An Efficient Data Mining Method for Learning Bayesian Networks Using an Evolutionary Algorithm Based Hybrid Approach

Man Leung Wong • Kwong Sak Leung

Department of Computing and Decision Sciences, Lingnan University, Tuen Mun, Hong Kong
Department of Computer Science and Engineering, The Chinese University of Hong Kong
mlwong@ln.edu.hk • ksleung@cse.cuhk.edu.hk

ABSTRACT

Given the explosive growth of data collected from current business environment, data mining can potentially discover new knowledge to improve managerial decision making. This study proposes a novel data mining approach that employs an evolutionary algorithm to discover knowledge represented in Bayesian networks. The approach is applied successfully to handle the business problem of finding response models from direct marketing data. Learning Bayesian networks from data is a difficult problem. There are two different approaches to the network learning problem. The first one uses dependency analysis, while the second one searches good network structures according to a metric. Unfortunately, both approaches have their own drawbacks. Thus, we propose a novel hybrid algorithm of the two approaches, which consists of two phases, namely, the Conditional Independence (CI) test and the search phases. In the CI test phase, dependency analysis is conducted to reduce the size of the search space. In the search phase, good Bayesian network models are generated by using an evolutionary algorithm. A new operator is introduced to further enhance the search effectiveness and efficiency. In a number of experiments and comparisons, the hybrid algorithm outperforms MDLEP, our previous algorithm which uses Evolutionary Programming (EP) for network learning, and other network learning algorithms. We then apply the approach to two data sets of direct marketing and compare the performance of the evolved Bayesian networks obtained by the new algorithm with those by MDLEP, the logistic regression models, the naïve Bayesian classifiers, and the tree-augmented naïve Bayesian network classifiers (TAN). In the comparison, the new algorithm outperforms the others.
(*Evolutionary Computation; Evolutionary Programming; Data Mining; Bayesian Networks*)

Footnote page

[1] We use 5,000 generations as the termination criterion.

[2] Since each DAG conforms to a topological ordering, the densely connected graph will have $n(n-1)/2$ edges.

[3] We reference the variable names as provided in [1]

[4] We use the g++ compiler with "-O2" optimization level.

[5] We have attempted different population sizes and found that the best performance is obtained when the population size is 500.

[6] Since PheGT$_2^R$ applies a steady-state genetic algorithm, ANG is estimated by first storing the number of individuals examined until the best-so-far solution is obtained, and then dividing the number by the population size.

[7] Since the population size of PheGT$_2^R$ is larger than those of HEA and MDLEP, their ANG values should be compared with caution.

[8] Since BNPC does not use the score-and-search approach, AFS and AIS are not applicable.

[9] Since BNPC and WinMine Toolkit execute on the Windows environment, their execution time cannot compare with that of HEA and MDLEP.

[10] ANG is not applicable because BNPC and WinMine Toolkit are not evolutionary algorithms.

[11] AME is not applicable because BNPC does not use the score-and-search approach.

[12] Since BNPC and WinMine are deterministic, they are executed once.

[13] Since WinMine Toolkit does not use the MDL score, AFS and AIS cannot compare with those of HEA and MDLEP.

[14] AME is not available in WinMine Toolkit.

[15] The odd ratio is the ratio of the probabilities of the event happening to not happening.

1. Introduction

Conventional business research is a process in which data are analyzed manually to explore the relationships among various factors defined by the researcher. Even with powerful computers and versatile statistical software, many hidden and potentially useful relationships may not be recognized by the analyst. Nowadays, such problems are more acute as many businesses are capable of generating and collecting a huge amount of data in a relatively short period. The explosive growth of data requires a more efficient way to extract useful knowledge. Thus, business research is a major area for applying data mining that aims at discovering novel, interesting, and useful knowledge from databases [2]. Through data mining, researchers can discover complex relationships among various factors and extract meaningful knowledge to improve the efficiency and quality of managerial decision making. In this paper, we propose a novel data mining approach that employs an evolutionary algorithm to discover knowledge represented in Bayesian networks and apply the approach to handle the business problem of finding response models from direct marketing data.

Recently, some researchers have employed evolutionary algorithms for data mining. Au, Chan, and Yao [3] proposed an algorithm, called data mining by evolutionary learning (DMEL), that induces classification rules for predicting the likelihood of each classification made. They performed several experiments to show that DMEL can discover interesting rules effectively. Moreover, they applied DMEL to a large database with 100,000 records to learn rules for churn prediction. Zhou et al. [4] employed gene expression programming (GEP) to learn classification rules. They evaluated their approach on several benchmark databases and demonstrated that accurate and compact rules can be induced. Cano et al. [5] compared four evolutionary and some non-evolutionary instance selection algorithms. The experimental results suggested that the evolutionary instance selection algorithms outperform the non-evolutionary ones. Atkinson-Abutridy et al. [6] proposed a novel approach for knowledge discovery from texts. The approach uses natural language techniques and genetic algorithms to generate novel explanatory hypotheses.

Bayesian networks are popular within the community of uncertainty in artificial intelligence. A Bayesian network is a graphical representation that depicts conditional independence among random variables in the domain and encodes the joint probability distribution [7]. With a network at hand, probabilistic inference can be performed to predict the outcome of some variables based on the observations of others. In light of this, Bayesian networks are widely used in diagnostic and classification systems. For example, MUNIN is used for diagnosing diseases in muscles and nerve, and PATHFINDER is used for diagnosing lymph node diseases [8]. Besides, they are also used in information retrieval [9] and printer troubleshooting [10].

Typically, a Bayesian network is constructed by eliciting knowledge from domain experts. To reduce imprecision due to subjective judgments, researchers start to be interested in constructing a Bayesian network from collected data or past observations in the domain. Recently, there is also increasing interest in applying Bayesian networks for data mining [11, 12, 13, 14, 15]. In the literature, there are two main approaches

to this network learning problem [16]. The first one is the dependency analysis approach [16, 1]. Since a Bayesian network describes conditional independence, we could make use of dependency test results to construct a Bayesian network that conforms to our findings. The second one, called the score-and-search approach [17, 18, 19], uses a metric to evaluate a candidate network structure. With the metric, a search algorithm is employed to find a network structure which has the best score. Thus, the learning problem becomes a search problem. Unfortunately, the two approaches both have their own drawbacks. For the former approach, an exponential number of dependency tests have to be performed. Moreover, some test results may be inaccurate [1]. For the latter approach, since the search space is huge, some Bayesian network learning algorithms [17] adopt greedy search heuristics which may easily make the algorithms get stuck in a local optimum [18].

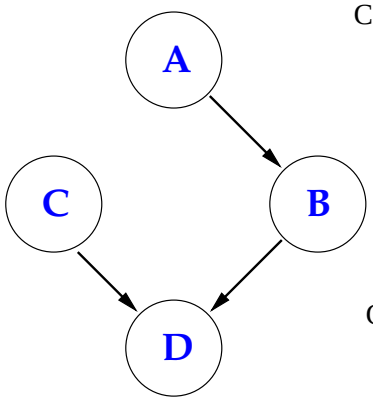
In this work, a hybrid approach is developed for the network learning problem. Simply put, dependency analysis results are used to reduce the search space of the score-and-search process. With such reduction, the search process would take less time for finding the optimal solution. Together with the introduction of a new operator and some modifications of our previous work, MDLEP [20], we call our new approach HEA (Hybrid Evolutionary Algorithm). We have conducted a number of experiments and compared HEA with MDLEP and other network learning algorithms. The empirical results illustrate that HEA outperforms these algorithms. Moreover, it is found that HEA executes much faster than MDLEP which is very important for real-life applications. HEA is also found to have the best results in a real-life application of direct marketing amongst similar state-of-the-art approaches. Some preliminary results of this work have been reported in [21, 22, 23, 24].

This paper is organized as follows. In Section 2, we present the backgrounds of Bayesian networks and the MDL metric. Different methods of applying evolutionary computation to learn Bayesian networks are presented in Section 3. In Section 4, we describe our new algorithm in detail. In Section 5, we present a comparison among the new algorithm and different network learning algorithms together with an analytical study. In Section 6, we apply the approach to two data sets of direct marketing and compare the performance of the evolved Bayesian networks obtained by HEA and MDLEP, the logistic regression models, the naïve Bayesian classifiers [25, 26], and the tree-augmented naïve Bayesian network classifiers (TAN) [25]. We then conclude the paper in Section 7.

2. Backgrounds

2.1 Bayesian Networks

A Bayesian network, G, has a directed acyclic graph (DAG) structure. As shown in Figure 1, each node in the graph corresponds to a discrete random variable in the domain. An edge, $X \leftarrow Y$, on the graph, describes a parent and child relation in which X is the child and Y is the parent. All parents of X constitute the



CPT of B

	A='F'	A='T'
B='F'	0.23	0.6
B='T'	0.77	0.4

CPT of D

	B='F', C='F'	B='F', C='T'	B='T', C='F'	B='T', C='T'
D='F'	0.65	0.54	0.78	0.01
D='T'	0.35	0.46	0.22	0.99

Figure 1: A Bayesian network example.

parent set of X which is denoted by Π_X. In addition to the graph, each node has a conditional probability table (CPT) specifying the probability of each possible state of the node given each possible combination of states of its parents. If a node contains no parent, the table gives the marginal probabilities of the node [7].

Since Bayesian networks are founded on the idea of conditional independence, it is necessary to give a brief description here. Let U be the set of variables in the domain and P be the joint probability distribution of U. Following Pearl's notation [7], a conditional independence (CI) relation is denoted by $I(X, Z, Y)$ where X, Y, and Z are disjoint subsets of variables in U. Such notation says that X and Y are conditionally independent given the *conditioning set*, Z. Formally, a CI relation is defined as [7]:

$$P(x \mid y, z) = P(x \mid z) \quad \text{whenever} \quad P(y, z) > 0, \tag{1}$$

where x, y, and z are any value assignments to the set of variables X, Y, and Z respectively. A CI relation is characterized by its *order*, which is simply the number of variables in the conditioning set Z.

By definition, a Bayesian network encodes the joint probability distribution of the domain variables, $U = \{N_1, \ldots, N_n\}$:

$$P(N_1, \ldots, N_n) = \prod_i P(N_i \mid \Pi_{N_i}). \tag{2}$$

2.2 Bayesian Network Learning

As mentioned before, researchers treat the network learning problem in two very different ways. They are called the dependency analysis and the search-and-scoring approaches respectively.

2.2.1 The Dependency Analysis Approach

The dependency analysis approach includes the algorithms in [16], [27], and [1]. It takes the view that Bayesian networks depict conditional independence relations among the variables. Hence, the approach tries to construct a Bayesian network using dependency information obtained from the data. Typically, the existence of a *perfect map* is presumed for a given distribution P [1]. In other words, it is assumed that there exists a Bayesian network, G, that captures all the conditional independence relations implied by P. Consequently, this approach constructs a network G by testing the validity of any independence assertions $I(X, Z, Y)$. If the statement $I(X, Z, Y)$ is supported by the data, it follows that X should be d-separated with Y by Z in G; otherwise, X is not d-separated with Y by Z [28, 7] .

The validity of an independence assertion $I(X, Z, Y)$ is tested by performing a conditional independence (CI) test. Statistical hypothesis testing procedure could be used in the CI test [29, 30, 1]. To begin with, the conditional independence assertion (i.e. $I(X, Z, Y)$) is modeled as the *null hypothesis*. Suppose that we use the likelihood-ratio χ^2 test and the χ^2 statistics is calculated by:

$$G^2 = -2 \sum \text{observed} * \log(\text{observed}/\text{expected}). \tag{3}$$

Simply put, the statistics calculates the discrepancies between the real occurrence, *observed*, and the expected count followed from the hypothesis, *expected* over every distinct events. In our case, because $I(X, Z, Y)$ implies:

$$\begin{aligned} P(x, y, z) &= P(x \mid y, z)\, P(y, z) \\ &= P(x \mid z)\, P(y, z) \qquad \text{(by equation 1)}, \end{aligned}$$

the statistics is computed by:

$$G^2 = -2 \sum_{x,y,z} P(x, y, z) \log \frac{P(x, y, z)}{P(y, z) P(x \mid z)}. \tag{4}$$

Suppose that the number of possible instantiations of the variables X, Y, and Z are respectively v_X, v_Y, and v_Z, G^2 follows a χ^2 distribution with $(v_X - 1) \times (v_Y - 1) \times v_Z$ degree of freedom. Checking our computed G^2 against the distribution, we obtain the p-value [31]. If the p-value is less than a predefined *cutoff value* α, the test shows strong evidence to reject the hypothesis; otherwise, the hypothesis cannot be rejected.

For example, the SGS algorithm [1] begins with a completely connected undirected graph. In other words, dependence between every pair of variables is assumed. Then, the CI tests between all pairs of connected nodes are conducted. When two nodes X and Y are found to be conditionally independent given Z, the undirected edge between them is removed so that $I(X, Z, Y)$ is not violated. When no more edges could be removed, the undirected edges in the graph are oriented according to some rules which conform with the conditional independence relations discovered previously. This produces the final Bayesian network.

In general, there are three problems in the dependency analysis approach. First, it is difficult to determine whether two nodes are dependent. Spirtes et al. stated that [1] "In general, two variables X and Y may be conditionally dependent given a set Z while independent on the subset or superset of Z." In the worst case, all possible combinations of the conditioning set need to be examined which would require an exponential number of tests. Second, results from CI test may not be reliable for high order CI tests when the size of the conditioning set is large [32, 1]. Hence, for algorithms that require high order CI tests, the results may be inaccurate. Third, because a network is constructed in a step by step manner, the construction algorithm may be *unstable* in the sense that an earlier mistake during construction is consequential [33, 1]. Moreover, this implies that the order of testing the CI relations is important, which will be a concern when one pursues for the optimal performance.

2.2.2 The Search-and-scoring Approach

The second approach is called the search-and-scoring approach. Recalling that a Bayesian network encodes a joint probability distribution (equation 2), we could derive a measure for assessing the goodness of such encoding. For instance, the measure could be derived from Bayesian statistics, information theory or the Minimum Description Length (MDL) principle [34]. Though their theoretical foundations are different, some studies [35, 36] show that different metrics are asymptotically equivalent under certain conditions.

Since we employ the MDL metric [19] in our work, we take it as an example for illustration. Basically, the metric is derived from information theory and incorporates the idea of the Minimum Description Length principle. The metric has two components: the network description length and the data description length. An optimal network is the one that minimizes the sum of the two components.

Formally, let $U = \{N_1, \ldots, N_n\}$ be the set of discrete variables, Π_{N_i} denotes the parent set of a node N_i in the candidate network, and v_i denotes the number of possible states of the variable N_i. The network description length is given by:

$$\sum_{i=1}^{n} \left[|\Pi_{N_i}| \log_2(n) + d(v_i - 1) \prod_{N_j \in \Pi_{N_i}} v_j \right],$$

where d is a constant denoting the number of bits used to store a numerical value. Intuitively, the network description length represents the structural complexity of the network which is evaluated by the number of bits required to encode the graphical structure and to store the conditional probability table at each node.

Meanwhile, the data description length is given by:

$$\sum_{i=1}^{n} \sum_{N_i, \Pi_{N_i}} M(N_i, \Pi_{N_i}) \log_2 \frac{M(\Pi_{N_i})}{M(N_i, \Pi_{N_i})},$$

where $M(.)$ is the count of the particular instantiation in the data set. In essence, the data description length evaluates the proximity of the distributions implied by the data and the candidate network, which is a measure of the accuracy of the candidate network.

Because the MDL metric is simply the sum of the two description lengths, it puts a balance between model complexity and model accuracy. In other words, the optimal network, with regard to the metric, should be simple while accurately represents the joint distribution.

As a property common to other metrics, the MDL metric is node-decomposable and could be written as in equation 5. One can observe that the score is simply the summation of the independent evaluation on the parent set, Π_{N_i}, of every node N_i in the domain U.

$$\mathrm{MDL}(G) = \sum_{N_i \in U} \mathrm{MDL}(N_i, \Pi_{N_i}). \tag{5}$$

With the defined metric, the network learning problem can be formulated as a search problem. The objective is to search for the network structure which has the optimal score. However, the problem is difficult as the search space, which contains all possible network structures, is huge. Chickering et al. proved that the search problem is NP-hard with the use of a particular metric [37]. Some algorithms, therefore, resort to greedy search heuristics [17, 19]. However, the drawback of these algorithms is that sub-optimal solutions may be obtained. Some others use systematic and exhaustive search, like branch-and-bound [38], to find the optimal solution. In the worst case, the time consumed would be considerable. Recently, some researchers attempt to use evolutionary computation to tackle the problem [39, 20].

3. Learning Bayesian Networks Using Evolutionary Computation

Evolutionary computation is a general stochastic search methodology. The principal idea is borrowed from evolution mechanisms proposed by Charles Darwin. Evolutionary computation is becoming popular as it often gives satisfactory results for various optimization problems in different areas. For example, it is applied in data mining, image processing, pattern recognition, and signal processing [4, 3, 6, 5, 40, 41, 42, 43].

Recently, there are two approaches [39, 20] that apply evolutionary computation to tackle the problem of learning Bayesian networks using the search-and-scoring approach. The first one uses genetic algorithms (GAs) [44, 45] while the second one uses evolutionary programming (EP) [46, 47].

3.1 Learning Bayesian Network Using Genetic Algorithms

Larrañaga et al. [39] proposed to use GAs [44, 45] to search for the optimal Bayesian network structure. In their research, the network structure (composed of n nodes) is represented by an $n \times n$ adjacency matrix C. Each element C_{ij} in the matrix is defined as:

$$C_{ij} = \begin{cases} 1, & \text{if node } j \text{ is a parent of node } i \\ 0, & \text{otherwise.} \end{cases}$$

With this representation, the i^{th} row in the matrix encodes the parent set of node N_i (i.e. Π_{N_i}). An illustration is given in Figure 2.

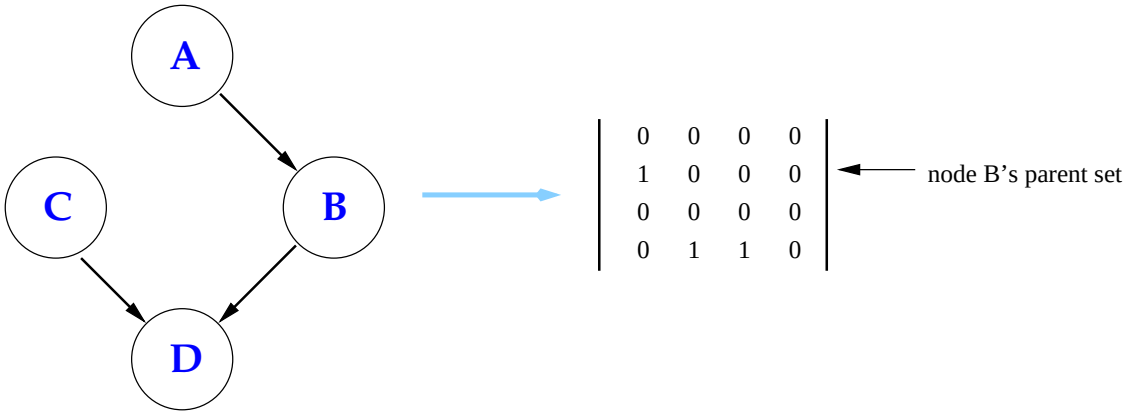


Figure 2: Matrix representation of a DAG.

