The Impact of Exact Probabilistic Learning Algorithms in EDAs Based on Bayesian Networks

Carlos Echegoyen[1], Roberto Santana[1], Jose A. Lozano[1], and Pedro Larrañaga[2]

[1] Intelligent Systems Group
Department of Computer Science and Artificial Intelligence
University of the Basque Country
Paseo Manuel de Lardizabal 1, 20018 Donostia - San Sebastian, Spain
{carlos.echegoyen,roberto.santana,ja.lozano}@ehu.es
[2] Department of Artificial Intelligence, Technical University of Madrid,
28660 Boadilla del Monte, Madrid, Spain
pedro.larranaga@fi.upm.es

Summary. This paper discusses exact learning of Bayesian networks in estimation of distribution algorithms. The estimation of Bayesian network algorithm (EBNA) is used to analyze the impact of learning the optimal (exact) structure in the search. By applying recently introduced methods that allow learning optimal Bayesian networks, we investigate two important issues in EDAs. First, we analyze the question of whether learning more accurate (exact) models of the dependencies implies a better performance of EDAs. Secondly, we are able to study the way in which the problem structure is translated into the probabilistic model when exact learning is accomplished. The results obtained reveal that the quality of the problem information captured by the probability model can improve when the accuracy of the learning algorithm employed is increased. However, improvements in model accuracy do not always imply a more efficient search.

1 Introduction

In estimation of distribution algorithms (EDAs) [21, 30] linkage learning, understood as the ability to capture the relationships between the variables of the optimization problem, is accomplished by detecting and representing probabilistic dependencies using probability models. EDAs are evolutionary algorithms that do not employ classical genetic operators such as mutation or crossover. Instead, machine learning methods are used to extract relevant features of the search space. The collected information is represented using a probabilistic model which is later employed to generate new points. During the sampling or generation step, the statistical dependencies between the variables are used for the construction of the new solutions.

In EDAs, the ability of learning an accurate representation of the relationships between the variables is related to the class of probabilistic models used

Y.-p. Chen, M.-H. Lim (Eds.): Linkage in Evolutionary Computation, SCI 157, pp. 109–139, 2008.
springerlink.com

and the methods employed to learn them. One class of model that has been extensively applied in EDAs is Bayesian networks [18]. Among the benefits of EDAs that use this type of models [14, 28, 33, 45] is that the complexity of the learned structure depends on the characteristics of the data (selected individuals). Additionally, the Bayesian networks learned during the search are suitable for human interpretation, aiding the discovery of unknown information about the problem structure.

Although, in the case of EDAs that use Bayesian networks, the role of the parameters that penalize the complexity of the networks has been studied [28, 32], a detailed analysis of the accuracy of the methods used for finding the best network and its influence in the behavior of EDAs has not been conducted. An initial attempt to investigate this problem was presented in [13], where exact Bayesian learning was introduced to EDAs.

Methods that do exact Bayesian structure learning [12, 20, 42, 43] compute, given a set of data and a prespecified score (in our case, the BIC score [41]), the network structure that optimizes the score. Since the problem of learning the optimal Bayesian network is NP-hard [6], these methods set constraints on the maximum number of variables and/or cases they can deal with. Usually, dynamic programming algorithms are used to learn the structure.

In this chapter, we extend the preliminary results presented in [13] and provide evidence that the methods for learning optimal (exact) Bayesian networks can be very useful to analyze the relationship between the search space and the structure of the learned probabilistic models. The advantage of using methods that learn exact models is because alternative approximate methods commonly applied to learn the models in EDAs are very often able to find only suboptimal solutions. Therefore, using exact learning makes it easier to investigate to what extent approximate learning algorithms are responsible for the loss in accuracy in the mapping between the problem structure and the model structure. In general, an exact learning algorithm can serve as a different framework to investigate the influence of the EDA components in the ability of the probability models to capture the problem structure.

The chapter is organized as follows. In the next section, Bayesian networks are presented, the general procedures to learn these networks from data are also discussed. In Section 3, we focus on the type of search strategies used to find the Bayesian network structure. Approximate and exact learning methods are analyzed. Section 4 introduces the EBNA algorithm. In Section 5, the experimental framework and functions used to evaluate the exact and local learning methods used by EBNA are introduced. Sections 6 and 7 respectively present experimental results on the time complexity analysis and convergence reliability of the two EBNA variants. Section 8 analyzes ways for using the Bayesian networks learned by EBNA as a source of problem knowledge and presents experimental results for several functions. Work related to our proposal is analyzed in Section 9. The conclusions of our paper are presented in Section 10.

2 Bayesian Networks

2.1 Notation

Let X be a random variable. A value of X is denoted x. $\mathbf{X} = (X_1, \ldots, X_n)$ will denote a vector of random variables. We will use $\mathbf{x} = (x_1, \ldots, x_n)$ to denote an assignment to the variables. We will work with discrete variables. The joint probability mass function of $\mathbf{x}$ is represented as $p(\mathbf{X} = \mathbf{x})$ or $p(\mathbf{x})$. $p(\mathbf{x}_S)$ will denote the marginal probability distribution for $\mathbf{X}_S$. We use $p(X_i = x_i \mid X_j = x_j)$ or, in a simplified form, $p(x_i \mid x_j)$, to denote the conditional probability distribution of X_i given $X_j = x_j$.

Formally, a Bayesian network [5] is a pair $(S, \boldsymbol{\theta})$ representing a graphical factorization of a probability distribution. The structure S is a directed acyclic graph which reflects the set of conditional (in)dependencies among the variables. The factorization of the probability distribution is codified by S:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|\mathbf{pa}_i)$$

where $\mathbf{pa}_i$ denotes a value of variable $\mathbf{Pa}_i$, the parent set of X_i (variables from which there exists an arc to X_i in the graph S), while $\boldsymbol{\theta}$ is a set of parameters for the local probability distributions associated with each variable. If the variable X_i has r_i possible values, $x_i^1, \ldots, x_i^{r_i}$, the local distribution $p(x_i|\mathbf{pa}_i^j, \boldsymbol{\theta}_i)$ is an unrestricted discrete distribution:

$$p(x_i^k|\mathbf{pa}_i^j, \boldsymbol{\theta}_i) \equiv \theta_{ijk}$$

where $\mathbf{pa}_i^1, \ldots, \mathbf{pa}_i^{q_i}$ denote the values of $\mathbf{Pa}_i$ and the term q_i denotes the number of possible different instances of the parent variables of X_i. In other words, the parameter θ_{ijk} represents the probability of variable X_i being in its k-th value, knowing that the set of its parent variables is in its j-th value. Therefore, the local parameters are given by $\boldsymbol{\theta}_i = (((\theta_{ijk})_{k=1}^{r_i})_{j=1}^{q_i})$.

2.2 Learning Bayesian Networks from Data

There are different strategies to learn the structure of a Bayesian network. We focus on a method called "score + search" which is the one used in the experiments presented in this paper. In this strategy, given a set of data D and a Bayesian network whose structure is denoted by S, a value (score) which evaluates how well the Bayesian network represents the probability distribution of the database D is assigned. Different scores can be used. In this work we have used the Bayesian Information Criterion score (BIC) [41] (based on penalized maximum likelihood).

A general formula for a penalized maximum likelihood score can be written as follows:

$$\log p(D|S,\hat{\boldsymbol{\theta}}) - f(N)dim(S)$$

where $dim(S)$ is the dimension –number of parameters needed to specify the model– of the Bayesian network with a structure given by S. Thus:

$$dim(S) = \sum_{i=1}^{n} q_i(r_i - 1)$$

and $f(N)$ is a non negative penalization function. The Jeffreys-Schwarz criterion, sometimes called BIC [41], takes into account $f(N) = \frac{1}{2}\log N$. Thus the BIC score can be written as follows:

$$BIC(S,D) = \log \prod_{w=1}^{N}\prod_{i=1}^{n} p(x_{w,i}|\mathbf{pa}_i^S, \hat{\boldsymbol{\theta}}_i) - \frac{1}{2}\log N \sum_{i=1}^{n} q_i(r_i - 1) \qquad (1)$$

To find the Bayesian network that optimizes the score implies solving an optimization problem. This can be done with exhaustive or heuristic search algorithms. In Section 3, we analyze two variants for finding the Bayesian network structures. Each structure is evaluated using the maximum likelihood parameters.

2.3 Learning of the Parameters

Once the structure has been learned, the parameters of the Bayesian network are calculated using the Laplace correction:

$$\hat{\theta}_{ijk} = \frac{N_{ijk} + 1}{N_{ij} + r_i} \qquad (2)$$

where N_{ijk} denotes the number of cases in D in which the variable X_i has the value x_i^k and $\boldsymbol{Pa}_i$ has its j^{th} value, and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

3 Methods for Learning Bayesian Networks

Once we have defined a score to evaluate Bayesian networks, we have to set a search process to find the Bayesian network that maximizes the score given the data. Approximate and exact methods can be used.

3.1 Learning an Approximate Model

In practical applications, we need to find an adequate model structure as quickly as possible. Therefore, a simple algorithm which returns a good structure, even if not optimal, is preferred. An algorithm that fulfills these criteria is Algorithm B [4] which is typically used by most of Bayesian network based EDAs. Algorithm B is a greedy search which starts with an arcless structure and, at each step, adds the arc with the maximum improvement in the score. The algorithm finishes when there is no arc whose addition improves the score.

3.2 Learning the Exact Model

Since learning the Bayesian network structure is an NP-hard problem, for a long time the goal of learning exact Bayesian networks was constrained to problems with a very reduced number of variables. In [20], an algorithm for learning the exact structure in less than super-exponential complexity with respect to n is introduced for the first time.

