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, $P(x_1, \cdots, x_n)$, 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:

$$P(x_1, \cdots, x_n) = \prod_{i=1}^{n} P(x_i|\pi_i),$$

where x_i represents the instantiation of the variable X_i and π_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) Assuming 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 continuous 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 Herkovits 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 χ^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 Herkovits 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): given a 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 v_3 is a root node, v_1 has as possible parent v_3, and the possible parents of v_2 are v_3, and v_1.

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 $\leftrightarrow$ 1, 5 $\leftrightarrow$ 6, and 6 $\leftrightarrow$ 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 $\leftrightarrow$ 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 $\leftrightarrow$ 6, and 6 $\leftrightarrow$ 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 $*$ $*$ $*$ $*$ $*$ $*$ 8). 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 we 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 6 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: (1 4 6 2 3 5 7 8) and (4 2 3 8 7 6 5 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 $i \in \{1, 2, \cdots, n\}$ the set of ith 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 $(1\ 2\ x\ x\ x\ 6)$. 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 7 5 1 6 8 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 4 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_{S^*} with maximal $P(B_S, D)$, where $P(B_S, D)$ 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: $(v_{i1}, \cdots, v_{ir_i})$. Let D be a database of cases of m cases, where each case contains a value assignment for each variable in Z. Let B_S denote a BN structure containing just the variables in Z. Each variable x_i in B_S has a set of parents, which are represented with a list of variables π_i. Let w_{ij} denote the jth unique instantiation of π_i relative to D. Suppose there are q_i such unique instantiations of π_i. Define N_{ijk} to be the number of cases in D in which variable x_i has the value v_{ik} and π_i is instantiated as w_{ij}. Let $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$. 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
      $\pi_i := 0$;
      $P_{\text{old}} := g(i, \pi_i)$:
      OKToProceed := TRUE
      WHILE OKToProceed AND $|\pi_i| < u$ DO
        BEGIN
          Let z be the node in $\text{Pred}(x_i) - \pi_i$ that maximizes $g(i, \pi_i \cup \{z\})$;
          $P_{\text{new}} := g(i, \pi_i \cup \{z\})$;
          IF $P_{\text{new}} > P_{\text{old}}$ THEN
            BEGIN
              $P_{\text{old}} := P_{\text{new}}$;
              $\pi_i := \pi_i \cup \{z\}$
            END
          ELSE OKToProceed := FALSE;
        END;
      WRITE('Node:', x_i, 'Parents of this node:', π_i)
    END;
END K2.

Fig. 1. The K2 algorithm.

the cases occur independently and the density function $f(B_P|B_S)$ is uniform, then it follows that

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

where

$$g(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!$$

□

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 case the number of searches to be carried out for a node j would increase from $\prod_{i=1}^{u}(n-j-i)$ to $\sum_{i=1}^{u}\binom{n-j-1}{i}$.

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 Whitley [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 α, if this gene has the same value in at least an α of the individuals in the population. A population converges at level β, if at least a β of the genes has converged. We choose α and β 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.

TABLE I
POPULATION SIZE 10. FOR ALL COMBINATIONS OF OPERATORS, RESPECTIVELY: THE BEST EVALUATION, THE AVERAGE EVALUATION, THE ACCOMPANYING STANDARD DEVIATION AND THE AVERAGE NUMBER OF EVALUATIONS BEFORE CONVERGENCE

	AP	CX	OX1	OX2	PMX	POS	VR
DM	14,456*	14,419	14,442	14,431	14,469	14,430	14,453
	14,576*	14,433	14,537	14,475	14,523	14,478	14,486
	84	14	59	28	35	36	22
	650	4124	317	606	276	791	795
EM	14,434	14,423	14,483	14,430	14,485	14,458	14,453
	14,567	14,439	14,552	14,482	14,552	14,486	14,501
	63	15	48	35	36	26	33
	658	4009	266	580	212	764	1685
ISM	14,475	14,423	14,454	14,435	14,454	14,430	14,450
	14,573	14,436	14,554	14,473	14,511	14,472	14,498
	97	14	58	21	38	22	36
	754	4483	353	748	454	1139	719
IVM	14,465	14,417	14,472	14,441	14,446	14,434	14,456
	14,581	14,437	14,543	14,478	14,546	14,466	14,493
	76	15	35	26	57	23	24
	550	4417	388	692	244	946	918
SIM	14,508	14,423	14,492	14,441	14,516	14,444	14,464
	14,653	14,437	14,561	14,500	14,584	14,518	14,525
	58	14	39	35	45	50	29
	812	3984	385	572	216	733	1109
SM	14,542	14,423	14,487	14,452	14,510	14,459	14,468
	14,647	14,435	14,564	14,515	14,576	14,510	14,516
	72	14	57	40	39	31	30
	699	3928	397	438	257	503	825