By flattening the matrix, the bit-string representation is obtained:

$$C_{11}C_{21}C_{31}\ldots C_{n1}C_{21}C_{22}\ldots C_{nn}.$$

A simple GA (with one-point crossover and mutation) is applied to search for the optimal solution represented in the bit-string representation. For the fitness function, they adopted the Bayesian score (referred as the BD score in [37]) in the K2 algorithm [17]. Note that since the genetic operators could generate illegal offspring structures (i.e. networks that are not DAG), cycles repairing is needed after an offspring is produced. Because it is rare to have a densely connected network in real-life problems, they imposed a limit on the number of parents that a node could have in their implementation. Even though such restriction greatly reduces the possible search space, the problem is still NP-hard [48]. Höffgen conducted a number of experiments to test the GA approach with different implementations under different parameter settings. Based on the results, several recommendations regarding the choice of implementation and parameters were made [48].

In another research work, Larrañaga et al. [49] considered the problem of finding node orderings. They represented a node ordering in a chromosome and used a genetic algorithm to evolve different node orderings. For each node ordering, it is passed to the K2 algorithm to obtain a network.

Cotta and Muruzábal proposed a number of recombination operators for Bayesian networks and applied these operators in some steady-state genetic algorithms to induce Bayesian networks [50].

Myers, Laskey, and DeJong [51] proposed a GA that learns Bayesian networks from incomplete data. This algorithm evolves the Bayesian network structures and the values of the missing data simultaneously. Myers, Laskey, and Levitt [52] introduced the Evolutionary Markov Chain Monte Carlo (EMCMC) algorithm to learn Bayesian networks from incomplete data. EMCMC combines the advantages of the canonical genetic algorithm and the Markov Chain Monte Carlo algorithm. Laskey and Myers [53] applied a hybrid algorithm called Population Markov Chain Monte Carlo (popMCMC) to induce Bayesian networks from data sets with missing observations and hidden variables. PopMCMC increases the rate of improvement in solutions

from the initial solution by exchanging information among solutions in a population of Metropolis-Hastings Samplers. It satisfies conditions ensuring ergodicity and convergence to the Boltzmann distribution on the given energy surface. They demonstrated that popMCMC had greater population diversity than an evolutionary algorithm.

3.2 MDLEP

Wong et al. [20] used EP to tackle the Bayesian network learning problem. Since they used the MDL metric [19] to evaluate the fitness value of a Bayesian network, they called their approach MDLEP.

EP is different from GAs mainly in the format of solution representation and the genetic operators used [46, 47]. Unlike the restricted use of string in GAs, EP does not have any restriction in solution representation. An individual in MDLEP is simply the adjacency matrix of the network. Furthermore, there is no crossover operation in EP, and the only operation is mutation. An outline of the MDLEP algorithm is given in Figure 3 [20].

---

1. Set t, the generation count, to 0.
2. Create an initial population, Pop(t) of m random DAGs (m is the population size).
3. Each DAG in the population is evaluated using the MDL metric.
4. While t is less than the maximum number of generations,
   - Each DAG in Pop(t) produces one offspring by performing a number of mutation operations. The probabilities of executing 1, 2, 3, 4, 5, or 6 mutations are 0.2, 0.2, 0.2, 0.2, 0.1, and 0.1, respectively. For each mutation, one of the four mutation operators (simple, reverse, move, and knowledge-guided mutations) is selected for execution according to a uniform distribution. If there are cycles, a randomly picked edge in each cycle is removed.
   - The DAGs in Pop(t) and all new offspring are stored in the intermediate population Pop$'(t)$. The size of Pop$'(t)$ is $2 \times m$.
   - Conduct a number of pairwise competitions over all DAGs in Pop$'(t)$. For each G_i in the population, q other individuals are selected. Then the fitness of G_i and the q individuals are compared. The score of G_i is the number of individuals (out of q) that has lower fitness than G_i.
   - Select the m highest score individuals from Pop$'(t)$ with ties broken randomly. The individuals are stored in Pop($t+1$).
   - increment t by 1.
5. Return the individual that has the lowest MDL metric in any generation of a run as the output of the algorithm.

Figure 3: The MDLEP algorithm.

In essence, MDLEP uses four mutation operators which include simple, reverse, move, and knowledge-guided mutations. The simple mutation operator randomly picks an edge, and if the edge is already present in the network, the edge is removed, otherwise, the edge is added. The reverse mutation operator randomly picks an edge from the network and reverses its direction. The move mutation operator modifies the parent

set of a node by replacing one of the parents with a non-parent. The knowledge-guided mutation operator is similar to the simple mutation except that an edge is selected with certain guidance. Briefly, each edge, $X \rightarrow Y$, is weighted by evaluating the MDL score of node Y having only X as its parent. These scores are computed and stored at the beginning. When the knowledge-guided mutation operator determines that an existing edge should be removed, it retrieves the stored MDL scores of all edges in the network and those edges with higher scores are deleted with higher probabilities. On the other hand, if the knowledge-guided mutation operator decides to add an edge to the network, it gets the stored MDL scores of the edges that are absent and those edges with lower scores will have higher probabilities of being added.

In their experiments, they tested their algorithm with data sets generated from two benchmark networks, ALARM and PRINTD. They compared their algorithm with Larrañaga et al.'s GA approach using the MDL metric, and they found that MDLEP performs better in many aspects. In general, those networks generated from MDLEP have smaller structural differences (in comparison with the original network), and smaller MDL scores. In addition, MDLEP is also faster as it requires fewer generations to converge and generates less invalid structures.

Tucker et al. [54] extended MDLEP to find good dynamic Bayesian network structures that can have large time lags.

3.3 Problems of the Previous Approaches

As reported in Wong et al.'s work [20], the EP formulation seems to have a clear advantage over Larrañaga et al.'s GA approach. To account for this, we conjecture that the performance gain is largely due to the different choice of genetic operators in producing the offspring, which, in effect, influences the exploration of the search space. On the one hand, the success of EP readily suggests that sheer mutations, which correspond to adding or removing an edge or the combination of the two, are good enough for generating new search points. On the other hand, the crossover operation, which plays an important role in GAs, seems to be ineffective. The reason is not difficult to understand as the one-point crossover recombines two graphs arbitrarily. In most cases, this could result in an invalid structure. In this regard, such recombination would seem to be insignificant and offspring do not properly *inherit*, which is supposedly the merit of the crossover operator. Although the idea of exchanging information among the population members is good, the one-point crossover cannot achieve the purpose in this problem. Similarly, this observation was independently reported by Yao and Liu [55]. They proposed a novel evolutionary system, called EPNet, for evolving architectures and weights of artificial neural networks(ANNs) simultaneously. EPNet emphasizes on evolving ANN behaviours and uses a number of techniques to maintain a close behavioral link between parents and their offspring. They performed several experiments and showed that EPNet can produce very compact ANNs with good generalization ability.

Despite that the EP approach performs better, we observe that it still requires a long execution time. For instance, to learn a network with 37 nodes from a given data set of 10,000 cases, MDLEP needs about an hour to find the solution [1], which is not practical for real-life data mining applications on large databases. At closer inspection, we find that a major cause of its long execution time is that there are much more worse offspring (comparing an offspring with its parental DAG) produced than *better offspring* on average. From our experience, if the population size is 50, we would have, on average, less than two *better offspring* produced in each generation. This implies that most of the mutation operations generate inferior network structures. Hence, we conjecture that MDLEP is still not efficient enough in finding good solutions.

4. Hybrid Evolutionary Algorithm (HEA)

The efficiency of MDLEP can be enhanced by employing a number of strategies. First, a hybrid approach is introduced so that the knowledge from dependency tests is exploited during searching. Second, previous search results are reused through a new merge operator. Third, in contrast to MDLEP where repairing is needed, the formation of cycle should be avoided altogether when producing new individuals.

Since a hybrid approach and an evolutionary algorithm are used in Bayesian network learning, this approach is called HEA (Hybrid Evolutionary Algorithm). In the following subsections, the ideas will be discussed in detail.

4.1 A Hybrid Approach

In dependency analysis approach, CI test is typically used to check the validity of a conditional independence assertion $I(X, Z, Y)$ of any two given nodes X, Y and a conditioning set Z. Assume that the χ^2 test is employed and the assertion is modeled as the null hypothesis. A χ^2 test generates a p-value, ranges between 0 and 1, which shows the least level of significance for which the given data lead to the rejection of the null hypothesis. In effect, if the p-value is less than a predefined cutoff value, α, the hypothesis $I(X, Z, Y)$ is rejected. Otherwise, if the p-value is greater than or equal to α, the hypothesis could not be rejected and $I(X, Z, Y)$ is assumed to be valid. Consequently, this implies that the two nodes, X and Y, cannot have a direct edge between them. In other words, the edges $X \leftarrow Y$ and $X \rightarrow Y$ cannot exist in the resultant network.

With such observation, a hybrid approach for learning Bayesian networks is formulated which consists of the CI test and the search phases.

4.1.1 The CI Test Phase

In this phase, low-order CI tests are conducted so that some edges could be removed. Only low-order CI tests are performed because their results are more reliable than higher order tests and the time complexity is bounded.

Initially, we let the possible parent set of each node to contain all other nodes and we attempt to reduce the size of the parent set of each node by discovering low order CI relations. For example, if the node X is found to be conditionally independent of node Y in a test, X will be removed from Y's parent set and vice versa. Alternatively, it could be viewed as though the edges $X \leftarrow Y$ and $X \rightarrow Y$ are both excluded for further consideration. Since higher-order CI tests may be unreliable, we only use order-0 and order-1 tests for discerning possible conditional independence relations.

In our implementation, we use the likelihood-ratio χ^2 test for testing. For a given assertion $I(X,\ Z,\ Y)$, a p-value is returned from the test (see Section 2.2.1 for details). If the p-value is greater than a predefined cutoff value α, the assertion cannot be rejected and we assume $I(X,\ Z,\ Y)$ to be valid.

Suppose that there are n variables. For a given pair of variables, we need to, in the worst case, conduct the order-0 test (i.e. $I(X, \Phi, Y)$) and all order-1 tests (i.e. test $I(X, Z, Y)$ for every $Z \in U \setminus \{X, Y\}$). Hence, the overall complexity of the CI test phase is bounded by $O(n^3)$ tests.

Although CI tests are very useful, incorrect results could be detrimental. In particular, if a crucial edge (which appears in the optimal network) is excluded due to the findings in the CI test phase, it is impossible to obtain the optimal solution in the subsequent search phase. In the following sub-section, we propose an approach to tackle this problem.

4.1.2 The Search Phase

In the search phase, a score-and-search approach is used together with the knowledge obtained in the CI test phase. In particular, the search space is reduced by excluding networks containing the edges $X \leftarrow Y$ or $Y \rightarrow X$ for which $I(X, Z, Y)$ is assumed to be valid. Since the search space is reduced, the learning problem becomes easier and less time will be needed for finding the best network.

This idea could be applied readily. After obtaining the test results, all candidate networks having invalid edges are prevented from being generated. Although such formulation can work fine, it must be emphasized that the choice of α is critical. If improper α is used, in the worst case, either all edges are pruned away or all edges are retained. Hence, although it is possible to impose the restrictions from CI tests as *global* constraints, there is the risk of assuming our choice of α is appropriate.

As an alternative, a novel realization of the hybrid approach is developed in which a different α is used for each individual in the population. Thus, each individual has, besides the network structure, a cutoff value α which is also subjected to be evolved. As the evolutionary search proceeds, individuals having improper values of α will eventually be eliminated. In general, a small value of α implies more constraints (less likely to reject an hypothesis) and results in a more restricted search space. Hence, if the value of α of an individual is too small which excludes some *important* edges, the individual will have greater chance of being eliminated. On the other hand, if the value of α of an individual is too large, it is less likely to find

the *right* edge (because there are many *wrong* alternatives) for its offspring. Consequently, the individual will also have higher chance of being eliminated.

This idea is implemented in the first phase by storing the largest p-value returned by the CI tests for every possible conditioning set, Z (restricted to order-0 and all order-1 tests) in a matrix, Pv. In the second phase, for a given individual G_i in the population with associated cutoff value α_i, an edge $X \leftarrow Y$ cannot be added if Pv_{XY} is greater than α_i (i.e. $I(X, Z, Y)$ is assumed to be valid). The value of each α_i is randomly initialized at the beginning. In subsequent generations, an offspring will inherit the cutoff value from its parental network with a possible increment or decrement by Δ_α. In other words, each individual conducts its search in different search space and this search space can be modified dynamically by changing its value of α. The advantage of this approach is that CI tests would not remove the essential edges permanently. If the value of α of an individual is so small that some essential edges are excluded, the search space of its offspring can be extended to include these edges by increasing the value of α.

van Dijk, Thierens, and van der Gaag have just reported a similar approach for learning Bayesian networks [56]. The approach consists of two phases. In the first phase, CI tests are performed to generate an undirected graph, called the *skeleton graph*, to reduce the search space. A GA is used in the second phase to search for Bayesian networks. They have compared their approach with the preliminary implementation of our hybrid approach reported in [24]. They have found that their approach is comparable with ours. However, their approach is generally slower. Moreover, they have used a fixed value of α. Thus, their approach may suffer from the problem discussed in the previous sub-section.

4.2 The Merge Operator

In addition to the four mutation operators mentioned in Section 3.2, a new operator called merge is introduced. Taking a parental network G_a and another network G_b as input, the merge operator attempts to produce a better network structure (in terms of MDL score) by modifying G_a with G_b. If no modification can be done, G_a is returned.

Let M_i^x denote the MDL score of the parent set $\Pi_{N_i}^x$ of node $N_i \in U$ in the network G_x. Recalling that the MDL score is decomposable and a network is an agglomeration of Π_{N_i} (for $i = 1, \ldots, n$). Thus, given two input networks G_a and G_b, a better network, G_c, could be generated by selecting $\Pi_{N_i}^c$ from $\Pi_{N_i}^a$ or $\Pi_{N_i}^b$ so that (1) there is no cycle in G_c and (2) the sum $\sum_{N_i \in U} M_i^c$ is less than $\sum_{N_i \in U} M_i^a$ and $\sum_{N_i \in U} M_i^b$. With such observation, the merge operator is devised and is the heuristics for finding a subset of nodes, $W \subset U$, with which $\Pi_{N_j}^a$ are replaced with $\Pi_{N_j}^b$ in G_a for every $N_j \in W$. Meanwhile, the replacement would not create cycles and has a MDL score smaller than those of G_a and G_b. The pseudo-code for the merge operator is presented in Figure 4.

For the two input networks G_a and G_b, the merge procedure produces a node ordering by sorting $\delta_i = M_i^a - M_i^b$ in descending order. Since positive δ_i means that $\Pi_{N_i}^b$ is better than $\Pi_{N_i}^a$, the procedure follows the

Procedure merge(G_a, G_b)

1. Find $\delta_i = M_i^a - M_i^b$ for every node $N_i \in U$.
2. Produce a node ordering L by sorting δ_i in descending order.
3. Set $W = \phi$.
4. While there are still nodes in L left unconsidered,
   - Get the next node, N_i, from L which is unconsidered.
   - Set $W' = \phi$.
   - Invoke the procedure `findSubset(`N_i`, `W'`)` which returns W' on completion.
   - Calculate the sum of δ_j for every node $N_j \in (W' - W)$.
   - If the sum is greater than zero
     - Mark every node $N_j \in W'$ in L as considered.
     - Replace $\Pi^a_{N_j}$ with $\Pi^b_{N_j}$ for every node $N_j \in (W' - W)$.
     - Set $W = W \cup W'$.
5. If the MDL score of the new network is smaller than those of G_a and G_b, return the new network. Otherwise, return the original G_a.

Figure 4: Pseudo-code for the merge operator.

ordering in considering the replacement of $\Pi^a_{N_i}$ with $\Pi^b_{N_i}$. Beginning with the first node, N_i, in the ordering, the merge procedure invokes the recursive procedure `findSubset(`N_i`, `W'`)` to find a subset of nodes W' such that by replacing $\Pi^a_{N_j}$ with $\Pi^b_{N_j}$ for every $N_j \in W'$ in G_a, the resultant graph is still acyclic. The pseudo-code for `findSubset(`N_i`, `W'`)` is shown in Figure 5.

Procedure findSubset(N_i, W')

1. If N_i is already in W', return immediately.
2. Insert N_i to W'.
3. For every parent N_k of N_i in G_b,
   - For every node N_j in W',
     - If $N_j \neq N_k$,
       - check if there is a directed path going from N_j to N_k in G_a, i.e. $N_j \rightarrow \cdots \rightarrow N_k$.
       - If a directed path exists, invoke `findSubset(`N_k`, `W'`)`. (recursion)

Figure 5: Pseudo-code for `findSubset`.

Initially, when `findSubset(`N_i`, `W'`)` is called by `merge(`G_a`, `G_b`)` for an input node N_i and $W' = \phi$, it checks if cycles will be created in G_a by replacing $\Pi^a_{N_i}$ with $\Pi^b_{N_i}$. The parent set replacement involves two steps: Step one removes $N_k \rightarrow N_i$ for every $N_k \in \Pi^a_{N_i}$ and Step two adds $N_k \rightarrow N_i$ for every $N_k \in \Pi^b_{N_i}$. However, since Step one will not form any cycles, it suffices to detect for cycles after executing Step two only. In other words, `findSubset` only examines if cycles will be produced by adding the edges $N_k \rightarrow N_i$ for every $N_k \in \Pi^b_{N_i}$ to G_a. Hence, `findSubset` checks whether there is a directed path going in the reverse direction (i.e. $N_i \rightarrow \cdots \rightarrow N_k$) in G_a for each edge addition. If cycles were not produced, `findSubset` could safely replace $\Pi^a_{N_i}$ with $\Pi^b_{N_i}$. However, if it finds that cycles will be created by adding $N_k \rightarrow N_i$, for some $N_k \in \Pi^b_{N_i}$, `findSubset` will consider replacing the parent set of node N_i and the parent sets of the N_k's

together by invoking `findSubset` recursively.

In the recursive invocation of `findSubset(`N_i`, `W'`)`, N_i still denotes the current input node being considered in the recursion. However, now W' will contain the sets of nodes that are being considered for parent set replacement. If N_i is not in W', it is added to W'. Otherwise, the procedure returns immediately which guarantees that the procedure would eventually terminate. Similar to the initial invocation, `findSubset` checks whether cycles will be produced by replacing $\Pi^a_{N_i}$ with $\Pi^b_{N_i}$. This time, however, the checking is more complicated because the procedure has to cater for the simultaneous replacements of the parent sets of all nodes in W'.

For example, consider the two Bayesian networks, G_a and G_b, in Figure 6 and assume that `findSubset(`N_i`, `W'`)` is invoked by `merge(`G_a`, `G_b`)` for $N_i = C$ and $W' = \phi$. It is not enough to replace Π^a_C with Π^b_C only, because cycles $A \rightarrow C \rightarrow A$ and $B \rightarrow C \rightarrow B$ will be created. In the first invocation of `findSubset(`N_i`, `W'`)`, since $N_i = C$ is not in W', $N_i = C$ is inserted to W' and W' becomes $\{C\}$. From G_b, the parent set of node C is $\{A,\ B\}$, `findSubset` checks if there are direct paths in G_a from C to A. Since the edge $C \rightarrow A$ exists in G_a, the first invocation of `findSubset(`N_i`, `W'`)` recursively calls `findSubset(`A`, `$\{C\}$`)`.