In [42], the most efficient method so far is presented and it is used in our work. This algorithm is feasible for $n < 33$ and it was shown to learn a best network for a data set of 29 variables.

In that work, the Bayesian network structure S is defined as a vector $\mathbf{S} = (S_1, .., S_n)$ of parent sets, where S_i is the subset of $\mathbf{X}$ from which there are arcs to X_i, for example S_1 is the parent set of X_1. Another necessary concept is the variable ordering. This is simply the variables of $\mathbf{X}$ in a determined order. In this order, the i^{th} element is denoted by ord_i. Therefore, the structure $\mathbf{S} = (S_1, .., S_n)$ is said to be consistent with an ordering ord when all the parents of the node precede the node in the ordering.

Another important concept in the algorithm is the sink node. Every DAG has at least one node with no outgoing arcs, so at least one node is not a parent of any other node. These nodes are called sinks of the network.

In this algorithm, the data set D is processed in a particular way and it uses two kinds of data tables. Given $\mathbf{W} \subseteq \mathbf{X}$, first it is defined the contingency table $CT(\mathbf{W})$ to be a list of the frequencies of different data-vectors $d^{\mathbf{W}}$ in $D^{\mathbf{W}}$, where $D^{\mathbf{W}}$ is the data set for $\mathbf{W}$ variables. However, the main task is to calculate conditional frequency tables $CFT(X_i, \mathbf{W})$ that record how many times different values of the variable X_i occur together with different vectors $d^{\mathbf{W}-\{X_i\}}$ in the data.

On the other hand, many popular scores such as BIC, AIC and BDe can be decomposed to local scores:

$$score(S) = \sum_{i=1}^{n} score_i(S_i) = \sum_{i=1}^{n} score(CFT(X_i, S_i)),$$

Thus, the score of the network is the sum of the local scores that only depend on the conditional frequency table for one variable and its parents. Algorithm 1 presents the main steps of the method:

The first step is the main procedure and the only one for which data is needed. It starts by calculating the contingency table for all the variables $\mathbf{X}$ and continues calculating contingency tables for all smaller variable subsets, marginalizing variables out of the contingency table. After that, for each contingency table, the conditional frequency table is calculated for each variable appearing in the contingency table. These conditional frequency tables can then be used to calculate the local scores for any parent set given a variable. All the $n2^{n-1}$ local scores are stored in a table which will be the basis of the algorithm.

Having calculated the local scores, the best parents for X_i given a candidate set C are either the whole candidate set C itself or one of the smaller candidate

Algorithm 1. Exact learning algorithm

1. Calculate the local scores for all $n2^{n-1}$ different (variable, variable set)-pairs
2. Using the local scores, find best parents for all $n2^{n-1}$ (variable, variable set)-pairs
3. Find the best sink for all 2^n variable sets
4. Using the results from Step 3, find a best ordering of the variables
5. Find a best network using results computed in Steps 2 and 4

sets $\{C\backslash\{c\}|c \in C\}$. It must be computed for all 2^{n-1} variable sets (parent candidate sets) related with X_i.

Step 3 of the algorithm is based on the following observation: The best network G^* for a variable set W must have a sink s. As G^* is a network with the highest score, sink s must have incoming arcs from its best possible set of parents. In this way, the rest of the nodes and the arcs must form the best possible network for variables $W\backslash\{s\}$. Therefore, the best sink for $\mathbf{W}$, $sink^*(\mathbf{W})$, is the node that maximizes the sum between the local score for s and the score for the network S without node s.

When we have the best sinks for all 2^n variable sets, it is possible to yield the best ordering ord^* in reverse order. Then, for each position from $|\mathbf{X}|$ to 1, in ord_i^* we have to store the best sink for the set $\bigcup_{j=i+1}^{|\mathbf{X}|}\{ord_j^*(\mathbf{X})\}$.

Having a best ordering and a table with the best parents for any candidate set, it is possible to obtain a best network consistent with the given ordering. For the i^{th} variable in the optimal ordering, the best parents from its predecessors are picked.

More details about the algorithm can be found in [42]. We use an implementation of Algorithm 1 given by the authors[1]. The computational complexity of the algorithm is $o(n^2 2^{n-2})$. The memory requirement of the method is 2^{n+2} bytes and the disk-space requirement is $12n2^{n-1}$ bytes.

4 Estimation of Distribution Algorithms Based on Bayesian Networks

The estimation of Bayesian networks algorithm (EBNA) allows statistics of unrestricted order in the factorization of the joint probability distribution. This distribution is encoded by a Bayesian network that is learned from the database containing the selected individuals at each generation. It has been applied with good results to a variety of problems [3, 19, 23, 24, 25]. Other algorithms based

[1] The C++ code of this implementation is available from http://www.cs.helsinki.fi/u/tsilande/sw/bene/download/

Algorithm 2. EBNA$_{BIC}$

1 $BN_0 \leftarrow (S_0, \boldsymbol{\theta}^0)$ where S_0 is an arc-less DAG, and $\boldsymbol{\theta}^0$ is uniform
2 $p_0(\mathbf{x}) = \prod_{i=1}^{n} p(x_i) = \prod_{i=1}^{n} \frac{1}{r_i}$
3 $D_0 \leftarrow$ Sample M individuals from $p_0(\mathbf{x})$ and evaluate them
4 t $\leftarrow 1$
5 **do** {
6 $D_{t-1}^{Se} \leftarrow$ Select N individuals from D_{t-1}
7 $S_t^* \leftarrow$ Using a search method find one network structure according to the BIC score
8 $\boldsymbol{\theta}^t \leftarrow$ Calculate θ_{ijk}^t using D_{t-1}^{Se} as the data set
9 $BN_t \leftarrow (S_t^*, \boldsymbol{\theta}^t)$
10 $D_t \leftarrow$ Sample M individuals from BN_t and evaluate them
11 } **until** Stopping criterion is met

on the use of Bayesian networks have been proposed in [28, 33, 45]. A pseudocode of EBNA is shown in Algorithm 2.

In the experiments presented in this paper, EBNA uses truncation selection and the number of selected individuals equals half of the population. The best solution at each generation is passed to the next population, therefore, at each generation $N - 1$ new solutions are sampled. The stopping criterion is changed according to the type of experiments conducted.

5 Experimental Framework and Function Benchmark

To investigate the impact of exact learning in the behavior of Bayesian network based EDAs, we compare the EBNA versions that use the two different Bayesian network learning schemes described in Section 3. We call them EBNA-Exact and EBNA-Local.

We used three different criteria to compare the algorithms. The time complexity, the convergence reliability and the way in which probabilistic dependencies are represented in the structure of the Bayesian network. In this section, we introduce a set of functions that represent different classes of problems and which are used in the following sections to test the behavior of EDAs.

5.1 Function Benchmark

Let $u(\mathbf{x}) = \sum_{i=1}^{n} x_i$, $f(\mathbf{x})$ be a unitation function if $\forall \mathbf{x}, \mathbf{y} \in \{0,1\}^n, u(\mathbf{x}) = u(\mathbf{y}) \Rightarrow f(\mathbf{x}) = f(\mathbf{y})$. A unitation function is defined in terms of its unitation value $u(\mathbf{x})$, or in a simpler way u.

Function $OneMax$:

$$OneMax(\mathbf{x}) = \sum_{i=1}^{n} x_i = u(\mathbf{x}) \tag{3}$$

Unitation functions are also useful for the definition of a class of functions where the difficulty is given by the interactions that arise among subsets of variables. One example of this class of deceptive functions is $f_{3deceptive}$ [15]:

$$f_{3deceptive}(\mathbf{x}) = \sum_{i=1}^{i=\frac{n}{3}} f^3_{dec}(x_{3i-2}, x_{3i-1}, x_{3i}) \tag{4}$$

where f^3_{dec} is defined as:

$$f^3_{dec}(u) = \begin{cases} 0.9 \; for \; u = 0 \\ 0.8 \; for \; u = 1 \\ 0.0 \; for \; u = 2 \\ 1.0 \; for \; u = 3 \end{cases}$$

Function $SixPeaks$ is a modification of the $FourPeaks$ problem [1] and it can be defined mathematically as:

$$SixPeaks(\mathbf{x}, t) = max\{tail(0, \mathbf{x}), head(1, \mathbf{x}), tail(1, \mathbf{x}), head(0, \mathbf{x})\} + \mathcal{R}(\mathbf{x}, t) \tag{5}$$

where

$tail(b, \boldsymbol{x})$ = number of trailing b's in $\boldsymbol{x}$
$head(b, \boldsymbol{x})$ = number of leading b's in $\boldsymbol{x}$

$$\mathcal{R}(\boldsymbol{x}, t) = \begin{cases} n & if \; tail(0, \boldsymbol{x}) > t \text{ and } head(1, \boldsymbol{x}) > t \text{ or} \\ & \quad tail(1, \boldsymbol{x}) > t \text{ and } head(0, \boldsymbol{x}) > t \\ 0 & \text{otherwise} \end{cases}$$

The goal is to maximize the function. For an even number of variables this function has 4 global optima, located at the points:

$$(\overbrace{0, \ldots, 0}^{t}, 1, \ldots, 1) \quad (0, \ldots, 0, \overbrace{1, \ldots, 1}^{t}) \quad (\overbrace{1, \ldots, 1}^{t}, 0, \ldots, 0) \quad (1, \ldots, 1, \overbrace{0, \ldots, 0}^{t})$$

These points are very difficult to reach because they are isolated. On the other hand, two local optima $(0, 0, \ldots, 0)$, $(1, 1, \ldots, 1)$ are very easily reachable. The value of t was set to $\frac{n}{2} - 1$.

