Data Mining of Bayesian Networks Using Cooperative Coevolution

Man Leung Wong
Department of Computing and Decision Sciences
Lingnan University
Hong Kong
mlwong@ln.edu.hk

Shing Yan Lee
Department of Computer Science
and Engineering
The Chinese University
of Hong Kong, Hong Kong
sylee@cse.cuhk.edu.hk

Kwong Sak Leung
Department of Computer Science
and Engineering
The Chinese University
of Hong Kong, Hong Kong
ksleung@cse.cuhk.edu.hk

Abstract

This paper describes a novel data mining algorithm that employs cooperative coevolution and a hybrid approach to discover Bayesian networks from data. A Bayesian network is a graphical knowledge representation tool. However, 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 approach searches good network structures according to a metric. Unfortunately, the two approaches both have their own drawbacks. Thus, we propose a novel algorithm that combines the characteristics of these approaches to improve learning effectiveness and efficiency. The new learning algorithm consists of 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 networks are generated by a cooperative coevolution genetic algorithm. We conduct a number of experiments and compare the new algorithm with our previous algorithm, Minimum Description Length and Evolutionary Programming (MDLEP), which uses evolutionary programming for network learning. The results illustrate that the new algorithm has better performance. We apply the algorithm to a large real-world data set and compare the performance of the discovered Bayesian networks with that of the back-propagation neural networks and the logistic regression models. This study illustrates that the algorithm is a promising alternative to other data mining algorithms.

Keywords

Cooperative Coevolution; Evolutionary Computation; Data Mining; Bayesian Networks

1. Introduction

Data Minining is defined as the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [19]. The whole process of Data Mining consists of several steps. Firstly, the problem domain is analyzed to determine the objectives. Secondly, data is collected and an initial exploration is conducted to understand and verify the quality of the data. Thirdly, data preparation such as selection is made to extract relevant data sets from the database. The data is preprocessed to remove noise and to handle missing data values. Transformation may be performed to reduce the number of variables under consideration. A suitable data mining algorithm is then employed on the prepared data to discover knowledge represented in different representations such as decision trees, rules, and Bayesian networks. Finally the result of data mining is interpreted and evaluated. If the discovered knowledge is not satisfactory, these steps will be iterated. The discovered knowledge is then applied in decision making.

Recently, there is increasing interest in discovering knowledge represented in Bayesian Networks [16, 25, 47, 54, 55], because Bayesian networks can handle incomplete data sets and facilitate the combination of domain knowledge and data. Moreover, Bayesian networks provide an efficient way for avoiding the over fitting problem and allow one to learn about causal relationships [25]. In this paper, we propose a novel data mining algorithm that employs cooperative coevolution and a hybrid approach to learn knowledge represented in Bayesian networks from data. A Bayesian network is a graphical representation that depicts conditional independence among random variables in the domain and encodes the joint probability distribution [40]. A Bayesian network is composed of a structure and a number of conditional probabilities as shown in Figure 1. With a network at hand, probabilistic inference can be performed to predict the outcome of some variables based on the observations of others [1]. In light of this, Bayesian networks are widely used in diagnostic and classification systems. For example, they are used for diagnosing diseases in muscles, nerve, and

[1]There are several commerical and free software such as HUGIN [32], Bayesian Network tools in Java [10], and Microsoft Belief Network Tools [37] that can perform probabilistic reasoning given a Bayesian network.

lymph nodes [39, 28]. Besides, they are also used in information retrieval [26] and printer troubleshooting problems [27].

The main task of learning Bayesian networks from data is to automatically find directed edges between the nodes. Once the network structure is constructed, the conditional probabilities are readily calculated based on the data. In the literature, there are two main approaches to learning network structure from data [14]. The first one is the dependency analysis approach [50, 14]. Since a Bayesian network describes conditional independence, we could make use of dependency test results to construct a Bayesian network structure that conforms to our findings. The second one, called the score-and-search approach [30, 34, 24], 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 should be performed. Moreover, some test results may be inaccurate [50]. For the latter approach, since the search space is huge, some Bayesian network structure learning algorithms [30] adopt greedy search heuristics which may easily make the algorithms get stuck in a local optimum [24].

In this work, a hybrid approach is developed for the network structure 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. A modular decomposition evolutionary search approach, cooperative coevolution [43, 44, 42], is employed to search for good Bayesian network structures. Our new algorithm for learning Bayesian network structures is called CCGA (Cooperative Coevolution Genetic Algorithm). We have conducted a number of experiments and compare CCGA with our previous algorithm, Minimum Description Length and Evolutionary Programming (MDLEP). The empirical results illustrate that CCGA outperforms MDLEP. Moreover, it is found that CCGA executes much faster than MDLEP which is very important for real-world applications. We evaluate CCGA on a real-world data set of direct marketing and compare the performance of the discovered Bayesian networks with that of the back-propagation neural networks and the logistic regression models.

