Bayesian Optimization Algorithm, Decision Graphs, and Occam's Razor

Martin Pelikan, David. E. Goldberg, and Kumara Sastry

IlliGAL Report No. 2000020
May 2000

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

Bayesian Optimization Algorithm, Decision Graphs, and Occam's Razor

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

Abstract

This paper discusses the use of various scoring metrics in the Bayesian optimization algorithm (BOA) which uses Bayesian networks to model promising solutions and generate the new ones. The use of decision graphs in Bayesian networks to improve the performance of the BOA is proposed. To favor simple models, a complexity measure is incorporated into the Bayesian-Dirichlet metric for Bayesian networks with decision graphs. The presented algorithms are compared on a number of interesting problems.

1 Introduction

Recently, the use of local structures as default tables and decision trees/graphs in context of learning the structure of Bayesian networks has been proposed and discussed (Friedman & Goldszmidt, 1999; Chickering, Heckerman, & Meek, 1997). Using local structures has shown to improve the performance of methods for learning Bayesian networks in terms of the likelihood of the resulting models on a number of benchmark data sets. However, none of these approaches was used to improve model building in the Bayesian optimization algorithm, which uses Bayesian networks to guide the exploration of the search space by using a model of promising solutions to generate new ones. Moreover, the use of various metrics in the BOA has not been investigated thoroughly.

The purpose of the paper is threefold. First, the paper briefly introduces the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1998). Various methods that can be used in order to construct and evaluate competing models in the BOA are presented. Second, the use of decision graphs in the model construction phase of the BOA is proposed to improve its performance. Third, to eliminate superfluously complex models, model complexity is incorporated into the Bayesian-Dirichlet scoring metric for Bayesian networks with decision graphs and it is empirically shown that the introduced pressure is sufficient to eliminate the necessity of bounding the complexity of models by the user. This is a significant contribution. The advantages of the minimum description length (MDL) metric, which favors simple models, are attained without having to sacrifice a possibility of using prior knowledge about the solved problem introduced by Bayesian metrics. The performances of the BOA with various scoring metrics and network construction algorithms, and the simple genetic algorithm are compared on a number of problems.

The paper starts by a brief overview of the recent developments in the field of optimization that guides the exploration of the search space by building probabilistic models of promising solutions

found so far. Thereafter, the Bayesian optimization algorithm is described. Section 3 provides basic theoretical background of learning a structure of Bayesian networks. Section 4 describes how decision graphs can be used as a core component of learning the structure of Bayesian networks and provides a Bayesian scoring metric for computing the likelihood of the evaluated network given the data. The results of our experiments are described in Section 5. The paper is summarized and concluded in Section 6.

2 Background

2.1 Genetic Algorithms

A genetic algorithm (GA) (Goldberg, 1989) is an optimization method that evolves a population of candidate solution to a given problem by repeatedly applying operators of selection, mutation, and recombination/crossover to the current population. The first population of solutions is generated randomly with a uniform distribution. Prior information about the problem can be used to bias the initial population to the regions that seem to be a promising starting point for the algorithm. From the current population the better solutions are selected at the expense of the worse ones, yielding the set of promising solutions. By combining and slightly perturbing the selected solutions, a set of new solutions is constructed. New solutions replace the old ones and the process is repeated until the termination criteria (e.g., convergence to a singleton, or a maximal number of iterations) are reached.

The measure of quality of candidate solutions is called the fitness function. The fitness can be evaluated by either a computer or a human expert. GAs attempt to maximize this value. The solutions are usually represented by fixed-length strings over a finite alphabet. The representation together with the fitness function define the problem that is being solved. To select promising solutions, various methods can be used. Tournament selection, for instance, repeatedly selects a particular number of individuals from the population at random, and always picks the best of them. The tournaments are repeated until enough solutions are selected. Fitness proportionate selection selects each solution with a probability proportional to its fitness. Other selection schemes include truncation selection which selects the portion of the best solutions from the population and the Boltzmann selection which selects individuals according to the Boltzmann distribution based on their fitness values.

Without recombination the number of fitness evaluations until the optimum is reached grows as a polynomial of order equal to the size of important partial solutions that must be considered and combined to reach the optimum. That is why for complex problems even with separable subfunctions contributing to the overall fitness local search algorithms based only on mutation perform quite poorly. We discuss this topic further in Section 5.3.

By applying recombination and mutation, GAs are manipulating a large number of promising partial solutions. However, fixed, problem independent, recombination and mutation operators often result in an inferior performance even on simple problems. Without knowing where the important partial solutions are and designing problem specific operators that take this information into account, the number of fitness evaluations and the required population sizes grow exponentially with the number of decision variables. We discuss this topic later in Section 5.3. That is why there has been a growing interest in methods that try to use the information acquired during the optimization to combine the promising solutions found so far more effectively. One of the approaches to resolve the above problem is based on using probability distributions to model promising solutions found so far and generating new solutions according to the estimated distribution. Probability

distributions can capture variables which are correlated and the ones which are independent. This can subsequently be used to combine the solutions in more effective manner.

2.2 Probabilistic Model-Building Genetic Algorithms

The set of selected solutions contains information about the good solutions and it can be seen as a sample from the space of solutions that we are interested in. Global information extracted from the set of promising solutions can be used to estimate their distribution and this estimate can be used in order to generate new solutions. In other words, the distribution of "good" points can be estimated and the new points can be simply generated according to the same distribution. The algorithms based on this principle are called the probabilistic model-building genetic algorithms (PMBGAs) (Pelikan, Goldberg, & Lobo, 2000), or the estimation of distribution algorithms (EDAs) (Mühlenbein & Paaß, 1996).

However, estimating a multivariate distribution is not an easy task. There is a trade off between the accuracy of the estimation and its computational cost. To use computationally efficient methods, one must make a number of assumptions, all of which decrease the generality of the used method. Next sub-sections describe basic principles of recently proposed algorithms that use probabilistic models of promising solutions to guide their search. For a more detailed overview, see Pelikan et al. (2000).

Probably the simplest way to estimate the distribution of good solutions is to assume that the variables in a problem are independent. New solutions can be generated by only preserving the proportions of the values of all variables independently of the context. This is the basic principle of the population based incremental learning (PBIL) algorithm (Baluja, 1994), the compact genetic algorithm (cGA) (Harik et al., 1998), and the univariate marginal distribution algorithm (UMDA) (Mühlenbein, 1997). Since these algorithms take into account only contributions of the values of each variable without considering the contexts where these contributions take place, it is natural to expect that the algorithms should work very well on linear problems, but experience great difficulties on problems where the variables are correlated and therefore the context does matter.

The first attempts to resolve this problem were the incremental algorithm using the so-called dependency trees in order to estimate the distribution of selected solutions (Baluja & Davies, 1997) and the population-based MIMIC algorithm using a simple chain distribution (De Bonet et al., 1997). Another population-based attempt to solve the problem of the disruption of building blocks of order two by using a slightly more general technique is the bivariate marginal distribution algorithm (BMDA) (Pelikan & Mühlenbein, 1999).

In the algorithms described above, to determine the contribution of a particular variable, the context of one other variable can be taken into account. Consequently, the algorithms can solve more complex problems efficiently. However, this has been shown to be still insufficient to solve problems with interactions of higher order efficiently (Pelikan & Mühlenbein, 1999; Bosman & Thierens, 1999). Covering pairwise interactions still does not preserve partial solutions of higher order. Moreover, interactions of higher order do not necessarily imply pairwise interactions that can be detected at the level of partial solutions of order two.

The first algorithm to tackle multivariate interactions was the factorized distribution algorithm (FDA) (Mühlenbein et al., 1998), which uses a fixed factorization of the distribution 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 of which the factorization of the distribution is known, this algorithm is a very powerful optimization tool. Unfortunately, the exact factorization of the distribution is often not available without computationally intensive 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 factorization.

This problem has been overcome with the Bayesian optimization algorithm (Pelikan, Goldberg, & Cantú-Paz, 1998), which uses Bayesian networks to model promising solutions and generate the new ones and the extended compact genetic algorithm (ECGA) (Harik, 1999) which uses the minimum description length metric to construct groups of correlated variables and generates new solutions accordingly.