The $Parity$ function [8] is a simple k-bounded additively separable function that has been used to investigate the limitations of linkage learning by probabilistic modeling. It can be seen as a generalization of the XOR function and the Walsh transform. In this case we will work with the concatenated parity function

(CPF) [8]. It is said that this problem is hard for EDAs in general. The *Parity* function can be defined mathematically as:

$$parity(\mathbf{x}) = \begin{cases} C_{even} & if\ u(\mathbf{x})\ is\ even \\ C_{odd} & otherwise \end{cases}$$

where C_{even} and C_{odd} are parameters of the function. The CPF is defined as m concatenated parity sub-functions,

$$CPF(\mathbf{x}) = \sum_{i=0}^{m-1} parity(x_{ik+1}, ..., x_{ik+k}) \tag{6}$$

Where k is the size for each sub-function. As in [8], we use $k = 5$, $C_{odd} = 5$ and $C_{even} = 0$. Notice that there are 2^{n-m} solutions where the function reaches the global optima.

Function *Cuban*5 [29] is a non-separable additive function. The second best value of this function is very close to the global optimum.

$$Cuban5(\mathbf{x}) = \\ F^5_{cuban1}(s_0) + \sum_{j=0}^{m}(F^5_{cuban2}(s_{2j+1}) + F^5_{cuban1}(s_{2j+2})) \tag{7}$$

where

$s_i = x_{4i}x_{4i+1}x_{4i+2}x_{4i+3}x_{4i+4}$ and $n = 4(2m+1)+1$

$$F^3_{cuban1}(\mathbf{x}) = \begin{cases} 0.595\ for\ \mathbf{x} = 000 \\ 0.200\ for\ \mathbf{x} = 001 \\ 0.595\ for\ \mathbf{x} = 010 \\ 0.100\ for\ \mathbf{x} = 011 \\ 1.000\ for\ \mathbf{x} = 100 \\ 0.050\ for\ \mathbf{x} = 101 \\ 0.090\ for\ \mathbf{x} = 110 \\ 0.150\ for\ \mathbf{x} = 111 \end{cases}$$

$$F^5_{cuban1}(\mathbf{x}) = \\ \begin{cases} 4F^3_{cuban1}(x_1, x_2, x_3)\ if\ x_2 = x_4\ and\ x_3 = x_5 \\ 0 \qquad\qquad\qquad\qquad otherwise \end{cases} \tag{8}$$

$$F^5_{cuban2}(\mathbf{x}) = \begin{cases} u(\mathbf{x}) & for \quad x_5 = 0 \\ 0 & for\ x_1 = 0, x_5 = 1 \\ u(\mathbf{x}) - 2 & for\ x_1 = 1, x_5 = 1 \end{cases}$$

The HP protein model

In our experiments we also use a class of coarse-grained protein folding model called the hydrophobic-polar (HP) model [11].

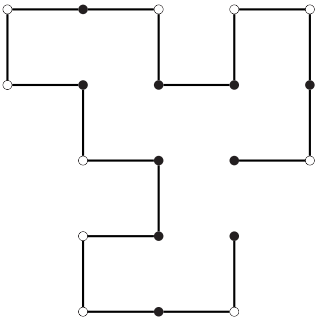


Fig. 1. An optimal solution of the HP model for sequence $HPHPPHHPHPPHPHHPPHPH$. The optimal energy corresponding to this sequence is -9.

Under specific conditions, a protein sequence folds into a native 3-d structure. The problem of determining the protein native structure from its sequence is known as the protein structure prediction problem. To solve this problem, a protein model is chosen and an energy is associated to each possible protein fold. The search for the protein structure is transformed into the search for the optimal protein configuration given the energy function.

The HP model considers two types of residues: hydrophobic (H) residues and hydrophilic or polar (P) residues. In the model, a protein is considered as a sequence of these two types of residues, which are located in regular lattice models forming self-avoided paths. Given a pair of residues, they are considered neighbors if they are adjacent either in the chain (connected neighbors) or in the lattice but not connected in the chain (topological neighbors). The total number of topological neighboring positions in the lattice (z) is called the lattice coordination number. Figure 1 shows one possible configuration of sequence $HPHPPHHPHPPHPHHPPHPH$ in the HP model.

A solution $\mathbf{x}$ can be interpreted as a walk in the lattice, representing one possible folding of the protein. We use a discrete representation of the solutions. For a given sequence and lattice, X_i will represent the relative move of residue i in relation to the previous two residues. Taking as a reference the location of the previous two residues in the lattice, X_i takes values in $\{0, 1, \ldots, z-2\}$, where $z-1$ is the number of movements allowed in the given lattice. These values respectively mean that the new residue will be located in one of the $z-1$ numbers of possible directions with respect to the previous two locations. If the encoded solution is self-intersecting, it can be repaired or penalized during the evaluation step using a recursive repairing procedure introduced in [9]. Therefore, values for X_1 and X_2 are meaningless. The locations of these two residues are fixed.

For the HP model, an energy function that measures the interaction between topological neighbor residues is defined as $\epsilon_{HH} = -1$ and $\epsilon_{HP} = \epsilon_{PP} = 0$. The HP problem consists of finding the solution that minimizes the total energy. More details about the representation and function can be found in [40].

6 Time Complexity Analysis

In our case, the time complexity analysis will refer to the study of the average number of generations needed by EBNA-Local and EBNA-Exact to find the optimum.

Experiments were conducted for three functions, $OneMax$, $f_{3deceptive}$ and $SixPeaks$. In the first function, there are no interactions between the variables. In the rest, interactions arise between variables that belong to the same definition set of the function.

To determine the average number of generations to find the optimum needed by EBNA-Local and EBNA-Exact, we start with a population of 10 individuals and the population size is increased by 10 until a maximum population size of 150 is reached. For each possible combination of function, number of variables n, and population size N, 50 experiments are conducted. For each execution of the algorithm, a maximum of 10^5 evaluations are allowed.

For the $OneMax$ function we conducted experiments for $n \in \{15, 20\}$. In order to increase the accuracy of the curves shown in the first figure, for $n = 15$ we exceptionally conducted 100 experiments. The original idea was to evaluate, under the dimension constraints imposed by the exact learning algorithm, the scalability of both EBNA versions with $n \in \{10, 12, 15, 20\}$. Nevertheless, in this work we only present the results achieved for the last two sizes. The results of the experiments for $n = 15$ are shown in Figures 2 (a) and the average results for $n = 20$ are shown in Figure 2 (b).

The analysis of Figure 2 reveals that both algorithms exhibit the same time complexity pattern. However, EBNA-Exact needs, in general, a higher number of evaluations than EBNA-Local to find the optimal solution for the first time. The difference in the number of generations is less evident when the population size approaches 150. For this simple function, it seems that the error in the learning of the model, introduced by the approximate learning algorithm, is beneficial for the search.

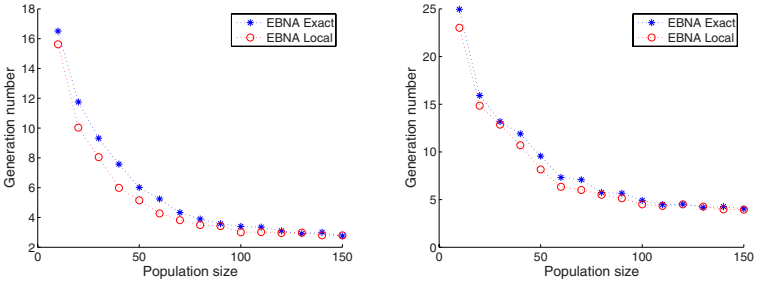


Fig. 2. Time complexity analysis for function $OneMax$,(a) $n = 15$ and (b) $n = 20$

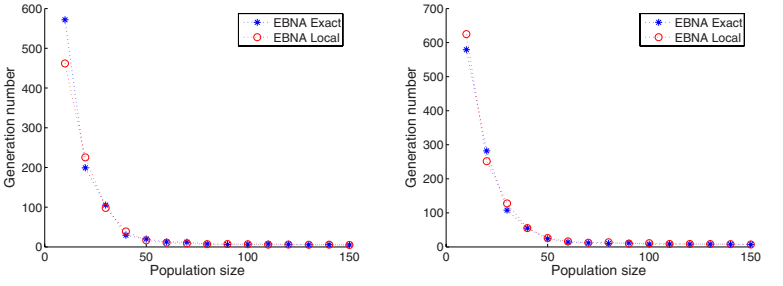


Fig. 3. Time complexity analysis for function $f_{3deceptive}$,(a) $n = 15$ and (b) $n = 18$

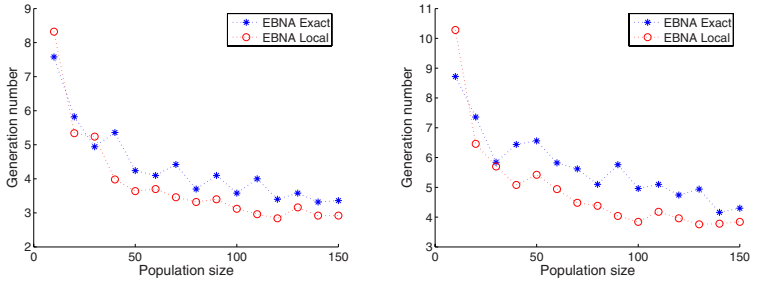


Fig. 4. Time complexity analysis for function $SixPeaks$, (a) $n = 14$ and (b) $n = 16$

In Figure 3 we can observe that both algorithms have an identical curve. For this function, and for the values of n investigated, the influence of the exact learning is not relevant. We can anticipate that the structures learned by both algorithms are similar. For this function, a small population size determines many generations are necessary to reach the optimum.