In the recursive invocation of `findSubset(`N_i`, `W'`)`, $N_i = A$ and $W' = \{C\}$. Since $A \notin \{C\}$, A is inserted to W' and W' becomes $\{A,\ C\}$. From G_b, node A has not parent, the recursive invocation of `findSubset(`N_i`, `W'`)` terminates and the control flow returns back to the first invocation of `findSubset(`N_i`, `W'`)`.

Currently, $N_i = C$ and $W' = \{A,\ C\}$, `findSubset` checks if there are direct paths in G_a from A to B or from C to B. Since the edge $C \rightarrow B$ exists in G_a, the first invocation of `findSubset(`N_i`, `W'`)` recursively invokes `findSubset(`B`, `$\{A,\ C\}$`)`.

In the recursive invocation of `findSubset(`N_i`, `W'`)`, $N_i = B$ and $W' = \{A,\ C\}$ Since $B \notin \{A,\ C\}$, B is inserted to W' and W' becomes $\{A,\ B,\ C\}$. From G_b, the parent set of node B is $\{A\}$, `findSubset` determines if there are direct paths in G_a from B to A or from C to A. Since the direct path from C to A exists, the recursive invocation of `findSubset(`N_i`, `W'`)` calls `findSubset(`A`, `$\{A,\ B,\ C\}$`)` which terminates immediately because $A \in \{A,\ B,\ C\}$. Then, the recursive invocation of `findSubset(`N_i`, `W'`)` terminates and the control flow returns back to the first invocation of `findSubset(`N_i`, `W'`)`. Finally, `findSubset(`N_i`, `W'`)` terminates with $W' = \{A,\ B,\ C\}$.

Although `findSubset(`N_i`, `W'`)` guarantees that the resultant network is acyclic, it does not guarantees that W' is minimal. A set of nodes W' is minimal if there is not proper subset W'' of W', such that by replacing $\Pi^a_{N_j}$ with $\Pi^b_{N_j}$ for every $N_j \in W''$ in G_a, the resultant network is acyclic. An example is given in the Appendix to illustrate this situation.

After obtaining W', the merge procedure calculates the sum $\sum_{N_j \in (W' - W)} \delta_j$. If the sum is greater than zero, it replaces $\Pi^a_{N_j}$ with $\Pi^b_{N_j}$ in G_a for every $N_j \in (W' - W)$, removes W' from the ordering and then inserts W' into W. The procedure repeatedly examines the next node in the ordering until all nodes are

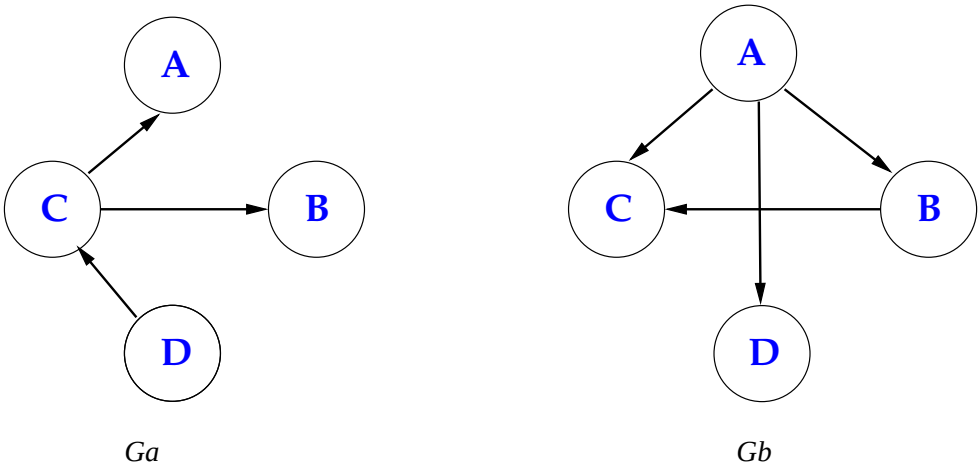


Figure 6: Two Bayesian networks used by `findSubset`.

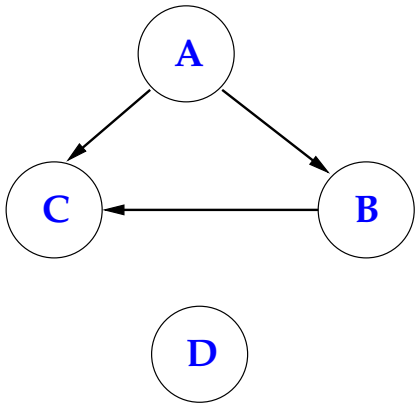


Figure 7: The resultant network obtained after the merge operator.

considered. Assume that the merge operator takes the two Bayesian networks, G_a and G_b, in Figure 6 as input, and W is $\{A,\ B,\ C\}$. The resultant network obtained is shown in Figure 7.

4.2.1 An Analysis of the Operator

The merge operator represents a heuristic method for selecting W. In other words, it is not guaranteed that the choice is optimal. In fact, if there are n variables, the optimal solution can be found by considering exhaustively all 2^n combinations. However, this would be very expensive because we have to check for cycle formation for every possible combinations. Thus, our primary concern in developing this operator is its computational cost. The worst case complexity of `findSubset` is $O(n^2)$ and this case occurs when `findSubset` follows every edge in the densest network [2]. Inside the `findSubset` procedure, the most expensive operation is the checking for the existence of a directed path. Since the directed path connectivity information about G_a is also stored in a *path number matrix* (see Section 4.3), the checking reduces to a trivial table lookup operation. Hence, the merge operator is a fast recombination procedure that has the worst case complexity of $O(n^3)$. In Section 5.3.4, we will show that the empirical complexity is much smaller than the worst case

complexity.

Essentially, the merge operator increases the efficiency in several ways. Since the score of the composite network structure can be readily calculated, it is not necessary to invoke the procedure for MDL score evaluation which is time-consuming. Thus, the merge operator offers an economical way to create new network structures. It also provides a way to combine and reuse the search results obtained in previous generations. Moreover, the merge operator ensures that the new network structure and its parental network structures belong to different equivalence classes of Bayesian networks [57]. Two network structures are equivalent if the set of distributions that can be represented using one of the structures is identical to the set of distributions that can be represented using the other. Since the equivalence relation is reflexive, symmetric, and transitive, it defines a set of equivalence classes over network structures. The MDL scoring criterion is *score equivalent* that assigns the same score to equivalent structures [58]. Search strategies will be inefficient if they spend most of their time within the same equivalence class. The merge operator can avoid this problem because it generates a network structure having different MDL score from those of its parental network structures. If network structures have different MDL scores, they belong to different equivalence classes. Thus, the merge operator improves the search effectiveness and efficiency by exploring more different equivalence classes in each generation.

Cotta and Muruzábal independently proposed a number of recombination operators for inducing Bayesian networks from data and developed some steady-state evolutionary algorithms (such as PheGT$_2^R$) that use these operators [50]. The worst case complexity of these operators is $O(n^5)$ while the empirical complexity is $O(n^3)$ [59]. Since these operators individually consider to include or exclude an edge based on the information of the two corresponding parental network structures, it is possible to produce an offspring having parent sets that are different from those in its parental network structures. Thus, the expensive procedure for score evaluation may be invoked for the offspring. On the other hand, our merge operator combines parent sets from the two parental network structures to obtain an offspring. Thus, it is not necessary to perform the MDL score evaluation procedure for the offspring. Furthermore, their operators do not ensure that the offspring and its parental network structures belong to different equivalence classes because they used a scoring criterion which is not score equivalent.

4.2.2 Merging in HEA

One of the reason that MDLEP runs slowly is that there are more worse offspring produced than *better offspring*. To improve this situation, we favor a heavy use of merging than mutation because it is both computationally efficient and effective. In our current implementation, we select half of the population randomly for merging and only keep the results if better networks are produced. For the unselected members of the population and the selected ones which fail to give better networks after merging, we will produce offspring by mutation as usual.

For the networks that are selected for merging, they are merged with dumped networks from the last generation. In such a way, we considerably reuse our previous search efforts, which are thrown away otherwise.

4.3 Prevention of Cycle Formation

Since MDLEP consumes much time in repairing networks that contain cycles, HEA prevents cycle formation in all candidate networks to handle this problem. In addition to the adjacency matrix representing an individual, HEA also maintains the *path number matrix* containing the number of directed paths between every pair of nodes. If $X \rightarrow \cdots \rightarrow Y$ exists in a network, HEA forbids adding the edge $X \leftarrow Y$ to the network. The matrix is updated when an edge is added or removed. Let $\mathcal{C}$ denotes the path number matrix and $\mathcal{C}_{XY}$ be the number of directed paths going from X to Y. $\mathcal{C}$ is updated when an edge is either added or removed. Using the convention that a node X is an immediate successor of itself, we have $\mathcal{C}_{XX} = 1$ for every node $X \in U$. The procedure for updating the matrix is shown in Figure 8. As can be seen, the overhead for the update has a complexity of $O(n^2)$, where n is the number of variables.

Finally, the algorithm of HEA is summarized in Figure 9.

Procedure UpdatePathNumberMatrix(X, Y, adding)

FOR each ancestor of X, X',
  FOR each successor of Y, Y',
    IF *adding*,
      $\mathcal{C}_{X'Y'} = \mathcal{C}_{X'Y'} + \mathcal{C}_{X'X} \times \mathcal{C}_{YY'}$
    ELSE,
      $\mathcal{C}_{X'Y'} = \mathcal{C}_{X'Y'} - \mathcal{C}_{X'X} \times \mathcal{C}_{YY'}$
    END
  END
END

Figure 8: Pseudo-code for `UpdatePathNumberMatrix`.

5. Evaluation of HEA

In this section, the performance of HEA is studied. In Section 5.1, the experiment methodology is described. In Section 5.2, the experimental results on comparing HEA with MDLEP, PheGT$_2^R$ [50], Bayesian Network PowerConstructor (BNPC) [16], and WinMine Toolkit [60] are reported. In Section 5.3, an analysis of the performance of HEA is presented. Finally, our findings are summarized in Section 5.4.

5.1 Experimental Methodology

A common practice to assess the performance of a Bayesian network learning algorithm is to test the algorithm on data sets generated from known network structures using probabilistic logic sampling [61].

CI test Phase

1. For every pair of nodes (X, Y),
   - Perform order-0 and all order-1 CI tests.
   - Store the highest p-value in the matrix Pv.

Evolutionary Algorithm Search Phase

1. Set t, the generation count, to 0.
2. Initialize the value of m, the population size.
3. For each individual G_i in the population Pop(t),
   - initialize the α value randomly.
   - refine the search space by checking the α value against the Pv matrix.
   - Inside the reduced search space, create a DAG randomly.
4. Each DAG in the population is evaluated using the MDL metric.
5. While t is less than the maximum number of generations,
   - If $t = 0$, all individuals are marked "NS" (not selected).
   - otherwise,
     - select $m/2$ individuals from Pop(t), the rest are marked "NS".
     - For each of the selected ones,
       * merge with a random pick from the dumped half in Pop$'(t-1)$.
       * If merge does not produce a new structure, mark the individual with "NS".
       * otherwise, regard the new structure as an offspring.
   - For each individual marked "NS",
     - produces an offspring by cloning.
     - alters the α value of the offspring randomly.
     - refine the search space by checking the α value against the Pv matrix.
     - changes the structure by performing a number of mutation operations. The probabilities of executing 1, 2, 3, 4, 5, or 6 mutations are 0.2, 0.2, 0.2, 0.2, 0.1, and 0.1, respectively. For each mutation, one of the four mutation operators (simple, reverse, move, and knowledge-guided mutations) is selected for execution according to a uniform distribution. Note that cycle formation is prohibited.
   - The DAGs in Pop(t) and all new offspring are stored in the intermediate population Pop$'(t)$. The size of Pop$'(t)$ is 2*m.
   - Conduct a number of pairwise competitions over all DAGs in Pop$'(t)$. For each G_i in the population, q other individuals are selected. The fitness of G_i is compared against the q individuals. The score of G_i is the number of individuals (out of q) that are worse than G_i.
   - Select the m highest score individuals from Pop$'(t)$ with ties broken randomly. The individuals are stored in Pop$(t+1)$.
   - increment t by 1.
6. Return the individual that has the lowest MDL metric in any generation of a run as the output of the algorithm.

Figure 9: The HEA algorithm.

Here, we follow the practice and test HEA on seven different data sets. All of the data sets are generated from the well-known benchmarks of Bayesian networks including the ALARM, the PRINTD, and the ASIA networks. Table 1 gives a summary of the data sets used in our experiments.

Data set	Original Network	Size	MDL Score of Original Network	Source
ALARM-1000	ALARM	1,000	18,533.5	MDLEP [20]
ALARM-2000	ALARM	2,000	34,287.9	MDLEP
ALARM-5000	ALARM	5,000	81,223.4	MDLEP
ALARM-10000	ALARM	10,000	15,8497.0	MDLEP
ALARM-O	ALARM	10,000	138,455.0	PowerConstructor [16]
ASIA-1000	ASIA	1,000	3,416.9	PowerConstructor
PRINTD-5000	PRINTD	5,000	106,541.6	MDLEP

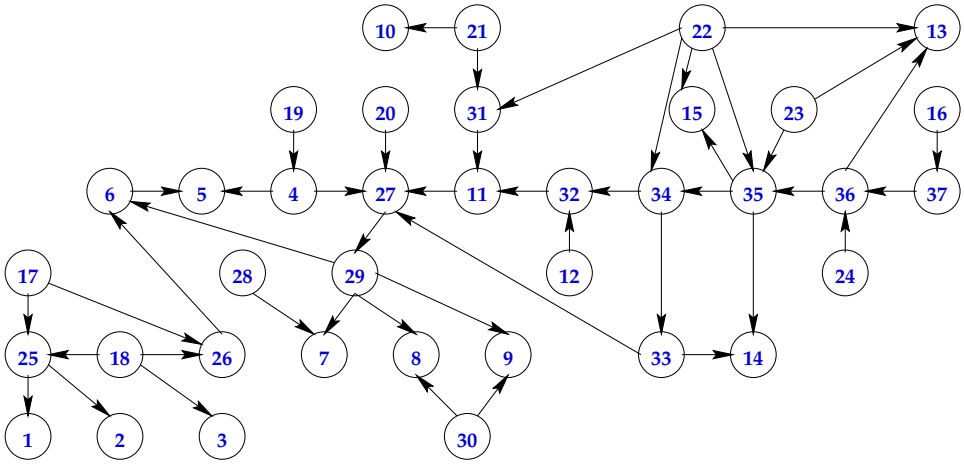
Table 1: Data sets used in the experiments.

ALARM-1000, ALARM-2000, ALARM-5000, ALARM-10000, and ALARM-O are generated from the ALARM network which has the structure shown in Figure 10 [3]. Originally, the ALARM network is used in the medical domain for potential anesthesia diagnosis in the operating room [62]. Because the network, with 37 nodes and 46 directed edges, has a complex structure, it is widely used for evaluating the performance of a Bayesian network learning algorithm. Examples include the K2 algorithm [17], the CB algorithm [63], the BENEDICT algorithm [64], and MDLEP [20]. The ALARM data sets used in our experiments are obtained from two different sources. One of the data sets (ALARM-O) containing 10,000 cases is obtained from BNPC [16]. ALARM-1000, ALARM-2000, ALARM-5000, and ALARM-10000 have been used for evaluating MDLEP [20]. The four data sets are of different sizes and contain 1,000, 2,000, 5,000, and 10,000 cases respectively.

Another data set with 5,000 cases is generated from the PRINTD network. The PRINTD network is primarily constructed for troubleshooting printer problems in the Windows^{TM} operating system [10]. The structure of the network is shown in Figure 11. It has 26 nodes and 26 edges.

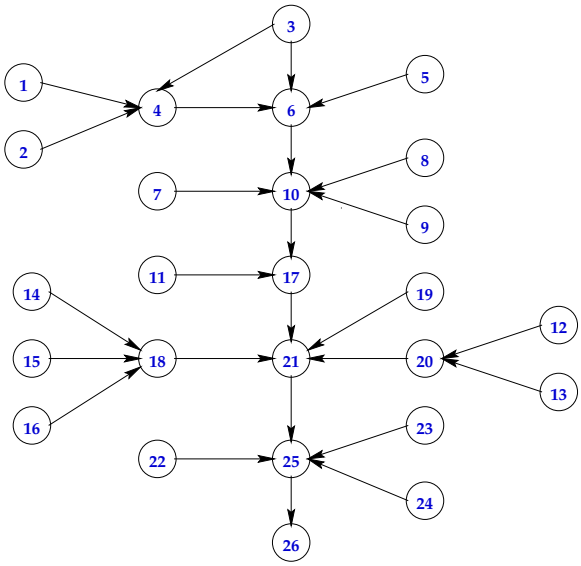
One of the data sets is generated from the ASIA network. As shown in Figure 12, the ASIA network is a relatively simple structure that contains eight nodes and eight edges. The network is also known as the "chest-clinic" network which describes a fictitious medical example on whether a patient has tuberculosis, lung cancer, or bronchitis, related to the attributes (X-ray, dyspnea, visit-to-Asia, and smoking) of the patient [65, 66]. The data set contains 1,000 cases.

In our experiments, we compare the performance of HEA with MDLEP, PheGT$_2^R$ [50], BNPC [16], and WinMine Toolkit [60]. We implement the steady-state GA proposed by Cotta and Muruzábal that applies a genetic recombination operator to learn Bayesian networks from data [50]. In their algorithm called PheGT$_2^R$, they used a fitness function which is not score equivalent [58]. On the other hand, we use the MDL scoring criterion in our implementation of PheGT$_2^R$, so that its performance can be directly compared with that of HEA and MDLEP. Moreover, we also implement the cycle prevention mechanism to improve its efficiency.



1. central venous pressure
2. pulmonary capillary wedge pressure
3. history of left ventricular failure
4. total peripheral resistance
5. blood pressure
6. cardiac output
7. heart rate obtained from blood pressure
8. heart rate obtained from electrocardiogram
9. heart rate obtained from oximeter
10. pulmonary artery pressure
11. arterial-blood oxygen saturation
12. fraction of oxygen in inspired gas
13. ventilation pressure
14. carbon-dioxide content of expired gas
15. minute volume, measured
16. minute volume, calculated
17. hypovolemia
18. left-ventricular failure
19. anaphylaxis
20. insufficient anesthesia or analgesia
21. pulmonary embolus
22. intubation status
23. kinked ventilation tube
24. disconnected ventilation tube
25. left-ventricular end - diastolic volume
26. stroke volume
27. catecholamine level
28. error in heart rate reading due to low cardiac output
29. true heart rate
30. error in heart rate reading due to electrocautery device
31. shunt
32. pulmonary-artery oxygen saturation
33. arterial carbon-dioxide content
34. alveolar ventilation
35. pulmonary ventilation
36. ventilation measured at endotracheal tube
37. minute ventilation measured at the ventilator

Figure 10: The ALARM network.