2.3 Bayesian Optimization Algorithm

The Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1998) uses Bayesian networks to model promising solutions and subsequently guide the exploration of the search space. In the BOA, the first population of strings is generated randomly with a uniform distribution. 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 of quality of networks and any search algorithm can be used to search over the networks in order to maximize/minimize the value of the used metric. Besides the set of good solutions, prior information about the problem can be used in order to enhance the estimation and subsequently improve convergence. New strings are generated according to the joint distribution encoded by the constructed network. The new strings are added into the old population, replacing some of the old ones. The pseudo-code of the BOA is shown in Figure 1.

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 pseudo-code of the Bayesian optimization algorithm.

3 Bayesian Networks

A Bayesian network (Pearl, 1988) is a directed acyclic graph with the nodes corresponding to the variables in the modeled data set (in our case, to the positions in solution strings). Mathematically, a Bayesian network encodes a joint probability distribution given by

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

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 given its parents Π_{X_i}. A directed edge relates the variables so that in the encoded distribution, a variable corresponding to the terminal node is conditioned on a variable corresponding to the initial node. More incoming edges into a node result in a conditional probability of the corresponding variable with conjunctional condition containing all its parents. The network encodes independence assumptions that each variable is independent of any of its antecedents given its parents.

Various methods can be used to construct the network given the set of selected solutions. All of these methods have two basic components: (1) a scoring metric which discriminates the networks according to their quality and (2) a search algorithm which searches over the networks to find the one with the best scoring metric value. The BOA can use any scoring metric and any search algorithm. In our recent experiments, we have used the Bayesian-Dirichlet metric (Heckerman, Geiger, & Chickering, 1994). The complexity of the considered models was bounded by the maximum number of incoming edges into any node denoted by k. To search the space of networks, a simple greedy algorithm was used due to its efficiency. Next two sections briefly discuss the Bayesian-Dirichlet metric that can be used to compute the probability of a network given the data. Thereafter, a simple greedy algorithm that can be used to maximize this measure is described. This algorithm does not guarantee global optimality of the constructed network; however, it can be implemented very efficiently and results in a very good models on many problems (Heckerman, Geiger, & Chickering, 1994).

3.1 Bayesian-Dirichlet Metric

The Bayesian Dirichlet (BD) metric (Heckerman et al., 1994) combines the prior knowledge about the problem and the statistical data from a given data set. The probability of a Bayesian network B given data D can be computed by applying Bayes theorem as

$$p(B|D) = \frac{p(B)p(D|B)}{p(D)}. \tag{2}$$

The higher the $p(B|D)$, the more likely the network B is a correct model of the data. Therefore, the value of $p(B|D)$ can be used to score different networks and measure their quality. This measure is called a Bayesian scoring metric, or the *posterior* probability of B given data D. Since we are only interested in comparing different networks (hypotheses) for a fixed data set D, we can eliminate the denominator of the above equation. The remaining two terms in the above equation are discussed in the following paragraphs.

The probability $p(B)$ is called the *prior* probability of the network B and it can be used to incorporate prior information about the problem by assigning higher probabilities to the networks

confirming our intuition or expert knowledge. Heckerman, Geiger, and Chickering (1994) suggest the following assignment to bias the search toward networks similar to a prior network specified by an expert:

$$p(B) = c\kappa^{\delta},$$

where c is a normalization constant, $\kappa \in (0, 1]$ is a constant factor penalizing the network for each unmatched edge with the prior network, and δ is the so-called symmetric difference between B and the prior network. By setting the prior network to an empty network the metric gives preference to simpler networks. However, judging from our own experience, this pressure is not strong enough to eliminate the upper boundary on the network complexity required for an efficient learning. A more effective assignment is discussed later in Section 4.2.

It is very difficult to find a closed expression for the probability $p(D|B)$ of the data D given the network B. Heckerman, Geiger, and Chickering (1994) derived the closed expression for $p(D|B)$ by making a number of assumptions on the data. First assumption is that the data is a multinomial sample. The second assumption is the assumption of parameter independence, which says that (1) the parameters associated with each variable are independent (also called *global parameter independence*, see Spiegelhalter & Lauritzen, 1990) and that (2) the parameters associated with each instance of the parents of a variable are independent (also called *local parameter independence*, see Spiegelhalter & Lauritzen, 1990). The assumption of parameter modularity states that the parameters (values in the conditional probability tables) associated with a variable depend only on the variable and its parents. The Dirichlet assumption restricts the parameter set for each variable to have a Dirichlet distribution where each exponent corresponds to one possible instance of the variable and its parents. The last assumption is the one of complete data, stating that the database D is complete, i.e. it contains no missing data. Under the above assumptions, the following closed expression can be derived for $p(D|B)$:

$$p(D|B) = \prod_{i=0}^{n-1} \prod_{\pi_i} \frac{\Gamma(m'(\pi_i))}{\Gamma\left(m'(\pi_i) + m(\pi_i)\right)} \prod_{x_i} \frac{\Gamma\left(m'(x_i, \pi_i) + m(x_i, \pi_i)\right)}{\Gamma(m'(x_i, \pi_i))}, \tag{3}$$

where the product over π_i runs over all instances π_i of the parents Π_i of X_i, and the product over x_i runs over all instances x_i of X_i. By $m(\pi_i)$, the number of instances in D with Π_i instantiated to π_i is denoted. When the set Π_i is empty, there is one instance of Π_i and the number of instances with Π_i instantiated to this instance is set to N (the size of the data set D). By $m(x_i, \pi_i)$, we denote the number of instances in D that have both X_i set to x_i as well as Π_i set to π_i. The metric computed according to the above equation is called the Bayesian-Dirichlet metric, since one of the assumptions made to compute the formula is that the parameters are distributed according to a Dirichlet distribution.

Terms $m'(x_i, \pi_i)$ and $m'(\pi_i)$ express our beliefs in frequencies $m(x_i, \pi_i)$ and $m(\pi_i)$, respectively, and can be used as another source of prior information. A simple prior for the parameters $m'(x_i, \pi_i)$ and $m'(\pi_i)$ is to assume $m'(x_i, \pi_i) = 1$ for all x_i and π_i, and compute $m'(\pi_i)$ according to the above assignment. The metric using this assignment is often referred to as the K2 metric.

3.2 Minimum Description Length Metric

A minimum description length metric is based on the philosophical rule called Occam's razor, claiming that the simplest of competing theories be preferred to the more complex ones. The MDL metric favors short models. A total description length of a data set D compressed according to

a given model is defined as the sum of the space, measured in bits, required by the model, its parameters (various frequencies), and the data compressed according to the model. In context of optimization the minimum description length was first time used by (Harik, 1999) in the extended compact genetic algorithm.

Let us use a network B with each node corresponding to one variable from $X = (X_0, \ldots, X_{n-1})$ as a model to compress the data set D of size N. To store the model, we need to store both the network structure (a directed acyclic graph) and the conditional probabilities used in the encoded distribution (terms $p(X_i|\Pi_i)$ from Equation 1). The length of the compressed data then depends on the values of conditional probabilities.

