edge reversal, or edge removal, we need $O(2^k N)$ steps, because the contribution corresponding to a particular node changes only when its set of parents is changed. 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 of the logarithm of the metric for an edge addition, edge removal, or edge reversal.

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 and Mühlenbein, 1999). Only the pairs that are not independent at 5% confidence level with respect to this statistic are considered. The metric used in the 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 order two (see Example 5.2). The BD metric is able to give preference to networks that cover correlations of a higher order, too. Experiments show that the BOA can even efficiently solve problems where higher order interactions are present, although the variables are not statistically strongly correlated at the level of probabilistic terms of length.

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. □

5.2.2 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 a restricted number of incoming edges denoted by k will be considered. We consider the following 3 cases:

a) $\boldsymbol{k = 0}$ This case is trivial. An empty network is the best one (and the only one possible).

b) $\boldsymbol{k = 1}$ 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 $O(n^2N)$ 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). Finding the best network with $k = 1$ simplifies to a special case of the maximal branching problem. To solve this problem, there exists a polynomial algorithm (Edmonds, 1967).

c) $\boldsymbol{k > 1}$ For $k > 1$ the problem becomes 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, edge reversal, and edge removal.

In the empirical part of this paper, we have used a simple greedy algorithm with only edge additions allowed. Starting with an empty network, improvement of the network for all edge additions is measured by an increase of the metric value. Only edge additions that keep the network acyclic and do not violate network complexity constraints are allowed. If no legal additions improve the network, the algorithm finishes and returns the current network. Otherwise, the addition that most improves the network score is performed, and the process is repeated until termination criteria are finally met.

Edge reversals and removals can be allowed in the greedy search as well. However, our experience suggests that introducing these operators does not significantly improve learning. The pseudocode of a greedy algorithm for constructing the Bayesian network is shown in Figure 3.

```
A greedy algorithm for the network construction
(1) initialize the network B (e.g., to an empty network)
(2) choose all simple graph operations that can be performed on the network without
    violating the constraints
(3) pick the operation that increases the score of the network the most
(4) perform the operation picked in the previous step
(5) if the network can no longer be improved under given constraints on its complexity
    or a maximal number of interactions has been reached, finish
(6) go to 2
```

Figure 3: The pseudocode of the greedy algorithm for the network construction.

The time complexity of the greedy learning algorithm we used in the experiments presented in this paper can be computed using the time complexity of a simple edge addition and the greatest number of edges to be processed. 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 $O(n^2N)$ 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 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 with the BD metric is then $O(k2^k n^2 N + kn^3)$. Assuming that k is constant, we get the overall time complexity $O(n^2N + n^3)$.

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 the maximal incoming edges constraint. Allowing other operations (e.g., an edge reversal or removal) would make the analyses more complicated.

The algorithm for constructing the networks does not depend on how the networks are discriminated by the used metric. It simply uses a given metric to guide its search. The BOA does not restrict the networks to be learned by a greedy algorithm. More robust techniques (e.g., simulated annealing, evolutionary programming, etc.) can be used.

5.3 Generating New Instances

In this section, the generation of new instances using a network B and the marginal frequencies for a few sets of variables in the modeled data set will be described. This corresponds

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.

to step (4) in the pseudocode of the BOA (see Figure 1). The network B encodes a joint probability distribution given by Equation 1. The conditional probabilities present in this distribution are given by

$$p(X_i|\Pi_{X_i}) = \frac{p(X_i, \Pi_{X_i})}{p(\Pi_{X_i})} \tag{8}$$

for all $i \in \{0, \dots, n-1\}$.

Since the network B is acyclic, it is easy to generate a new instance. First, the conditional probabilities of each possible instance of each variable, given all instances of its parents in a given data set, are computed. The conditional probabilities are used to generate each new instance. Each iteration, the values of the variables 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 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.

The overall complexity of constructing the network and generating the new instances according to this network is therefore $O(k2^kn^2N + kn^3)$, and assuming that k is constant, we get the overall time complexity $O(n^2N+n^3)$. This process is performed each generation of the BOA. Using a different metric or a search procedure for the network construction, the time complexity analysis would also change.

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 great difficulty solving some decomposable problems. This topic is addressed in this section as well. However, to precisely define what is difficult for the GAs as well as other methods is not an easy task.

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 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:

$$f(X) = \sum_{i=0}^{l-1} f_i(S_i), \qquad (9)$$

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 that can be decomposed by using only nonoverlapping 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 9 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 nonoverlapping, each subfunction can be considered independently. The optimum can be obtained by simply optimizing each of the subfunctions f_i by enumeration. With binary variables, 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 fully deceptive subfunctions (Deb and 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 tend 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 and Goldberg, 1993). The relation between the number of building blocks and the population size remains qualitatively the same. However, 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 and Goldberg, 1993).

