Learning Bayesian Network Structures by Searching for the Best Ordering with Genetic Algorithms

Pedro Larrañaga, Cindy M. H. Kuijpers, Roberto H. Murga, and Yosu Yurramendi

Abstract—In this paper we present a new methodology for inducing Bayesian network structures from a database of cases. The methodology is based on searching for the best ordering of the system variables by means of genetic algorithms. Since this problem of finding an optimal ordering of variables resembles the traveling salesman problem, we use genetic operators that were developed for the latter problem. The quality of a variable ordering is evaluated with the structure-learning algorithm K2. We present empirical results that were obtained with a simulation of the ALARM network.

I. INTRODUCTION

Bayesian networks (BN's) constitute a reasoning method based on probability theory. They model causal relations between events.

A BN consists of a set of nodes and a set of arcs which together constitute a directed acyclic graph (DAG). The nodes represent random variables, all of which have a finite set of states. The arcs indicate the existence of direct causal connections between the linked variables, and the strengths of these connections are expressed in terms of conditional probabilities.

To specify the probability distribution of a Bayesian network, \( P(x_1, \ldots, x_n) \), one must give prior probabilities for all root nodes (nodes without predecessors) and conditional probabilities for all other nodes, given all possible combinations of their direct predecessors. These numbers in conjunction with the DAG, specify the BN completely. The joint probability of any particular instantiation of all \( n \) variables in a BN can be calculated as follows:

\[
P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|\pi_i),
\]

where \( x_i \) represents the instantiation of the variable \( X_i \) and \( \pi_i \) represents the instantiation of the parents of \( X_i \). Excellent introductions on BN's can be found in [1]–[3].

The construction of a BN consists of two subproblems, namely of the structure learning or search for the DAG that best reflects all interdependence relations between the system variables, and of the parameter learning, i.e., the determination of the conditional probabilities belonging to the network.

In this paper we consider the problem of the automatic structure learning of BN's from a database of cases (observations). This problem is an interesting one because the construction of a BN exclusively from the information provided by an expert is time-consuming and subject to mistakes. Therefore, and due to the fact that large databases become more accessible, algorithms for automatic learning can be of great help. We are not the first to look at this problem: a considerable amount of research has been done on the induction of causal structures, BN's and other graphical models. In the structure learning of BN's often an ordering between the nodes
of the structure is assumed, in order to reduce the search space. This means that a node \( x_i \) can only have node \( x_j \) as a parent node if in the ordering node \( x_j \) comes before node \( x_i \).

We propose to search for the best ordering and we choose to do this using a genetic algorithm. For developing this algorithm, we use results of the research carried out on the application of genetic algorithms in tackling the intensively studied traveling salesman problem (TSP), since the problem of the search for an optimal ordering of system variables is not very different from the TSP. We evaluate the orderings of the variables with the K2 structure-learning algorithm of Cooper and Herskovits [4].

The structure of this paper is as follows. In Section II, we revise the most important structure-learning algorithms that were proposed in literature. Genetic algorithms are introduced in Section III. In Section IV we consider the resemblance of the problem of the search for an optimal ordering of variables and the TSP. In Section V, we explain the crossover and mutation operators that we use for our experiments. In Section VI, we describe our algorithm. Empirical results with a simulation of the ALARM network [5] are presented in Section VII. There can be seen that our algorithm is robust, for all combinations of parameters it manages to obtain results close to the evaluation of the ALARM network. In a final Section VIII concluding remarks are given.

II. RELATED WORK

A. Trees and Poly-Trees

Chow and Liu [6] show how to recover an undirected Markov tree from empirical observations using the maximum weight spanning tree algorithm.

Suzuki [7] proposes to carry out structure search using the MDL (Minimum Description Length) principle of Rissanen [8]. Suzuki focuses on tree structures, in which case his method is a generalization of the one of Chow and Liu.

Rebane and Pearl [9] showed that the algorithm of Chow and Liu can also be used for recovering the topology of a poly-tree. They also developed an algorithm for recovering the direction of the branches.