A directed acyclic graph can be encoded by storing a set of parents of each node. The set of parents of a particular node can be encoded by the number of the parents followed by the index of the set of parents in some agreed-upon enumeration of all possible sub-sets of variables of the corresponding cardinality. Since each node can have at most $(n-1)$ parents, to encode a number of parents of each node in a binary code, $\log_2 n$ bits can be used. There are $\binom{n}{|\Pi_i|}$ possible sub-sets of variables of the cardinality $|\Pi_i|$, where $|\Pi_i|$ is the number of parents of X_i. Therefore, to encode the set of parents of X_i, $\log_2 \binom{n}{|\Pi_i|}$ bits can be used. The number of bits needed to encode a network structure B, denoted by $length\,(B)$, can be then computed as

$$length\,(B) = \sum_{i=0}^{n-1} \left(\log_2 n + \log_2 \binom{n}{|\Pi_i|} \right). \tag{4}$$

To store the conditional probabilities according to the distribution encoded by the network, we need to store all combinations of all but one values x_i of each variable X_i and all possible instances π_i of its parents Π_i. For each such combination of x_i and π_i the corresponding conditional probability $p(x_i|\pi_i)$ must be stored. For binary variables, there are $2^{|\Pi_i|}$ possible combinations of values of the variable and its parents (excluding one value x_i for each π_i, e.g. $x_i = 1$, for which $p(x_i|\pi_i)$ can be computed from the remaining conditional probabilities). This is an upper bound and can be reduced by using more sophisticated data structures to encode the conditional probability tables. To accurately encode each conditional probability, we can use $\frac{1}{2}\log_2 N$ bits (Friedman & Yakhini, 1996). Thus, the overall number of bits needed to store the table of conditional probabilities for the network B, denoted by $length\,(X, \Pi)$, is given by

$$length\,(X, \Pi) = \frac{1}{2}\log_2 N \sum_{i=0}^{n-1} 2^{|\Pi_i|}. \tag{5}$$

Given the conditional probabilities $p(x_i|\pi_i)$ for all values x_i and π_i of X_i and its parents Π_i, respectively, the overall number of bits needed to store the data set D by using Huffman coding for the instances in D, denoted by $length(D|B)$, can be approximated (Cover & Thomas, 1991) by

$$length(D|B) = -N \sum_{i=0}^{n-1} \sum_{x_i} \sum_{\pi_i} p(x_i, \pi_i) \log_2 p(x_i|\pi_i), \tag{6}$$

where $p(x_i, \pi_i)$ is the probability of $X_i = x_i$ and $\Pi_i = \pi_i$, the sum over x_i runs over all instances x_i of X_i, and the sum over π_i runs over all instances π_i of Π_i. The total length of the model, its parameters, and the data set compressed according to this model, denoted by $length\,(B, D)$, is then given by

$$length\,(B, D) = length\,(B) + length\,(X, \Pi) + length\,(D|B)\,. \tag{7}$$

The lower the above measure, the shorter the description length of the data D given the model B. Therefore, when constructing a network, one must minimize the above measure. A major advantage of the MDL metric is that it favors simple models so that no upper bound on the model complexity has to be specified. This bound comes up naturally. However, when using a greedy algorithm for model construction, the problem of finding a valid model can become more difficult. Moreover, the MDL metric does not allow the use of prior information about the problem. In many real-world problems the utilization of expert knowledge (which is often available in some form) may be unavoidable. Section 4.2 presents another way of dealing with the complexity of models by specifying the prior probability of each model inversely proportionally to its complexity.

A similar metric, called the Bayesian Information Criterion (BIC), was used in the EBNA (Etxeberria & Larrañaga, 1999) and the LFDA (Mühlenbein & Mahnig, 2000) algorithms.

3.3 Constructing a Network

The problem of finding the best network given a scoring metric has been proven to be NP-complete for most Bayesian and non-Bayesian metrics (Chickering, Geiger, & Heckerman, 1994). However, since most of the commonly used metrics (such as the BD and MDL metrics) can be decomposed into independent terms each of which corresponds to one variable (node), to construct a network, a greedy algorithm, local hill-climbing, or simulated annealing can be used very efficiently. Moreover, empirical results show that more sophisticated search algorithms do not improve the obtained result significantly (Heckerman, Geiger, & Chickering, 1994).

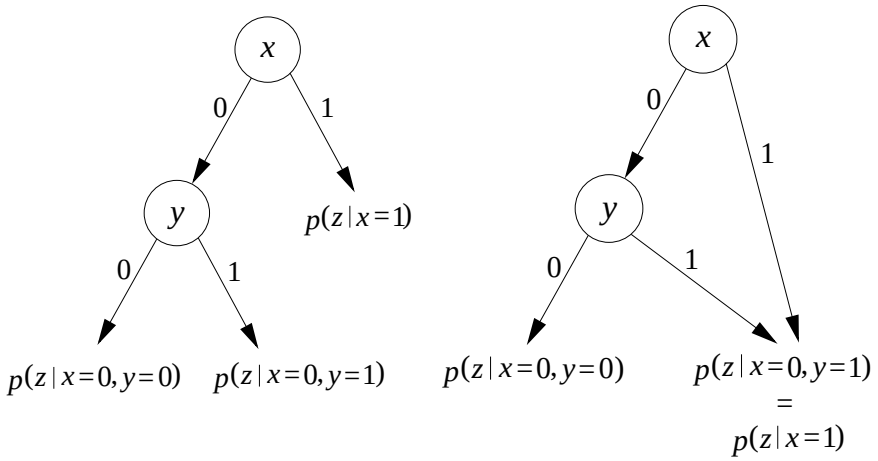
In our recent experiments we have used a simple greedy search algorithm. Each iteration of the algorithm, the graph operation that improves the network score the most is performed. The simple operations that can be performed on a network include edge additions, edge reversals, and edge removals. Only operations that keep the network acyclic are allowed and the number of parents of each node is bound by a constant in order to avoid superfluously complex models. The construction finishes when no operations are allowed or no applicable graph operation improves the score.

4 Decision Graphs in Bayesian Networks

Recently, the use of local structures as decision trees, decision graphs, and default tables to represent equalities among parameters was proposed (Friedman & Goldszmidt, 1999; Chickering, Heckerman, & Meek, 1997). This results in that the number of parameters required to fully encode the distribution can be significantly decreased. This section explains how decision graphs can be used to improve the expressiveness of Bayesian networks and the learning of Bayesian network structure. It provides a metric for computing the likelihood of Bayesian networks with decision graphs and a method for constructing such structures. The network construction algorithm (Chickering, Heckerman, & Meek, 1997) takes an advantage of using decision graphs by directly manipulating the network structure through the graphs. This is a major difference from the way the decision trees/graphs are usually used in context of the MDL metrics as in (Friedman & Goldszmidt, 1999). Additionally, an assignment of the prior probabilities that takes into account model complexity is presented.

4.1 Decision Graphs

A decision tree is a directed acyclic graph where each node except for the root has exactly one parent. The root has no parents. Non-leaf nodes of the tree are labeled by a variable (feature) on



(a) Decision tree (b) Decision graph

Figure 2: Example decision tree (a) and decision graph (b) for encoding a conditional probability distribution of $p(z|x, y)$.

which we want to split. When a node is labeled by a variable v, we say that this node is a split on v. Edges from a split on v are labeled by non-empty distinct exhaustive subsets of possible values of v.

To traverse a tree given an assignment of all the variables, we start in a root and on each split on v we continue to the child along the edge which contains the current value of v. For each instance (an assignment of all the variables), there is only one possible way of traversing the tree to a leaf. This is a consequence of that the edges leading to different children must be associated with distinct subsets of values.

A leaf of a decision tree contains a quantity or information of our interest, associated with all instances that end up a traversal through the tree in the leaf. To use decision trees for representing conditional probabilities of a particular variable, the leaves contain the conditional probabilities of the values of the variable given that the variables contained in a path from the root are fixed according to the path.

An example adopted from Chickering et al. (1997) of a decision tree that encodes the conditional probability distribution $p(z|x, y)$ is shown in Figure 2(a). All variables in this figure are binary and thus we can split only to two children, one for 0 and one for 1. Instance $(x = 1, y = 1, z = 0)$ would traverse the tree to the right-most leaf. Instance $(x = 0, y = 1, z = 0)$ would result in the middle leaf. This decision tree represents, for instance, the equality constraints $p(z|x = 1, y = 0) = p(z|x = 1, y = 1)$.

A decision graph is an extension of a decision tree in which each non-root node can have multiple parents. By a decision graph, any set of equality constraints can be encoded. This can be shown by simply constructing a complete tree and merging all leaves that are equal. An example of a decision graph is shown in Figure 2(b). This decision graph can be obtained by merging the leaves $p(z|x = 0, y = 1)$ and $p(z|x = 1)$ which represents another equality constraint. It is important to note that the equality constraints, in fact, represent independence constraints. Moreover, each leaf in the decision graph for a variable represents independence assumptions of any variable not contained in the path from the root to this leaf, given the constraints specified by the corresponding

path to this leaf.