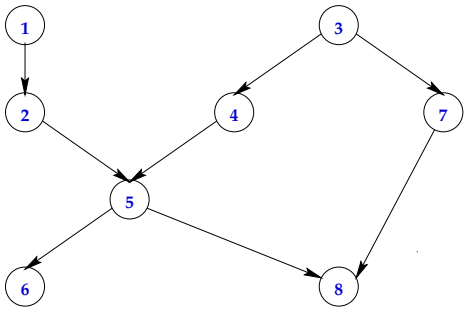


1. Spool Process OK
2. Local Disk Space Adequate
3. Application Output OK
4. Spooled Data OK
5. Print Spooling On
6. GDI Data Input OK
7. Correct Driver
8. Uncorrupted Driver
9. Correct Driver Settings
10. GDI Data Output OK
11. Correct Printer Selected
12. Correct Local Port
13. Local Cable Connected
14. Network Up
15. Correct Printer Path
16. Network Cable Connected
17. Print Data OK
18. Network Path OK
19. Network/Local Printing
20. Local Path OK
21. PC to Printer Transport OK
22. Printer On and Online
23. Paper Loaded
24. Printer Memory Adequate
25. Printer Data OK
26. Printer Output OK

Figure 11: The PRINTD network.

HEA, MDLEP, and PheGT$_2^R$ are implemented in the C++ language and are compiled using the same compiler [4]. Besides, the same MDL metric evaluation routine is used so that the difference among implementations is minimized. To enhance efficiency, they are implemented with the same hashing mechanism so that each computed MDL metric query is stored in a hash table. By a *query*, we refer to the evaluation of the MDL score of the given parent set of a node. For HEA, MDLEP, and PheGT$_2^R$, the allowable parent set size is limited to five. Since these algorithms are stochastic in nature, they are executed 40 times for each testing instance. All these experiments are conducted on the same Sun Ultra-5 workstation with a 270MHz UltraSparc IIi processor and 128M memory running Solaris 8 operating system.

BNPC is an algorithm based on the dependency analysis approach that executes on the Windows environment. Since it is a deterministic algorithm, it is executed once for each testing instance. WinMine Toolkit is a set of tools for the Windows environment that build statistical models from data. It can generate dependency networks or Bayesian networks from data. When it is used to learn a Bayesian network with conditional probability tables, it performs a greedy search algorithm starting with a Bayesian network containing no edges. It adds, deletes, and reverses edges in the Bayesian network until a local maximum is reached. In our experiments, it is executed once for each testing instance because it is deterministic. In the following sections, we shall present our experimental results.



1. Visit to Asia
2. Tuberculosis
3. Smoking
4. Lung Cancer
5. Tuberculosis or Cancer
6. X-Ray Result
7. Bronchitis
8. Dyspnea

Figure 12: The ASIA network.

5.2 Comparing HEA with Other Algorithms

In this section, we compare the performance of different algorithms on all of the data sets. Our main objective in these experiments is to determine whether HEA is more efficient and effective than MDLEP. For both algorithms, the population size is 50, the maximum number of generations is 5000, and the tournament size (q) is 7. For HEA, we set Δ_α to be 0.02. We estimate the performance of the two algorithms using six measures, which include:

- the average MDL score of the final solutions, the smaller the better (AFS).
- the average MDL score of the best network obtained in the first generation (AIS).
- the average execution time in seconds (AET).
- the average generation that the best-so-far solution is obtained (ANG),
- the average number of MDL metric evaluations in a run (AME). AME is *not* the count of MDL metric evaluations invoked during a run. Since HEA and MDLEP use the same hashing mechanism (i.e. store every computed query) for MDL metric evaluation, counting the number of invocations to the MDL evaluation function does not have much significance. AME measures the number of stored queries which is a count of all distinct MDL metric evaluations that have taken place.
- the average structural difference, i.e. number of edges added, omitted and reversed, between the final solution and the original network (ASD).

For PheGT$_2^R$, the population size is 500 [5], the maximum number of individuals examined is 250,000, the crossover rate is 0.9, the mutation rate is $1/n^2$ where n is the number of variables. A tournament selection with tournament size = 3 is used. Except the first two parameters, all other parameter values are the same as those used in [50].

Table 2 provides a summary of the performance comparison among different algorithms. Recall that HEA, MDLEP, and PheGT$_2^R$ are executed forty times for each data set, the figures are, therefore, an average

of forty trials. The MDL score of the original network is also included (just below the name of each data set) as reference. Numbers in parentheses are the standard deviations.

In the table, only the structural differences between the networks obtained by BNPC and the original networks are presented, because BNPC uses the dependency analysis approach and AFS, AIS, and AME are not applicable. ANG is not available because it is not an evolutionary algorithm. Its execution time cannot compare with that of HEA, MDLEP, and PheGT$_2^R$ because it executes on the Windows environment. Similarly, we only present the structural differences between the networks obtained by WinMine Toolkit and the original networks.

Comparing with MDLEP, HEA could always find better or equally good network structures for all the data sets in terms of both MDL score and structural difference. Apart from the ASIA-1000 and the PRINTD-5000 data sets, the differences for all the data sets are statistically significant at the 0.05 level using the Mann-Whitney test [31]. If we compare the ANG statistics, it is found that HEA uses much less generations to obtain the final solution (statistically significant at 0.05 level) for all the data sets. Given that HEA and MDLEP essentially use the same formulation in searching, the experimental results readily suggest that HEA is more efficient as it uses fewer generations to obtain similar, or better, solutions.

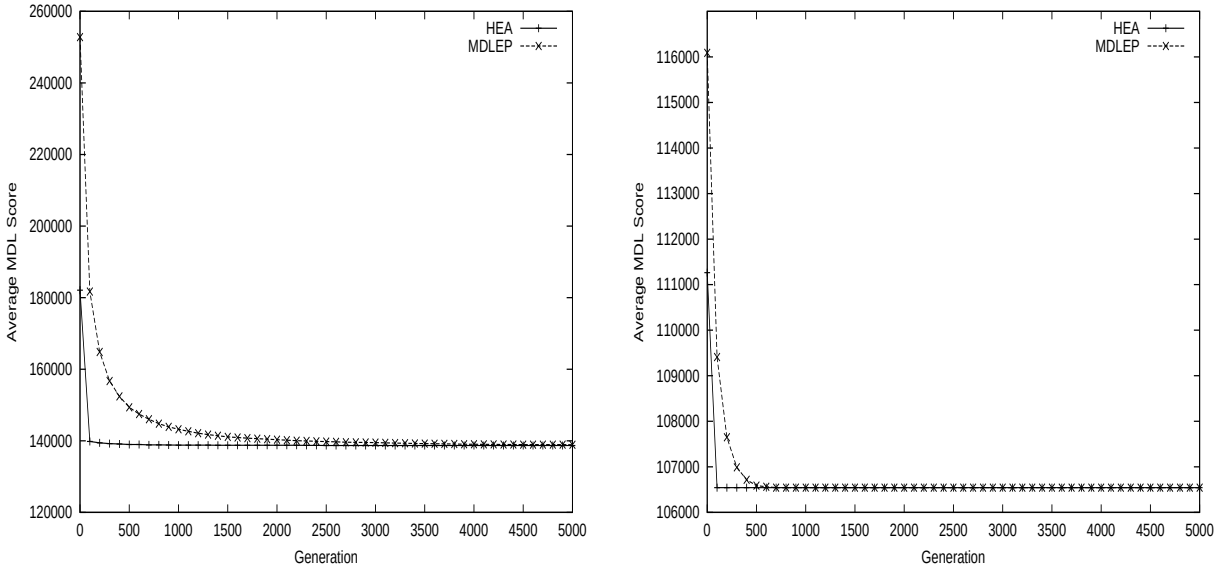
¿From the AET statistics, HEA uses less time to finish than MDLEP under the same termination criterion. Despite that the time spent on the CI phase is also counted in HEA, we can easily deduce that a *generation* in HEA takes less time than in MDLEP. But if similar operations (i.e. mutations) are performed on the same amount of individuals, what constitutes the speedup? An explanation for this is due to the prevention of cycle formation in HEA (see Section 5.3.1). From empirical experience, cycle repairing in MDLEP often takes a significant portion of the total running time. By avoiding to create illegal networks (i.e. that contains cycle), HEA does not need to spend time on cycle repairing for every offspring as MDLEP does. Although there is overhead involved (see Section 4.3), the results suggest that cycle prevention is more efficient than cycle repairing. Another possible reason for the speedup is because HEA issues fewer MDL metric evaluations. When comparing the count of the metric evaluations (AME), we observe that HEA makes orders of magnitude less evaluations than MDLEP. The situation could be accounted for by the reduction of the search space due to the hybrid approach.

In Figure 13, we compare the typical runs of HEA and MDLEP for the two data sets, ALARM-O and PRINTD-5000. For each algorithm, we measure the MDL score of the best-so-far solution averaged over forty runs as the generations proceed. Although we are testing on two very different data sets, we obtain similar observation that HEA converges much faster than MDLEP. Besides, for the same number of generations, HEA is observed to perform better than MDLEP in terms of the average score of the final solution obtained.

To validate that HEA improves over MDLEP because HEA produces more *better offspring*, we obtain the results as shown in Figure 14. The testing is based on the ALARM-O data set. For both algorithms, we record the numbers of *better offspring* (i.e. better than all of its parental network(s)) produced at each

Data Set		AFS	AIS	AET	ANG	AME	ASD
ALARM-1000 (18533.5)	HEA	17,871.6 (30.1)	36,829.6 (7,341.0)	306.4 (8.8)	913.1 (1,260.5)	3,024.3 (818.7)	10.8 (2.1)
	MDLEP	17,990.5 (73.1)	30,831.0 (795.6)	1,003.9 (70.8)	4,301.2 (654.3)	22,133.8 (619.3)	19.4 (4.2)
	PheGT$_2^R$	18,016.9 (107.9)	31,310.6 (725.0)	935.5 (22.4)	62.8 [6 7] (56.6)	16,489.9 (133.4)	17.0 (3.1)
	BNPC	– [8]	– [8]	– [9]	– [10]	– [11]	17 [12]
	WinMine	– [13]	– [13]	– [9]	– [10]	– [14]	34 [12]
ALARM-2000 (34287.9)	HEA	33,773.9 (1.6)	55,484.4 (6,914.5)	346.0 (18.1)	206.8 (155.1)	3,114.8 (374.2)	7.6 (0.6)
	MDLEP	33,932.6 (215.8)	56,896.6 (1,259.5)	1,307.8 (125.1)	4,046.6 (634.1)	25,905.8 (911.3)	12.9 (4.9)
	PheGT$_2^R$	34,079.4 (125.8)	57,382.0 (999.4)	998.8 (25.6)	86.8 (111.3)	16,421.5 (132.4)	12.2 (3.1)
	BNPC	–	–	–	–	–	13
	WinMine	–	–	–	–	–	30
ALARM-5000 (81233.4)	HEA	81,004.0 (0.0)	123,147.6 (11,460.9)	393.4 (19.9)	331.0 (465.2)	3,237.4 (218.2)	6.1 (0.4)
	MDLEP	81,287.6 (419.9)	134,487.2 (1,836.0)	1,843.2 (359.0)	3,946.3 (651.2)	29,570.8 (1,016.3)	10.7 (4.9)
	PheGT$_2^R$	81,489.1 (283.2)	132,707.5 (1,768.7)	1,223.1 (30.9)	61.7 (68.4)	16,243.7 (144.2)	10.7 (3.8)
	BNPC	–	–	–	–	–	8
	WinMine	–	–	–	–	–	23
ALARM-10000 (158497.0)	HEA	158,466.5 (164.9)	236,032.0 (20,827.9)	483.3 (36.2)	865.1 (843.1)	4,032.2 (679.9)	4.2 (2.1)
	MDLEP	158,704.4 (513.1)	256,946.2 (3,843.7)	2,435.1 (350.1)	3,596.7 (720.0)	32,160.8 (1,538.0)	8.7 (5.1)
	PheGT$_2^R$	158,766.5 (344.1)	252,736.3 (3,427.6)	1,483.7 (43.1)	57.5 (65.1)	16,114.8 (134.9)	8.1 (3.1)
	BNPC	–	–	–	–	–	9
	WinMine	–	–	–	–	–	19
ALARM-O (138455.0)	HEA	138,672.4 (500.8)	228,518.0 (25,904.9)	448.0 (9.4)	877.3 (817.9)	4,181.3 (642.5)	8.9 (5.0)
	MDLEP	138,913.4 (460.8)	252,818.4 (5,862.0)	4,209.9 (2,021.3)	4,523.8 (482.1)	34,309.5 (1,327.5)	17.5 (6.9)
	PheGT$_2^R$	138,861.8 (535.7)	244,124.7 (5,263.2)	1,762.8 (67.2)	55.8 (36.7)	17,007.6 (181.0)	10.2 (5.2)
	BNPC	–	–	–	–	–	9
	WinMine	–	–	–	–	–	25
ASIA-1000 (3416.9)	HEA	3,398.7 (0.2)	3,683.8 (192.3)	57.0 (1.2)	29.7 (21.0)	224.7 (26.1)	3.1 (0.8)
	MDLEP	3,398.6 (0.0)	3,590.2 (48.5)	76.3 (0.4)	79.6 (30.2)	656.8 (9.2)	3.5 (0.5)
	PheGT$_2^R$	3,398.9 (0.2)	3,554.5 (17.7)	422.6 (20.7)	6.4 (2.2)	771.9 (11.0)	2.7 (1.0)
	BNPC	–	–	–	–	–	3
	WinMine	–	–	–	–	–	2
PRINTD-5000 (106541.6)	HEA	106,541.6 (0.0)	115,148.1 (2,371.2)	243.4 (5.4)	47.0 (14.1)	2,441.5 (558.9)	0 (0.0)
	MDLEP	106,541.6 (0.0)	116,089.6 (546.4)	704.5 (13.8)	512.1 (95.8)	17,688.4 (373.7)	0.0 (0.0)
	PheGT$_2^R$	107,071.5 (91.4)	115,236.9 (383.3)	840.5 (27.5)	146.2 (107.2)	11,137.6 (129.1)	34.8 (4.4)
	BNPC	–	–	–	–	–	4
	WinMine	–	–	–	–	–	27

Table 2: Performance comparison among HEA, MDLEP, PheGT$_2^R$, BNPC, and WinMine.



(a) ALARM-O data set

(b) PRINTD-5000 data set

Figure 13: Typical runs of both algorithms.

generation averaged over forty runs. From the figure, we can clearly observe that HEA is able to produce more *better offspring* than MDLEP. In addition, after a number of generations, we observe that MDLEP could rarely produce *better offspring*. On the other hand, because of the merge operator, HEA still manages to produce a fair amount of *better offspring*. We will give a detailed analysis on the effectiveness of the merge operator in Section 5.3.4.

Comparing with PheGT$_2^R$, HEA could always find better network structures in terms of MDL score for all the data sets except the ASIA-1000 data set. The difference is statistically significant at 0.05 level. ¿From the AET statistics, HEA uses less time to finish than PheGT$_2^R$. We conclude that HEA is more efficient since they generate the same number of individuals. When comparing the count of the metric evaluations (AME), we observe that HEA makes much less MDL score evaluations than PheGT$_2^R$. The situation could be accounted for by the reduction of the search space due to the hybrid approach.

When we compare HEA with BNPC and WinMine Toolkit, the ASD of HEA are smaller than those of BNPC for all the data sets except the ASIA-1000 data set. In the forty trials of executing HEA on the ALARM-1000 data set, only one trial returns a Bayesian network that is worse than that generated by BNPC. The information for the other data sets is summarized in Table 3. Similarly, the ASD of HEA are smaller than those of WinMine Toolkit for all the data sets except the ASIA-1000 data set. In Table 3, we summarize the number of HEA trials generating a Bayesian network that is worse than that obtained by WinMine Toolkit. It can be observed that HEA performs better than BNPC and WinMine Toolkit for most of the data sets.

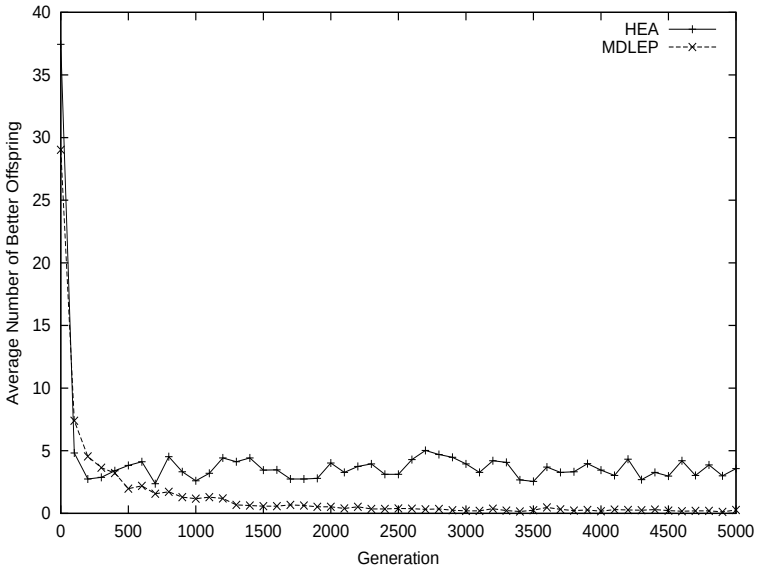


Figure 14: Comparing the numbers of better offspring produced.

	ALARM-1000	ALARM-2000	ALARM-5000	ALARM-10000	ALARM-O	ASIA-1000	PRINTD-5000
BNPC	1	0	0	2	16	11	0
WinMine	0	0	0	0	0	35	0

Table 3: The number of HEA trials that return worse Bayesian networks.

5.3 Performance Analysis of HEA

We study the factors that affect the performance of HEA. Particularly, there are four issues that we wish to investigate: (1) the contributions of the hybrid approach and the merge operator, (2) the effect of the population size, (3) the effect of $\Delta\alpha$, and (4) the effectiveness and the efficiency of the merge operator. We conduct all our experiments on the ALARM-O data set and use the same set of random seed for the forty trials. We assume the following default settings: the termination criterion is 5,000 generations; the population size is 50; the tournament size is 7; and Δ_α is 0.02.

5.3.1 The Hybrid Approach and the Merge Operator

Recall that we have proposed three strategies of improvements in HEA, it is important to investigate the actual merits of each of them. As the experimental results suggest, the cycle prevention scheme seems to be indispensable because it contributes largely to the observed speedup. However, we note that the cycle prevention scheme is only an efficiency enhancement in implementation, which implies that it does not actually improve the effectiveness as a better search methodology. Hence, we concentrate on the effectiveness of the hybrid approach and the merge operator.