For the $SixPeaks$ function, the optimal value is reached in a significantly lower number of generations as can be appreciated in Figure 4. It can also be observed that EBNA-Local is able to reach the optimum earlier than EBNA-Exact. From this analysis we deduce that for $SixPeaks$ function, the structures learned by both algorithms could be different. We will later further analyze this behavior of the algorithms when discussing the structures of the models they learn for the $f_{3deceptive}$ and $SixPeaks$ functions.

To illustrate the complexity of the function and to study in more detail the algorithms, we introduce Figures 5 and 6 which show the total number of executions needed by each algorithm in order to succeed, i.e. to find the optimum. It should be noticed that in every run, the maximum number of evaluations is bounded by 10^5. This constraint strongly influences the behavior of the algorithms.

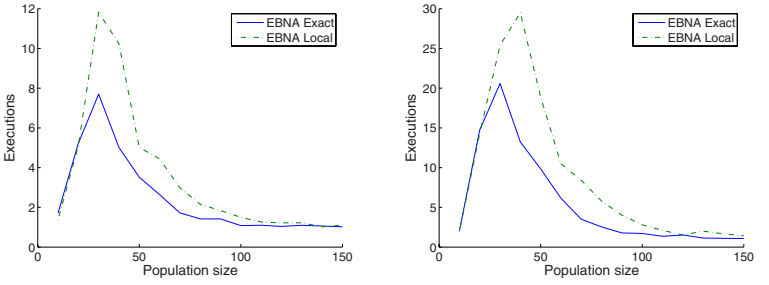


Fig. 5. Number of executions, for each population size, in order to obtain one optimal value for $f_{3deceptive}$ function, (a) $n = 15$, (b) $n = 18$

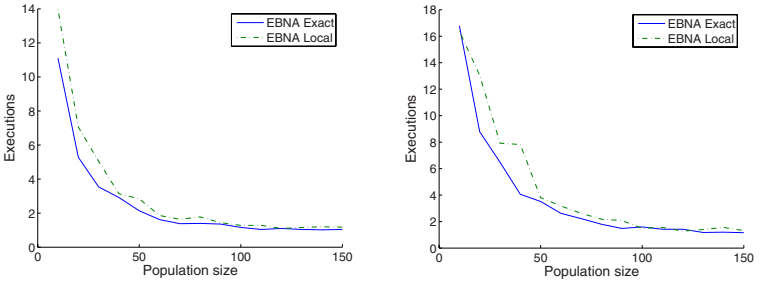


Fig. 6. Number of executions, for each population size, in order to obtain one optimal value for $SixPeaks$ function, (a) $n = 14$, (b) $n = 16$

7 Convergence Reliability

In the analysis of the convergence reliability, we focus on the critical population size needed by the EDAs to achieve a predefined convergence rate. In the experiments conducted, the goal was to determine the minimum population size needed by the two different variants of EBNA to find the optimum in 20 consecutive experiments. We investigated the behavior of the algorithms for functions $Cuban5$ ($n = 13$), $SixPeaks$ ($n \in \{10, 12, 14\}$) and $f_{3deceptive}$ ($n \in \{9, 12, 15\}$).

The algorithm begins with a population size $N = 16$ which is doubled until the optimal solution has been found in 20 consecutive experiments. The maximum number of evaluations allowed is 10^4. For each function and value of n, 25 experiments are carried out. Table 1 shows the mean and standard deviation of the critical population size found.

Table 1 shows that for function $Cuban5$, EBNA-Exact requires a slightly higher population size than EBNA-Local. The picture is drastically changed for functions $SixPeaks$ and $f_{3deceptive}$, for which EBNA-Exact needs a much smaller

Table 1. Mean, standard deviation and p-value of the critical population size for different functions and number of variables

$function$	n	$EBNA-$Exact		$EBNA-$Local		$T-Test$
		$mean$	std	$mean$	std	$p-value$
$Cuban5$	13	118.40	53.07	**109.44**	57.26	0.57
$SixPeaks$	10	**153.60**	52.26	215.04	109.11	0.014
$SixPeaks$	12	**209.92**	110.11	389.12	249.19	0.019
$SixPeaks$	14	**312.32**	133.64	604.16	318.97	0.001
$f_{3deceptive}$	9	**135.68**	38.40	168.96	60.94	0.025
$f_{3deceptive}$	12	**168.96**	60.94	261.12	86.50	0.001
$f_{3deceptive}$	15	**220.16**	58.66	296.96	95.79	0.0013

population size. This difference is particularly evident for function $SixPeaks$. Another observation is that the standard deviation of EBNA-Local is always higher than that of EBNA-Exact. Since the only difference between EBNA-Exact and EBNA-Local is in the class of algorithm used to learn the models, the difference of behaviors is due to the ability of EBNA-Exact to learn a more accurate model of the dependencies. Therefore, at least for functions $SixPeaks$ and $f_{3deceptive}$, learning a more accurate model determines a better performance of EBNA.

To determine if the population sizes obtained for each algorithm are significantly different, we have carried out a Student's t-test over the two sets of 25 population sizes for each function and value of n. In the last column of Table 1, the probability values of the test are reported.

If we consider a significance level of 0.05, we would have to reject the null hypotesis for all cases except for $Cuban5$ where there are not significant differences. An explanation of the similar behavior achieved with both algorithms for Cuban5 will be presented in the next section, where the structures of the probabilistic models learned by the algorithms are studied. Moreover, for $SixPeaks$, using the highest value of n, and for $f_{3deceptive}$, using the two highest problem sizes, the difference between the algorithms is statistically significant at the 1% level.

8 Problem-Knowledge Extraction from Bayesian Networks

The objective of this section is to show a number of ways in which knowledge about the problem structure can be extracted from the analysis of the Bayesian networks learned by EBNA. In particular, we investigate the difference between the structures learned using exact and approximate learning algorithms. We also analyze the changes in the pattern (number and type) of the dependencies captured by the algorithms during their evolution.

8.1 Probabilistic Models as a Source of Knowledge About the Problem

Although the main objective in EDAs is to obtain a set of optimal solutions, the analysis of the models learned by the algorithms during the evolution can reveal previously unknown characteristics of the problem. There is a variety of information that can be obtained from the analysis of the models. Just to cite a few examples, it could be possible to extract:

- A description of sets of dependent or interacting variables.
- Probabilistic information about most likely configurations for subsets of the problem variables which can be translated into most-probable partial solutions of the problem.
- Evidence on the existence of different types of problem symmetry.
- Identification of conflicting partial solutions in problems with frustration.
- In addition, by considering the change of the models during the evolution (a dynamical perspective), it is also possible to identify patterns in the formation of optimal structures.

A central problem in EDAs is the design of methods for extracting and interpreting this information from the models. There are a number of approaches that have been proposed to treat this issue for different classes of probabilistic models used in EDAs. We postpone a review of some of these approaches for the next section and focus now on the extraction of information from Bayesian networks. We identify three main sources of information:

1. The structure of the Bayesian network: By inspecting the topological characteristics of the graphs (e.g. most frequent arcs), we identify structural relationships between the variables.
2. The probabilistic tables of the Bayesian networks: By analyzing the probability associated to variables linked in the network, it is possible to identify promising and also poor configurations of the partial solutions.
3. Most probable configurations given the network: These are the solutions with the highest probability given the model. Thus, they condense the structural and parametrical information stored by the Bayesian network and have not necessarily been generated during the evolution of the EDA.

In this paper we focus on the analysis of network structures.

8.2 Analysis of the Bayesian Structures Learned by EBNA

To investigate the type of dependencies learned by EBNA-Exact and EBNA-Local, we saved the structures of the Bayesian networks learned during the evolutionary process by both variants of the algorithm for functions $f_{3deceptive}$, $SixPeaks$, $Cuban5$, CPF and $Protein$.

In all the following experiments, we start by running EBNA-Local and EBNA-Exact and choose 30 executions in which the optimum was found and the algorithms did not converge in the first generation. The stopping criterion is a

maximum number of 10^5 evaluations. In each of these experiments, the structures of the Bayesian networks learned in each generation are stored. From the structures, the frequency in which each arc appeared in the Bayesian network was calculated. Since we are not interested in the direction of the dependencies, we add the frequency of the two arcs that involves the same pair of variables. The matrices that store this information are called frequency matrices.

Two different ways of showing the information contained in the frequency matrices are used. The first way to represent the frequencies is using images where lighter color indicate a higher frequency. As another means to visualize the patterns of interactions, we use contour maps in which dependencies with a similar frequency are joined with lines. In this way, it is possible to identify areas of similar strength of dependency. In addition, the number of contours is a parameter that can be tuned to focus the attention on the set of the strongest dependencies.

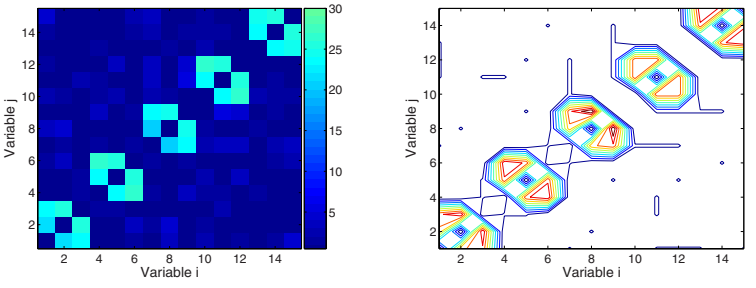
In the following, for each function and variant of EBNA employed, two figures are shown. The first figure shows the image graph of the dependencies learned by the model in the last generation and contained in the corresponding frequency matrix. The second figure shows the contour graph corresponding to a matrix that stores all the arcs learned by all the models during the evolution. We call this second matrix the cumulative frequency matrix. In order to fairly compare both algorithms using the contour figures, we normalized the frequencies of the arcs by the highest value among the two cumulative matrices learned by each algorithm. The normalized values are later discretized in ten levels. This way the contour lines refer to the same levels of frequencies.