There are four major advantages of using decision graphs in learning Bayesian networks. First, much less parameters can be used to represent a model. This saves memory and time requirements of both model construction as well as its utilization. Second, the use of decision graphs allows learning more complex class of models, called Bayesian multinets. Third, the construction of a Bayesian network with decision graphs performs smaller and more specific steps what results in better models with respect to their likelihood. The last but not least advantage is that the network complexity measure can be easily incorporated into the scoring metric and it results in a measure that is still based on Bayesian statistics and thus allows the use of prior information unlike the MDL metric, and is as robust as the MDL metric when no such information is used. This is not the case with the previously discussed network construction algorithm. We will discuss this topic shortly.

4.2 Bayesian Score for Networks with Decision Graphs

In this section we shortly present how to compute a Bayesian score for Bayesian networks where conditional probabilities and independence assumptions for each variable are encoded by decision graphs. Conditional probabilities for a variable X_i are stored in a decision graph G_i, i.e. for each variable there is one decision graph.

It has been shown that the Bayesian score can be computed for Bayesian networks where the independence constraints are encoded by a decision graph for each of the variables in a very similar way (Chickering, Heckerman, & Meek, 1997). The outer product from Equation 3 remains the same. The middle product runs over all leaves of the decision graph G_i corresponding to the variable X_i. The inner-most product runs over all possible instances of the variable X_i. Thus,

$$p(D|B) = \prod_{i=0}^{n-1} \prod_{l \in L_i} \frac{\Gamma(m'(i,l))}{\Gamma(m(i,l)+m'(i,l))} \prod_{x_i} \frac{\Gamma(m(x_i,i,l)+m'(x_i,i,l))}{\Gamma(m'(x_i,i,l))}, \tag{8}$$

where L_i is the set of leaves in the decision graph G_i for X_i, $m(i,l)$ is the number of instances in D which end up the traversal through the graph G_i in the leaf l, $m(i,l)$ is the number of instances that have $X_i = x_i$ and end up the traversal of the graph G_i in the leaf l, the $m'(i,l)$ represents our prior knowledge about the value of $m(i,l)$, and $m'(x_i,i,l)$ represents our prior knowledge about the value of $m(x_i,i,l)$. The Bayesian score is then given by using Bayes theorem (see Equation 2).

To adjust the prior probability of each network according to its complexity, we first compute the description length of the parameters required by the networks. To encode one frequency in the data set of size N, it is sufficient to use $0.5 \log_2 N$ bits (Friedman & Goldszmidt, 1999). Therefore, to encode all parameters, we need $0.5 \log_2 N \sum_i |L_i|$ bits, where $\sum_i |L_i|$ is the total number of leaves in all decision graphs. To favor simpler networks to the more complex ones we can set the prior probability of a network to decrease exponentially with the description length of the set of parameters they require. Thus,

$$p(B) = c 2^{-0.5 \log_2 N \sum_i |L_i|}, \tag{9}$$

where c is a normalization constant required for the prior probabilities of all networks to sum to 1. The value of a normalization constant does not affect the result, since we are only interested in relative comparisons of networks and not the absolute value of their likelihood. As we will see in the next section, the assignment in the last equation is sufficient to bias the model construction to networks with less parameters and avoid superfluously complex network structures without having

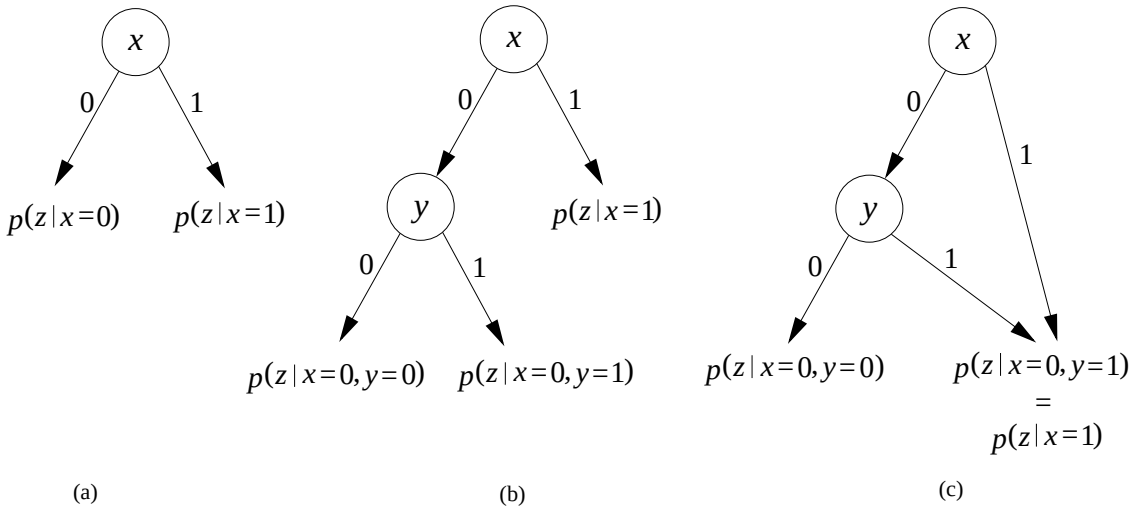


Figure 3: Example of applying various operators to a decision graph.

to determine the maximal number of incoming edges in advance. This eliminates another degree of freedom for setting the parameters of the algorithm and thus makes the algorithm easier to use.

The above assignment can be extended or fully replaced by the one that takes into account our prior knowledge about the problem by favoring models that are more similar to the prior network.

4.3 Operators on Decision Graphs

To construct a decision graph on binary variables, two operators are sufficient. The first operator is a *split*, which splits a leaf on some variable and creates two new children of the leaf, connecting each of them with an edge associated with one possible value of this variable, in our case, 0 or 1. The second operator is a *merge*, which merges two leaves into a single leaf and introduces a new equality constraint on the parameter set. See Figure 3 for an example of both operators. The decision graph (b) in this figure is created by splitting the leaf containing $p(z|x = 0)$ of the graph (a) on variable y. The graph (c) can be obtained by merging leaves $p(z|x = 1)$ and $p(z|x = 0, y = 1)$ of the decision graph (b) in this figure. It does not make sense to split a leaf on a variable that was encountered on the path from the root to this leaf and therefore these operators will not be allowed.

For variables that can obtain more than two values, two versions of the split operator can be considered: (1) a complete split which creates one child for each possible value of the variable (as above), and (2) a binary split, which creates one child correspond to one particular value and another child for all the remaining values. These two operators are equivalent in case of binary variables. Other alternatives can also be considered, including splitting the node on a variable so that each of the newly created children corresponds to a subset of values of this variable.

4.4 Constructing Bayesian Networks with Decision Graphs

The greedy algorithms described in this section differs from the one presented in Section 3.3 by that it does not manipulate the constructed network directly but only by modifying the decision graphs corresponding to each variable. The network B is initialized to an empty network that contains no

edges. The decision graph G_i for each variable X_i is initialized to a single-leaf graph, containing only probabilities $p(X_i)$.

Each iteration, all operators (e.g., all possible merges and splits) that can be performed on all decision graphs G_i are examined. The operator that improves the score the most is performed on the corresponding decision graph. The operators that can be performed include (1) splitting a leaf of some decision graph on a variable that was not encountered on the path from the root to the leaf and (2) merging two leaves into a single leaf.

When performing a split operator, we must make sure that no cycles appear in the network B. To guarantee that the final network remains acyclic, we can continuously update the network B each time we perform a split. Once we split a leaf of the graph G_i on a variable X_j, we add an edge (X_j, X_i) to the network B. If a cycle would appear in case of this addition, we ignore the operator and consider alternative ones. This requirement could be alleviated. The use of decision trees allows Bayesian multinets (Geiger & Heckerman, 1996) with one or more distinguished variables. However, it is computationally inefficient to take this under consideration.

(1) Initialize a decision graph G_i for each node X_i to a graph containing only a single leaf.

(2) Initialize the network B into an empty network.

(3) Choose the best split or merge that does not result in a cycle in B.

(4) If the best operator does not improve the score, finish.

(5) Execute the chosen operator.

(6) If the operator was a split, update the network B as described above.

(7) Go to (3).