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 $(\lambda, a_2, a_3, a_4, p_c, p_m, a_7)$ where λ 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_7 the reduction criterion for reducing the population to its original size, then we can describe our algorithm as follows: λ = 10, 50; a_2 = based on the rank of the objective function; a_3 = AP, CX, OX1, OX2, PMX, POS, VR; a_4 = DM, EM, ISM, IVM, SIM, SM; $p_c = 1$; $p_m = 0.01$; a_7 = *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 Beinlinch *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 *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 II
POPULATION SIZE 50. FOR ALL COMBINATIONS OF OPERATORS, RESPECTIVELY: THE BEST EVALUATION, THE AVERAGE EVALUATION, THE ACCOMPANYING STANDARD DEVIATION AND THE AVERAGE NUMBER OF EVALUATIONS BEFORE CONVERGENCE

	AP	CX	OX1	OX2	PMX	POS	VR
DM	14,422*	14,422	14,422	14,422	14,423	14,423	14,424
	14,441*	14,428	14,443	14,433	14,436	14,430	14,451
	19	10	15	13	14	11	11
	7921	13,447	6569	3862	4350	4049	10,052
EM	14,422	14,423	14,424	14,423	14,423	14,423	14,424
	14,449	14,425	14,447	14,436	14,444	14,437	14,446
	17	4	11	13	15	14	13
	6529	14,047	6685	3489	3614	3842	9406
ISM	14,422	14,423	14,417	14,423	14,423	14,423	14,424
	14,447	14,428	14,448	14,432	14,437	14,435	14,445
	17	10	14	12	14	13	12
	7782	13,336	7148	3742	4331	3905	9872
IVM	14,423	14,422	14,417	14,423	14,417	14,423	14,424
	14,447	14,425	14,442	14,433	14,433	14,439	14,442
	22	6	15	12	13	13	13
	8355	15,467	7331	3944	4683	3898	9445
SIM	14,426	14,423	14,427	14,423	14,424	14,423	14,433
	14,512	14,424	14,457	14,442	14,446	14,439	14,460
	39	1	13	11	15	12	15
	3397	12,321	4848	3240	3165	3639	8161
SM	14,442	14,417	14,430	14,423	14,427	14,423	14,432
	14,518	14,426	14,455	14,437	14,453	14,442	14,455
	41	9	15	12	15	12	12
	3498	15,022	5050	3113	2983	3242	7910

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, OX1, AP for population size 10 and CX, OX2, POS, PMX, OX1, VR, AP 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: DM, ISM, IVM, EM, SM, SIM for $\lambda = 10$ and IVM, DM, ISM, EM, SIM, SM for $\lambda = 50$.

If we apply the Kruskal–Wallis test for comparing the behavior of the crossover operators, statistically significant differences are found ($p < 0.0001$) for both $\lambda = 10$ as well as for $\lambda = 50$. For the mutation operators we obtain the same result.

For all operators considered, the performance of the algorithm becomes better as the population size grows. For the crossover operators, however, this tendency is stronger than for the mutation operators.

The evaluation found for the structure induced by the K2 algorithm when this algorithm is applied to the order that was used for creating the database of cases is $-1.4412e04$.

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 $-1.4442e04$, while 4 combinations give orderings the structure of which is $-1.4417e04$.

In the Tables I and II also the convergence velocity of the algorithm is represented. Ranking the crossover operators from the fastest to the slowest, we find: PMX, OX1, OX2, AP, POS, VR, CX for $\lambda = 10$ and OX2, POS, PMX, AP, OX1, VR, CX for $\lambda = 50$. For the mutation operators, we find: SM, DM, SIM, IVM, EM, ISM for $\lambda = 10$ and SIM, SM, EM, ISM, DM, IVM for $\lambda = 50$. We observe that the CX operator, which gives the best results, implies a slow convergence, while the OX2 operator, which is the second best operator, results in a considerably faster algorithm. However, we also see that the CX operator only needs a small population size to give good results while the other crossover operators need larger population sizes. With respect to the convergence velocity of the mutation operators, we see that the SM operator, which is one of the fastest ones, gives the worst results.

VIII. CONCLUDING REMARKS

We have presented a method for structure learning of BN's from a database of cases with which it is not necessary to assume an ordering 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 that we presented in [16], where we also tackled the structure learning of BN's with genetic algorithms, 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.