We first describe the experiment which analyses the efficiency and effectiveness of the cycle prevention

	AFS	AIS	AET	ANG	AME	ASD
MDLEP	138,913.4 (460.8)	252,818.4 (5,862.0)	4,209.9 (2,021.3)	4,523.8 (482.1)	34,309.5 (1,327.5)	17.5 (6.9)
MDLEP+CP	139,089.0 (672.3)	253,895.1 (7,065.6)	531.9 (23.1)	4,846.0 (150.1)	56,583.8 (2,847.9)	19.4 (7.7)
HEA	138,672.4 (500.8)	228,518.0 (25,904.9)	448.0 (9.4)	877.3 (817.9)	4,181.3 (642.5)	8.9 (5.0)

Table 4: Effectiveness of the cycle prevention scheme.

scheme. In Table 4, we name the approach that has only cycle prevention (without the hybrid approach and the merge operator) as "MDLEP+CP". When we compare the time used (AET) between MDLEP and MDLEP+CP, it is evident that the cycle prevention scheme helps to provide a speedup. However, when we compare the quality of the final solutions (AFS), the difference is not statistically significant (p-value equals 0.2789) which suggests that only similar quality solutions are obtained. Furthermore, as the ANG statistics suggest, MDLEP+CP would require even more generations to obtain the final solution. Thus, we conclude that the cycle prevention scheme merely helps to improve the speed, yet provides no obvious improvement on the search methodology. On the other hand, if we compare MDLEP+CP with HEA, we find that there are statistically significant differences between the quality of the final solutions (p-value equals 6.729×10^{-4}), the ANG (p-value equals 1.489×10^{-14}) and even the time used (p-value equals 1.468×10^{-14}). Hence, we conclude that it is the hybrid approach and the merge operator that provide the actual improvement on the efficiency and effectiveness.

In fact, if either the hybrid approach or the merge operator is good enough, it is not necessary to bring them together. Hence, the objective of our experiment is to justify the HEA approach. To conduct the experiment, we compare HEA with two other implementations that called "HEA$-$merge" and "EA+merge." In HEA$-$merge, we keep the hybrid approach but we do not use the merge operator; while in EA+merge, we do not perform any CI tests but keep the use of the merge operator. We present our results in Table 5. In this table, AST is the average execution time in seconds that the best-so-far solution is obtained. Note that for both implementations, the cycle prevention scheme is used.

When we compare MDLEP+CP in Table 4 and EA+merge in Table 5, we find that EA+merge produces significantly better solutions than MDLEP+CP (p-value equals 8.261×10^{-4}). The AET, ANG, AME, and ASD of EA+merge are significantly smaller than those of MDLEP+CP (p-values are respectively 2.397×10^{-14}, 1.382×10^{-14}, 1.382×10^{-14}, and 1.995×10^{-3}). Thus, the merge operator does improve on the efficiency and effectiveness.

In comparing the average execution time of the three implementations, we observe that there are clear distinctions among them. Since EA+merge does not apply the hybrid approach to reduce the search space, it takes the longest running time. On the other hand, the hybrid alone approach, HEA$-$merge, takes the shortest time while HEA comes in between. It is apparent that HEA has a significantly smaller ANG (the average number of generations to obtain the best-so-far solution). This implies that HEA can obtain

	AFS	AIS	AET	AST	ANG	AME	ASD
EA+merge	138,688.8 (393.6)	280,583.8 (13,383.1)	462.6 (12.1)	190.6 (92.6)	1,687.6 (1,062.0)	20,158.5 (782.1)	13.4 (7.7)
HEA−merge	138,571.0 (321.4)	228,518.0 (25,904.9)	307.6 (5.7)	205.6 (44.5)	2,293.4 (1,207.6)	4,484.6 (643.8)	9.3 (4.1)
HEA	138,672.4 (500.8)	228,518.0 (25,904.9)	448.0 (9.4)	175.1 (55.4)	877.3 (817.9)	4,181.3 (642.5)	8.9 (5.0)

Table 5: Performance comparison of different implementations.

	Best Score	Lower quartile	Worst score	Average
EA+merge	138,275 (9)	138,870	140,260	138,688.8
HEA−merge	138,275 (12)	138,668	140,228	138,571.0
HEA	138,275 (18)	138,692	139,912	138,672.4

Table 6: A comparison of the final scores obtained for different implementations.

the best-so-far solutions at much earlier generation than the other two algorithms. When we examine the AST (the average execution time to obtain the best-so-far solution) statistics, HEA can find the best-so-far solutions much faster than the other two algorithms. The AST value of HEA is significantly smaller than that of HEA−merge (p-value is 1.09×10^{-4}).

To compare the final solutions obtained, we present a more detailed analysis in Table 6. In the table, we show the best score obtained among the forty trials, with the number in parenthesis indicating the number of occurrences. Besides, the lower quartile is included for which better or equally good scores are obtained for 75% of the forty trials ($40 \times 75\% = 30$). The worst score is the worst one out of the forty trials while the average is the AFS measure shown previously in Table 5.

¿From the table, it can be observed that EA+merge has smaller chance of finding the recorded-best solution (the one with MDL score of 138,275 and structural difference of 4) and it has the worst average score. Moreover, by comparing its lower quartile with those of the other two implementations, it has higher chance of obtaining worse solutions. On the other hand, HEA has the highest chance of obtaining the recorded-best solution, although its average score is worse than that of HEA−merge.

It must be said that both EA+merge and HEA−merge have their drawbacks. For EA+merge, in addition to that it is inferior in obtaining the recorded-best solution, it also requires much more MDL metric evaluations than both HEA and HEA−merge. The reason is obvious as the absence of the hybrid approach renders EA+merge to search in the entire search space. Consequently, EA+merge will sample much more different solutions and hence a much larger number of metric evaluations. Although the consequence is not obvious in the current data set, it makes a difference when the size of the data set grows. Because it takes $O(n_r)$ time, where n_r is the number of records in the data set, to evaluate the MDL score of a node, it follows that time needed for one evaluation increases with increasing data set size. When the size of the data set is huge, we expect that the contribution due to the hybrid approach will manifest while EA+merge will run much slower.

For the HEA−merge implementation, we notice that the best-so-far solution will often appear later than

	AFS	AET	AST	ANG	AME	ASD
$m = 30$	138,747.5 (579.4)	325.3 (25.9)	173.0 (39.7)	1,218.3 (873.4)	3,791.9 (747.6)	9.3 (5.2)
$m = 50$	138,672.4 (500.8)	448.0 (9.4)	175.1 (55.3)	887.3 (817.9)	4,181.3 (642.5)	8.9 (5.0)
$m = 70$	138,479.0 (272.0)	597.8 (12.9)	177.5 (62.6)	622.3 (675.5)	4,510.3 (843.4)	7.5 (4.2)
$m = 90$	138,483.9 (336.8)	740.8 (15.2)	177.1 (67.6)	453.7 (533.9)	5,015.8 (711.1)	7.1 (4.3)
$m = 110$	138,529.3 (324.4)	886.1 (32.3)	172.5 (42.4)	301.3 (269.7)	5,516.1 (999.8)	7.7 (4.1)
$m = 130$	138,534.6 (352.6)	1,022.2 (30.2)	248.6 (213.9)	689.3 (1,195.8)	5,593.8 (590.3)	7.8 (4.3)
$m = 150$	138,406.6 (183.7)	1,174.4 (28.7)	229.2 (164.7)	477.6 (772.0)	6,401.6 (1,408.8)	6.8 (4.0)
$m = 170$	138,403.3 (225.4)	1,308.8 (42.7)	397.0 (349.6)	1,151.2 (1,527.7)	6,358.9 (691.5)	6.3 (3.7)
$m = 190$	138,432.7 (232.5)	1,447.9 (35.3)	333.2 (303.4)	785.1 (1,186.1)	6,715.9 (1,097.6)	7.1 (4.1)
$m = 210$	138,372.4 (162.2)	1,612.9 (49.2)	256.6 (265.8)	422.1 (944.8)	7,165.4 (1,071.6)	6.0 (3.5)
$m = 230$	138,371.6 (170.1)	1,784.5 (50.5)	274.0 (234.1)	432.3 (737.1)	7,297.4 (659.7)	6.1 (3.7)
$m = 250$	138,391.3 (180.4)	1,891.2 (40.7)	355.8 (354.6)	633.6 (1,056.4)	7,888.6 (1,414.8)	6.6 (4.1)

Table 7: Performance of HEA with different population sizes.

HEA in terms of the number of generations (ANG). Apart from the difference in mean, the Mann-Whitney test suggests there is a significant difference between HEA−merge and HEA (p-value equals 1.809×10^{-9}). From our experiment, it is evident that both algorithms are able to obtain similar final solutions in the time span of 5,000 generations. However, in another set of experiments, where we set the termination criterion to be 1,000 generations, HEA easily outperforms HEA−merge. Another drawback of HEA−merge relates to the reliance on the hybrid approach. If ever the CI tests were not useful, the performance of HEA−merge would be greatly affected. Although HEA could also suffer from the same problem, the use of the merge operator will provide a bottom line performance boost as manifested in EA+merge.

With the drawbacks of HEA−merge and EA+merge mentioned, a combination of both strategies in HEA is justified and forms the most stable algorithm amongst the three in terms of efficiency and effectiveness.

5.3.2 Effect of Different Population Sizes

In this experiment, we test HEA with twelve different population sizes (m): 30, 50, 70, 90, 110, 130, 150, 170, 190, 210, 230, and 250. For all settings, the tournament size is set to 7. We summarize our results in Table 7.

A large population means that more search points are employed and hence, as reflected by the average MDL metric evaluation statistics (AME), there are more different solutions examined than in a smaller population. Consequently, the time consumed (AET) also increases proportionally with increasing population size. We observe that a larger population would help to obtain the solution earlier. If we compare the average number of generations needed to obtain the best-so-far solution (ANG), we observe that it decreases for increasing population sizes and reaches the minimum when population size equals 110. It fluctuates between

	Best Score	Lower quartile	Worst score	Average
$m = 30$	138,275 (16)	138,692	140,310	138,747.5
$m = 50$	138,275 (18)	138,692	139,912	138,672.4
$m = 70$	138,275 (22)	138,692	139,328	138,479.0
$m = 90$	138,275 (24)	138,668	139,744	138,483.9
$m = 110$	138,275 (20)	138,668	139,328	138,529.3
$m = 130$	138,275 (20)	138,668	139,720	138,534.6
$m = 150$	138,275 (26)	138,578	138,692	138,406.6
$m = 170$	138,275 (28)	138,578	139,328	138,403.3
$m = 190$	138,275 (25)	138,668	139,328	138,432.7
$m = 210$	138,275 (29)	138,578	138,692	138,372.4
$m = 230$	138,275 (30)	138,275	138,692	138,371.6
$m = 250$	138,275 (28)	138,668	138,692	138,391.3

Table 8: A comparison of the final scores obtained for different population sizes.

422.1 and 1,151.2 for larger population sizes. The ANG for population size equals 110 is not significantly different from those for population sizes equal 130, 150, 190, 210, 230, and 250. On the other hand, it is significantly smaller than those for the other population sizes.

¿From the average MDL score of the final solutions (AFS), the best value is obtained when population size equals 230. It is significantly better than the AFS values for the other population sizes except the ones for population sizes of 90, 150, 170, 190, 210, and 250. To have a close examination on their performance, we present a more detailed comparison in Table 8.

¿From the table, we observe that the performance of HEA is very similar for population sizes equal 210, 230, and 250. Essentially, many of the runs (out of forty) could return the recorded-best score (i.e. 138,275) while the same result (i.e. 138,692) is obtained in the worst run. In conclusion, we recommend a population size of about 230 if the users can afford longer execution time, otherwise a population size of about 150 would be sufficient, because 26 of the runs could return the recorded-best score and the score of the worst run is the same as those for population sizes of 210, 230, and 250. The users may also consider to use a larger population size with a smaller number of generations because we note in Table 7 that the ANG values generally decrease for increasing population sizes.

5.3.3 Effect of Different Δ_α

As described in Section 4.1.2, each individual has its own cutoff value α. An offspring will inherit the cutoff value from its parental network with a possible increment or decrement by Δ_α. Thus, each individual performs its search in a different search space and this search space can be modified dynamically by changing its cutoff value. The parameter Δ_α affects the difference between the search spaces of a parental network and its offspring. In this experiment, we investigate the effect of varying the value of Δ_α. We perform experiments using the values of 0.0, 0.005, 0.01, 0.02, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, and 0.35. The results are presented in Table 9.

¿From the table, we notice that the best final results (AFS) is obtained for $\Delta_\alpha = 0.25$. However, it is not significantly different from those for other values of Δ_α. The smallest AET value is achieved for $\Delta_\alpha = 0.02$.

	AFS	AET	AST	ANG	AME	ASD
$\Delta_\alpha = 0.0$	138,587.3 (409.2)	446.1 (32.8)	186.1 (61.9)	904.4 (797.8)	5,083.1 (1,734.8)	8.6 (4.8)
$\Delta_\alpha = 0.005$	138,574.7 (400.7)	465.7 (26.9)	197.1 (78.6)	1,083.6 (1,128.1)	4,326.5 (815.6)	8.2 (4.5)
$\Delta_\alpha = 0.01$	138,639.7 (494.6)	469.0 (27.8)	187.8 (77.2)	911.7 (1,076.0)	4,352.4 (1,180.2)	8.4 (4.8)
$\Delta_\alpha = 0.02$	138,672.4 (500.8)	448.0 (9.4)	175.1 (55.3)	877.3 (817.9)	4,181.3 (642.5)	8.9 (5.0)
$\Delta_\alpha = 0.05$	138,508.6 (396.3)	467.9 (23.6)	194.0 (85.7)	1,029.5 (1,230.9)	4,114.5 (474.2)	7.3 (4.2)
$\Delta_\alpha = 0.1$	138,596.0 (430.4)	471.2 (32.7)	192.6 (71.7)	947.5 (915.5)	5,156.0 (2,539.2)	7.9 (4.6)
$\Delta_\alpha = 0.15$	138,600.1 (391.1)	467.1 (18.4)	188.9 (76.7)	952.3 (1,043.9)	5,816.9 (1,288.3)	8.5 (4.6)
$\Delta_\alpha = 0.2$	138,545.3 (333.0)	477.9 (21.2)	206.7 (96.9)	1,171.7 (1,341.9)	8,639.9 (3,488.6)	8.6 (5.1)
$\Delta_\alpha = 0.25$	138,503.7 (357.4)	499.6 (39.3)	209.8 (63.3)	958.5 (856.9)	10,267.8 (3,503.1)	7.7 (5.7)
$\Delta_\alpha = 0.3$	138,517.8 (355.6)	511.0 (43.8)	226.5 (76.3)	1,135.4 (903.9)	11,154.7 (3,093.7)	8.2 (5.5)
$\Delta_\alpha = 0.35$	138,641.3 (456.9)	512.9 (29.5)	229.0 (87.8)	1,222.3 (1,034.2)	12,599.0 (2,865.8)	8.7 (5.5)

Table 9: Performance of HEA with different Δ_α.

	Best Score	Lower quartile	Worst score	Average
$\Delta_\alpha = 0.0$	138,275 (19)	138,692	139,834	138,587.3
$\Delta_\alpha = 0.005$	138,275 (19)	138,692	139,744	138,574.7
$\Delta_\alpha = 0.01$	138,275 (18)	138,692	140,178	138,639.7
$\Delta_\alpha = 0.02$	138,275 (18)	138,692	139,912	138,672.4
$\Delta_\alpha = 0.05$	138,275 (23)	138,578	139,912	138,508.6
$\Delta_\alpha = 0.1$	138,275 (20)	138,692	139,758	138,596.0
$\Delta_\alpha = 0.15$	138,275 (18)	138,692	139,744	138,600.1
$\Delta_\alpha = 0.2$	138,275 (18)	138,602	139,455	138,545.3
$\Delta_\alpha = 0.25$	138,275 (23)	138,578	139,654	138,503.7
$\Delta_\alpha = 0.3$	138,275 (21)	138,602	139,654	138,517.8
$\Delta_\alpha = 0.35$	138,275 (18)	138,602	139,654	138,641.3

Table 10: A comparison of the final scores obtained for different Δ_α.

It is significantly smaller than those for the other values of Δ_α. The AME measure shows an interesting trend upon different values of Δ_α: it reaches the minimum at a moderate value of Δ_α (i.e. $\Delta_\alpha = 0.05$) and increases with both increasing or decreasing values of Δ_α. A more detailed comparison is presented in Table 10. ¿From the table, we observe that the performance of HEA is almost alike for $\Delta_\alpha = 0.05$ and $\Delta_\alpha = 0.25$. Thus, we recommend a Δ_α value of 0.05 because the corresponding AFS value is comparable to the best AFS value when $\Delta_\alpha = 0.25$ and its AME value is minimum.

5.3.4 Effectiveness and Efficiency of the Merge Operator

In this section, we study the number of successful merge operations during a run. In our current implementation, we designate a fixed number of individuals (half of the population) to undergo merging. When merging fails to produce a better individual, the result is discarded which would mean a waste of effort. Hence, it is important to note the success rate as this indicates the effectiveness of the merge operator. In Figure 15, we plot the number of successful merge operations in each generation (sampling at every ten generations) averaged over forty runs.

As shown in the figure, the average number of successful merge operations fluctuates between 3 and 21.5

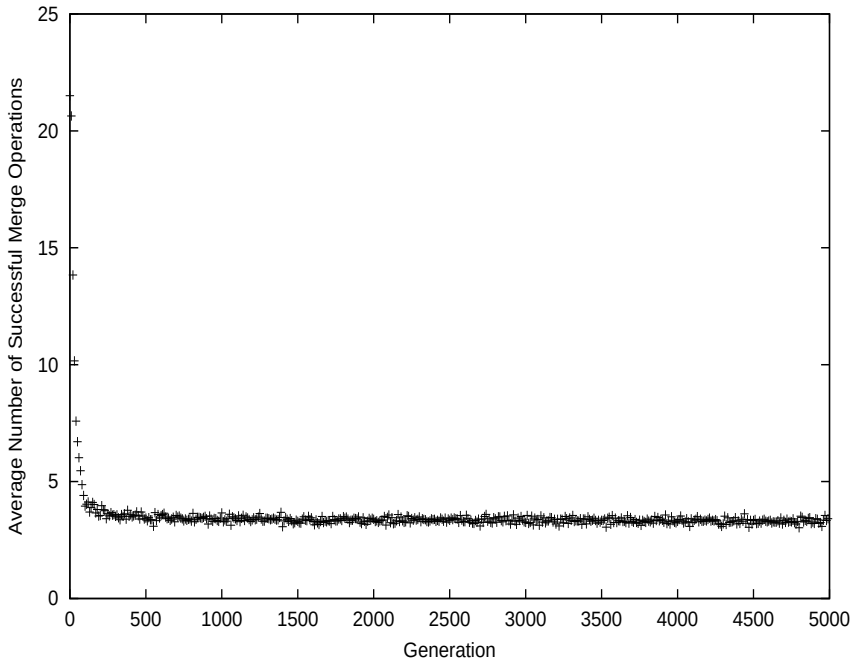


Figure 15: Successful merging in a run.

	ALARM-1000	ALARM-2000	ALARM-5000	ALARM-10000	ALARM-O	ASIA-1000	PRINTD-5000
number of nodes	37	37	37	37	37	8	26
empirical complexity	40.94	43.16	45.49	46.66	47.73	9.28	27.92

Table 11: Empirical complexity of the merge operator.

and converges to about 3.5. This is a good behavior as the successful rate is almost constant which implies that the effectiveness of the merge operator does not decline over a period of time. If we look at the statistics, we obtain an average number of 3.51 and the successful rate is therefore over one-seventh (= 3.51/25).