It is important to notice the difference between the algorithm that directly modifies the network and the one which modifies the decision graphs. Adding an edge into a Bayesian network and using a full conditional probability table to store the corresponding probabilities corresponds to splitting all leaves of the decision graph corresponding to the terminal node of the edge on the variable corresponding to the initial node of the edge. However, by modifying only the decision graph, finer steps can be performed which may positively affect the quality of the resulting model.

5 Experiments

5.1 Specification of the Experiments

We have performed experiments on a number of functions. A simple linear function, called one-max, simply sums all bits, i.e.

$$f_{onemax}(X) = \sum_{i=0}^{i<n} X_i,$$

where $X = (X_0, \ldots, X_{n-1})$ is an input binary vector of length n.

The 3-deceptive function is additively composed of subfunctions of order 3, applied to disjoint

blocks of 3 consecutive bits in the input string. Each of these subfunctions is defined as

$$f^3_{dec}(X) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

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

The 3-deceptive function is then computed as

$$f_{3deceptive}(X) = \sum_{i=0}^{i<\frac{n}{3}} f^3_{dec}(X_{3i}, X_{3i+1}, X_{3i+2}),$$

where X is an input vector of n variables and n is a multiple of 3. The 3-deceptive function has one global optimum in $(1, 1, \ldots, 1)$ and $2^{\frac{n}{3}}$ local optima in all points where $(X_{3i} + X_{3i+1} + X_{3i+2}) \in \{000, 111\}$ for all $i \in \{0, \ldots, \frac{n}{3}\}$. The above function is deceptive in a sense that an average value over all strings that contain 0's in two particular positions is greater than the corresponding average with the 1's in these two positions. As a result of this, whenever we cut a variable out of the context of the remaining two variables of the corresponding subfunction, the algorithm is deceived and converges to local optima. That is why deceptive functions are often used as benchmarks for genetic and other evolutionary algorithms.

Another problem we have tested our algorithm on is the max-sat where the goal is to find a truth assignment of n boolean variables that maximizes the number of satisfied clauses of a given formula in a conjunctive normal form. A number of instances both generated at random as well as the instances known to be hard for stochastic optimization methods were tested. Random instances were generated so that the solution was generated first. Then, the instance was generated by repeatedly generating clauses. For each clause to be added it is required that it be satisfiable. We have found that random instances are very easy for both the simple genetic algorithm as well as the BOA. The hard instances have shown to be much more difficult for both algorithms.

Another problem we tested the algorithms on is the graph bisection, where the goal is to split the nodes of a given graph into two equally-sized groups so that the number of edges between the two groups is minimized. We have tried grid-like graph structures (see Figure 4 for an example) and the so-called caterpillar graphs (see Figure 5)which are known to be quite difficult due to only one global and many local optima (Schwarz & Ocenasek, 1999; Schwarz & Ocenasek, 2000). Moreover, the search space is symmetrical, which adds another source difficulty for genetic and other evolutionary methods that construct new solutions by combining previously found solutions of high quality (Pelikan & Goldberg, 2000).

Each candidate solution in the graph bisection problem is represented by a binary string with one bit per node in the graph. The value of a particular bit determines the class where the corresponding node in the graph is classified. Each class corresponds to one part the split. If a candidate solution does not contain exactly half ones and half zeroes, and thus it does not split the nodes of the graph in halves, a required number of the bits that are in majority are picked at random with a uniform distribution and subsequently flipped. In this fashion, the solution is repaired to split the nodes of the graph in halves.

An Ising spin-glass function is defined as

$$f_{Ising}(X) = \sum_{i=0}^{n-1} \sum_{j \in N(i)} Y_i J_{i,j} Y_j \tag{11}$$

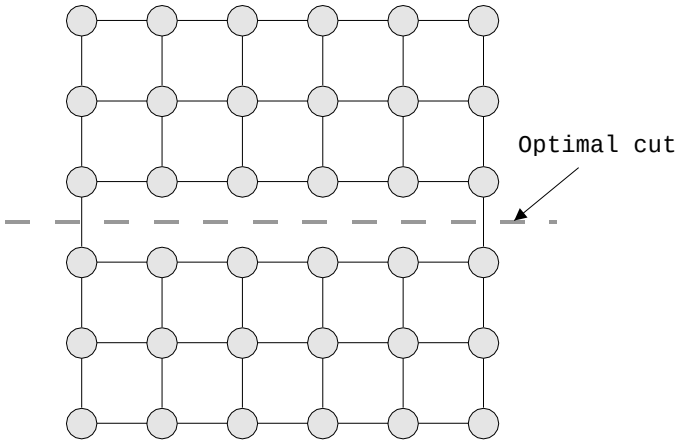


Figure 4: An example grid-like graph bisection problem with $n = 36$ nodes.

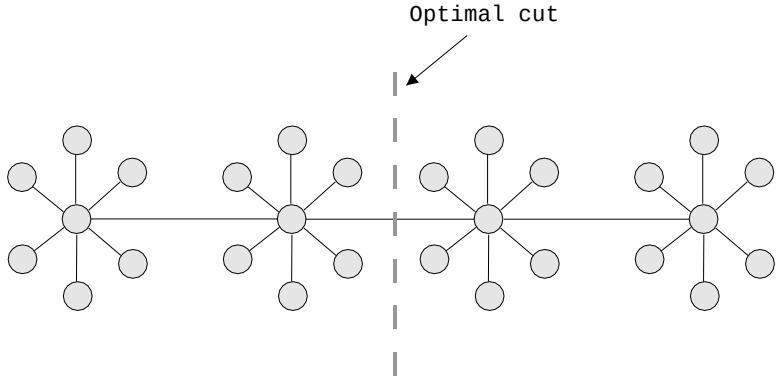


Figure 5: An example caterpillar graph bisection problem with $n = 28$ nodes.

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 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.

5.2 Compared Algorithms and the Parameters

We have compared the simple GA, the BOA with both the MDL metric as well as the Bayesian-Dirichlet metric, and the BOA using decision graphs to construct the model and encode its parameters. The complexity measure was incorporated into the metric for the use with decision graphs as described in above sections.

The population size in each algorithm was set as the minimal population size required for the particular algorithm to converge in all of 30 independent runs. Binary tournament selection was used in all experiments. The number of offspring is equal to a half of the population size and the generated offspring replace the worst half of the original population. In this fashion, the runs are more stable and elitism is introduced. The probability of crossover in the simple GA was determined according to empirical evidence presented elsewhere (Pelikan, Goldberg, & Cantú-Paz, 1999) as $p_c = 1$. Except for the one-max function, the mutation was not used, since it seems to not pay off on the tested problems. On one-max problem, $p_m = 0.01$ was used.

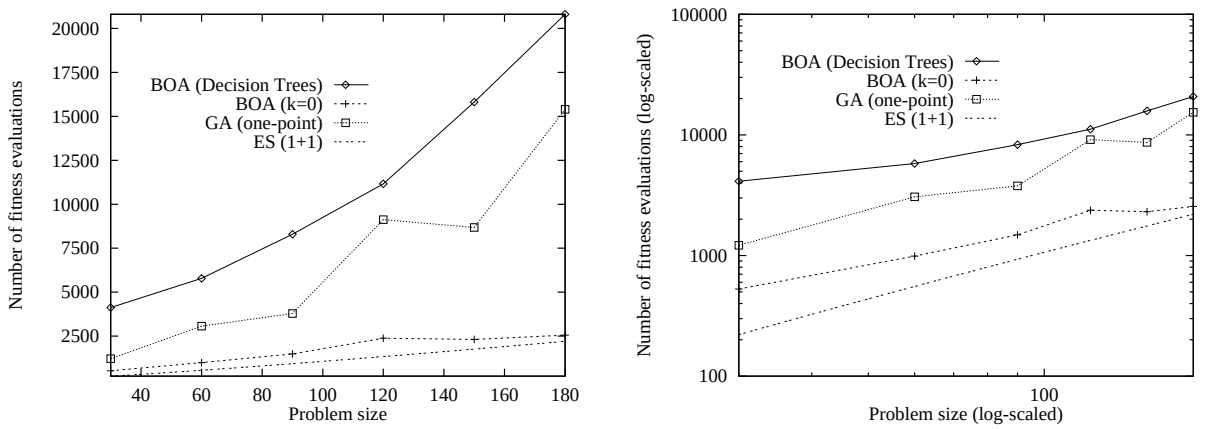
To show how difficult the problems tested in this paper can get for local optimization methods we have included the so-called (1+1) evolution strategy, referred to as (1+1)-ES in the following text. The (1+1)-ES evolves a single solution which is initially generated at random with uniform distribution. Each trial, a bit-wise mutation which flips each bit with a given probability is performed and the new solution replaces the old one in case it is better. In case the created solution is worse, the old solution is kept. For simple linear problems as one-max, the (1+1)-ES converges to the optimum in a linear time (Mühlenbein, 1992). However, as the order of subfunctions contained in the fitness function increases, its performance gets worse and the number of fitness evaluations grows proportionally to n^k, where n is the size of a problem and k is the size of the subfunctions (building blocks). The expected number of fitness evaluations required by the (1+1)-ES can be obtained by Markov-chain analysis (Mühlenbein, 1992). We will use these theoretical results to estimate the expected number of fitness evaluations to convergence.