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. In general, we cannot consider each subfunction independently, optimize it, and get the global optimum by combining the partial solutions together as with nonoverlapping 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 considered independently. A possible solution to this problem is the FDA (Mühlenbein and Mahnig, 1998), which is able to combine important building blocks together even if they are overlapping. However, the FDA needs as input an exact or an approximate factorization of a problem, and without this information it is not applicable.

However, by scaling function values corresponding to the overlapping sets of variables according to some function of a problem size, deceptive building blocks of order growing with the size of a problem can be created. For such problems, the exponential complexity of solving these problems cannot be avoided by any of the mentioned algorithms based on the concepts of reproducing and mixing promising building blocks according to a problem decomposition. An example of such function was presented by Mühlenbein et al. (1998) (here, see Equation 23 in Section 7.2), where the FDA is applied to solve the problem. According to the presented results, the number of fitness evaluations until successful convergence grows faster than exponentially, even with the FDA, which uses an exact problem factorization (requiring complete information about problem decomposition) to solve this problem. This implies that even a valid problem factorization does not guarantee that the corresponding decomposable problem can be solved in polynomial time. The reasons why this is the case are presented in Section 7.2. The problem can only be avoided by generating the initial population according to a distribution factorization instead of using a uniform distribution. However, given no prior information about the problem, this is impossible. The connection between the order of problem decomposition and the order of building blocks that need to be considered is far from trivial. Only for separable problems, where the set of all the variables in a problem is decomposed into nonoverlapping subsets of variables, is the order of building blocks that need to be considered equal to the order of problem decomposition.

7 Experiments

The experiments were done for decomposable problems composed of functions of unitation and a highly overlapping two-dimensional Ising spin-glass problem instance. For all problems except the Ising spin-glass, the scalability of the proposed algorithm was shown. The following sections describe the optimized functions and present results of the experiments.

7.1 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.

A simple $OneMax$ function of order 1 is defined for a single bit as its value, i.e.,

$$f^1_{onemax}(X) = X, \tag{10}$$

where X is a binary variable.

A deceptive function of order 3 is defined as

$$f^3_{deceptive}(X) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise}, \end{cases} \tag{11}$$

where X is a vector of 3 binary variables, and u is the sum of the input variables.

A trap function of order 5 is defined as

$$f^5_{trap}(X) = \begin{cases} 4 - u & \text{if } u < 5 \\ 5 & \text{otherwise}, \end{cases} \tag{12}$$

where X is a vector of 5 binary variables, and u is the sum of the input variables.

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

$$f^6_{bipolar}(X) = f^3_{deceptive}(|3 - u|), \tag{13}$$

where X is a vector of 6 binary variables, and u is the sum of the input variables.

Following, functions of order 3 will be used in the 0-*peak* function. The first function will be denoted by f_1. It is defined as

$$f^3_1(X, l) = \begin{cases} l & \text{if } u = 0 \\ l - 1 & \text{if } u = 3 \\ 0 & \text{otherwise}, \end{cases} \tag{14}$$

where X is a vector of 3 binary variables, u is the sum of the input variables, and l is an integer. The second function used in the 0-*peak* will be denoted by f_2, and it is defined as

$$f^3_2(X, l) = \begin{cases} l & \text{if } u = 3 \\ 0 & \text{otherwise}, \end{cases} \tag{15}$$

where X is a vector of 3 variables, u is the sum of the input variables, and l is an integer.

7.2 Test Functions

In this section, the functions used in experiments will be described. All functions except the 2-dimensional Ising spin-glass function are constructed with functions of unitation presented in the previous section.

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

$$f(X) = \sum_{i=0}^{l-1} f_k(S_i), \tag{16}$$

where X is the set of n variables, and S_i for $i \in \{0, \dots, l-1\}$ are subsets of k variables from X. Sets S_i can be either overlapping or nonoverlapping, 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 used to construct the function. Each variable will be required to contribute to the function through some of the subfunction, i.e.,

$$X = \bigcup_{i=0}^{l-1} S_i \tag{17}$$

The $OneMax$ function returns the number of ones in an input string, i.e.,

$$f_{onemax}(X) = \sum_{i=0}^{n-1} f^1_{onemax}(X_i), \tag{18}$$

where $X = (X_0, \dots, X_{n-1})$ is a vector of variables. $OneMax$ is a unimodal function with optimum in $X_{opt} = (1, \dots, 1)$. There are no interactions in a problem, and therefore $k = 0$ is sufficient for the BOA in order to find the distribution that guarantees fast and reliable convergence.