CASTLE (Causal STructures from inductive LEarning), which was developed by Acid et al. [10] learns poly-tree structures from examples, using the maximum weight spanning tree heuristic in combination with some metric to estimate the undirected graph and a conditional independence test for the determination of the direction of the branches.

B. Multiple Connected Structures

1) Assum ing an Ordering Between the Nodes: Srinivas et al. [11] proposed an algorithm for the automatic construction of sparse BN’s from information about the domain provided by an expert. The network is constructed by incrementally adding nodes. The information of the expert, together with a greedy heuristic that intends to minimize the number of arcs, guide, in each step, the search for a next node.

Herskovits and Cooper [12] developed the system KUTATÓ, which incorporates a module for constructing belief networks based on entropy calculations. KUTATÓ constructs an initial network in which all variables in the database are assumed to be marginally independent. In every step, the arc is added that, maintaining acyclicity, minimizes the entropy of the resulting network. This process continues until an entropy-based threshold is reached.

A Bayesian version of the last described algorithm was developed by the same authors. Cooper and Herskovits [4] proposed K2, an algorithm which searches for the most probable belief network structure given a database of cases. The K2 algorithm is described in detail in Section VI.

Chickering et al. [13] reviewed the BDe metric (Bayesian metric with Dirichlet priors) described by Heckerman et al. [14] under the name CH, which has a property useful for inferring causation called likelihood equivalence, which says that two networks that represent the same assertions of conditional independence have the same likelihood.

Bouckaert [15] proposed a measure for the quality of a structure based on the MDL principle, using a search algorithm similar to K2.

Larrañaga et al. [16] tackled the problem of the search for a BN structure that maximizes the metric proposed by Cooper and Herskovits with hybrid genetic algorithms.

2) Solving the Restriction of the Ordering: Bouckaert [17] presented an algorithm that manipulates the ordering of the variables with operations similar to arc reversal. These operations are only applied in case the resulting DAG represents at least the independences that were already present in the structure before the application of the operator. In this way the set of independences increases incrementally.

Singh and Valtorta [18] developed the CB algorithm (Conditional independence + Bayesian learning) with which they intended to integrate two of the existing trends in the learning of BN’s. The algorithm first uses a conditional independence test based on the \( \chi^2 \)-distribution for obtaining an ordering between the variables. Next, given this ordering, a structure is obtained by means of K2 after which, again with K2, the structures are obtained that correspond to orderings that are compatible with the partial ordering implied by the structure found with the first application of K2.

Lam and Bacchus [19] described a method for learning unrestricted multiply-connected belief networks based on the MDL principle, which permits to trade off accuracy and complexity. The method can be seen as a generalization of other approaches based on the cross entropy of Kullback and Leibler and can be interpreted from a Bayesian point of view, where the a priori probability to be assigned to a structure is inversely proportional to its complexity.

In [20] Lam and Bacchus improved the algorithm of [19], by considering partial information available about the domain.

Larrañaga et al. [21] presented a genetic algorithm that used the metric that was proposed by Cooper and Herskovits for evaluating the quality of an induced structure. They used a repair operator for converting offspring structures that were not acyclical into DAG’s.

Provan and Singh [22] proposed an algorithm called K2-AS (K2 + Attribute Selection) in which not all variables (or attributes) about which information is present are considered, but only a subset of them. That subset should maximize the predictable capacity of the network. In this way the generated networks are computationally easy to evaluate and their predictability is comparable with the networks that consider all variables.

C. Other Graphical Models

Andersen et al. [23] developed STENO, an expert system for medical diagnosis, which combines expert knowledge concerning associations between entities with knowledge generated by a statistical analysis of data relating these entities. It uses the model search strategy described by Kreiner [24].

Fung and Crawford [25] developed CONSTRUCTOR, a system which integrates techniques and concepts of the probabilistic networks, artificial intelligence and statistics, in order to induce Markov networks.

Lauritzen et al. [26] presented results of a medical diagnostic system. They compared the diagnostic power of different block recursive graphical models induced using the information criterion.
of Akaike [27], and criteria based on statistical tests. The model construction is carried out by means of backward selection.