5.3 Results

This section presents the results of our experiments. First, the results of (1+1)-ES are compared to other algorithms. It is argued that on complex problems the performance of local methods without any recombination is qualitatively worse than the performance of evolutionary algorithms which use recombination in a reasonable way. A necessity to learn what variables are correlated and use this information to improve recombination is stressed.

Figure 6(a) suggests that on a linear one-max problem the (1+1)-ES performs the best. This result is expectable since for linear problems the local changes always bias the search in the right direction. However, Figure 6(b) shows that the performance curves of all compared algorithms have a very similar slope and thus their scale-up behavior does not differ much. A slight increase in the slope of the simple GA with one-point crossover is caused by the slower mixing of this type of crossover. All algorithms require from about $O(n)$ to $O(n^{1.5})$ function evaluations with respect to the number of decision variables in the problem, denoted by n.

However, figures 7(a) and 7(b) show that as the order of subfunctions increases (here the order is 3), the performance of (1+1)-ES significantly decreases. Figures 8(a) and 8(b) show the results on a function of order 5 where the gap between the recombination-based methods and the local methods increases even more. These results suggest that with increased function difficulty, the importance of crossover increases. For both problems, the BOA converges to the optimum in about $O(n^{1.5})$ to $O(n^{1.7})$ function evaluations. The simple GA with one-point crossover converges to the optimum in about $O(n^{2.5})$ function evaluations because this problem allows one-point crossover to combine the solutions quite effectively. On the other hand, the (1+1)-ES requires $O(n^3)$ function evaluations on the 3-deceptive and $O(n^5)$ function evaluations on the trap-5 function. As the order of building blocks (or, the subfunctions composing the optimized function) increases, the performance of the BOA decreases by a constant factor as the recent theory suggests (Pelikan, Goldberg, & Cantú-Paz, 2000). However, the performance of local search methods increases with a polynomial of the



(a) Normal scale.

(b) Log scale.

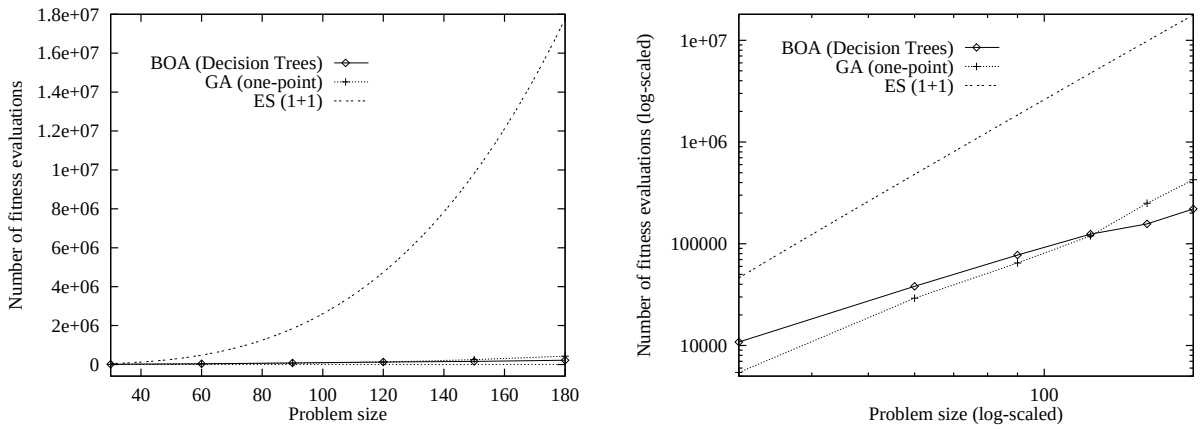
Figure 6: Comparison of (1+1)-ES and some more advanced evolutionary algorithms on the onemax function.

order of subfunctions used to construct the optimized function. This is a major difference in the performance. The time required by all methods in addition to a linear time is caused by that the algorithms must discover the interactions in a problem. If these were known in advance, only linear time would be required.

With a recombination that is independent of the coding (e.g., uniform crossover), the population size required by the simple genetic algorithm to converge to the optimum grows exponentially with the problem size for any deceptive problem of bounded difficulty (and, subsequently, for any more complex problem) (Thierens & Goldberg, 1993). The slope of the growth depends on the size of the subfunctions, but even with subfunctions of order 3, the growth of the required population size is exponential. When using one-point crossover the situation is the same and the number of fitness evaluations until convergence to the optimum grows exponentially in the number of decision variables, unless the variables are ordered so that the interacting variables are close to each other. The tighter the subfunctions, the better the performance. However, for many problems we do not have enough information to design such a coding and the corresponding operators. That is why there is a need for methods as the BOA that are able to discover interactions of the variables on the fly.

In next paragraphs among with the result of the BOA with various settings we also present the results of the simple GA. The best coding for the used one-point crossover is used. Therefore, this is a lower bound on the performance of the simple GA with one-point crossover. As the subfunctions would get looser, the performance of the simple GA would suffer and the algorithm would require exponentially many function evaluations until convergence to the optimum with respect to the number of decision variables.

The results on the simple linear one-max function are shown in Figure 9. One-max is a linear function and thus it is expectable that all the algorithms perform very well. The BOA with an optimal $k = 0$ (which is equivalent to the UMDA) performs the best, because it does not take into account any interactions and in this problem all the variables are independent. Thus, the model used by the BOA with $k = 0$ is an accurate one. Other algorithms try to model some interactions and that is why they perform worse. However, all algorithms seem to scale very well. The use of



(a) Normal scale.

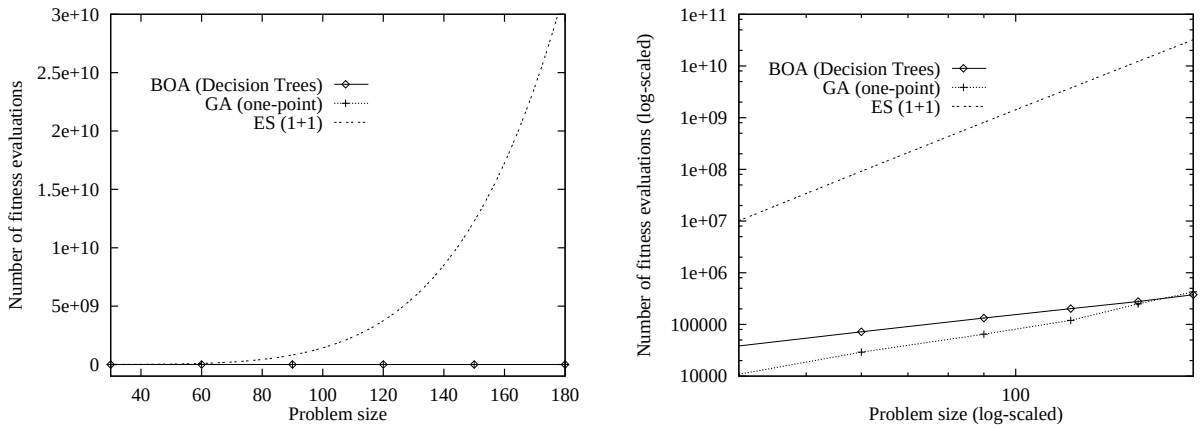
(b) Log scale.