A deceptive function composed of nonoverlapping deceptive functions of order 3 (see Equation 11) will be referred to as 3-*deceptive* **without** overlapping. It is defined as

$$f_{3deceptive}(X) = \sum_{i=0}^{n/3-1} f^3_{deceptive}(S_i), \tag{19}$$

where $X = (X_0, \dots, X_{n-1})$ is a vector of variables, and $S_i = (X_{3i}, X_{3i+1}, X_{3i+2})$. This function has one global optimum in $X_{opt} = (1, 1, \dots, 1)$. For this problem, probabilistic terms of length 3 are needed in order to cover all interactions. Therefore, $k = 2$ is sufficient.

A deceptive function composed of deceptive functions of order 3 that are overlapping in one variable in a chain-like structure will be referred to as 3-*deceptive* **with** overlapping. It is defined as follows:

$$f_{3dec-overlap}(X) = \sum_{i=0}^{(n-3)/2} f^3_{deceptive}(S_i), \tag{20}$$

where $X = (X_0, \dots, X_{n-1})$ is a vector of variables, and $S_i = (X_{2i}, X_{2i+1}, X_{2i+2})$. This function has one global optimum in $X_{opt} = (1, 1, \dots, 1)$. Analogically to the previous problem, $k = 2$ is sufficient in order to model the selected strings well.

A trap function composed of nonoverlapping trap functions of order 5 will be referred to as *trap*-5. It is defined as

$$f_{trap5}(X) = \sum_{i=0}^{n/5-1} f^5_{trap}(S_i). \tag{21}$$

$X = (X_0, \dots, X_{n-1})$ and $S_i = (X_{5i}, X_{5i+1}, X_{5i+2}, X_{5i+3}, X_{5i+4})$, where X is a vector of variables. This function has one global optimum in $X_{opt} = (1, \dots, 1)$. Probabilistic terms of length 5 are needed to cover the interactions in this problem. Therefore, $k = 4$ is sufficient.

A bipolar function composed of nonoverlapping bipolar functions of order 6 will be referred to as 6-*bipolar*. It is defined as

$$f_{6bipolar}(X) = \sum_{i=0}^{n/6-1} f^{6}_{bipolar}(S_i). \tag{22}$$

$X = (X_0, \dots, X_{n-1})$ and $S_i = (X_{6i}, X_{6i+1}, X_{6i+2}, X_{6i+3}, X_{6i+4}, X_{6i+5})$, where X is a vector of variables. This function has $2^{\frac{n}{6}}$ global optima in all strings that have blocks of bits corresponding to each set S_i set to either $(0, \dots, 0)$ or $(1, \dots, 1)$. The 6-*bipolar* function is highly multimodal. It has $\binom{6}{3}^{\frac{n}{6}}$ local optima besides $2^{\frac{n}{6}}$ global ones. Therefore, for a problem size of $n = 120$, there are only something more than 10^6 global optima, but more than 10^{26} places to get stuck. To cover the interactions in this problem, the probabilistic terms of order 6 are required. Therefore, $k = 5$ is sufficient.

A 0-*peak* function composed of functions f^3_1 and f^3_2 is defined as

$$f_{0-peak}(X) = \sum_{i=0}^{l-2} f^3_1(S_i) + f^3_2(S_{l-1}), \tag{23}$$

where $X = (X_0, \dots, X_{n-1})$ is a vector of variables, $S_i = (X_{2i}, X_{2i+1}, X_{2i+2})$, and $l = (n-1)/2$. This function is very difficult to optimize. It has one global optimum in $X_{opt} = (1, \dots, 1)$ with function value $f_{0-peak}(X_{opt}) = l(l-1) + 1$. The second best optimum is in $X_{2nd} = (0, \dots, 0)$, i.e., in exactly opposite point, although the function value is only 1 less than the optimal function value, i.e., $f_{0-peak}(X_{2nd}) = l(l-1)$. All schemata of order $(n-3)$ with "don't care" positions in one of the sets from the function decomposition are deceiving the algorithm from the global optima into the local one for $l > 21$. For instance, the average fitness of schema ***000...0 is higher than the average fitness of ***111...1. The same condition is satisfied for schemata of lower order. This closely resembles the definition of fully deceptive functions. Moreover, the larger the problem, the stronger the deception. Using an algorithm that reproduces and mixes the building blocks of order 3 according to the function decomposition, wrong schemata 000 are reproduced and mixed for all but the last building block in the chromosome. In fact, the building blocks in this problem are not really of order 3 as would intuitively follow from the problem definition, but they are growing linearly with the problem size. Since they are also deceptive, the algorithms that take into account only blocks of bits of order 3 are deceived for the same reasons as the UMDA would be deceived for deceptive building blocks growing with the problem size. These are the main reasons why 0-*peak* is very difficult to optimize and the algorithms based on using only partial information from a relatively small and inaccurate set of promising solutions require exponential time to find a global optimum. Mostly, the algorithms are misled, and only after finding the optimum by chance do they further reproduce it.