Madigan et al. [28] proposed a Bayesian method for finding graphical models, in which they, instead of only one model, consider several good ones, combining the results from them.

Mechling and Valtorta [29] proposed an algorithm that constructs Markov networks in a similar way to CONSTRUCTOR.

Provan [30] presented an algorithm for the automatic construction of a temporal influence diagram, i.e., a union of a sequence of influence diagrams, each of which model the system during a certain interval of time in which the system is supposed to have a static behavior.

III. GENETIC ALGORITHMS

Holland [31] introduced the genetic algorithms. In these algorithms, the search space of a problem is represented as a collection of individuals. These individuals are represented by character strings, which are often referred to as chromosomes. The purpose of the use of a genetic algorithm is to find the individual from the search space with the best "genetic material." The quality of an individual is measured with an evaluation function. The part of the search space to be examined is called the population.

Roughly, a genetic algorithm works as follows. Firstly, the initial population is chosen, and the quality of this population is determined. Next, in every iteration parents are selected from the population. These parents produce children, which are added to the population. For all newly created individuals of the resulting population a probability near to zero exists that they "mutate," i.e., that they change their hereditary distinctions. After that, some individuals are removed from the population according to a selection criterion in order to reduce the population to its initial size. One iteration of the algorithm is referred to as a generation.

The operators which define the child production process and the mutation process are called the crossover operator and the mutation operator, respectively. Mutation and crossover play different roles in the genetic algorithm. Mutation is needed to explore new states and helps the algorithm to avoid local optima. Crossover should increase the average quality of the population. By choosing adequate crossover and mutation operators, the probability that the genetic algorithm provides a near-optimal solution in a reasonable number of iterations is enlarged. Under certain circumstances, the genetic algorithms evolve to the optimum with probability 1 [32]-[34].

Further descriptions of genetic algorithms can be found in [35] and [36].

IV. RESEMBLANCE TO THE TSP

The search for an optimal ordering between the variables resembles the intensively studied traveling salesman problem (TSP): a given collection of cities, determine the shortest tour that visits each city precisely once and then returns to its starting point.

Both problems are ordering problems. However, between both problems a difference exists: in the TSP, in general, only the relative order is assumed to be important while in our problem the absolute order also matters. For example, in the 6-cities TSP, in general, the string (1 2 3 4 5 6) is assumed to represent the same tour as the string (4 5 6 1 2 3). In the 6-variables ordering problem both strings represent different variable orderings. We remark that the variable ordering problem is an asymmetrical problem; the string (1 2 3 4 5 6) does not represent the same variable ordering as the string (6 5 4 3 2 1). The TSP is often assumed to be symmetrical.

Because of the similarities between our problem of finding an optimal variable ordering and the TSP, we use the results of the research carried out on the TSP with genetic algorithms. For a review on representations and operators that have been used in tackling the TSP with genetic algorithms, see [37].

We choose to use, what in relation with the TSP is called, the path representation. Therefore, we represent an ordering between the variables by a list of numbers, where the ith element of the list is a j if variable j has the ith place in the ordering. For example, the string (3 1 2) represents the ordering in which v3 is a root node, v1 has as possible parent v2, and the possible parents of v2 are v3 and v1.

The genetic operators that we use for our experiments (see Section V) have all but the AP operator already been used for tackling the TSP.