ACKNOWLEDGMENT

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

REFERENCES

[1] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Mateo, CA: Morgan Kaufmann, 1988.
[2] R. E. Neapoli, *Probabilistic Reasoning in Expert Systems. Theory and Algorithms.* New York: Wiley, 1990.
[3] F. V. Jensen, "Introduction to Bayesian networks," Dept. of Mathematics and Computer Science, Univ. of Aalborg, Denmark, Tech. Report IR 93–2003, 1993.
[4] G. F. Cooper and E. A. Herskovits, "A Bayesian method for the induction of probabilistic networks from data," *Mach. Learning,* vol. 9, no. 4, pp. 309–347, 1992.
[5] I. A. Beinlinch, H. J. Suermondt, R. M. Chavez, and G. F. Cooper, "The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks," in *Proc. 2nd Europ. Conf. on Artificial Intelligence in Medicine,* 1989, pp. 247–256.
[6] C. K. Chow and C. N. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Trans. Inform. Theory,* vol. 14, no. 3, pp. 462–467, 1968.
[7] J. Suzuki, "A construction of Bayesian networks from databases based on an MDL principle," in *Proc. 9th Conf. Uncertainty in Artificial Intelligence,* 1993, pp. 266–273.
[8] J. Rissanen, "Modeling by shortest data description," *Automatica,* vol. 14, pp. 465–471, 1978.
[9] G. Rebane and J. Pearl, "The recovery of causal poly-trees from statistical data," in *Uncertainty in Artificial Intelligence 3,* 1989, pp. 175–182.
[10] S. Acid, L. M. de Campos, A. González, R. Molina, and N. Pérez de la Blanca, "Learning with CASTLE," *Symbolic and Quantitative Approaches to Uncertainty, Lectures Notes in Comput. Sci. 548,* R. Kruse and P. Siegel Eds. Berlin: Springer-Verlag, 1991, pp. 99–106.
[11] S. Srinivas, S. Russell, and A. Agogino, "Automated construction of sparse Bayesian networks from unstructured probabilistic models and domain information," in *Uncertainty in Artificial Intelligence 5,* Windsor, Ontario, Canada, 1990, pp. 295–308.
[12] E. Herskovits and G. Cooper, "KUTATÓ: An entropy-driven system for construction of probabilistic expert systems from databases," Knowledge Systems Laboratory, Medical Computer Science, Stanford Univ., Stanford, CA, Rep. KSL-90-22, 1990.

[13] D. M. Chickering, D. Geiger, and D. Heckerman, "Learning Bayesian networks: Search methods and experimental results," in *Preliminary Papers 5th Int. Workshop on Artificial Intelligence and Statistics,* 1995, pp. 112–128.

[14] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," in Microsoft, Technical Report MSR-TR-94-09, 1994.

[15] R. R. Bouckaert, "Properties of Bayesian belief networks learning algorithms," in *Proc. 10th Annual Conf. Uncertainty in Artificial Intelligence,* Washington, 1994, pp. 102–109.

[16] P. Larrañaga, R. H. Murga, M. Poza, and C. M. H. Kuijpers, "Structure learning of Bayesian networks by hybrid genetic algorithms," in *Preliminary Papers 5th Int. Workshop on Artificial Intelligence and Statistics,* 1995, pp. 310–316.

[17] R. R. Bouckaert "Optimizing causal orderings for generating DAG's from data," in *Proc. 8th Conf. Uncertainty in Artificial Intelligence,* 1992, pp. 9–16.

[18] M. Singh and M. Valtorta, "An algorithm for the construction of Bayesian network structures from data," in *Proc. 9th Conf. Uncertainty in Artificial Intelligence,* Washington, DC, 1993, pp. 259–265.

[19] W. Lam and F. Bacchus, "Learning Bayesian belief networks. An approach based on the MDL principle," *Computational Intelligence,* vol. 10, no. 4, 1994.

[20] ——, "Using causal information and local measures to learn Bayesian networks," in *Proc. 9th Conf. Uncertainty in Artificial Intelligence,* Washington, DC, 1993, pp. 243–250.

[21] P. Larrañaga, M. Poza, Y. Yurramendi, R. H. Murga, and C. M. H. Kuijpers, "Structure learning of Bayesian networks by genetic algorithms: A performance analysis of control parameters," *IEEE Trans. Pattern Anal. Mach. Intell.*, in press.

[22] G. M. Provan and M. Singh, "Learning Bayesian networks using feature selection," in *Preliminary Papers 5th Int. Workshop on Artificial Intelligence and Statistics,* FL, 1995, pp. 450–456.