To study the empirical complexity of the merge operator, we estimate the average number of executions of the inner loop in step 3 of `findSubset` for each merge operation. In Table 11, we summarize the empirical complexity for different data sets. It can be observed that the empirical complexity is much smaller than the worst case complexity, which is $O(n^3)$, where n is the number of nodes.

5.4 Summary

In this section, we have presented various experimental results on comparing the performance of different learning algorithms and on analyzing HEA. In comparing the performance of HEA against MDLEP, it is evident that HEA is more efficient than MDLEP. Most importantly, it executes significantly faster than MDLEP which is important in real-life applications. HEA performs better than PheGT$_2^R$, Bayesian Network PowerConstructor, and WinMine Toolkit for most data sets.

Since both the hybrid approach and the merge operator help to improve the effectiveness and efficiency

of HEA, we have investigated their respective independent contribution towards the performance of HEA. We note the merits of bringing both strategies together which justifies the HEA approach. Concerning the choice of parameters, we recommend using a population size of about 230 and $\Delta\alpha = 0.05$. Finally, the successful rate of the merge operator of about one-seventh confirms its effectiveness and the empirical complexity suggests that it is a fast recombination operator.

6. Application in Direct Marketing

In this section, we apply HEA to induce Bayesian networks on a real-life data mining problem. The problem is related to direct marketing, the objective of which is to predict buyers from a list of customers. Advertising campaign, which includes mailing of catalogs or brochures, is then targeted on the most promising prospects. Hence, if the prediction is accurate, it can help to enhance the *response rate* of the advertising campaign and increase the return of investment (ROI). The direct marketing problem requires ranking the customer list by the likelihood of purchase [67, 68]. Given that Bayesian networks can estimate the posterior probability of an instance (a customer) belonging to a particular class (active or inactive respondents), they are particularly suitable for handling the direct marketing problem.

6.1 The Direct Marketing Problem

Direct marketing concerns communication with prospects, so as to elicit response from them. In contrast to the mass marketing approach, direct marketing is targeted at a group of individuals that are potential buyers and are likely to respond. In retrospect, direct marketing emerged because of the prevalence of mail ordering in the nineteenth century [69]. As technology advances, marketing is no longer restricted to mailing but includes a variety of media. Nevertheless, the most important issue in the business remains to be the maximization of the profitability, or ROI, of a marketing campaign.

In a typical scenario, we often have a huge list of customers. The list could be records of existing customers or data bought from *list brokers*. But among the huge list, there are usually few real buyers which amount to only a few percents [70]. Since the budget of a campaign is limited, it is important to focus the effort on the most promising prospects so that the response rate could be improved.

Before computers became widely used, direct marketers often used simple heuristics to enhance the response rate. One straightforward approach is to use common sense to make the decision. In particular, we could match prospects by examining the demographics of the customers in the list. For example, in the life insurance industry, it is natural to target the advertising at those who are rich and aging. Another common approach to enhance the response rate is to conduct list testing by evaluating the response of samplings from the list. If a certain group of customers gives a high response rate, the actual campaign may be targeted at the customers similar to this group. A more systematic approach, which was developed in 1920s but is still

being used today, is to differentiate potential buyers from non-buyers using the recency-frequency-monetary model (RFM) [69]. In essence, the profitability of a customer is estimated by three factors including the recency of buying, the frequency of buying, and the amount of money spent. Hence, only individuals that are profitable will be the targets of the campaign.

With the advancement of computing and database technology, people seek for computational approaches to assist in decision making. From the data set that contains demographic details of customers, the objective is to develop a *response model* and use the model to predict promising prospects. In certain sense, response models are similar to classifiers in the classification problem. However, unlike the classifier which makes a dichotomous decision (i.e. active or inactive respondents), the response model needs to score each customer in the data set with the likelihood of purchase. The customers are then ranked according to the score. A ranked list is desirable because it allows decision makers to select the portion of customer list to roll out [67]. For instance, out of the 200,000 customers on the list, we might wish to send out catalogs or brochures to the most promising 30% of customers so that the advertising campaign is cost-effective (the 30% of the best customers to be mailed is referred to as the *depth-of-file*) [68]. Hence, one way to evaluate the response model is to look at its performance at different depth-of-file. In the literature, there are various approaches proposed for building the response model. Here, we give a brief review in the following sub-section.

6.2 Response Models

Apart from the RFM model, which could be categorized as a computational approach for direct marketing, there are other approaches that use statistical analysis, machine learning, and artificial intelligence in the response model construction.

Earlier attempts often adopted a statistical analysis approach. Back in 1967, a company already used multiple regression analysis to build the response model. In 1968, the Automatic Interaction Detection (AID) system was developed which essentially uses tree analysis to divide consumers into different segments [69]. Later, the system was modified and became the Chi-Squared Automatic Interaction Detector (CHAID). One statistical analysis technique, which is still widely used today, is logistic regression. Essentially, the logistic regression model assumes that the *logit* (i.e. the logarithm of the *odd ratios* [15]) of the dependent variable (active or inactive respondents) is a linear function of the independent variables (i.e. the attributes). Because the approach is popular, newly proposed models are often compared with the logistic regression model as the baseline comparison [68, 71, 72].

Zahavi and Levin [72] examined the possibility of learning a back-propagation neural network as the response model. However, due to a number of practical issues and that the empirical result did not improve over a logistic regression model, it seems that the neural network approach does not bring much benefit.

Because there are striking similarities between classification and the direct marketing problem, it is straightforward to apply classification algorithms to tackle the problem. As an example, Ling and Li [73]

used a combination of two well-known classifiers, the naïve Bayesian classifier and C4.5, to construct the response model. Because scoring is necessary, they modified the C4.5 classifier so that a prediction (i.e. active and inactive respondents) comes with a *certainty factor*. To combine the two classifiers, they applied ada-boosting [74] to both classifiers in learning. When they evaluated their response model across three different real-life data sets, the result showed that their approach are effective for solving the problem.

Bhattacharyya formulated the direct marketing problem as a multi-objective optimization problem [68, 71]. He noted that the use of a single evaluation criterion, which is to measure the model's accuracy, is often inadequate [71]. For practical concern, he suggested that the evaluation criterion needs to include the performance of the model at a given depth-of-file. In an early attempt, he proposed to use GAs to learn the weights of a linear response model while the evaluation function is a weighted average of the two evaluation criteria. When comparing the learnt model with the logit model on a real-life data set, the new approach indicates a superior performance [68]. Recently, he attempted to use genetic programming to learn a tree-structured symbolic rule form as the response model [71]. Instead of using a weighted average criterion function, his new approach searches for *Pareto-optimal* solutions. From the analysis, he found that the GP approach outperforms the GA approach and is effective at obtaining solutions with different levels of trade-offs [71].

6.3 Experiment

Because Bayesian networks can estimate the probability of an object belonging to certain class(es), they are also suitable to handle the direct marketing problem. By assuming the estimated probability to be equal to the likelihood of purchase, a Bayesian network is readily applicable to the direct marketing problem. Thus, it is interesting to evaluate the empirical performance of Bayesian network response models. Specifically, we compare the performance of the evolved Bayesian network models obtained by HEA and MDLEP, the logistic regression models, the naïve Bayesian classifier (NB) [25, 26], and the tree-augmented naïve Bayesian network classifier (TAN) [25]. NB simplifies the estimation of the joint probability distribution by assuming that each attribute is conditionally independent of others given the class variable. Although the assumption behind the naïve Bayesian classifier seems unrealistic [25], the classifier often exhibits surprisingly good and robust performance in many real-life problems [26]. TAN contains augmented edges which form a spanning tree. It is regarded as the state-of-the-art Bayesian network classifier [75].

For both HEA and MDLEP, the population size is 50, the maximum number of generations is 5000, and the tournament size (q) is 7. For HEA, Δ_α is set to 0.02.

6.3.1 Experimental Methodology

The response models are evaluated on two real-life direct marketing data sets. The first data set contains records of customers of a specialty catalog company, which mails catalogs to good customers on a regular

basis. There is a total of 106,284 customers in the data set and each entry is described by 361 attributes. The response rate is 5.4%.

Typically in any data mining process, it is necessary to reduce the dimension of the data set by selecting the attributes that are considered relevant and necessary. Towards this feature selection process, there are many possible options. For instance, we could use either a *wrapper* or a *filter* selection process [76]. In a wrapper selection process, different combinations are iteratively tried and evaluated by building an actual model out of the selected attributes. In a filter selection process, a certain evaluation function, which is based on information theory or statistics, is defined to score a particular combination of attributes. Then, the final combination is obtained in a search process. In this experiment, we have used the forward selection criteria of logistic regression [29] to select eleven relevant attributes out of the 361 attributes.

The second data set contains 101,532 records of customers and each record is described by 94 attributes. The response rate is 9.6%. We have applied the forward selection criteria of logistic regression to select twenty three attributes out of the 94 attributes.

To compare the performance of different response models, we use decile analysis which estimates the enhancements of the response rates for marketing at different depth-of-file. Essentially, the ranked list is equally divided into ten deciles. Customers in the first decile are the top ranked customers that are most likely to give response. On the other hand, customers in the tenth decile are ranked lowest. Then, a *gains table* is constructed to describe the performance of the response model. In a gains table, we collect various statistics at each decile, including [77]:

Predicted Probability of Active: It is the average of the predicted probabilities of active respondents in the decile by the response model.

Percentage of Active: It is the actual percentage of active respondents in the decile. For a good model, the predicted probability should approximate the percentage of active respondents.

Cumulative Percentage of Active: It is the cumulative percentage of active respondents from decile 0 to this decile.

Actives: It is the number of active respondents in the decile.

Percentage of Total Actives: It is the ratio of the number of active respondents in the decile to the number of all active respondents.

Cumulative Actives: It is the total number of active respondents from decile 0 to this decile.

Cumulative Percentage of Total Actives: It is the percentage of cumulative active respondents (from decile 0 to this decile) over the total number of records.

Lift: It is calculated by dividing the percentage of active respondents by the response rate of the file. Intuitively, it estimates the enhancement by the response model in discriminating active respondents over a random approach for the current decile.

Cumulative Lift: It is calculated by dividing the cumulative percentage of active respondents by the response rate of the file. Intuitively, this evaluates how good the response model is for a given depth-of-file over a random approach. The measure provides an important estimate of the performance of the model.

6.3.2 Cross-Validation Results for the First Data Set

To make a comparison concerning the robustness of the response models, we adopt a cross-validation approach for performance estimation. Specifically, we employ a 10-fold cross-validation where the ten folds are partitioned randomly. In Tables 12 and 13, the experimental results for the Bayesian networks evolved by HEA (HEA models) and the logistic regression models are shown respectively. We tabulate the statistics at each decile averaged over the ten runs. Numbers in parentheses are the standard deviations. Table 12 shows that the HEA models have cumulative lifts of 400.60, 278.20, and 221.40 in the first three deciles respectively, suggesting that by mailing to the top three deciles alone, the HEA models generate over twice as many respondents as a random mailing without a model. Table 13 indicates the first three deciles have cumulative lifts of 340.90, 249.90, and 211.10 respectively for the logistic regression models. The cumulative lifts of the HEA models are significantly higher than those of the logistic regression models at 0.05 level (p-values are 1.085×10^{-5}, 6.497×10^{-5}, and 1.44×10^{-3} respectively).

Table 14 shows that the Bayesian networks generated by MDLEP (MDLEP models) have cumulative lifts of 379.40, 291.0, and 220.20 in the first three deciles respectively. The cumulative lift of the HEA models in the first decile is significantly higher than that of the MDLEP models at 0.05 level (p-value is 2.621×10^{-2}), while the former is significantly lower than the latter in the second decile (p-value is 2.621×10^{-2}). Moreover, the lift in the fifth decile declines sharply to 5.4, which is lower than the next decile (94.0), suggesting instability in the MDLEP models.

Table 15 shows that the TAN classifiers have cumulative lifts of 381.50, 278.70, and 219.10 in the first three deciles respectively. The cumulative lift of the HEA models in the first decile is significantly higher than that of the TAN classifiers at 0.05 level (p-value is 2.621×10^{-2}).

Table 16 shows that the NB classifiers have cumulative lifts of 351.10, 266.60, and 218.90 in the first three deciles respectively. The cumulative lifts of the HEA models in the first two deciles are significantly higher than those of the NB classifiers at 0.05 level (p-values are 1.624×10^{-4} and 2.621×10^{-2} respectively). To facilitate direct comparison, the cumulative lifts of different models are summarized in Table 17. In this table, the highest cumulative lift in each decile is highlighted in bold. The superscript + represents the

cumulative lift of the HEA models is significant higher at 0.05 level than that of the corresponding models. The superscript $-$ represents the cumulative lift of the HEA models is significant lower at 0.05 level than that of the corresponding models. Overall, the HEA models perform significantly better than the other models in predicting consumer response to direct mailing promotions.

Decile	Records	Prob. of Active	Percent Active	Cum. Percent Active	Actives	% of Total Actives	Cum. Actives	Cum. % of Tot. Actives	Lift	Cum. Lift
0	1063	26.25% (1.18%)	21.64% (1.21%)	21.64% (1.21%)	230.00 (12.86)	40.10% (2.22%)	230.00 (12.86)	40.10% (2.22%)	400.60 (22.30)	400.60 (22.30)
1	1063	10.50% (0.48%)	8.47% (1.44%)	15.05% (0.87%)	90.000 (15.36)	15.65% (2.27%)	320.00 (18.60)	55.75% (2.17%)	156.00 (22.69)	278.20 (10.80)
2	1063	6.31% (0.27%)	5.88% (0.58%)	11.99% (0.68%)	62.50 (6.17)	10.88% (0.81%)	382.50 (21.86)	66.63% (2.15%)	108.30 (8.17)	221.40 (7.12)
3	1063	4.60% (0.21%)	5.04% (0.83%)	10.26% (0.53%)	53.60 (8.82)	9.34% (1.50%)	436.10 (22.37)	75.97% (1.72%)	92.80 (15.11)	189.30 (4.37)
4	1063	3.44% (0.10%)	3.60% (0.72%)	8.93% (0.38%)	38.30 (7.63)	6.69% (1.41%)	474.40 (20.21)	82.66% (1.26%)	66.40 (14.15)	164.80 (2.49)
5	1062	2.49% (0.07%)	3.00% (0.57%)	7.94% (0.35%)	31.90 (6.05)	5.55% (0.98%)	506.30 (22.18)	88.21% (1.01%)	54.90 (9.61)	146.40 (1.78)
6	1063	1.80% (0.06%)	2.15% (0.37%)	7.11% (0.32%)	22.80 (3.97)	3.97% (0.63%)	529.10 (23.85)	92.18% (0.98%)	39.10 (6.26)	131.10 (1.37)
7	1063	1.31% (0.06%)	1.67% (0.31%)	6.43% (0.28%)	17.70 (3.30)	3.09% (0.61%)	546.80 (23.24)	95.27% (0.86%)	30.40 (6.08)	118.60 (1.07)
8	1063	0.88% (0.05%)	1.41% (0.34%)	5.87% (0.25%)	15.00 (3.59)	2.61% (0.61%)	561.80 (23.74)	97.88% (0.45%)	25.70 (6.20)	108.30 (0.67)
9	1062	0.46% (0.03%)	1.15% (0.26%)	5.40% (0.23%)	12.20 (2.74)	2.12% (0.45%)	574.00 (24.55)	100.00% (0.00%)	20.70 (4.42)	100.00 (0.00)
Total	10,628				574.00					

Table 12: Gains table of the HEA models for the first data set.

Decile	Records	Prob. of Active	Percent Active	Cum. Percent Active	Actives	% of Total Actives	Cum. Actives	Cum. % of Tot. Actives	Lift	Cum. Lift
0	1063	24.89% (0.30%)	18.41% (1.03%)	18.41% (1.03%)	195.70 (10.97)	34.15% (2.33%)	195.70 (10.97)	34.15% (2.33%)	340.90 (23.31)	340.90 (23.31)
1	1063	15.59% (0.12%)	8.61% (0.65%)	13.51% (0.56%)	91.50 (6.87)	15.94% (1.07%)	287.20 (11.83)	50.09% (2.40%)	158.90 (10.69)	249.90 (12.09)
2	1063	12.67% (0.08%)	7.25% (1.22%)	11.42% (0.57%)	77.10 (12.93)	13.38% (1.76%)	364.30 (18.16)	63.47% (1.77%)	133.30 (17.40)	211.10 (5.97)
3	1063	10.45% (0.09%)	6.03% (0.99%)	10.08% (0.51%)	64.10 (10.51)	11.15% (1.58%)	428.40 (21.56)	74.62% (0.88%)	110.90 (15.62)	186.00 (2.26)
4	1063	8.50% (0.13%)	3.85% (0.54%)	8.83% (0.41%)	40.90 (5.76)	7.14% (1.08%)	469.30 (21.90)	81.75% (0.93%)	70.90 (10.72)	163.00 (1.70)
5	1062	6.63% (0.15%)	3.03% (0.45%)	7.86% (0.36%)	32.20 (4.76)	5.62% (0.84%)	501.50 (22.86)	87.37% (1.20%)	55.70 (8.22)	145.00 (2.00)
6	1063	4.89% (0.12%)	2.37% (0.63%)	7.08% (0.35%)	25.20 (6.71)	4.38% (1.13%)	526.70 (25.94)	91.75% (1.48%)	43.20 (11.34)	130.60 (2.01)
7	1063	3.45% (0.09%)	1.83% (0.47%)	6.43% (0.33%)	19.50 (4.99)	3.39% (0.80%)	546.20 (27.59)	95.13% (1.07%)	33.30 (8.06)	118.40 (1.35)
8	1063	2.36% (0.07%)	1.48% (0.28%)	5.87% (0.27%)	15.70 (3.02)	2.75% (0.56%)	561.90 (25.91)	97.88% (0.69%)	27.10 (5.61)	108.10 (0.74)
9	1062	1.38% (0.04%)	1.14% (0.36%)	5.40% (0.23%)	12.10 (3.81)	2.12% (0.69%)	574.00 (24.55)	100.00% (0.00%)	20.70 (6.91)	100.00 (0.00)
Total	10,628				574.00					

Table 13: Gains table of the logistic regression models for the first data set.