V. GENETIC OPERATORS

A. Crossover Operators

The partially-mapped crossover (PMX) [38] transmits ordering and value information from the parent strings to the offspring. A portion of one parent string is mapped onto a portion of the other parent string and the remaining information is exchanged. Consider, for example, the following two parents: (1 2 3 4 5 6 7 8) and (3 7 5 1 6 8 2 4). The PMX operator creates an offspring in the following way. It begins by selecting uniformly at random two cut points along the strings, which represent the parents. Suppose, for example, that the first cut point is selected between the third and the fourth string element, and the second one between the sixth and the seventh string element. Hence, (1 2 3 4 5 6 7 8) and (3 7 5 1 6 8 2 4). The substrings between the cut points are called the mapping sections. In our example, they define the mappings 4 <-> 1, 5 <-> 6, and 6 <-> 8. Now the mapping section of the first parent is copied into the second offspring, and the mapping section of the second parent is copied into the first offspring: offspring 1: (x x x | 1 6 8 | x x) and offspring 2: (x x x | 4 5 6 | x x). Then offspring i (i = 1, 2) is filled up by copying the elements of the ith parent. In case a number is already present in the offspring it is replaced according to the mappings. For example, the first element of offspring 1 would be a 1, like the first element of the first parent. However, there is already a 1 present in offspring 1. Hence, because of the mapping 1 <-> 4 we choose the first element of offspring 1 to be a 4. The second, third and seventh elements of offspring 1 can be taken from the first parent. However, the last element of offspring 1 would be an 8, which is already present. Because of the mappings 8 <-> 6, and 6 <-> 5, it is chosen to be a 5. Hence, offspring 1: (4 2 3 | 1 6 8 | 7 5). Analogously, we find offspring 2: (3 7 8 | 4 5 6 | 2 1). The absolute positions of some elements of both parents are preserved.

The cycle crossover (CX) [39] attempts to create an offspring from the parents where every position is occupied by a corresponding element from one of the parents. For example, consider again the parents (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1). Now we choose the first element of the offspring equal to either the first element of the first parent string or the first element of the second parent string. Hence, the first element of the offspring has to be a 1 or a 2. Suppose we choose it to be 1, (1 * * * * * * *). Now consider the last element of the offspring. Since this element has to be chosen from one of the parents, it can only be an 8 or a 1. However, if a 1 were selected, the offspring would not represent a legal individual. Therefore, an 8 is chosen, (1 * * * * * * *). Analogously, we find that the fourth and the second element of the offspring also have to be selected from the first parent, which results in (1 2 4 8 * * * *). The positions of the elements chosen up to now are said to be a cycle. Now consider the third element of the offspring. This element may choose from any of the parents. Suppose that we select it to be
from parent 2. This implies that the fifth, sixth and seventh elements of the offspring also have to be chosen from the second parent, as they form another cycle. Thus, we find the following offspring: (1 2 4 7 5 3 8). The absolute positions of on average half the elements of both parents are preserved.

The order crossover operator (OX1) [40] constructs an offspring by choosing a substring of one parent and preserving the relative order of the elements of the other parent. For example, consider the following two parent strings: (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1), and suppose that we select a first cut point between the second and the third bit and a second one between the fifth and the sixth bit. Hence, (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1). The offspring are created in the following way. Firstly, the string segments between the cut point are copied into the offspring, which gives (** | 3 4 5 | * * *) and (** | 6 8 7 | * * * ). Next, starting from the second cut point of one parent, the rest of the elements are copied in the order in which they appear in the other parent, also starting from the second cut point and omitting the elements that are already present. When the end of the parent string is reached, we continue from its first position. In our example this gives the following children: (8 7 | 3 4 5 | 1 2 6) and (4 5 | 6 8 7 | 1 2 3).

The order-based crossover operator (OX2), [41] which was suggested in connection with schedule problems, is a modification of the OX1 operator. The OX2 operator selects at random several positions in a parent string, and the order of the elements in the selected positions of this parent is imposed on the other parent. For example, consider again the parents (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1), and suppose that in the second parent the second, third, and sixth positions are selected. The elements in these positions are 4, 6 and 5 respectively. In the first parent these elements are present at the fourth, fifth and sixth positions. Now the offspring is equal to parent 1 except in the fourth, fifth and sixth positions: (1 2 3 * * * 7 8). We add the missing elements to the offspring in the same order in which they appear in the second parent. This results in (1 2 3 4 6 5 7 8). Exchanging the role of the first parent and the second parent gives, using the same selected positions, (2 4 3 8 7 5 6 1).