Results for $f_{3deceptive}$ and $SixPeaks$ functions

In the initial experiments we use functions $f_{3deceptive}$ ($n = 15$) and $SixPeaks$ ($n = 16$). We relate the behavior exhibited by these functions and analyze some patterns identified in the structures of the models learned. We also try to link the behavior in both studies.

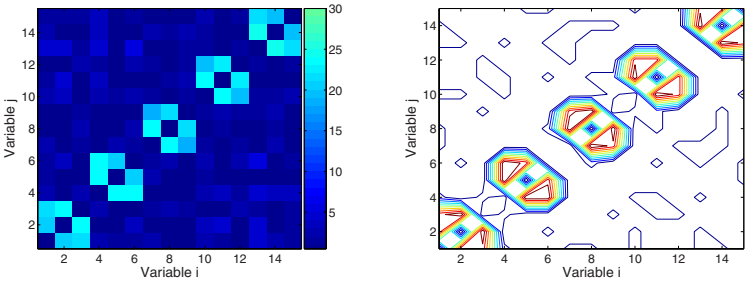
We start by using a population size of 150, which was the highest population size used on the complexity experiments shown in previous sections. Figure 7 and 8 respectively show the frequency matrices corresponding to EBNA-Local and EBNA-Exact for function $f_{3deceptive}$. Both algorithms are able to capture the dependencies corresponding to the problem interactions. This fact may explain the similar behavior exhibited in the time complexity experiments.

It can be noticed that the models include a number of additional spurious correlations which are not determined by the function structure. This is particularly evident for the EBNA-Exact algorithm and is explained by the fact that exact learning is more sensitive to the overfitting of the data when the population size is small. Therefore, we increase the population size to $N = 500$ and repeat the same experiment for this function. The frequency matrices obtained are shown in Figures 9 and 10. They reveal the effect of increasing the population size in the dependencies learned. It can be appreciated that spurious correlations



a) b)

Fig. 7. Frequency matrices calculated from the models learned by EBNA-Local for function $f_{3deceptive}$ with $N = 150$ (a) Last generation (b) All generations



a) b)

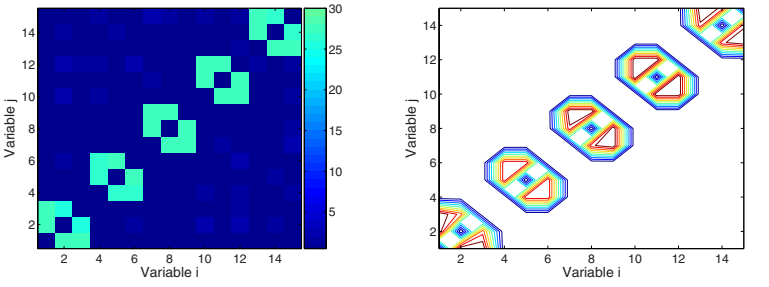
Fig. 8. Frequency matrices calculated from the models learned by EBNA-Exact for function $f_{3deceptive}$ with $N = 150$ (a) Last generation (b) All generations

have almost disappeared from the models. Both algorithms are able to learn an accurate model with a population size of $N = 500$.

We conduct a similar analysis for function *SixPeaks*. Figures 11 and 12 respectively show the frequency matrices calculated for EBNA-Local and EBNA-Exact with a population size $N = 150$. It can be seen that both algorithms are unable to learn the accurate structure. As in the case of the $f_{3deceptive}$ function, EBNA-Exact learns more spurious dependencies than EBNA-Local. This fact is specially evident in Figure 12. In this case the patterns of dependencies is spread along the matrix while dependencies learned by EBNA-Local are grouped around the diagonal. This fact may explain the better results achieved by EBNA-Local in the time complexity experiments done for this function.

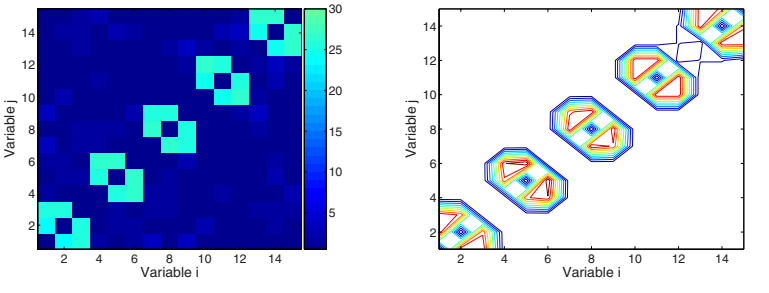
Insufficient population size might be the main reason that explains the poor quality in the mapping between the function structure and the model structure.

The experiments presented in [13] showed that EBNA-Exact was able to learn an accurate model for function *SixPeaks* with a smaller number of variables.



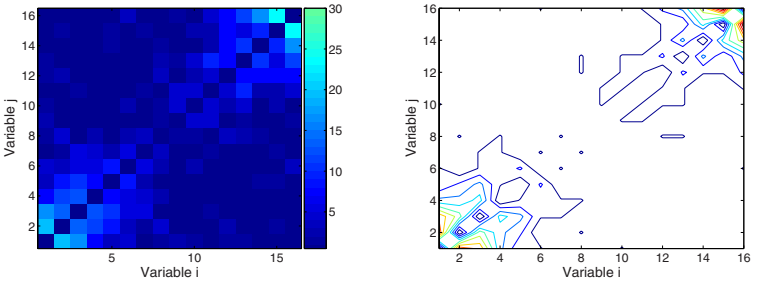
a) b)

Fig. 9. Frequency matrices calculated from the models learned by EBNA-Local for function $f_{3deceptive}$ with $N = 500$ (a) Last generation (b) All generations



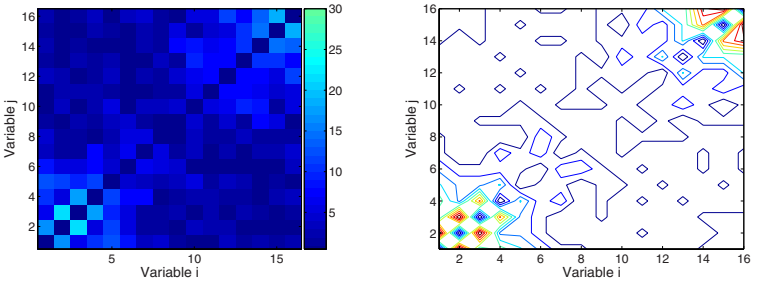
a) b)

Fig. 10. Frequency matrices calculated from the models learned by EBNA-Exact for function $f_{3deceptive}$ with $N = 500$ (a) Last generation (b) All generations



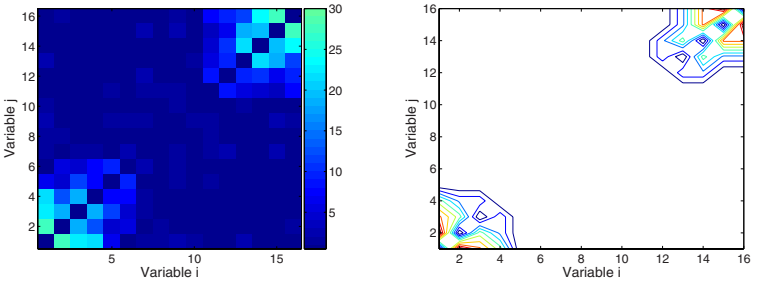
a) b)

Fig. 11. Frequency matrices calculated from the models learned by EBNA-Local for function $SixPeaks$ with $N = 150$ (a) Last generation (b) All generations



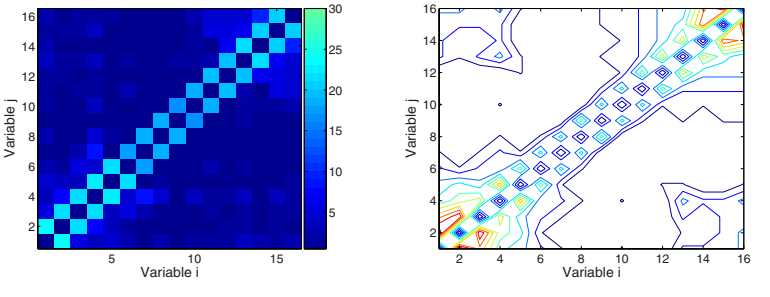
a) b)

Fig. 12. Frequency matrices calculated from the models learned by EBNA-Exact for function *SixPeaks* with $N = 150$ (a) Last generation (b) All generations



a) b)

Fig. 13. Frequency matrices calculated from the models learned by EBNA-Local for function *SixPeaks* with $N = 500$ (a) Last generation (b) All generations



a) b)

Fig. 14. Frequency matrices calculated from the models learned by EBNA-Exact for function *SixPeaks* with $N = 500$ (a) Last generation (b) All generations

Therefore, we repeat the experiment using a population size $N = 500$. Results are shown in Figures 13 and 14.

The image graph reveals that by increasing the population size EBNA-Exact is able to learn a very accurate structure. The model learned captures all the short-order dependencies of the function. This fact is corroborated by inspecting the contour graph in Figure 14 (b) where there is evidence that exact learning has gains in accuracy with respect to a smaller population size. On the other hand, EBNA-Local does not achieve a similar improvement. Furthermore, the accuracy of the approximation is lower than when a population size $N = 150$ was used, as can be seen comparing Figures 11 and 13.