For the above reasons, although the problem decomposition looks similar to that of the 3-*deceptive* function with overlapping, to solve this problem, a processing of much higher order building blocks is necessary. The order of building blocks that need to be taken into account is growing linearly with the problem size. To ensure the building block supply in the initial population, the required population sizes need to grow exponentially with the problem size. However, we test this problem for $k = 2$, therefore, covering interactions of order 3 at maximum. The results confirm the above analysis.

An Ising spin-glass function is defined as

$$f_{Ising}(X) = \sum_{i=0}^{n-1} \sum_{j \in N(i)} Y_i J_{i,j} Y_j, \tag{24}$$

where $X = (X_0, \ldots, X_{n-1})$ is a binary vector, $Y = (Y_0, \ldots, Y_{n-1})$ is transformation of the vector X such that $Y_i = -1$ for $X_i = 0$ and $Y_i = +1$ for $X_i = 1$, $J = (J_{i,j})_{i,j=0}^{n^2-1}$ is a fixed matrix of coefficients from $\{-1, 1\}$, and the sum over j runs over the neighbors of i. In our 2-dimensional problem, the neighbors of the variable X_i were defined as the adjacent variables when the input vector is sequentially filled in a 2-D grid of the size $\sqrt{n} \times \sqrt{n}$. In a general Ising spin-glass instance, coefficients $J_{i,j}$ are real numbers from $[-1, 1]$. However, in our experiments, we have used a simple instance with discrete coefficients since, for this problem, there exists a polynomial algorithm for finding the optimum (Toulouse, 1977), and therefore the results can be easily verified. Since the Ising spin-glass problem is a minimization problem, we will first transform the coefficient to exactly opposite values, i.e., $J_{i,j} := -J_{i,j}$, so that the problem transforms to maximization. In order to accurately model the problem, probability terms growing with a square root of problem size would be required. However, allowing each node in the grid to interact with two neighbors is sufficient for the algorithm to converge efficiently. Therefore, in our experiments, we set $k = 2$.

7.3 Results of the 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 the 6-*bipolar* and 2-D Ising spin-glass functions, 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, 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 the BOA, no prior information except for a maximal order of covered interactions was incorporated into the algorithm. All networks were treated equally (the κ parameter from Equation 4 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 the UMDA, the corresponding results were adopted from Pelikan and Mühlenbein (1999). The results for the simple GA with both types of crossover were obtained from the same source. 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 function evaluations until successful convergence. The simple GA with uniform crossover performs slightly worse than the BOA with $k = 0$. The simple GA with one-point crossover performs similarly to the BOA with $k = 1$. Both the GA with one-point crossover and the BOA with $k = 1$ perform worse than the BOA with $k = 0$

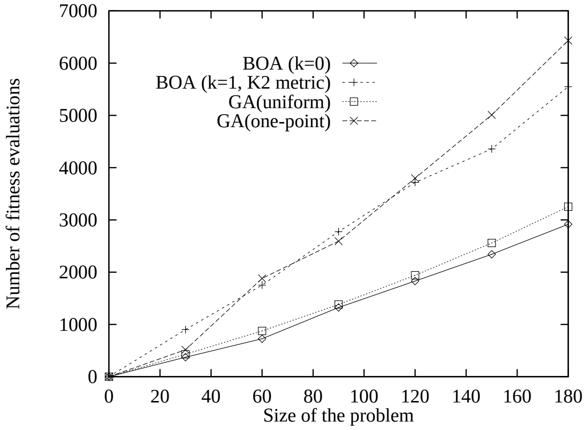


Figure 5: Results for $OneMax$.

and the simple GA with uniform crossover. This is caused by slower mixing. However, the differences between the performance of all compared algorithms are not significant. All algorithms appear to converge in linear time. The population sizes for the BOA ranged from $N = 50$ for $n = 30$ to $N = 170$ for $n = 180$ with $k = 0$ and from $N = 120$ for $n = 30$ to $N = 330$ for $n = 180$ for $k = 1$. 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 $n = 180$.

In Figure 6, the results for 3-$deceptive$ function without overlapping are presented. In this function, the deceptive building blocks are of order 3. The building blocks are nonoverlapping 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. Eventually, for the building blocks randomly spread throughout the solution strings, the simple GA with one-point crossover would require exponential time. 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 can be found in Pelikan and Mühlenbein (1999). The BOA with $k = 2$ and the K2 metric performs the best of the compared algorithms in terms of the number of function evaluations until successful convergence. It converges to the global optimum in linear time with respect to the size of the problem. 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. The population sizes for the simple GA ranged from $N = 400$ for $n = 30$ to $N = 7700$ for $n = 180$. For the BOA, the population sizes ranged from $N = 1000$ for $n = 30$ to $N = 7700$ for $n = 180$.