The position-based crossover operator (POS), [41] which was also suggested in connection with schedule problems, is a second modification of the OX1 operator. It also starts with selecting a random set of positions in the parent strings. However, this operator imposes the position of the selected elements on the corresponding elements of the other parent. For example, consider the parents (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1), and suppose that the second, third and the sixth positions are selected. This leads to the following offspring: (4 6 2 3 5 7 8) and (4 2 3 8 7 5 6 1).

The voting recombination crossover operator (VR) [42] can be seen as a p-sexual crossover operator, where p is a natural number greater than, or equal to, 2. It starts by defining a threshold, which is a natural number smaller than, or equal to, p. Next, for every \(i \in \{1, 2, \ldots, n\}\) the set of \(i\)th elements of all the parents is considered. If in this set an element occurs at least the threshold number of times, it is copied into the offspring. For example, if we consider the parents (p = 4) (1 4 3 5 2 6), (1 2 4 3 5 6), (3 2 1 5 4 6), (1 2 3 4 5 6) and we define the threshold to be equal to 3 we find (1 2 * * * * 6 7). The remaining positions of the offspring are filled with mutations. Hence, our example might result in (1 2 4 5 3 6).

The alternating-position crossover operator (AP) [43] creates an offspring by selecting alternately the next element of the first parent and the next element of the second parent, omitting the elements already present in the offspring. For example, if parent 1 is (1 2 3 4 5 6 7 8) and parent 2 is (3 5 1 2 6 8 7 2 4), the AP operator gives the following offspring (1 3 2 7 5 4 6 8) [41]. Exchanging the parents results in (3 1 7 2 5 4 6 8).

B. Mutation Operators

The displacement mutation operator (DM) (e.g., [44]) first selects a substring at random. This substring is removed from the string and inserted in a random place. For example, consider the string (1 2 3 4 5 6 7 8), and suppose that the substring (3 4 5) is selected. Hence, after the removal of the substring we have (1 2 6 7 8). Suppose that we randomly select element 7 to be the element after which the substring is inserted. This gives (1 2 6 7 3 4 5 8).

The exchange mutation operator (EM) (e.g., [45]) randomly selects two elements in the string that represents the individual and exchanges them. For example, consider the string (1 2 3 4 5 6 7 8), and suppose that the third and the fifth element are randomly selected. This results in (1 2 5 4 3 6 7 8).

The insertion mutation operator (ISM) (e.g., [44]) randomly chooses an element in the string that represents the individual, removes it from this string, and inserts it in a randomly selected place. For example, consider again the string (1 2 3 4 5 6 7 8), and suppose that the insertion mutation operator selects element 4, removes it, and randomly inserts it after element 7. The resulting offspring is (1 2 3 5 6 7 4 8).

The simple-inversion mutation operator (SIM) (e.g., [31]) selects randomly two cut points in the string that represents the individual, and it reverses the substring between these two cut points. For example, consider the string (1 2 3 4 5 6 7 8), and suppose that the first cut point is chosen between element 2 and element 3, and the second cut point between the fifth and the sixth element. This results in (1 2 5 3 6 7 8).

The inversion mutation operator (IVM) (e.g., [46]) randomly selects a substring, removes it from the string and inserts it in reversed order, in a randomly selected position. Consider again our example string (1 2 3 4 5 6 7 8), and suppose that the substring (3 4 5) is chosen, and that this substring is inserted immediately after element 7. This gives (1 2 6 7 5 4 3 8).

The scramble mutation operator (SM) (e.g., [41]) selects a random substring and scrambles the elements in it. For example, consider the string (1 2 3 4 5 6 7 8), and suppose that the substring (4 5 6 7) is chosen. This might result in (1 2 3 5 6 7 4 8).

VI. PROPOSED APPROACH

Our approach is based on joining the genetic algorithms and the algorithm K2 (see Fig. 1). We search for a near-optimal ordering between the variables, with a genetic algorithm that creates new variable orderings by means of the crossover and mutation operators described in the previous section. The quality of an ordering is the evaluation of the BN structure that K2 creates from it.