*Cuban*5 function

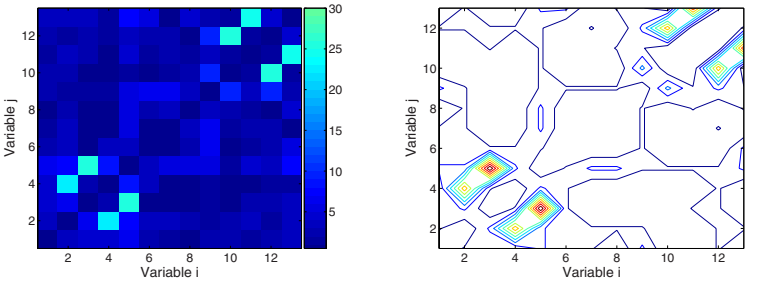
We analyze the $Cuban5$ function ($m = 1$, $n = 13$). For $m = 1$, $Cuban5$ is equal to the sum of three subfunctions:

$$Cuban5(\mathbf{x}) = F^5_{cuban1}(s_0) + F^5_{cuban2}(s_1) + F^5_{cuban1}(s_2) \tag{9}$$

The interactions are determined by two different functions, F^5_{cuban1} and F^5_{cuban2}. Therefore, we expect $Cuban5$ to exhibit a different pattern of interactions than those previously analyzed. As in previous experiments, we start with a population size $N = 150$. The frequency matrices corresponding to EBNA-Local and EBNA-Exact are respectively shown in Figures 15 and 16.

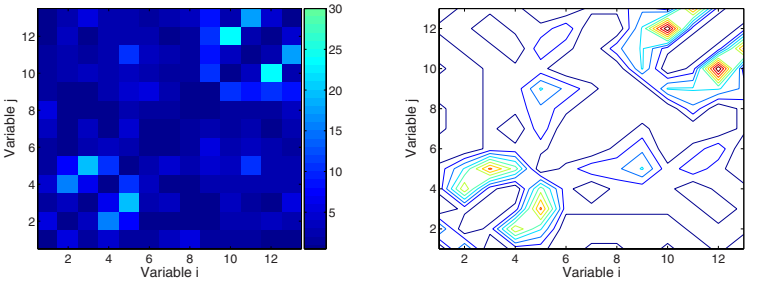
It can be seen in the images calculated from the frequency matrices of the last generation that only some of the dependencies determined by function F^5_{cuban1} are captured by both algorithms. However, the cumulative frequencies clearly show the existence of dependencies related to function F^5_{cuban2}. There are no important differences between the two EDAs.

The frequency matrices obtained by increasing the population size to $N = 500$ are shown in Figures 17 and 18. In this case, the dependencies determined by function F^5_{cuban2} are easier to recognize in the frequency matrices of the last



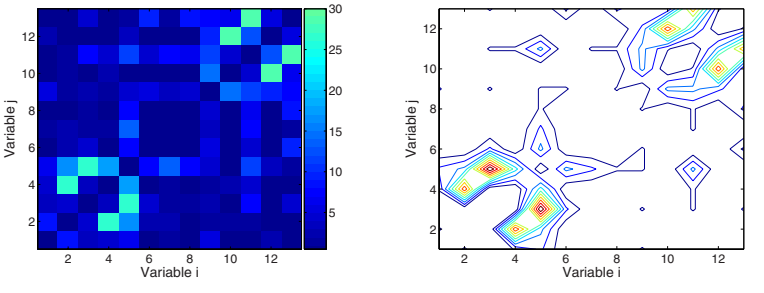
a) b)

Fig. 15. Frequency matrices calculated from the models learned by EBNA-Local for function $Cuban5$ with $N = 150$ (a) Last generation (b) All generations



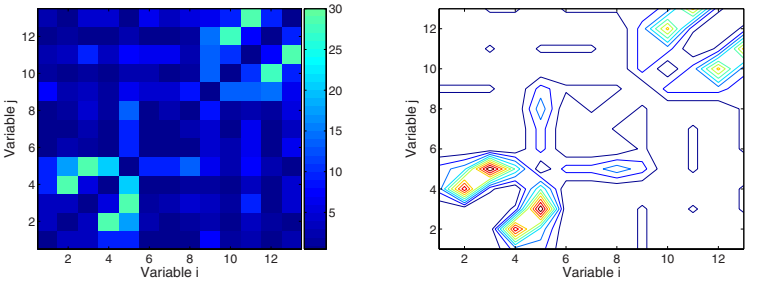
a) b)

Fig. 16. Frequency matrices calculated from the models learned by EBNA-Exact for function $Cuban5$ with $N = 150$ (a) Last generation (b) All generations



a) b)

Fig. 17. Frequency matrices calculated from the models learned by EBNA-Local for function $Cuban5$ with $N = 500$ (a) Last generation (b) All generations



a) b)

Fig. 18. Frequency matrices calculated from the models learned by EBNA-Exact for function $Cuban5$ with $N = 500$ (a) Last generation (b) All generations

generation. However, although the population sizes have grown, for this function both algorithms have learned a similar structure. This fact could explain the results achieved in the study of the convergence reliability presented in Section 7.

Results for the CPF function

The CPF function represents another interesting class of functions. It has been shown that Bayesian network based EDAs such as BOA are deceived by this function. Being CPF a decomposable function of bounded complexity, BOA has an exponential scaling for it. Furthermore, in [8] it is shown that increasing the population size does not always produce an improvement in the algorithm's behavior. Authors point to the fact that the learning algorithm used by BOA may fail to detect the higher order type of interactions that occurs in the CPF function.

We will investigate whether there are differences between exact and local learning for the CPF function with parameters: $n = 15$, $k = 5$, $c_{odd} = 5$ and $c_{even} = 0$. For these parameters, the optimum can be reached in 2^{12} different points. As a consequence, it is very likely that EBNA reaches the global solution in the first generation. On the other hand, the limitations of the exact learning algorithm do not allow to deal with a higher number of variables. Therefore, we will only analyze the models learned in the first generations of the EDAs, disregarding whether the optimum has been found or not. The models have been calculated using 30 independent experiments.

We start with a population size $N = 150$. For this value and higher values of the population size, none of the algorithms was able to recover any type of structure. However, for $N = 1000$, EBNA-Exact was able to detect some structure. The frequency matrices calculated for EBNA-Local and EBNA-Exact are shown in Figure 19.

Surprisingly, EBNA-Exact was able to recover an almost perfect structure while EBNA-Local was not. These results reveal that, for problems such as

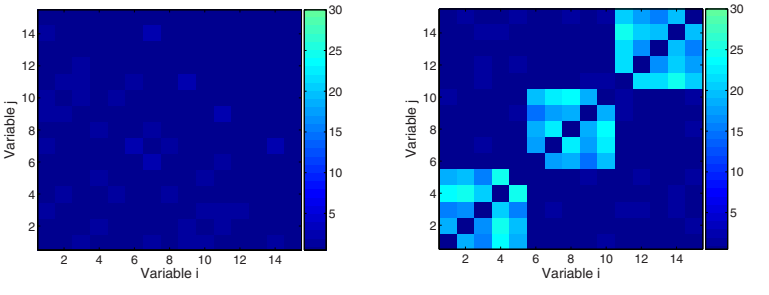


Fig. 19. Frequency matrices calculated from the models learned in the first generation of the EDAs for function CPF with $N = 1000$ (a) EBNA Local (b) EBNA-Exact

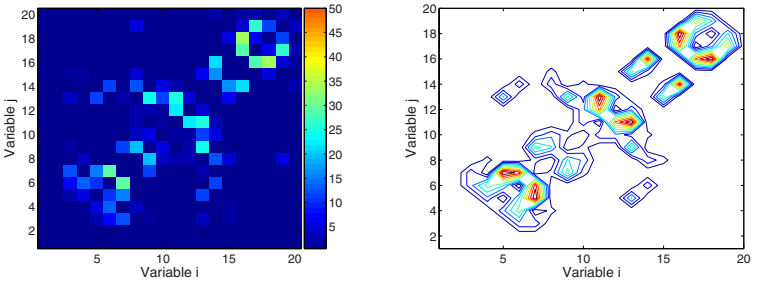
CPF, an accurate learning of the model might be essential to recover the correct structure of the problem. It also shows, that even with exact learning, the population size required to discover the problem structure is higher than for the other additive functions considered.

HP protein model

The HP model has served as a benchmark for studying different issues related with the behavior of EDAs [35, 38, 40]. It is a non-binary, non-decomposable problem for which extensive investigation using evolutionary and other heuristic algorithms have been conducted (see [10, 40] and references therein). We use one instance of the HP model to investigate the impact of exact learning. Figure 1 shows one optimal folding for the chosen sequence $HPHPPHHPHPPHPHHPPHPH$.

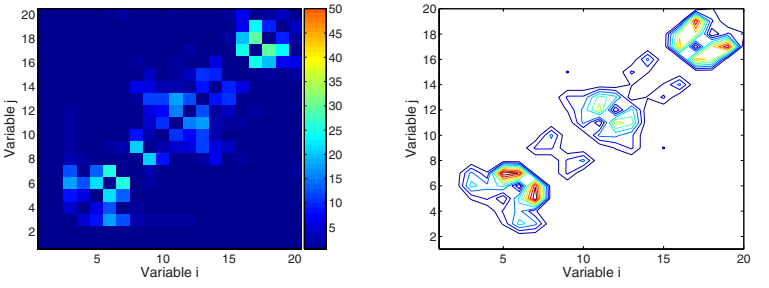
In the evaluation of the HP model, two variants are considered. In the first one, infeasible individuals are assigned a penalty. In the second variant, individuals are first repaired and after that the HP function (from now on *Protein* function) is used to evaluate them. In all the experiments conducted for the *Protein* function, $N = 200$ and 50 independent experiments of EBNA-Local and EBNA-Exact were run.