In Figure 7, the results for 3-$deceptive$ function with overlapping building blocks are presented. The building blocks are tightly mapped to strings representing the solutions.

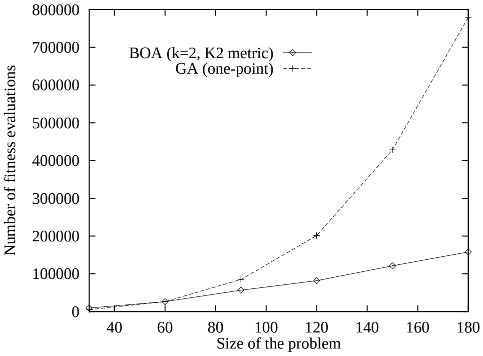


Figure 6: Results for 3-*deceptive* without overlapping.

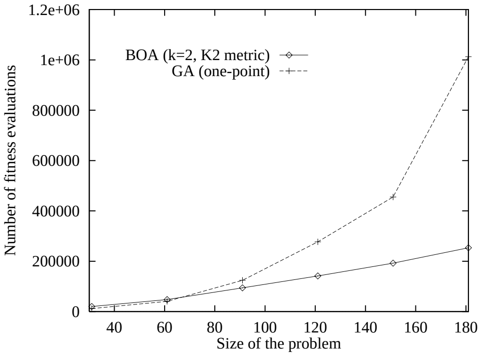


Figure 7: Results for 3-*deceptive* with overlapping.

Again, as with the same function with nonoverlapping building blocks, the BOA algorithm converges to the global optimum in close to linear time, while as the problem size grows, the simple GA requires qualitatively more computational time with respect to the number of fitness evaluations. Since the number of building blocks in the 3-*deceptive* function with overlapping grows 50% faster than in the same function without overlapping, the results also suggest that, with respect to the number of building blocks in a problem, the BOA is able to solve some of the problems with overlapping building blocks without scaling as fast as it can solve similar problems without overlapping. The population sizes for the simple GA ranged from $N = 900$ for $n = 31$ to $N = 11900$ for $n = 181$. For the BOA, the population sizes ranged from $N = 2100$ for $n = 31$ to $N = 12300$ for $n = 181$.

In Figure 8, the results for the *trap*-5 function with nonoverlapping building blocks are presented. The building blocks are nonoverlapping 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 the 3-*deceptive* function without overlapping. The BOA converges to the global optimum in almost linear time with respect to the problem size. The population sizes for the simple GA ranged from $N = 600$ for $n = 30$ to $N = 8100$ for $n = 180$. The population sizes for the BOA with the K2 metric ranged from $N = 1300$ for $n = 30$ to $N = 11800$ for $n = 180$.

In Figure 9, the results for a 6-*bipolar* function are presented. 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. The BOA converges to the global optima in time linearly proportional to the problem size. In addition to the faster convergence with the BOA for larger problems, the BOA algorithm discovers a number of solutions out of totally $2^{\frac{n}{6}}$ global optima of the 6-*bipolar* function instead of converging into a single solution. This effect could be further magnified by using niching methods. The population sizes for the simple GA ranged from $N = 360$ for $n = 30$ to $N = 4800$ for $n = 180$. The population sizes for the BOA ranged from $N = 900$ for $n = 30$ to $N = 5000$ for $n = 180$.

On the $0 - peak$ function, the algorithm performs significantly worse than on the remaining problems. This is caused by its deceptiveness of order growing linearly with the

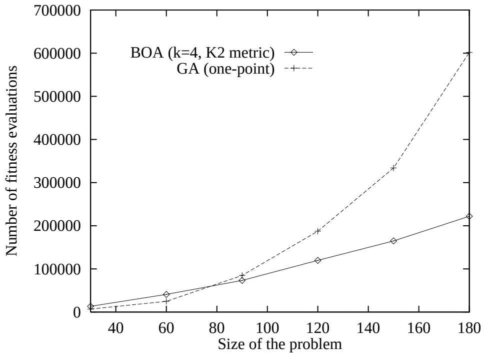


Figure 8: Results for $trap$-5.