K2 is an algorithm that creates and evaluates a BN from a database of cases once an ordering between the system variables is given. For the evaluation of the network that it constructs, the formula of Cooper and Herskovits is used.

K2 searches, given a database D for the BN structure B∗, with maximal F(B∗, D), where F(B∗, D) is as described in the following theorem proved in [4].

Theorem: Let Z be a set of n discrete variables, where a variable \(x_i\) in Z has \(r_i\) possible value assignments: \((v_{1i}, \ldots, v_{r_i})\). Let D be a database of cases of \(m\) cases, where each case contains a value assignment for each variable in Z. Let B be a BN structure containing just the variables in Z. Each variable \(x_i\) in B has a set of parents, which are represented with a list of variables \(\pi_i\). Let \(w_{ij}\) denote the jth unique instantiation of \(x_i\) relative to B. Suppose there are \(q_i\) such unique instantiations of \(x_i\). Define \(N_{i,j}k\) to be the number of cases in D in which variable \(x_i\) has the value \(v_{ik}\) and \(x_i\) is instantiated as \(w_{ij}\). Then \(N_{i,j} = \sum_{k=1}^{q_i} N_{i,j}k\). If given a BN model,
Algorithm K2

INPUT: A set of \( n \) nodes, an ordering on the nodes, an upper bound \( u \) on the number of parents a node may have, and a database \( D \) containing \( m \) cases.

OUTPUT: For each node, a printout of its parent nodes.

BEGIN K2
FOR \( i := 1 \) TO \( n \) DO
BEGIN
\( \pi_i := 0; \)
\( P_{old, i} := g(i, \pi_i); \)
OKToProceed := TRUE
WHILE OKToProceed AND \( |\pi_i| < u \) DO
BEGIN
Let \( z \) be the node in \( \text{Pred}(x_i) \) - \( \pi_i \) that maximizes \( g(i, \pi_i \cup \{z\}); \)
\( P_{new} := g(i, \pi_i \cup \{z\}); \)
IF \( P_{new} > P_{old, i} \) THEN
BEGIN
\( P_{old, i} := P_{new}; \)
\( \pi_i := \pi_i \cup \{z\} \)
END
ELSE OKToProceed := FALSE;
END
WRITE('Node: \( i \), Parents of this node: \( \pi_i \);
END;
END K2.

Fig. 1. The K2 algorithm.

The cases occur independently and the density function \( f(B_S|D) \) is uniform, then it follows that

\[
P(B_S|D) = P(B_S) \prod_{i=1}^{n} g(i, \pi_i),
\]

where

\[
g(i, \pi_i) = \prod_{j=1}^{n} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{\pi_i} N_{ijk}^{-1}.
\]

The K2 algorithm assumes that an ordering on the variables is available and that, a priori, all structures are equally likely. It searches, for every node, the set of parent nodes that maximizes \( g(i, \pi_i) \). K2 is a greedy heuristic. It starts by assuming that a node does not have parents, after which in every step it adds incrementally that parent whose addition most increases the probability of the resulting structure. K2 stops adding parents to the nodes when the addition of a single parent can not increase the probability. Obviously, this approach does not guarantee the selection of a structure with the highest probability.

A possible improvement of K2 could be the determination of the best combination of at most \( u \) parent nodes in which the number of searches to be carried out for a node \( j \) would increase from \( \prod_{i=1}^{n} (n - j - i) \) to \( \sum_{i=1}^{n} (n - j - i) \).

For our experiments, we let the K2 algorithm only construct networks which nodes have at most 4 parent nodes. The genetic algorithm we use, is an algorithm based on the principles of GENITOR, which was developed by Whiteley [47]. In every generation two orderings are selected for crossover, where the probability of an ordering to be selected depends on the rank of its objective function value. The newly created offspring substitutes, in case it is better, the worst ordering in the population.