Figure 7: Comparison of (1+1)-ES and some more advanced evolutionary algorithms on the 3-deceptive function.

decision graphs does not seem to pay off in this case because of that the noise misleads the model building which is very sensitive when the decision graphs are used. Other modeling techniques make bigger steps and it is harder to mislead them. Even though this feature is likely to make the BOA work less efficiently on simpler linear problem, it can be very useful when solving difficult problems where making little steps while building the model pays off. It is important to note that using binary tournament selection on this simple problem decreases the performance of all algorithms when compared to the results with the truncation selection with a threshold $\tau = 50\%$. This is caused by that the truncation selection with $\tau = 50\%$ decreases the diversity in the population (Cantú-Paz, 2000) more than the binary tournament selection and since this problem is very simple, this seems to pay off here. However, as we will see shortly, tournament selection has a positive effect on most of the compared algorithms when solving more difficult problems.

The results on the 3-deceptive function are shown in Figure 10. The results suggest that all versions of the BOA perform similarly and as the problem size grows, they outperform the simple genetic algorithm with one-point crossover. This suggests that they scale up better. It is important to note that the coding chosen for this problem is the best one for the simple genetic algorithm. If the building blocks (the bits corresponding to each deceptive subfunction) would be coded more loosely, the performance of the simple GA would get worse, and, eventually, for a random ordering of the variables in the strings representing solutions the simple GA would require exponential time (Thierens & Goldberg, 1993). However, the BOA is independent of the ordering of the bits in strings, and thus its performance would remain the same independently of the ordering we choose. The BOA with both the decision graphs and the MDL metric performs very similarly. This behavior is very interesting because the metrics are coming from two different paradigms.

Another important observation is that the performance of the BOA with the BD metric and a given maximal order of interactions in the problem decreases when we use the binary tournament selection instead of the truncation selection with a threshold $\tau = 50\%$ (Pelikan, Goldberg, & Cantú-Paz, 1999). This result is expectable, because the truncation selection used in our recent experiments has a higher selection intensity than the binary tournament selection. However, the results of all other compared algorithms get better. The reason for this behavior is again that the



(a) Normal scale.

(b) Log scale.

Figure 8: Comparison of (1+1)-ES and some more advanced evolutionary algorithms on the trap-5 function.

truncation selection decreases the diversity in the population (Cantú-Paz, 2000). The decrease in the diversity seems to pay off when some information about the problem is given as in the BOA with $k = 2$ and the BD metric; however, it also seems to threaten the performance of the BOA and simple GA when it must discover this information by itself.

The results on the grid-like graph bisection problem are shown in Figure 11. Recent empirical results (Schwarz & Ocenasek, 1999) suggest that good performance can be achieved by setting the maximal number of incoming edges in the network to $k = 3$. We have also applied the BOA with the MDL metric and decision graphs with a pressure to simple networks as described above. The results show that setting $k = 3$ and not considering the network complexity in the scoring metric yields the best performance. The BOA with the MDL metric and decision graphs perform very similarly. The differences between the performance of all the alternative settings are very small and the algorithm seems to scale very well with either of them. The simple genetic algorithm does not converge on this problem even with very huge populations and many generations and therefore it is not included in the comparisons.

The results on the caterpillar graph bisection are shown in Figure 12. It is interesting that even though the MDL performed quite well on the grid-like graph bisection, it performed about twice slower than the BOA with decision trees on the caterpillar graph bisection of sizes $n = 28$ and $n = 56$ and it was not able to solve problem of $n = 84$ with even very big populations. That is why we do not include any results of the BOA with MDL metric for problems of sizes $n = 84$ and more. However, the BOA with decision graphs performed quite well and solved all tested problem instances very efficiently and reliably.

The results on the spin-glass system are provided in Table 5.3. The BOA with the MDL metric performs the best. The BOA with $k = 2$ performs the worst. The simple GA does not reach the optimum even with huge populations and a large number of generations. This problem is very difficult for the simple GA. It is interesting that the MDL metric performs the best and outperforms both the BOA with decision graphs as well as a bound on the network complexity $k = 2$ and the K2 metric. We have also performed experiments with higher values of k. As the k increases to 4, the performance of the BOA with the K2 metric improves. The BOA with $k = 4$ and the

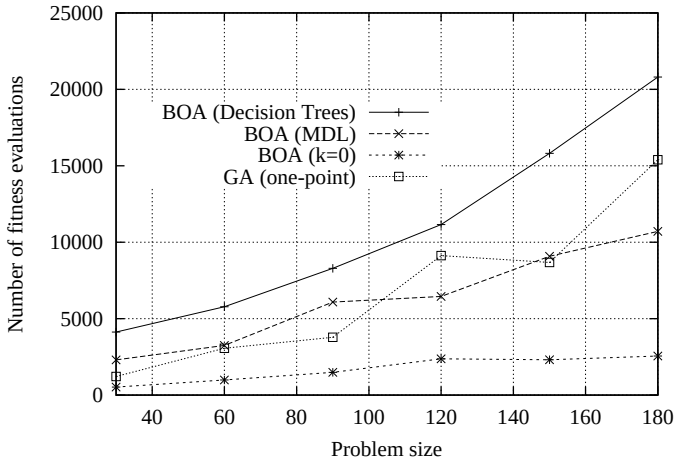


Figure 9: Results on the one-max function of sizes $n = 30$ to $n = 180$ in steps of 30.

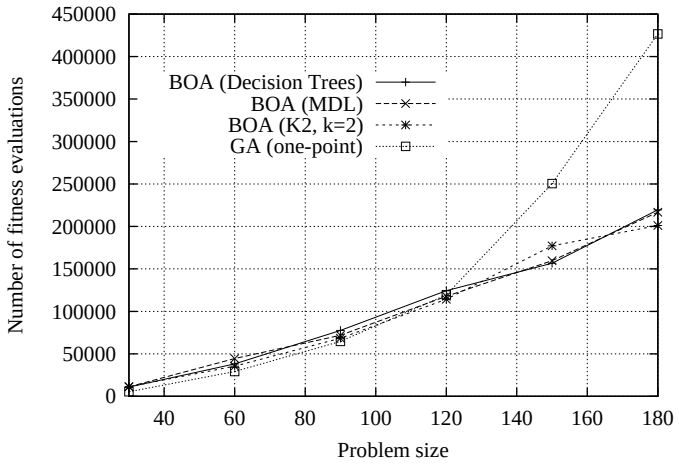


Figure 10: Results on the 3-deceptive function of sizes $n = 30$ to $n = 180$ in steps of 30.

K2metric performs the best of all compared algorithms. However, with $k = 5$ the performance again decreases. This suggests that the value of $k = 4$ is a good choice for this problem.

On randomly generated max-sat problem instances, all algorithms converged very fast without getting into any local optima. This suggests that randomly generated instances of max-sat are not very hard for stochastic optimization methods even when the ration between the number of clauses and variables increases to 4 or more. However, the performance of all algorithms has significantly decreased for the instances that are generated as hard for stochastic optimization methods. We have acquired two such instances from the web-page of T. Uchida and O. Watanabe (`http://www.is.titech.ac.jp/ watanabe/gensat/a2/`). The first instance has 75 variables and 335 clauses, and the second instance has 81 variables and 361 clauses. On the aforementioned web-page the first instance is listed as `F-5-19-29` and the second one is listed as `F34-5-19-29`. The simple genetic algorithm didn't converge of any of the two hard instances. The BOA with decision trees converged on the first instance in all 30 independent runs in 120400 function evaluations (a

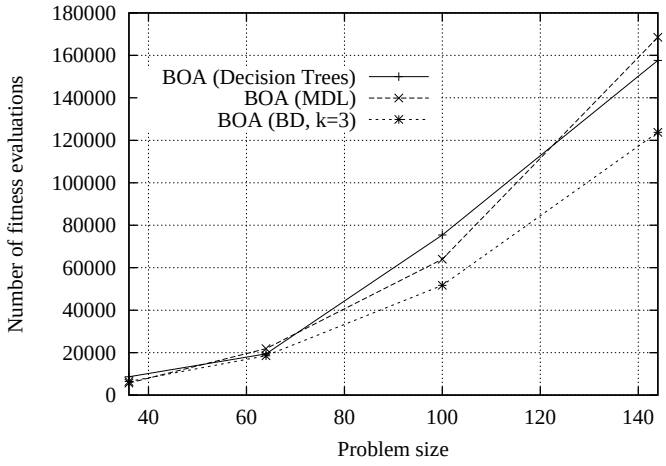


Figure 11: Results on the graph bisection of size $n = 36$, $n = 64$, $n = 100$, and $n = 144$.