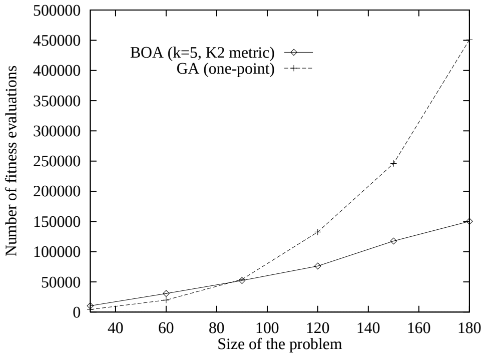


Figure 9: Results for 6-$bipolar$.

problem size. Although the function is additively composed of subfunctions of order 3, it is not sufficient to consider only BBs of order 3. Since for larger problem instances, the computational requirements of the algorithm were extremely high, only results for problem sizes up to $n = 91$ are presented. These results are shown in Table 2. Only results for the BOA are presented, since the simple genetic algorithm failed in most of the runs even with very large populations. It performed significantly worse than the BOA, and only results for small problem instances were computed.

Table 2: Results for 0-$peak$ function. The population sizes for the BOA ranged from $N = 1200$ for $n = 30$ to $N = 8900$ for $n = 91$.

n	**Population size**	**Fitness eval.**
31	1200	11040
61	3300	40920
91	8900	143735

For the 2-D Ising spin-glass function, an instance of size $n = 121$ with the size of the grid 11×11 previously examined by Mühlenbein et al. (1998) was used. For this instance, the function values in global optima are 178. The BOA converged to global optima in all of 30 independent runs with population size of $N = 2600$. An average number of fitness evaluations until successful convergence was 48620. This result is worse than the results achieved with the FDA by Mühlenbein et al. (1998), where the FDA converged to the optima in 11000 fitness evaluations. This is caused by the fact that in Mühlenbein et al. (1998), the initial population was generated according to a local approximation of the conditional marginal distributions and not by uniform random distribution as in the BOA, and the selection pressure in the FDA was stronger. Moreover, the FDA got the distribution estimate as input, while the BOA had to learn this on the run given no problem-specific knowledge.

8 BOA and the Existing Theory

The class of distribution that can be encoded by Bayesian networks does not seem as rich as the class of general conditional distributions used in the FDA. However, this section will formally show that this is not the case. For each conditional distribution (see Equation 28), there exists an identical distribution in the form given by Equation 1. In this section, the equivalence of the classes of distributions used by the FDA and the BOA will be formally proven first. Then, the most important result of the theory of the FDA that can be applied for a valid factorized distributions in this algorithm will be briefly presented. The presented theory can be applied to an ideal case with the BOA algorithm when the network constructed by the BOA is identical or approximately equal to a valid distribution factorization. It also demonstrates the power of the used class of distributions to solve additively decomposable problems.

Before proving that the class of conditional distributions is identical to the class of distributions used in the BOA, we will prove the following lemma.

LEMMA 8.1: *Each conditional probability $P(X_0, \ldots, X_{k-1}|R)$ can be decomposed into a product of conditional probabilities with at most one variable on the left side so that*

$$p(X_0, \ldots, X_{k-1}|R) = \prod_{j=0}^{k-1} p(X_j|R_j), \tag{25}$$

where R is a subset of variables from $X = (X_k, \ldots, X_{n-1})$, and R_j for each $j \in \{0, \ldots, k-1\}$ is a subset of variables from $X = (X_0, \ldots, X_{n-1}) \setminus X_j$.

PROOF: We will decompose the conditional probability by induction on the length of the left side of the probability with the inductive hypothesis for $m \geq 1$, denoted by $IH(m)$, defined as

$$p(X_0, \ldots, X_{m-1}|R_{m-1}) = \prod_{j=0}^{m-1} p(X_j|R_j) \tag{26}$$

for some sets R_j for all $j \in \{1, \ldots, m-2\}$.

(1) $IH(1)$: $p(X_0|R_0)$ is already of the required form. The case is trivial.

(2) $m > 1, IH(m)$ is true.

$IH(m+1)$: Inductive step can be performed as follows:

$$\begin{aligned} p(X_0, \ldots, X_{m+1}|R_{m+1}) &= \frac{p(X_0, \ldots, X_{m+1}, R_{m+1})}{p(R_{m+1})} \\ &= \frac{p(X_0, \ldots, X_{m+1}, R_{m+1})}{p(R_{m+1})} \cdot \frac{p(X_{m+1}, R_{m+1})}{p(X_{m+1}, R_{m+1})} \\ &= p(X_{m+1}|R_{m+1}) p(X_0, \ldots, X_{m-1}|R_m, X_m) \end{aligned} \tag{27}$$

By using $IH(m)$ we get $IH(m+1)$. □

The following theorem proves that the class of distributions used in the BOA is identical to the class of general conditional distributions considered in the FDA defined by

$$p(X) = \prod_{j=0}^{l-1} p(L_i|R_i), \tag{28}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of variables, L_i are disjoint subsets of variables from X, each variable from X is in exactly one of these subsets, R_i are subsets of variables for which $R_i \subseteq L_0 \cup \ldots \cup L_{i-1}$, and $p(L_i|R_i)$ is the conditional probability of the variables L_i conditioned on the variables R_i.