The stop criterion is based on the definition of convergence of a population formulated by De Jong [48]. We say that a gene has converged at level \( \alpha \), if this gene has the same value in at least an \( \alpha \) of the individuals in the population. A population converges at level \( \beta \), if at least a \( \beta \) of the genes has converged. We choose \( \alpha \) and \( \beta \) to be equal to 95 and 100, respectively. This convergence criterion does not always guarantee the termination of the algorithm. Therefore, we decide that the population has also converged if in a certain number of subsequent iterations the average fitness of the population has not improved.

VII. RESULTS OF THE EXPERIMENTS

We study the behavior of the algorithm described with respect to the different combinations of crossover and mutation operators of Section V.

If we consider the genetic algorithm as a 7-tuple GA \((\lambda, a_2, a_3, a_4, p_c, p_m, a_r)\) where \( \lambda \) is the population size, \( a_2 \) is the selection criterion, \( a_3 \) the crossover operator, \( a_4 \) the mutation operator, \( p_c \) crossover probability, \( p_m \) mutation rate, \( a_r \) the reduction criterion for reducing the population to its original size, then we can describe our algorithm as follows: \( \lambda = 10, 50; a_2 = \text{based on the rank of the objective function}; a_3 = \text{AP, CX, OX1, OX2, PMX, POS, VR}; a_4 = \text{DM, EM, ISM, IVM, SIM, SM}; p_c = 1; p_m = 0.01; a_r = \text{elitist}.\)

For all \( 84 (2 \times 7 \times 6) \) parameter combinations to be considered we carry out 20 searches.

For the experiments we use a simulation, consisting of the 3000 first cases obtained by Herskovits [49], of the ALARM network, which was designed by Beinlich et al. [5] for modeling a problem in a medical field. The objective function which expresses the quality of the structures is the natural logarithm of the \textit{a posteriori} probability of the database of cases, given the structure to be evaluated, following the formula of Cooper and Herskovits [4].

The best and average evaluations as well as the accompanying standard deviations obtained with the different combinations of