Since the *Protein* function is not decomposable, a detailed description of the problem structure is not available and we can not contrast the dependencies learned with a perfect model of the interactions. However, previous research on the application of EDAs to the HP problem [40] has shown that important dependencies between adjacent variables arise. These dependencies are in part determined by the codification used, in which each residue's position depend on the position of the previous two. Thus, the objective of our experiments is twofold. Firstly, to compare the class of models learned by EBNA-Local and EBNA-Exact. Secondly, to investigate the effect that the application of the repair mechanism has in the number and patterns of the interactions learned by the EDAs.



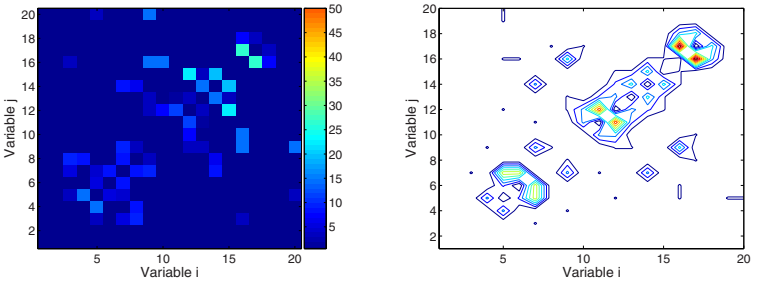
a) b)

Fig. 20. Frequency matrices calculated from the models learned by EBNA-Local for function *Protein*, repairing procedure with $N = 200$ (a) Last generation (b) All generations



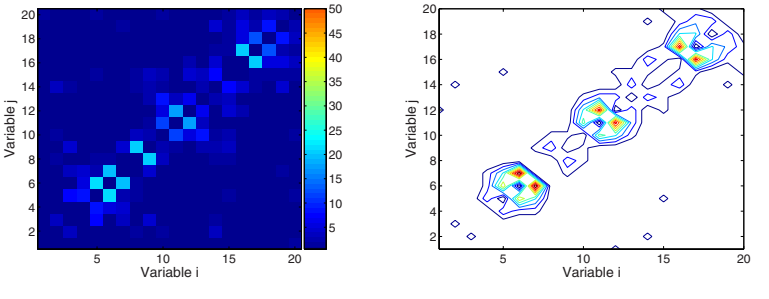
a) b)

Fig. 21. Frequency matrices calculated from the models learned by EBNA-Exact for function *Protein*, repairing procedure with $N = 200$ (a) Last generation (b) All generations



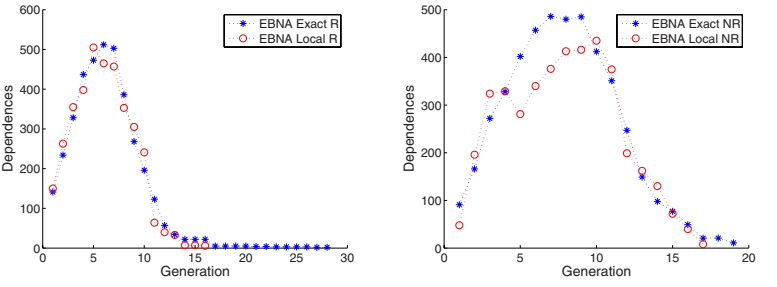
a) b)

Fig. 22. Frequency matrices calculated from the models learned by EBNA-Local for function *Protein*, without repairing procedure with $N = 200$ (a) Last generation (b) All generations



a) b)

Fig. 23. Frequency matrices calculated from the models learned by EBNA-Exact for function *Protein*, without repairing procedure with $N = 200$ (a) Last generation (b) All generations



a) b)

Fig. 24. Number of dependencies learned, by EBNA-Exact and EBNA-Local, at each generation for the *Protein* function, (a) with repairing procedure (b) without repairing procedure

Figures 20 and 21 respectively show the frequency matrices learned by EBNA-Local and EBNA-Exact when the repairing procedure is applied. Figures 22 and 23 show frequencies corresponding to the variant in which the repairing procedure is not applied.

An analysis of the figures reveal that EBNA-Exact learns a pattern of interactions more localized around the diagonal representing the dependencies between adjacent variables. The dependencies found by EBNA-Local are more spread-out, away from the diagonal.

We also observe some differences due to the application of the repairing procedure. These differences are particularly noticeable from the analysis of the contour graphs. Taking as an accuracy criterion the connectness of the adjacent variables in the problem representation, we see that repairing helps EBNA-Local to learn more accurate structures. Without repairing, the pattern of interactions is more fragmented. However, repairing does not help EBNA-Exact, which is able to recover a more connected structure without the application of the repairing procedure.

We also analyze the number of dependencies learned by the algorithms at each generation. Figures 24 (a) and (b) respectively show the sum, for the 50 experiments, of the number of dependencies learned by EBNA-Exact and EBNA-Local with and without the application of the repairing procedure.

EBNA-Local and EBNA-Exact have a similar behavior. In the initial generations, the number of dependencies learned increases until a maximum is reached and then the number of dependencies starts to diminish.

9 Related Work

In [3], an empirical comparison of EBNAs that use different learning algorithms has been presented. Also in [44] different variants of learning algorithms have been evaluated in the context of EDAs that use polytree models (a constrained

class of Bayesian networks). The use of exact learning algorithms of Bayesian networks was introduced to EDAs in [13], where preliminary results were presented.

Our work is part of an ongoing research trend that investigates the relationship between the problem structure and the class of structure learned during the search by the probabilistic models. A number of researchers have studied the most frequent dependencies learned by the probabilistic models in EDAs and analyzed their mapping with the function structure [2, 22, 27, 34, 37]. A promising related idea is the use of the dependency relationships represented by the probabilistic model to define functions with a desired degree of interactions [31].

The relationship between problem structure and dependencies is analyzed from two different perspectives in [36]. First, using Pearson's chi-square statistics as a measure of the strength of the interactions between pairs of variables in EDAs, the arousal of dependencies due to the selection operator is shown. Secondly, it is shown that for some problems, only a subset of the dependencies may be needed to solve the problem.

More recently, some work has been devoted to analyzing the way in which the different components of the EDA influence the arousal of dependencies [16] and to use the probabilistic models obtained by EDAs to speed up the solution of similar problems in the future [17]. However, the accuracy of the learning algorithm to recover the problem structure from the data has not been investigated in these papers. For instance, for the spin glass problem used as testbeds in [16], most of the dependencies found by hBOA are short dependencies between neighbors in the grid but some long range interactions also appear. We point out that the approximate learning algorithm may produce models that are only an approximate representation of the actual dependencies that arise in the population. The error introduced by the learning method in the estimation of the dependencies should also be taken into account.

10 Conclusions

In this work we have accomplished a detailed analysis of the use of exact learning of the Bayesian network structure in the study of EDAs. We have conducted systematic experiments for several functions. Results show that the type of learning algorithm (whether exact or approximate) may produce significant differences in the class of models learned and in the performance of the EBNA. This fact is important because usually Bayesian network models learned using approximate algorithms are thought to accurately reflect the dependencies that arise in the population. As the example of the CPF function illustrates, this might not be the case for functions with a particular type of higher order dependencies.

On the other hand, we have shown that whenever the size of the problem is manageable, exact learning of Bayesian networks is a more appropriate option for theoretical analysis of the probabilistic dependencies. We have shown that the analysis of the probabilistic models can reveal the effect that some EDA

components, such as repairing procedures, have in the arousal of dependencies. By using exact learning, we have confirmed the critical effect that an inadequate population size may have to capture an accurate probabilistic model.

Among the trends for future research we identify the followings:

- Design of feasible approaches to apply Bayesian network exact learning algorithms to problems with a higher number of variables. A possible alternative will treat these problems by initially identifying interacting sets of variables of manageable size and applying exact learning in each set to obtain a more accurate model of the interactions. Efficient methods for clustering the variables according to the mutual information have already been applied in EDAs [39].
- Application of more advanced techniques for extracting and visualizing the information contained in the models. As the importance of using the information contained in the probabilistic models learned by EDAs is acknowledged, it becomes more necessary to apply more advanced tecniques for information extraction and data visualization.
- Use of the most probable configurations to investigate the influence of the learning algorithms and other EDA components. Procedures that use the probabilistic models learned by EDAs either take advantage of the problem structure or use the probabilistic tables corresponding to some sets of marginal and conditional probabilities. However, information contained in the models can, in many cases, be translated into a set of most probable configurations (with their associated probabilities), which are usually not generated during the evolution. Most probable configurations can help to improve EDA behavior [26] but they could also be used to investigate the algorithms and extract relevant problem information.
- Another way to improve the results of the learning algorithms, particularly of the exact variant, in the discovery of accurate models, could be to increase the quality of the information contained in the population size. Research on this direction has been reported in [7].
- Exact learning could be used to investigate the effects that the existence of constraints, such as those imposed by repairing procedures, have in the arousal of dependencies: Constrained problems remain an important challenge for EDAs. While it is generally difficult to represent the constraints in the probabilistic model, the use of repairing procedures may introduce undesired bias in the construction of solutions. By investigating the structure of the model, it could be possible to detect the bias introduced and conceive ways of correcting it.

Finally, we emphasize that the study of the relationship between the problem structure and the dependencies captured by the probabilistic model should provide answers for the fundamental question of how to select appropriate probabilistic models to optimize a given problem in the framework of EDAs.

Acknowledgments

This work has been partially supported by the Etortek, Saiotek and Research Groups 2007-2012 (IT-242-07) programs (Basque Government), TIN2005-03824 and Consolider Ingenio 2010 - CSD2007-00018 projects (Spanish Ministry of Education and Science) and COMBIOMED network in computational biomedicine (Carlos III Health Institute).