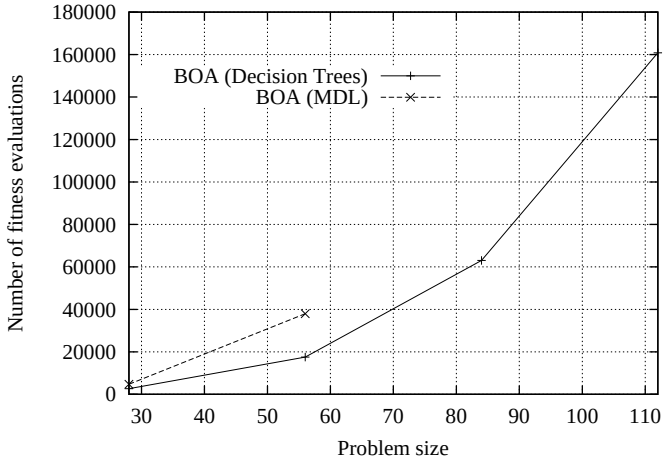


Figure 12: Results on the graph bisection of size $n = 28$, $n = 56$, $n = 84$, and $n = 112$.

standard deviation was 6483.9). Due to the lack of time we have not tested other algorithms on the same instance.

On the second instance no algorithm converged even with huge population sizes. This suggests a deceptive nature of the problem which does not seem to be solvable by combining promising partial solutions. We are currently investigating the effect of using prior information in form of the structure of the CNF to improve the performance of the BOA. An interesting observation was made with max-sat instances. All satisfiable instances (we have tested only satisfiable instances) are very easy to solve by the Davis-Putnam algorithm. Nonetheless, we believe that it is possible to construct special instances that would deceive this algorithm to local optima which would make it use exponential time to recover due to the nature of the algorithm. However, this algorithm fails for unsatisfiable instances where it must search the entire (exponential) space. This is when the BOA and other evolutionary algorithms could outperform the Davis-Putnam algorithm and reach the optimum. Moreover, it is important to note that the tested instances were created to be

Table 1: Results on the spin-glass system with $n = 121$.

Algorithm	Fitness evaluations	Standard deviation
BOA (k=2, K2 metric)	75833.33	4649.03
BOA (k=3, K2 metric)	42733.33	2993.48
BOA (k=4, K2 metric)	36606.67	2045.04
BOA (k=5, K2 metric)	48960.00	2099.85
BOA (MDL metric)	38091.67	2317.69
BOA (Decision trees)	57960.00	3109.62

difficult and were not randomly select from the space of all max-sat problem instances.

6 Summary and Conclusions

The paper has shed some light on the effects using alternative approaches to model construction in the Bayesian optimization algorithm. It presented and discussed various metrics and algorithms for network construction, and their use in the Bayesian optimization algorithm. Basic theoretical background and references to the related work were presented. Alternative settings of the BOA were tested on a number of interesting problems. It was argued why it makes sense to approach problems in this fashion. The argument was supported by empirical results. The design as well as the results of the performed experiments were discussed.

The use of local structures to represent conditional probability tables has three major advantages. First, the number of parameters required to store probabilities with a large conditional part can decrease significantly. This makes the method work more efficiently as we increase the complexity of models. Second, by using decision graphs to guide the network construction, one can discover more complicated relationships which may not be evident when directly modifying the network (e.g., see results on the caterpillar graph bisection).

Third, complexity of the models can be controlled by making prior probabilities of competing models be inversely proportional to their complexity. Our experiments suggest that setting the prior probability of a network to be inversely proportional to the number of bits required to store the parameters of the network (the frequencies) works very well. By using Bayesian scoring metric containing a complexity measure as described above, one can both (1) use prior knowledge about the problem in network construction and (2) eliminate the need for a bound on the network complexity. In this fashion one can get best of the two approaches.

Acknowledgments

The authors would like to thank David Heckerman, David Chickering, Martin V. Butz, Erick Cantú-Paz, Josef Schwarz, and Jiri Ocenasek for useful comments and help with the project.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-94-1-0103, F49620-95-1-0338, F49620-97-1-0050, and F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. Support 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 US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Martin Pelikan was partially supported by grant VEGA 1/7654/20 of Slovak Grant Agency.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U.S. Government.

References

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

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

Bosman, P. A. N., & Thierens, D. (1999). Linkage information processing in distribution estimation algorithms. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Volume I (pp. 60–67). Orlando, FL: Morgan Kaufmann Publishers, San Fransisco, CA.

Cantú-Paz, E. (2000). *Comparing selection methods of evolutionary algorithms using the distribution of fitness* (Technical Report). University of California Lawrence Livermore National Laboratory.

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

Chickering, D. M., Heckerman, D., & Meek, C. (1997). *A Bayesian approach to learning Bayesian networks with local structure* (Technical Report MSR-TR-97-07). Redmond, WA: Microsoft Research.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory.* Wiley Series in Telecommunications. New York, NY: John Wiley & Sons.

De Bonet, J. S., Isbell, C. L., & Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. In Mozer, M. C., Jordan, M. I., & Petsche, T. (Eds.), *Advances in neural information processing systems*, Volume 9 (pp. 424). The MIT Press, Cambridge.

Etxeberria, R., & Larrañaga, P. (1999, March). Global optimization using Bayesian networks. In *Second Symposium on Artificial Intelligence (CIMAF-99)* (pp. 332–339). Habana, Cuba.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (1 ed.). (pp. 421–459). Cambridge, MA: MIT Press.

Friedman, N., & Yakhini, Z. (1996). On the sample complexity of learning Bayesian networks. In Horvitz, E., & Jensen, F. (Eds.), *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)* (pp. 274–282). San Francisco: Morgan Kaufmann Publishers.

Geiger, D., & Heckerman, D. (1996). Beyond Bayesian networks: Similarity networks and Bayesian multinets. *Artificial Intelligence*, *82*, 45–74.

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

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

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

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

Mühlenbein, H. (1992). How genetic algorithms really work: I.Mutation and Hillclimbing. In Männer, R., & Manderick, B. (Eds.), *How genetic algorithms really work: I.Mutation and Hillclimbing* (pp. 15–25). Amsterdam, The Netherlands.

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

Mühlenbein, H., & Mahnig, T. (2000). Evolutionary algorithms using marginal distributions. *New Generation Computing*, *18*, 157–166.

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

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

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

Pelikan, M., & Goldberg, D. E. (2000). *Genetic algorithms, clustering, and the breaking of symmetry* (IlliGAL Report No. 2000013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1998). *Linkage problem, distribution estimation, and Bayesian networks* (IlliGAL Report No. 98013). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, Volume I (pp. 525–532). Orlando, FL: Morgan Kaufmann Publishers, San Fransisco, CA.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (2000). *Bayesian optimization algorithm, population sizing, and time to convergence* (IlliGAL Report No. 2000001). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2000). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*. In press.

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

Schwarz, J., & Ocenasek, J. (1999). Experimental study: Hypergraph partitioning based on the simple and advanced algorithms BMDA and BOA. In *Proceedings of the Fifth International Conference on Soft Computing* (pp. 124–130). Brno, Czech Republic: PC-DIR.

Schwarz, J., & Ocenasek, J. (2000). A problem-knowledge based evolutionary algorithm kboa for hypergraph partitioning. Personal communication.

Spiegelhalter, D. J., & Lauritzen, S. L. (1990). Sequential updating of conditional probabilities on directed graphical structures. *Networks*, *20*, 579–605.

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

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