In Figure 16, the models are compared in Receiver Operating Characteristic (ROC) curves that depict the true positive rate on the Y axis and the false positive rate on the X axis. These ROC curves are obtained by applying the averaging method proposed by Provost et al. [78]. It can be observed from the curves that the HEA models are more stable than the MDLEP models. Moreover, the HEP models are better than the

Decile	Records	Prob. of Active	Percent Active	Cum. Percent Active	Actives	% of Total Actives	Cum. Actives	Cum. % of Tot. Actives	Lift	Cum. Lift
0	1063	1.44% (0.03%)	20.54% (1.67%)	20.54% (1.67%)	218.30 (17.76)	38.00% (2.06%)	218.30 (17.76)	38.00% (2.06%)	379.40 (20.75)	379.40 (20.75)
1	1063	0.67% (0.01%)	10.95% (1.58%)	15.74% (0.86%)	116.40 (16.79)	20.33% (3.14%)	334.70 (18.32)	58.33% (2.49%)	202.70 (31.50)	291.00 (12.41)
2	1063	0.34% (0.01%)	4.27% (0.91%)	11.92% (0.57%)	45.40 (9.73)	7.89% (1.56%)	380.10 (18.24)	66.22% (1.31%)	78.30 (15.49)	220.20 (4.54)
3	1063	0.23% (0.00%)	7.51% (1.03%)	10.82% (0.63%)	79.80 (11.00)	13.87% (1.49%)	459.90 (26.83)	80.08% (1.70%)	138.00 (14.93)	199.70 (4.22)
4	1063	0.22% (0.00%)	0.29% (0.47%)	8.71% (0.45%)	3.10 (5.00)	0.56% (0.90%)	463.00 (24.12)	80.64% (1.23%)	5.40 (8.73)	160.70 (2.45)
5	1062	0.17% (0.00%)	5.10% (0.55%)	8.11% (0.41%)	54.20 (5.79)	9.45% (0.95%)	517.20 (26.46)	90.09% (1.41%)	94.00 (9.53)	149.70 (2.31)
6	1063	0.15% (0.00%)	0.08% (0.18%)	6.96% (0.36%)	0.90 (1.91)	0.15% (0.32%)	518.10 (26.88)	90.24% (1.39%)	1.40 (3.17)	128.40 (1.96)
7	1063	0.13% (0.01%)	2.89% (0.45%)	6.45% (0.31%)	30.70 (4.79)	5.35% (0.84%)	548.80 (26.56)	95.59% (0.88%)	53.00 (8.51)	119.20 (1.14)
8	1063	0.10% (0.00%)	2.35% (0.47%)	6.00% (0.26%)	25.00 (4.94)	4.37% (0.92%)	573.80 (24.62)	99.97% (0.11%)	43.10 (9.27)	110.90 (0.32)
9	1062	0.09% (0.00%)	0.02% (0.06%)	5.40% (0.23%)	0.20 (0.63)	0.04% (0.11%)	574.00 (24.55)	100.00% (0.00%)	0.30 (0.95)	100.00 (0.00)
Total	10,628				574.00					

Table 14: Gains table of the MDLEP models for the first data set.

logistic regression, TAN, and NB models.

The average and the standard deviations of the execution time for different methods are summarized in Table 18. Although HEA is slower than logistic regression, TAN, and NB, it is much faster than MDLEP. Moreover, HEA is able to learn Bayesian networks from a large data set in one minute. Thus, it can be used in real-life data mining applications.

6.3.3 Cross-Validation Results for the Second Data Set

Similarly, we employ a 10-fold cross-validation approach to compare the performance of different models for the second data set. The cumulative lifts of different models are summarized in Table 19.

It can be observed from Table 19 that the HEA models have cumulative lifts of 501.50, 314.60, and 239.70 in the first three deciles respectively. On the other hand, the cumulative lifts of logistic regression models are respectively 467.10, 302.2, and 232.2 in the first three deciles. The cumulative lifts of the HEA models in the first three deciles are significantly higher than those of the logistic regression models at 0.05 level (p-values are 5.424×10^{-6}, 3.791×10^{-5}, and 1.028×10^{-4} respectively).

The Bayesian networks generated by MDLEP (MDLEP models) have cumulative lifts of 482.40, 312.30, and 241.90 in the first three deciles respectively. The cumulative lift of the HEA models in the first decile is significantly higher than that of the MDLEP models at 0.05 level (p-value is 3.626×10^{-4}).

The TAN classifiers have cumulative lifts of 492.70, 314.10, and 237.40 in the first three deciles respectively. The cumulative lift of the HEA models in the first decile is significantly higher than that of the TAN classifiers at 0.05 level (p-value is 3.151×10^{-2}).

The NB classifiers have cumulative lifts of 353.30, 263.70, and 214.90 in the first three deciles respectively. The cumulative lifts of the HEA models in the first three deciles are significantly higher than those of the

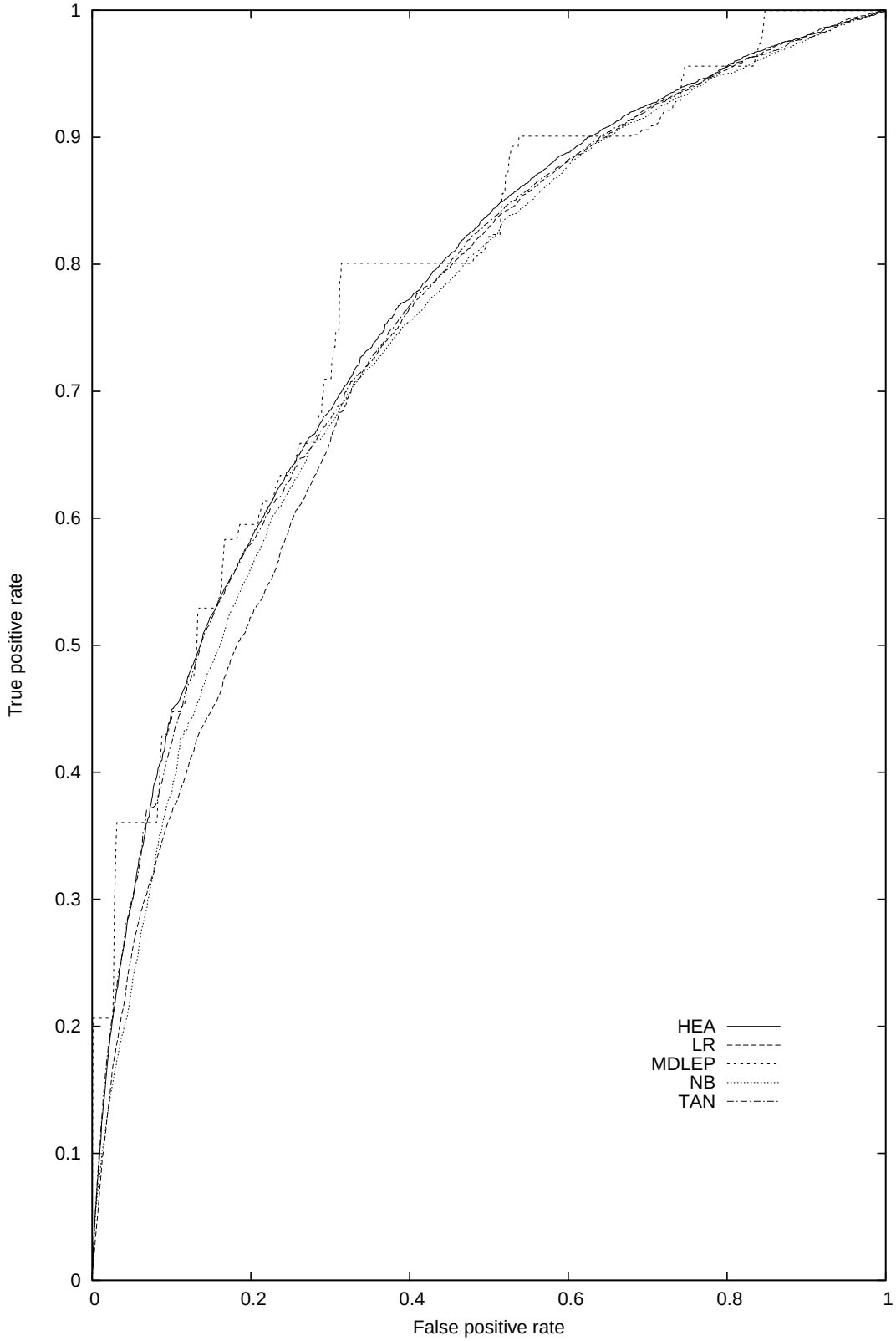


Figure 16: ROC curves of different models for the first data set.

Decile	Records	Prob. of Active	Percent Active	Cum. Percent Active	Actives	% of Total Actives	Cum. Actives	Cum. % of Tot. Actives	Lift	Cum. Lift
0	1063	35.92% (0.66%)	20.61% (0.83%)	20.61% (0.83%)	219.10 (8.88)	38.21% (1.69%)	219.10 (8.88)	38.21% (1.69%)	381.50 (16.96)	381.50 (16.96)
1	1063	12.00% (0.20%)	9.54% (0.92%)	15.08% (0.71%)	101.40 (9.73)	17.65% (1.32%)	320.50 (15.05)	55.86% (2.06%)	176.10 (13.27)	278.70 (10.47)
2	1063	5.36% (0.05%)	5.45% (0.71%)	11.86% (0.56%)	57.90 (7.55)	10.07% (1.13%)	378.40 (17.90)	65.94% (1.84%)	100.00 (11.40)	219.10 (6.26)
3	1063	3.77% (0.08%)	4.94% (0.83%)	10.13% (0.43%)	52.50 (8.87)	9.15% (1.48%)	430.90 (18.40)	75.08% (1.33%)	91.00 (14.87)	187.30 (3.27)
4	1063	2.72% (0.05%)	3.96% (0.53%)	8.90% (0.42%)	42.10 (5.59)	7.33% (0.84%)	473.00 (22.41)	82.41% (1.84%)	72.80 (8.40)	164.20 (3.68)
5	1062	1.98% (0.04%)	2.77% (0.61%)	7.88% (0.35%)	29.40 (6.52)	5.12% (1.12%)	502.40 (21.97)	87.53% (1.24%)	50.80 (11.06)	145.40 (2.07)
6	1063	1.45% (0.04%)	2.26% (0.39%)	7.08% (0.33%)	24.00 (4.11)	4.17% (0.60%)	526.40 (24.74)	91.70% (0.96%)	41.20 (5.96)	130.60 (1.51)
7	1063	1.05% (0.03%)	1.97% (0.31%)	6.44% (0.29%)	20.90 (3.31)	3.65% (0.59%)	547.30 (24.61)	95.34% (0.88%)	36.00 (6.13)	118.70 (1.34)
8	1063	0.74% (0.02%)	1.32% (0.47%)	5.87% (0.26%)	14.00 (5.01)	2.44% (0.88%)	561.30 (24.46)	97.78% (0.52%)	24.00 (8.77)	108.00 (0.82)
9	1062	0.43% (0.01%)	1.20% (0.28%)	5.40% (0.23%)	12.70 (2.98)	2.22% (0.52%)	574.00 (24.55)	100.00% (0.00%)	21.70 (5.42)	100.00 (0.00)
Total	10,628				574.00					

Table 15: Gains table of the TAN models for the first data set.

NB classifiers at 0.05 level (p-values are 5.424×10^{-6}).

¿From the cumulative lift analysis, the HEA models perform significantly better than the other models for this data set. The average and the standard deviations of the execution time for different methods are summarized in Table 20. It can be observed that HEA is much faster than MDLEP. It is able to discover Bayesian networks from this huge data set in five minutes, while MDLEP takes about 5 hours for processing.

In Figure 17, the models are compared in Receiver Operating Characteristic (ROC) curves. It can be found from the ROC curves that the HEA models perform better than the other models if the probability threshold values are high (i.e. the left hand side of the figure). On the other hand, when the threshold values become smaller, the MDLEP, TAN, and NB models perform better than the HEA models (i.e. the right hand side of the figure). However, it should be noticed that classifiers using small threshold values may not be useful in direct marketing, because they will incorrectly classify several inactive respondents as active.

6.4 Summary

An advertising campaign often involves huge investment, a response model which can categorize more prospects into the target list is valuable as it will enhance the response rate. From the experimental results, it seems that the HEA models are more desirable than the other models.

7. Conclusion

In this paper, we have described a new algorithm, HEA, for learning Bayesian networks effectively and efficiently. The algorithm is tested on a number of Bayesian networks learning problems to show that it can discover good Bayesian networks. We have compared HEA with MDLEP, PheGT$_2^R$, Bayesian Network

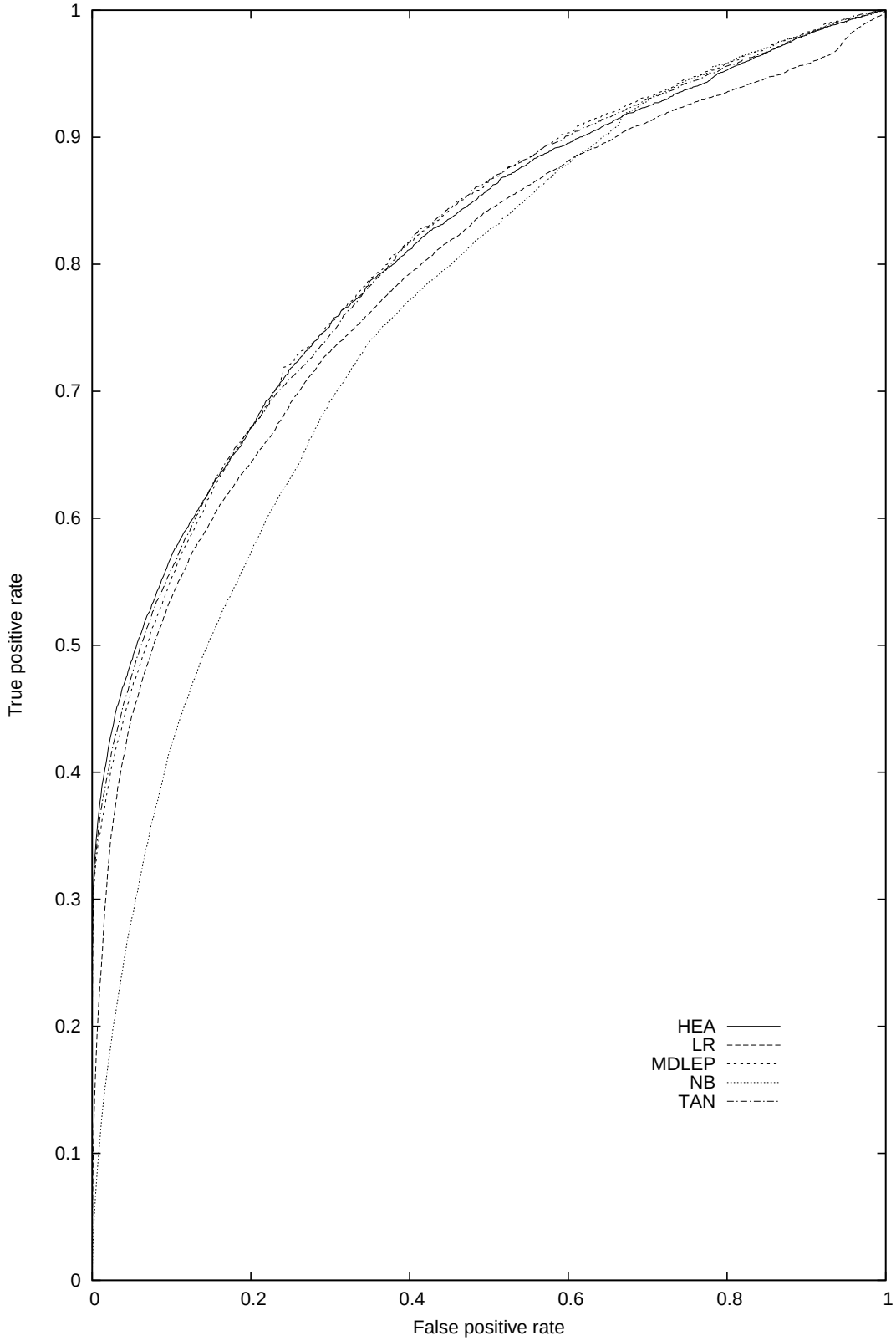


Figure 17: ROC curves of different models for the second data set.

Decile	Records	Prob. of Active	Percent Active	Cum. Percent Active	Actives	% of Total Actives	Cum. Actives	Cum. % of Tot. Actives	Lift	Cum. Lift
0	1063	57.92% (1.12%)	18.96% (1.18%)	18.96% (1.18%)	201.50 (12.51)	35.16% (2.52%)	201.50 (12.51)	35.16% (2.52%)	351.10 (25.22)	351.10 (25.22)
1	1063	24.37% (0.62%)	9.88% (1.36%)	14.42% (0.63%)	105.00 (14.46)	18.27% (2.15%)	306.50 (13.51)	53.43% (1.96%)	182.00 (21.47)	266.60 (9.83)
2	1063	10.25% (0.36%)	6.73% (1.04%)	11.85% (0.47%)	71.50 (11.00)	12.46% (1.90%)	378.00 (15.12)	65.89% (2.00%)	124.10 (18.97)	218.90 (6.64)
3	1063	4.40% (0.16%)	4.48% (0.86%)	10.01% (0.47%)	47.60 (9.14)	8.26% (1.33%)	425.60 (19.95)	74.15% (1.33%)	82.10 (13.30)	184.90 (3.28)
4	1063	2.12% (0.09%)	3.64% (0.73%)	8.74% (0.41%)	38.70 (7.76)	6.74% (1.30%)	464.30 (21.90)	80.89% (1.45%)	66.90 (12.99)	161.20 (2.90)
5	1062	1.17% (0.04%)	3.20% (0.33%)	7.82% (0.34%)	34.00 (3.53)	5.93% (0.67%)	498.30 (21.55)	86.82% (1.38%)	58.80 (6.68)	144.10 (2.38)
6	1063	0.66% (0.03%)	2.44% (0.64%)	7.05% (0.30%)	26.00 (6.78)	4.53% (1.17%)	524.30 (22.36)	91.35% (0.99%)	44.80 (11.69)	129.90 (1.45)
7	1063	0.37% (0.01%)	1.92% (0.51%)	6.40% (0.27%)	20.40 (5.42)	3.56% (0.92%)	544.70 (22.78)	94.90% (0.78%)	35.00 (9.19)	118.20 (1.03)
8	1063	0.20% (0.01%)	1.42% (0.25%)	5.85% (0.25%)	15.10 (2.64)	2.63% (0.42%)	559.80 (23.79)	97.53% (0.54%)	25.80 (4.21)	107.80 (0.63)
9	1062	0.07% (0.01%)	1.34% (0.30%)	5.40% (0.23%)	14.20 (3.22)	2.47% (0.54%)	574.00 (24.55)	100.00% (0.00%)	24.10 (5.43)	100.00 (0.00)
Total	10,628				574.00					

Table 16: Gains table of the NB models for the first data set.

Decile	HEA	Logistic regression	MDLEP	TAN	NB
0	**400.6**	340.9^{+}	379.4^{+}	381.5^{+}	351.1^{+}
1	278.2	249.9^{+}	**291.0**$^{-}$	278.7	266.6^{+}
2	**221.4**	211.1^{+}	220.2	219.1	218.9
3	189.3	186.0^{+}	**199.7**$^{-}$	187.3	184.9^{+}
4	**164.8**	163.0	160.7^{+}	164.2	161.2^{+}
5	146.4	145.0	**149.7**$^{-}$	145.4	144.1^{+}
6	**131.1**	130.6	128.4^{+}	130.6	129.9
7	118.6	118.4	**119.2**	118.7	118.2
8	108.3	108.1	**110.9**$^{-}$	108.0	107.8
9	100.0	100.0	100.0	100.0	100.0

Table 17: Cumulative lifts of different models for the first data set. The highest cumulative lift in each decile is highlighted in bold. The superscript + represents the cumulative lift of the HEA models is significant higher at 0.05 level than that of the corresponding models. The superscript − represents the cumulative lift of the HEA models is significant lower at 0.05 level than that of the corresponding models.

PowerConstructor, and WinMine Toolkit and found that HEA is much more efficient and effective than MDLEP and PheGT$_2^R$. Moreover, HEA usually discovers better Bayesian networks than the other algorithms. With an efficient and effective algorithm, it enables us to explore interesting applications of Bayesian networks on real-life data mining problems.