References

1. Baluja, S., Davies, S.: Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In: Proceedings of the 14th International Conference on Machine Learning, pp. 30–38. Morgan Kaufmann, San Francisco (1997)
2. Bengoetxea, E.: Inexact Graph Matching Using Estimation of Distribution Algorithms. PhD thesis, Ecole Nationale Supérieure des Télécommunications (2003)
3. Blanco, R., Lozano, J.A.: Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. In: Larrañaga, P., Lozano, J.A. (eds.) An empirical comparison of discrete Estimation of Distribution Algorithms, pp. 163–176. Kluwer Academic Publishers, Dordrecht (2002)
4. Buntine, W.: Theory refinement on Bayesian networks. In: Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence, pp. 52–60 (1991)
5. Castillo, E., Gutierrez, J.M., Hadi, A.S.: Expert Systems and Probabilistic Network Models. Springer, Heidelberg (1997)
6. Chickering, D.M., Geiger, D., Heckerman, D.: Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research, Redmond, WA (1994)
7. Chuang, C.Y., Chen, Y.P.: On the effectiveness of distribution estimated by probabilistic model building. Technical Report: NCL-TR-2008001, Natural Computing Laboratory NCLab. Department of Computer Science. National Chiao Tung University (February 2008)
8. Coffin, D.J., Smith, R.E.: The limitations of distribution sampling for linkage learning. In: Proceedings of the 2007 Congress on Evolutionary Computation CEC 2007, pp. 364–369. IEEE Press, Los Alamitos (2007)
9. Cotta, C.: Protein structure prediction using evolutionary algorithms hybridized with backtracking. In: Mira, J., Álvarez, J.R. (eds.) IWANN 2003. LNCS, vol. 2687, pp. 321–328. Springer, Heidelberg (2003)
10. Cutello, V., Nicosia, G., Pavone, M., Timmis, J.: An immune algorithm for protein structure prediction on lattice models. IEEE Transactions on Evolutionary Computation 11(1), 101–117 (2007)
11. Dill, K.A.: Theory for the folding and stability of globular proteins. Biochemistry 24(6), 1501–1509 (1985)
12. Eaton, D., Murphy, K.: Exact Bayesian structure learning from uncertain interventions. In: Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (2007)
13. Echegoyen, C., Lozano, J.A., Santana, R., Larrañaga, P.: Exact Bayesian network learning in estimation of distribution algorithms. In: Proceedings of the 2007 Congress on Evolutionary Computation CEC 2007, pp. 1051–1058. IEEE Press, Los Alamitos (2007)

14. Etxeberria, R., Larrañaga, P.: Global optimization using Bayesian networks. In: Ochoa, A., Soto, M.R., Santana, R. (eds.) Proceedings of the Second Symposium on Artificial Intelligence (CIMAF 1999), Havana, Cuba, pp. 151–173 (1999)
15. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
16. Hauschild, M., Pelikan, M., Lima, C., Sastry, K.: Analyzing probabilistic models in hierarchical BOA on traps and spin glasses. In: Thierens, D., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2007, vol. I, pp. 523–530. ACM Press, London (2007)
17. Hauschild, M., Pelikan, M., Sastry, K., Goldberg, D.E.: Using previous models to bias structural learning in the hierarchical BOA. MEDAL Report No. 2008003, Missouri Estimation of Distribution Algorithms Laboratory (MEDAL) (2008)
18. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning 20, 197–243 (1995)
19. Inza, I., Larrañaga, P., Sierra, B.: Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. In: Larrañaga, P., Lozano, J.A. (eds.) Feature weighting for nearest neighbor by estimation of distribution algorithms, pp. 291–308. Kluwer Academic Publishers, Dordrecht (2002)
20. Koivisto, M., Sood, K.: Exact Bayesian structure discovery in Bayesian networks. Journal of Machine Learning Research 5, 549–573 (2004)
21. Larrañaga, P., Lozano, J.A. (eds.): Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic Publishers, Dordrecht (2002)
22. Lima, C.F., Pelikan, M., Goldberg, D.E., Lobo, F.G., Sastry, K., Hauschild, M.: Influence of selection and replacement strategies on linkage learning in BOA. In: Proceedings of the 2007 Congress on Evolutionary Computation CEC 2007, pp. 1083–1090. IEEE Press, Los Alamitos (2007)
23. Lozano, J.A., Larrañaga, P., Inza, I., Bengoetxea, E. (eds.): Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms. Springer, Heidelberg (2006)
24. Lozano, J.A., Sagarna, R., Larrañaga, P.: Parallel estimation of Bayesian networks algorithms. In: Evolutionary Computation and Probabilistic Graphical Models. Proceedings of the Third Symposium on Adaptive Systems (ISAS 2001), Havana, Cuba, pp. 137–144 (March 2001)
25. Mendiburu, A., Lozano, J., Miguel-Alonso, J.: Parallel implementation of EDAs based on probabilistic graphical models. IEEE Transactions on Evolutionary Computation 9(4), 406–423 (2005)
26. Mendiburu, A., Santana, R., Lozano, J.A.: Introducing belief propagation in estimation of distribution algorithms: A parallel framework. Technical Report EHU-KAT-IK-11/07, Department of Computer Science and Artificial Intelligence, University of the Basque Country (October 2007)
27. Mühlenbein, H., Höns, R.: The estimation of distributions and the minimum relative entropy principle. Evolutionary Computation 13(1), 1–27 (2005)
28. Mühlenbein, H., Mahnig, T.: Evolutionary synthesis of Bayesian networks for optimization. In: Patel, M., Honavar, V., Balakrishnan, K. (eds.) Advances in Evolutionary Synthesis of Intelligent Agents, pp. 429–455. MIT Press, Cambridge (2001)
29. Mühlenbein, H., Mahnig, T., Ochoa, A.: Schemata, distributions and graphical models in evolutionary optimization. Journal of Heuristics 5(2), 213–247 (1999)

30. Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I. Binary parameters. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 178–187. Springer, Heidelberg (1996)
31. Ochoa, A., Soto, M.R.: Linking entropy to estimation of distribution algorithms. In: Lozano, J.A., Larrañaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms, pp. 1–38. Springer, Heidelberg (2006)
32. Pelikan, M.: Hierarchical Bayesian Optimization Algorithm. Toward a New Generation of Evolutionary Algorithms. Studies in Fuzziness and Soft Computing. Springer, Heidelberg (2005)
33. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: 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 1999, Orlando, FL, vol. I, pp. 525–532. Morgan Kaufmann Publishers, San Francisco (1999)
34. Santana, R.: Estimation of distribution algorithms with Kikuchi approximations. Evolutionary Computation 13(1), 67–97 (2005)
35. Santana, R., Larrañaga, P., Lozano, J.A.: Protein folding in 2-dimensional lattices with estimation of distribution algorithms. In: Barreiro, J.M., Martín-Sánchez, F., Maojo, V., Sanz, F. (eds.) ISBMDA 2004. LNCS, vol. 3337, pp. 388–398. Springer, Heidelberg (2004)
36. Santana, R., Larrañaga, P., Lozano, J.A.: Interactions and dependencies in estimation of distribution algorithms. In: Proceedings of the 2005 Congress on Evolutionary Computation CEC 2005, Edinburgh, U.K, pp. 1418–1425. IEEE Press, Los Alamitos (2005)
37. Santana, R., Larrañaga, P., Lozano, J.A.: The role of a priori information in the minimization of contact potentials by means of estimation of distribution algorithms. In: Marchiori, E., Moore, J.H., Rajapakse, J.C. (eds.) EvoBIO 2007. LNCS, vol. 4447, pp. 247–257. Springer, Heidelberg (2007)
38. Santana, R., Larrañaga, P., Lozano, J.A.: Component weighting functions for adaptive search with EDAs. In: Proceedings of the 2008 Congress on Evolutionary Computation CEC 2008, Hong Kong. IEEE Press, Los Alamitos (accepted for publication, 2008)
39. Santana, R., Larrañaga, P., Lozano, J.A.: Estimation of distribution algorithms with affinity propagation methods. Technical Report EHU-KZAA-IK-1/08, Department of Computer Science and Artificial Intelligence, University of the Basque Country (January 2008), `http://www.sc.ehu.es/ccwbayes/technical.htm`
40. Santana, R., Larrañaga, P., Lozano, J.A.: Protein folding in simplified models with estimation of distribution algorithms. IEEE Transactions on Evolutionary Computation (to appear, 2008)
41. Schwarz, G.: Estimating the dimension of a model. Annals of Statistics 7(2), 461–464 (1978)
42. Silander, T., Myllymaki, P.: A simple approach for finding the globally optimal Bayesian network structure. In: Proceedings of the 22th Annual Conference on Uncertainty in Artificial Intelligence (UAI 2006). Morgan Kaufmann Publishers, San Francisco (2006)
43. Singh, A., Moore, A.: Finding optimal Bayesian networks by dynamic programming. Technical report, Carnegie Mellon University (June 2005)

44. Soto, M.R., Ochoa, A.: A factorized distribution algorithm based on polytrees. In: Proceedings of the 2000 Congress on Evolutionary Computation CEC 2000, La Jolla Marriott Hotel La Jolla, California, USA, July 6-9, pp. 232–237. IEEE Press, Los Alamitos (2000)
45. Soto, M.R., Ochoa, A., Acid, S., Campos, L.M.: Bayesian evolutionary algorithms based on simplified models. In: Ochoa, A., Soto, M.R., Santana, R. (eds.) Proceedings of the Second Symposium on Artificial Intelligence (CIMAF 1999), Havana, Cuba, pp. 360–367 (March 1999)