<table>
<thead>
<tr>
<th>AP</th>
<th>CX</th>
<th>OX1</th>
<th>OX2</th>
<th>PMX</th>
<th>POS</th>
<th>VR</th>
</tr>
</thead>
<tbody>
<tr>
<td>14,456</td>
<td>14,419</td>
<td>14,442</td>
<td>14,431</td>
<td>14,469</td>
<td>14,430</td>
<td>14,453</td>
</tr>
<tr>
<td>14,576</td>
<td>14,433</td>
<td>14,537</td>
<td>14,475</td>
<td>14,523</td>
<td>14,478</td>
<td>14,486</td>
</tr>
<tr>
<td>84</td>
<td>14</td>
<td>59</td>
<td>28</td>
<td>35</td>
<td>36</td>
<td>22</td>
</tr>
<tr>
<td>650</td>
<td>4124</td>
<td>317</td>
<td>606</td>
<td>276</td>
<td>791</td>
<td>795</td>
</tr>
<tr>
<td>14,434</td>
<td>14,423</td>
<td>14,483</td>
<td>14,430</td>
<td>14,485</td>
<td>14,458</td>
<td>14,453</td>
</tr>
<tr>
<td>14,567</td>
<td>14,430</td>
<td>14,552</td>
<td>14,482</td>
<td>14,552</td>
<td>14,486</td>
<td>14,501</td>
</tr>
<tr>
<td>63</td>
<td>15</td>
<td>48</td>
<td>35</td>
<td>36</td>
<td>26</td>
<td>33</td>
</tr>
<tr>
<td>658</td>
<td>4009</td>
<td>266</td>
<td>580</td>
<td>212</td>
<td>764</td>
<td>1685</td>
</tr>
<tr>
<td>14,475</td>
<td>14,423</td>
<td>14,454</td>
<td>14,435</td>
<td>14,454</td>
<td>14,430</td>
<td>14,450</td>
</tr>
<tr>
<td>14,573</td>
<td>14,430</td>
<td>14,554</td>
<td>14,473</td>
<td>14,511</td>
<td>14,472</td>
<td>14,498</td>
</tr>
<tr>
<td>97</td>
<td>14</td>
<td>58</td>
<td>21</td>
<td>38</td>
<td>22</td>
<td>36</td>
</tr>
<tr>
<td>794</td>
<td>4483</td>
<td>353</td>
<td>748</td>
<td>454</td>
<td>1139</td>
<td>719</td>
</tr>
<tr>
<td>14,465</td>
<td>14,417</td>
<td>14,472</td>
<td>14,441</td>
<td>14,446</td>
<td>14,434</td>
<td>14,456</td>
</tr>
<tr>
<td>14,581</td>
<td>14,437</td>
<td>14,543</td>
<td>14,478</td>
<td>14,546</td>
<td>14,466</td>
<td>14,493</td>
</tr>
<tr>
<td>76</td>
<td>15</td>
<td>35</td>
<td>26</td>
<td>57</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>350</td>
<td>4417</td>
<td>388</td>
<td>692</td>
<td>244</td>
<td>946</td>
<td>918</td>
</tr>
<tr>
<td>14,508</td>
<td>14,423</td>
<td>14,492</td>
<td>14,441</td>
<td>14,516</td>
<td>14,444</td>
<td>14,464</td>
</tr>
<tr>
<td>14,653</td>
<td>14,437</td>
<td>14,561</td>
<td>14,500</td>
<td>14,584</td>
<td>14,518</td>
<td>14,525</td>
</tr>
<tr>
<td>58</td>
<td>14</td>
<td>39</td>
<td>29</td>
<td>45</td>
<td>50</td>
<td>29</td>
</tr>
<tr>
<td>612</td>
<td>3964</td>
<td>385</td>
<td>572</td>
<td>216</td>
<td>733</td>
<td>1109</td>
</tr>
<tr>
<td>14,542</td>
<td>14,423</td>
<td>14,487</td>
<td>14,452</td>
<td>14,510</td>
<td>14,459</td>
<td>14,468</td>
</tr>
<tr>
<td>14,647</td>
<td>14,435</td>
<td>14,564</td>
<td>14,515</td>
<td>14,576</td>
<td>14,510</td>
<td>14,516</td>
</tr>
<tr>
<td>72</td>
<td>14</td>
<td>57</td>
<td>40</td>
<td>39</td>
<td>31</td>
<td>90</td>
</tr>
<tr>
<td>699</td>
<td>3928</td>
<td>397</td>
<td>438</td>
<td>257</td>
<td>503</td>
<td>825</td>
</tr>
</tbody>
</table>
genetic operators for the population sizes 10 and 50 are presented in the Tables I and II, respectively. If we order the crossover operators with respect to their average evaluations, from best to worst we find: CX, OX2, POS, VR, PMX, AP for population size 10 and CX, OX2, POS, PMX, AP, VO for population size 50. Noticeable is that as the average evaluation increases, the standard deviation also grows. Ordering the mutation operators in the same way, we obtain: SM, DM, SIM, IVM, EM, ISM for population size 10 and DM, ISM, SIM, IVM, EM, ISM for population size 50. For the crossover operators, we obtain the same result.

As can be observed in the Tables I and II, none of the best orderings obtained in the searches is able to improve the evaluation of this initial ordering. For population size 50, however, the worst best evaluation obtained is $-1.4442 \times 10^4$, while 4 combinations give orderings that are statistically significant of $p < 0.0001$ for both $A = 10$ as well as for $A = 50$. For the mutation operators, statistically significant differences are found between the system variables since the method is based on searching for the optimal ordering of variables. For this search we have proposed a genetic algorithm that uses the K2 algorithm for evaluating the orderings and that creates new offspring orderings by applying the genetic operators that were already used in the genetic tackling of the TSP.

The empirical results obtained are comparable with the results we presented in [16], where we also tackled the structure learning of BN’s, however, assuming an ordering between the variables.

It would be interesting to see which results would be obtained if the best orderings found with the method described in this paper were used as an input for an order-assuming (genetic) algorithm for learning the structure of BN’s.

The authors thank G. F. Cooper for providing his simulation of the ALARM network.

REFERENCES