[23] L. R. Andersen, J. H. Krebs, and J. D. Andersen, "STENO: An expert system for medical diagnosis based on graphical models and model search," *J. Appl. Stat.,* vol. 18, no. 1, pp. 139–153, 1991.

[24] S. Kreiner, "On tests of conditional independence," Statistical Research Unit, University of Copenhagen, Res. Rep. 89/14, 1989.

[25] R. M. Fung and S. L. Crawford, "Constructor: A system for the induction of probabilistic models," in *Proc. AAAI,* Boston, MA, 1990, pp. 762–769.

[26] S. L. Lauritzen, B. Thiesson, and D. J. Spiegelhalter, "Diagnostic systems created by model selection methods—A case study," in *Preliminary Papers 4th Int. Workshop on Artificial Intelligence and Statistics,* 1993, pp. 93–105.

[27] H. Akaike, "New look at the statistical model identification," *IEEE Trans. Automat. Contr,* vol. 19, pp. 716–722, 1974.

[28] D. Madigan, A. E. Raffery, J. C. York, J. M. Bradshaw, and R. G. Almond, "Strategies for graphical model selection," in *Preliminary Papers 4th Int. Workshop on Artificial Intelligence and Statistics,* 1993, pp. 331–336.

[29] R. Mechling and M. Valtorta, "PaCCIN: A parallel constructor of Markov networks," in *Preliminary Papers 4th Int. Workshop on Artificial Intelligence and Statistics,* 1993, pp. 405–410.

[30] G. M. Provan, "Model selection for diagnosis and treatment using temporal influence diagrams," in *Preliminary Papers 5th Int. Workshop on Artificial Intelligence and Statistics*, 1995, pp. 469–480.

[31] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: The Univ. of Michigan Press, 1975.

[32] U. K. Chakraborty and D. G. Dastidar, "Using reliability analysis to estimate the number of generations to convergence in genetic algorithms," *Inform. Proc. Lett.,* vol. 46, no. 4, pp. 199–209, 1993.

[33] A. E. Eiben, E. H. L. Aarts, and K. M. van Hee, "Global convergence of genetic algorithms: An infinite Markov chain analysis," *Computing Science Notes,* Eindhoven Univ. of Tech., 1990.

[34] G. Rudolph, "Convergence analysis of canonical genetic algoritms," submitted to *IEEE Trans. Neural Networks.*

[35] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[36] L. Davis, Ed., *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991.

[37] P. Larrañaga, C. M. H. Kuijpers, and R. H. Murga, "Evolutionary algorithms for the travelling salesman problem: A review of representations and operators," submitted to *Artif. Intell. Rev.*

[38] D. E. Goldberg and J. R. Lingle, "Alleles, loci and the traveling salesman problem," in *Proc. Int. Conf. Genetic Algorithms and Their Applications,* Pittsburgh, PA, 1985, pp. 154–159.

[39] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the TSP," in *Proc. 2nd Int. Conf. on Genetic Algorithms and Their Applications,* Cambridge, MA, 1987, pp. 224–230.

[40] L. Davis, "Applying adaptive algorithms to epistatic domains," in *Proc. Int. Joint Conf. on Artificial Intelligence,* Los Angeles, CA, 1985, pp. 162–164.

[41] G. Syswerda, "Schedule optimization using genetic algorithms," in [36], pp. 332–349.

[42] H. Mühlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proc. 3rd Int. Conf. on Genetic Algorithms,* Arlington, VA, 1989, pp. 416–421.

[43] P. Larrañaga, C. M. H. Kuijpers, M. Poza, and R. H. Murga, "Optimal decomposition of Bayesian networks by genetic algorithms," Dept. of Com. Science and Art. Intel., Univ. of the Basque Country, Int. Rep. EHU-KZAA-IKT-3-94, 1994.

[44] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin: Springer-Verlag, 1992.

[45] W. Banzhaf, "The "molecular" traveling salesman," *Biol. Cybern.,* vol. 64, pp. 7–14, 1990.

[46] D. B. Fogel, "A parallel processing approach to a multiple traveling salesman problem using evolutionary programming," in *Proc. 4th Annual Parallel Processing Symp.*, Fullerton, CA, 1990, pp. 318–326.

[47] D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," in *Proc. 3rd Int. Conf. on Genetic Algorithms,* Arlington, VA, 1989, pp. 116–121.

[48] K. A. de Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. Dissertation, Univ. of Michigan, 1975.

[49] E. H. Herskovits, "Computer based probabilistic-network construction," Doctoral Dissertation, Dept. Medical Information Sciences, Stanford University, Stanford, CA, 1991.