THEOREM 8.1: *The classes of distributions used in the BOA and the FDA are equivalent, i.e., for each conditional distribution, there exists an equivalent joint distribution encoded by a Bayesian network, and for each distribution encoded by a Bayesian network, there exists an equivalent conditional distribution.*

PROOF: By definition, each distribution encoded by a Bayesian network is automatically a conditional distribution (see Equations 1 and 28).

The opposite direction of the proof is somewhat more complicated. We will decompose each component in the conditional distribution using the previous lemma. It is easy to see that the resulting distribution is in the form defined in Equation 1. □

Since we have proven that the classes of distributions used by the BOA and the FDA are equivalent, the theory of the FDA can be used to demonstrate the power of the proposed algorithm in case it finds the right model. We close this section by presenting the main result of the theoretical investigation of the FDA performed by Mühlenbein et al. (1998).

Mühlenbein (1997) has shown that for the UMDA with an infinite population and truncation selection, the number of generations G until convergence on the $OneMax$ function that simply counts the bits in an input string is given by

$$G = \left(\frac{\pi}{2} - \arcsin(2p_0 - 1)\right) \frac{\sqrt{n}}{I}, \tag{29}$$

where p_0 is the proportion of 1's on each string position in the initial population, n is the size of a problem, and I is the selection intensity. The selection intensity (Mühlenbein, 1997) at a certain generation is given by

$$I = \frac{\bar{f}_s - \bar{f}}{\sigma}, \tag{30}$$

where the $\bar{f}_s$ is an average fitness of the selected strings, $\bar{f}$ is an average of the fitness of all strings, and σ is the standard deviation of the fitness values in the population. For most of the common selection methods, the selection intensity is constant during the optimization.

On uniformly-scaled separable problems, the FDA behaves very similarly to the UMDA on the $OneMax$ function (Mühlenbein et al., 1998). This can be proven by mapping the meta-variables in the FDA to the single variables in the UMDA (Mühlenbein and Mahnig, 1998). Some empirical results indicate that this might be the case for some decomposable problems with overlapping building blocks as well (Mühlenbein et al., 1998). Therefore, the latter theoretical result can be, with certain boundaries, used with the FDA and the BOA. For uniformly-scaled separable problems (i.e., problems that are decomposable in disjoint sub-problems), there are three important consequences of this result:

- For a population large enough, the algorithm using a valid distribution factorization converges to the optimum.
- The number of generations until convergence is proportional to $\sqrt{n}$.
- The number of generations until convergence is inversely proportional to I.

For the BOA, there is another consequence of this result. For k large enough for considered Bayesian networks to encode a valid factorization distribution, the algorithm has enough power to solve any uniformly-scaled separable problem in a number of generations proportional to the square root of a problem size $\sqrt{n}$ and inversely proportional to the selection intensity I. Our empirical results suggest that this result holds for some decomposable problems with overlapping building blocks, too.

For exponentially scaled subfunctions, the algorithm should converge in linear time with respect to the number of building blocks, with building blocks taking over in a domino-like fashion from the most salient building block to the least salient one (Thierens et al., 1998). Therefore, we expect the algorithm to converge in a number of generations proportional to n. In general, the convergence should be approximately proportional to something between $\sqrt{n}$ and n and inversely proportional to I.

A critical problem for convergence is the required population size. In the FDA, for populations larger than a critical population size, the number of generations until convergence remains constant. It depends only on a problem size and the selection intensity (see Equation 29). In the BOA, the population has to be somewhat larger so that the algorithm is able to find a good model for the problem. Under the assumption of a good model, the population-sizing theory of GAs may be used in order to determine the required population size (Harik et al., 1999). We are currently investigating this topic.

9 Future Work

In spite of promising empirical results, there are a number of challenging problems to be resolved in order to improve the proposed algorithm. We are currently investigating how the population size affects the solution quality as well as performing the experiments in order to confirm the statements about the expected number of generations until convergence from Section 8.

Another question that remains unanswered is the optimal choice of the metric and the network construction algorithms in order to eliminate the parameter k, which is the only prior information about the problem that the BOA requires. We have performed a number of experiments with recently suggested metrics that are biased toward simpler networks and allowed to use local structures (Friedman and Goldszmidt, 1999). The experiments have not shown significant improvement yet and a number of additional difficulties have arisen. The parameter k can be seen, however, as a useful feature. It is the user who decides how "deep" in modeling the problem we can go. By exploring a number of different settings, a general picture of how difficult the problem is can be obtained, and the optimal settings can be used in any future uses of the algorithm on a problem of similar properties. Moreover, the BDe metric allows the use of prior knowledge about the problem. In this way, the user no longer disregards prior knowledge but instead can use this knowledge to effectively bias the search, thereby further improving the results.