We have applied HEA to two data sets of direct marketing and compare the Bayesian networks obtained by HEA and the models generated by other methods. From the experimental results, the HEA models predict more accurately than the other models. This study shows that HEA can potentially become a powerful and efficient data mining tool for direct marketing problems.

Appendix: An example that `findSubset` does not return a minimal subset

Consider the two Bayesian networks, G_a and G_b, in Figure 18 and assume that `findSubset` is invoked with $N_i = A$. `findSubset` returns $W' = \{A,\ B,\ C\}$. Thus, the resultant network has the same structure as

	HEA	Logistic regression	MDLEP	TAN	NB
Average execution time (sec.)	53.791	10.0	2552.27	17.789	17.441
Standard deviations	1.117	0.745	235.528	0.951	0.914

Table 18: The execution time for different methods for the first data set.

Decile	HEA	Logistic regression	MDLEP	TAN	NB
0	**501.5** (10.11)	467.1^{+} (13.22)	482.4^{+} (9.66)	492.7^{+} (12.23)	353.3^{+} (8.62)
1	**314.6** (5.58)	302.2^{+} (3.58)	312.3 (4.55)	314.1 (7.22)	263.7^{+} (8.39)
2	239.7 (2.95)	232.2^{+} (3.94)	**241.9** (5.51)	237.4 (4.77)	214.9^{+} (5.15)
3	197.2 (3.39)	191.6^{+} (2.22)	**197.6** (4.74)	196.7 (3.53)	186.8^{+} (3.99)
4	168.1 (2.73)	164.4^{+} (2.07)	**169.5**$^{-}$ (2.80)	**169.5** (2.76)	161.2^{+} (2.78)
5	147.5 (2.17)	144.7^{+} (1.49)	**148.1** (2.60)	148.0 (1.89)	143.7^{+} (2.26)
6	131.0 (1.33)	129.0^{+} (1.15)	**131.9** (1.91)	131.3 (1.83)	131.0 (1.41)
7	118.4 (0.84)	116.0^{+} (1.15)	118.6 (1.43)	118.4 (1.26)	**118.9** (0.88)
8	**108.5** (0.71)	105.6^{+} (1.07)	**108.5** (0.71)	108.2 (0.79)	**108.5** (0.53)
9	100.0 (0.00)	100.0 (0.00)	100.0 (0.00)	100.0 (0.00)	100.0 (0.00)

Table 19: Cumulative lifts of different models for the second data set. The highest cumulative lift in each decile is highlighted in bold. The superscript + represents the cumulative lift of the HEA models is significant higher at 0.05 level than that of the corresponding models. The superscript − represents the cumulative lift of the HEA models is significant lower at 0.05 level than that of the corresponding models.

G_b. However, it is clear that another acyclic network can be obtained by replacing Π_A^a with Π_A^b and Π_B^a with Π_B^b. In other words, $W' = \{A,\ B,\ C\}$ is not minimal. This situation occurs because `findSubset` decides to add B to W' by examining if there are direct paths from A to B in G_a and determines to insert C to W' by checking if there are direct paths from A to C in G_a.

Another implementation of `findSubset` replaces Π_A^a with Π_A^b to generate an intermediate network G'_a. Since there is a cycle $A \rightarrow B \rightarrow C$ in G'_a, The implementation replaces $\Pi_B^{a'}$ with Π_B^b to create another intermediate network G''_a. In G''_a, the edge $A \rightarrow B$ is removed and there is no cycle. Thus, it is not necessary to place C to W'. This implementation can find the minimal subset of nodes $\{A,\ B\}$, but it cannot guarantee that the minimal subset of nodes will lead to the optimal solution. Moreover, it is more expensive than our implementation in HEA because the former has to create many intermediate networks.

Acknowledgment

The authors are grateful to Prof. Xin Yao and anonymous referees for their constructive suggestions for improving the paper. This research was supported by the Earmarked Grant LU 3012/01E from the Research Grant Council of the Hong Kong Special Administrative Region.

	HEA	Logistic regression	MDLEP	TAN	NB
Average execution time (sec.)	281.03	68.61	18,707.91	95.37	92.87
Standard deviations	9.227	0.167	2,111.281	6.338	6.338

Table 20: The execution time for different methods for the second data set.

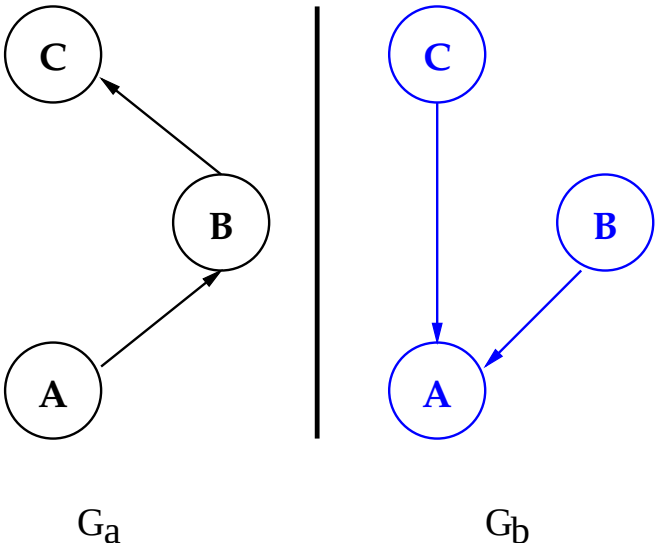


Figure 18: Two Bayesian networks used by `findSubset`.

References

[1] P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction, and Search*. MA: MIT Press, second ed., 2000.

[2] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, eds., *Advances in Knowledge Discovery and Data Mining*. AAAI Press, 1996.

[3] W.-H. Au, K. C. C. Chan, and X. Yao, "A novel evolutionary data mining algorithm with applications to churn prediction," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 532–545, December 2003.

[4] C. Zhou, W. Xiao, T. M. Tirpak, and P. C. Nelson, "Evolving accurate and compact classification rules with gene expression programming," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 519–531, December 2003.

[5] J. R. Cano, F. Herrera, and M. Lozano, "Using evolutionary algorithms as instance selection for data reduction in kdd: An experimental study," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 561–575, December 2003.

[6] J. Atkinson-Abutridy, C. Mellish, and S. Aitken, "A semantically guided and domain-independent evolutionary model for knowledge discovery from texts," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 546–560, December 2003.

[7] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[8] F. V. Jensen, *An Introduction to Bayesian Network*. University of College London Press, 1996.

[9] D. Heckerman and E. Horvitz, "Inferring informational goals from free-text queries: A Bayesian approach," in *Proceedings of Fourteenth Conference of Uncertainty in Artificial Intelligence* (G. F. Cooper and S. Moral, eds.), (Wisconsin, USA), pp. 230–237, Morgan Kaufmann, July 1998.

[10] D. Heckerman and M. P. Wellman, "Bayesian networks," *Communications of the ACM*, vol. 38, pp. 27–30, March 1995.

[11] G. F. Cooper, "A simple constraint-based algorithm for efficiently mining observational databases for causal relationships," *Data Mining and Knowledge Discovery*, vol. 1, pp. 203–224, 1997.

[12] D. Heckerman, "Bayesian networks for data mining," *Data Mining and Knowledge Discovery*, vol. 1, pp. 79–119, 1997.

[13] C. Silverstein, S. Brin, R. Motwani, and J. Ullman, "Scalable techniques for mining causal structures," *Data Mining and Knowledge Discovery*, vol. 4, pp. 163–192, 2000.

[14] M. L. Wong, W. Lam, K. S. Leung, P. S. Ngan, and J. C. Y. Cheng, "Discovering knowledge from medical databases using evolutionary algorithms," *IEEE Engineering in Medicine and Biology Magazine*, vol. 19, no. 4, pp. 45–55, 2000.

[15] Y. Xiang and T. Chu, "Parallel learning of belief networks in large and difficult domains," *Data Mining and Knowledge Discovery*, vol. 3, pp. 315–339, 1999.

[16] J. Cheng, R. Greiner, J. Kelly, D. Bell, and W. Liu, "Learning Bayesian network from data: An information-theory based approached," *Artificial Intelligence*, vol. 137, pp. 43–90, 2002.

[17] G. Cooper and E. Herskovits, "A Bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, no. 4, pp. 309–347, 1992.

[18] D. Heckerman, "A tutorial on learning Bayesian networks," tech. rep., Microsoft Research, Advanced Technology Division, March 1995.

[19] W. Lam and F. Bacchus, "Learning Bayesian belief networks-an approach based on the MDL principle," *Computational Intelligence*, vol. 10, no. 4, pp. 269–293, 1994.

[20] M. L. Wong, W. Lam, and K. S. Leung, "Using evolutionary programming and minimum description length principle for data mining of Bayesian networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, pp. 174–178, February 1999.

[21] S. Y. Lee, K. S. Leung, and M. L. Wong, "Improving the efficiency of using evolutionary programming for Bayesian Network learning," in *Late Breaking Papers of the 2001 Genetic and Evolutionary Computation Conference*, pp. 252–259, 2001.

[22] M. L. Wong, S. Y. Lee, and K. S. Leung, "A hybrid approach to discover Bayesian networks from databases using evolutionary programming," in *Proceedings of the 2002 IEEE International Conference on Data Mining*, pp. 498–505, 2002.

[23] M. L. Wong, S. Y. Lee, and K. S. Leung, "A hybrid approach to learn Bayesian networks using evolutionary programming," in *Proceedings of 2002 Congress on Evolutionary Computation*, pp. 1314–1319, 2002.

[24] M. L. Wong, S. Y. Lee, and K. S. Leung, "A hybrid data mining approach to discover Bayesian networks using evolutionary programming," in *Proceedings of the 2002 Genetic and Evolutionary Computation Conference*, pp. 214–221, 2002.

[25] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine Learning*, vol. 29, pp. 131–163, 1997.

[26] P. Langley and S. Sage, "Induction of selective Bayesian classifier," in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence* (R. L. de Mantaras and D. Poole, eds.), (Seattle, Washington), pp. 399–406, Morgan Kaufmann, July 1994.

[27] R. M. Fung and S. L. Crawford, "Constructor: A system for the induction of probabilistic models," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 762–769, 1990.

[28] R. Neapolitan, *Probabilistic Reasoning in Expert Systems*. New York: Wiley, 1990.

[29] A. Agresti, *Categorical Data Analysis*. New York: Wiley, 2002.

[30] C. Spatz and J. O. Johnston, *Basic statistics: tales of distribution*. Monterey, California: Brooks/Cole Publishing Company, second ed., 1981.

[31] G. P. Beaumont and J. D. Knowles, *Statistical Tests: An introduction with MINITAB commentary*. Prentice-Hall, 1996.

[32] L. M. de Campos and J. F. Huete, "Approximating causal orderings for Bayesian networks using genetic algorithms and simulated annealing," tech. rep., Department of Computer Science and Artificial Intelligence, University of Granada, Spain, May 1999.

[33] D. Dash and M. J. Druzdzel, "A hybrid anytime algorithm for the construction of causal models from sparse data," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (K. B. Laskey and H. Prade, eds.), (Stockholm, Sweden), pp. 142–149, Morgan Kaufmann, July–August 1999.

[34] J. Rissanen, "Modelling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[35] R. R. Bouckaert, *Bayesian Belief Networks: from Inference to Construction*. PhD thesis, Utrecht University, June 1995.

[36] J. Suzuki, "Learning Bayesian belief networks based on the minimum description length principle: Basic properties," in *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E82-A, pp. 2237–2245, November 1999.

[37] D. M. Chickering, D. Geiger, and D. Heckerman, "Learning Bayesian network is NP-Hard," tech. rep., Microsoft Research, 1994.

[38] J. Tian, "A branch-and-bound algorithm for MDL learning Bayesian networks," in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (C. Boutilier and M. Goldszmidt, eds.), (San Francisco, California), pp. 580–588, Morgan Kaufmann, June–July 2000.

[39] P. Larrañaga, M. Poza, Y. Yurramendi, R. Murga, and C. Kuijpers, "Structural learning of Bayesian network by genetic algorithms: A performance analysis of control parameters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, pp. 912–926, September 1996.

[40] M. Brameier and W. Banzhaf, "A comparison of linear genetic programming and neural networks in medical data mining," *IEEE Transactions on Evolutionary Computation*, vol. 5, pp. 17–26, February 2001.

[41] S. Bhattacharyya, O. V. Pictet, and G. Zumbach, "Knowledge-intensive genetic discovery in foreign exchange markets," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 169–181, April 2002.

[42] T. Bäck, D. B. Fogel, and Z. Michalewicz, eds., *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, 2000.

[43] T. Bäck, D. B. Fogel, and Z. Michalewicz, eds., *Evolutionary Computation 2: Advanced Algorithms and Operators*. Institute of Physics Publishing, 2000.

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

[45] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[46] D. B. Fogel, *Evolutionary Computation: Toward a New Philosohpy of Machine Intelligence*. IEEE Press, 2000.

[47] L. Fogel, A. Owens, and M. Walsh, *Artificial Intelligence Through Simulated Evolution*. John Wiley and Sons, 1966.

[48] K.-U. Höffen, "Learning and robust learning of product distributions," *ACM COLT*, pp. 77–83, 1993.

[49] P. Larrañaga, C. Kuijpers, R. Mura, and Y. Yurramendi, "Learning Bayesian network structures by searching for the best ordering with genetic algorithms," *IEEE Transactions on System, Man and Cybernetics*, vol. 26, pp. 487–493, April 1996.

[50] C. Cotta and J. Muruzábal, "Towards a more efficient evolutionary induction of Bayesian networks," in *Proceedings of the seventh international conference on Parallel Problem Solving From Nature: PPSN VII* (J. M. Guervós, P. Adamidis, H.-G. Beyer, and J.-L. F.-V. H.-P. Schwefel, eds.), (Granada, Spain), pp. 730–739, Springer-Verlag, September 2002.

[51] J. W. Myers, K. B. Laskey, and K. A. DeJong, "Learning Bayesian networks from incomplete data using evolutionary algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference* (W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, eds.), vol. 1, (Orlando, Florida, USA), pp. 458–465, Morgan Kaufmann, 1999.

[52] J. W. Myers, K. B. Laskey, and T. S. Levitt, "Learning bayesian networks from incomplete data with stochastic search algorithms," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (K. B. Laskey and H. Prade, eds.), vol. 1, (San Francisco, Califonia, USA), pp. 476–485, Morgan Kaufmann, 1999.

[53] K. B. Laskey and J. W. Myers, "Population markov chain monte carlo," *Machine Learning*, vol. 50, pp. 175–196, 2003.

[54] A. Tucker, X. Liu, and A. Ogden-Swift, "Evolutionary learning of dynamic probabilistic models with large time lags," *The International Journal of Intelligent Systems*, vol. 16, no. 5, pp. 621–646, 2001.

[55] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, pp. 694–713, May 1997.

[56] S. van Dijk, D. Thierens, and L. G. van der Gaag, "Building a GA from design principles for learning Bayesian networks," in *Proceedings of the 2003 Genetic and Evolutionary Computation Conference, Part I* (E. Cantú-Paz, J. Foster, K. Deb, L. Davis, U.-M. O. R. Roy, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska, and J. Miller, eds.), vol. 2723 of *Lecture Notes in Computer Science*, pp. 886–897, Springer-Verlag, 2003.

[57] S. A. Andersson, D. Madigan, and M. D. Perlman, "A characterization Markov equivalence classes for acyclic digraphs," *Annals of Statistics*, vol. 25, pp. 505–541, 1997.

[58] D. M. Chickering, "Learning equivalence classes of Bayesian-network structures," *Journal of Machine Learning Research*, vol. 2, pp. 445–498, 2002.

[59] C. Cotta and J. M. Troya, "Analyzing directed acyclic graph recombination," in *Computational Intelligence: Theory and Applications* (B. Reusch, ed.), vol. 2206 of *Lecture Notes in Computer Science*, pp. 739–748, Springer-Verlag, 2001.

[60] D. M. Chickering, "The WinMine toolkit," Tech. Rep. MSR-TR-2002-103, Microsoft Research, Redmond, Washington, October 2002.

[61] M. Henrion, "Propagating uncertainty in Bayesian networks by logic sampling," in *Proceedings of the Second Conference on Uncertainty in Artificial Intelligence* (J. Lemmar and L. Kanal, eds.), (Amsterdam, North Holland), pp. 149–163, 1988.

[62] I. Beinlinch, H. Suermondt, R. Chavez, and G. Cooper, "The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks," in *Proceedings of Second European Conference Artificial Intelligence in Medicine*, pp. 247–256, 1989.

[63] M. Singh and M. Valtorta, "An algorithm for the construction of Bayesian network structures from data," in *Proceedings of the Ninth Conference of Uncertainty in Artificial Intelligence* (D. Heckerman and E. H. Mamdani, eds.), (San Mateo, CA), pp. 259–265, Morgan Kaufmann, 1993.

[64] S. Acid and L. M. de Campos, "BENEDICT:an algorithm for learning probabilistic belief networks," in *Proceedings of the IPMU-96 Conference*, 1996.

[65] "Norsys Bayes net library." http://www.norsys.com/net_library.htm.

[66] S. L. Lauritzen and D. J. Spieglhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of Royal Statistics Society*, vol. 50, no. 2, pp. 157–194, 1988.

[67] J. Zahavi and N. Levin, "Issues and problems in applying neural computing to target marketing," *Journal of Direct Marketing*, vol. 11, no. 4, pp. 63–75, 1997.

[68] S. Bhattacharyya, "Direct marketing response models using genetic algorithms," in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 144–148, 1998.

[69] L. A. Petrison, R. C. Blattberg, and P. Wang, "Database marketing: Past present, and future," *Journal of Direct Marketing*, vol. 11, no. 4, pp. 109–125, 1997.

[70] P. Cabena, P. Hadjinian, R. Stadler, J. Verhees, and A. Zansi, *Discovering Data Mining: From Concept to Implementation*. Prentice-Hall Inc., 1997.

[71] S. Bhattacharyya, "Evolutionary algorithms in data mining: Multi-objective performance modeling for direct marketing," in *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, pp. 465–473, August 2000.

[72] J. Zahavi and N. Levin, "Applying neural computing to target marketing," *Journal of Direct Marketing*, vol. 11, no. 4, pp. 76–93, 1997.

[73] C. X. Ling and C. Li, "Data mining for direct marketing: Problems and solutions," in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 73–79, 1998.

[74] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156, 1996.

[75] E. J. Keogh and M. J. Pazzani, "Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches," in *Proceedings of the Seventh International Workshop on AI and Statistics* (D. Heckerman and J. Whittaker, eds.), (Fort Lauderdale, Florida), pp. 225–230, Morgan Kaufmann, January 1999.

[76] M. Singh, *Learning Bayesian Networks for Solving Real-World Problems*. PhD thesis, University of Pennsylvania, 1998.

[77] O. P. Rud, *Data Mining Cookbook: modeling data for marketing, risk and customer relationship management*. New York: Wiley, 2001.

[78] F. Provost, T. Fawcett, and R. Kohavi, "The case against accuracy estimation for comparing induction algorithms," in *Proceedings of 15th International Conference on Machine Learning*, pp. 445–453, Morgan Kaufmann, San Francisco, CA, 1998.