Another direction of future research on the BOA is to tackle hierarchical problems and to formally define the types of problems that the BOA can and cannot solve efficiently. Problems with various codings (e.g., real-coded, combinatorial, etc.) can be solved by using learning of Bayesian networks with continuous variables and other kinds of models.

10 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 modeling data by Bayesian networks. The algorithm can cover interactions up to a specified order. In the 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 this was not investigated in this paper.

The BOA is designed to solve decomposable problems of bounded difficulty. 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 even able to efficiently solve problems where the pairwise interactions are not transparent, although there are higher order interactions present in the problem (e.g., bipolar deceptive functions).

Acknowledgments

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

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. Technical Report No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Baluja, S. and Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In Fisher, D., editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 30–38, Morgan Kaufmann, San Francisco, California.

Bernardo, J. M. and Smith, A. F. M. (1994). *Bayesian Theory*. John Wiley and Sons, Chichester, England.

Buntine, W. L. (1991). Theory refinement of Bayesian networks. In D'Ambrosio, B. D., Smets, P. and Bonissone, P. P., editors, *Uncertainty in Artificial Intelligence: Proceedings of the Seventh Conference*, pages 52–60, Morgan Kaufmann, San Mateo, California.

Chickering, D. M., Geiger, D. and Heckerman, D. (1994). Learning Bayesian networks is NP-hard. Technical Report No. MSR-TR-94-17, Microsoft Research, Redmond, Washington.

De Bonet, J. S., Isbell, C. L. and Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. In Mozer, M. C., Jordan, M. I. and Petsche, T., editors, *Advances in Neural Information Processing Systems*, volume 9, page 424, MIT Press, Cambridge, Massachusetts.

Deb, K. and 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. *Journal of Research, National Bureau of Standards*, 71B:233–240.

Friedman, N. and Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I., editor, *Graphical Models*, 1st Edition, pages 421–459, MIT Press, Cambridge, Massachusetts.

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

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

Harik, G. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Ph.D. thesis, Department of Computer Science, University of Michigan, Ann Arbor, Michigan. Also IlliGAL Report No. 97005, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, Illinois.

Harik, G. (1999). Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report No. 99010, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, Illinois.

Harik, G. R. and Goldberg, D. E. (1996). Learning linkage. In Belew, R. and Vose, M., editors, *Foundations of Genetic Algorithms IV*, pages 247–262, Morgan Kaufmann, San Mateo, California.

Harik, G. R., Lobo, F. G. and Goldberg, D. E. (1998). The compact genetic algorithm. In *Proceedings of the International Conference on Evolutionary Computation 1998 (ICEC '98)*, pages 523–528, IEEE Service Center, Piscataway, New Jersey.

Harik, G., Cantú-Paz, E., Goldberg, D. E. and Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253.

Heckerman, D., Geiger, D. and Chickering, M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report No. MSR-TR-94-09, Microsoft Research, Redmond, Washington.

Henrion, M. (1988). Propagation of uncertainty in Bayesian networks by logic sampling. In Lemmer, J. F. and Kanal, L. N., editors, *Uncertainty in Artificial Intelligence 2*, pages 149–163, Elsevier, Amsterdam, The Netherlands.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan.

Howard, R. A. and Matheson, J. E. (1981). Influence diagrams. In Howard, R. A. and Matheson, J. E., editors, *Readings on the Principles and Applications of Decision Analysis*, volume II, pages 721–762, Strategic Decisions Group, Menlo Park, California.

Kargupta, H. (1995). *SEARCH, polynomial complexity, and the fast messy genetic algorithm*. Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana, Illinois.

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

Kvasnicka, V., Pelikan, M. and 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. and Mahnig, T. (1998). Convergence theory and applications of the factorized distribution algorithm. *Journal of Computing and Information Technology*, 7(1):19–32.

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

Mühlenbein, H., Mahnig, T. and 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*. Morgan Kaufmann, San Mateo, California.

Pelikan, M. and Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In Roy, R., Furuhashi, T. and Chawdhry, P. K., editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, Springer-Verlag, London, England.

Pelikan, M., Goldberg, D. E. and Lobo, F. (1999). A survey of optimization by building and using probabilistic models. IlliGAL Report No. 99018, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, Illinois.

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

Thierens, D. and Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 38–45, Morgan Kaufmann, San Mateo, California.

Thierens, D., Goldberg, D. E. and Pereira, A. G. (1998). Domino convergence, drift, and the temporal-salience structure of problems. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*, pages 535–540, IEEE Press, Piscataway, New Jersey.

Toulouse, G. (1977). Optimal graph matching for 2-D Ising models. *Commun. Phys.*, 2:115–119.