This paper is organized as follows. In Section 2, we present the backgrounds of Bayesian networks, the Minimum Description Lenght (MDL) metric and cooperative coevolution. Different methods of applying evolutionary computation to learn Bayesian network structures

are presented in Section 3. In Section 4, we describe our algorithm in detail. In Section 5, we present a comparison between the new algorithm and our previous algorithm (MDLEP) together with a parameter study. In Section 6, We apply the algorithms to a data set of direct marketing and compare the performance of different models. We conclude the paper in Section 7.

2. Backgrounds

2.1 Bayesian Network Learning

A Bayesian network, G, has a directed acyclic graph (DAG) structure. 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 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 [40].

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 let P be the joint probability distribution of U. Following Pearl's notation, 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 with [40]:

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

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

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

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

2.1.1 The Dependency Analysis Approach

As mentioned before, researchers treat the network structure learning problem in two very different ways. The first approach is called the dependency analysis approach which includes the algorithms in [50], [22], and [14]. The approach tries to construct a Bayesian network structure using dependency information obtained from the data. It produces 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 [38, 40] .

A conditional independence (CI) test checks the validity of an independence assertion $I(X, Z, Y)$ by performing a statistical hypothesis testing procedure [50, 49, 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, the χ^2 statistics is calculated by:

$$g^2 = -2 \sum_{x,y,z} P(x, y, z) \log \frac{P(x, y, z)}{P(y, z)P(x \mid z)}. \tag{3}$$

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 [6]. 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 (Spirtes, Glymour, and Scheines) algorithm [50] begins with a completely connected undirected graph. In other words, dependence between every pair of variables is assumed. Then, 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 structure.

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 [50] "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, every possible combinations of the conditioning set needs 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 [50, 18]. As described in equation 3, the χ^2 statistics depends on $P(x, y, z)$, $P(y, z)$, and $P(x \mid z)$ where x, y, and z are respectively different value assignments to the sets of variables X, Y, and Z. If there are many variables in the conditioning set Z, there may be very few examples in the data set that satisfy a particular value assignment z, and $P(x \mid z)$ may be inaccurate if there is noise in the examples. Similar issues may occur for $P(x, y, z)$ and $P(y, z)$. Hence, for algorithms that require high order CI tests, the results may be inaccurate. Third, because a network structure 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 [50, 17]. Moreover, this suggests that the order of testing the CI relations is important, which will be a concern when one pursues for the optimal performance.

2.1.2 The Search-and-scoring Approach

The second approach makes use of a metric which evaluates the quality of a Bayesian network structure with respect to the given data. Such metric may be derived from information theory, Bayesian statistics, or Minimum Description Length (MDL) principle [45]. Though their theoretical foundations are different, some studies [11, 51] show that different metrics are asymptotically equivalent under certain conditions.

Since we employ the MDL metric [34] in our work, we take it as an example for illustration. Basically, the metric is derived from information theory and incorporates the Minimum Description Length principle. It tries to balance between model accuracy and model complexity. Hence, the best network needs to be both accurate and simple. Using the metric, a better network structure would have a smaller score.

Similar to other metrics, the MDL score for a Bayesian network, G, is *decomposable* [24] and could be written as in equation 4. Let $U = \{N_1, \ldots, N_n\}$ be the set of nodes and let Π_{N_i} denotes the parent set of node N_i. The MDL score of the network is simply the summation of the MDL score of Π_{N_i} of every node N_i in the network.

$$\mathrm{MDL}(G) = \sum_{N_i \in U} \mathrm{MDL}(N_i,\ \Pi_{N_i}) \tag{4}$$

while the method for computing $\mathrm{MDL}(N_i,\ \Pi_{N_i})$ can be found in reference [34].

With the defined metric, the network structure 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, that 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 [15]. Some algorithms, therefore, resort to greedy search heuristics [30, 34]. 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 [52], to find the optimal solution. In the worst case, the time consumed would be considerable. Recently, some researchers attempted [35, 53] to use evolutionary computation to tackle the problem.

2.2 Evolutionary Computation

Evolutionary computation is a general stochastic search methodology. The principal idea borrows from evolution mechanisms proposed by Charles Darwin. Evolutionary computation is becoming popular as it often gives satisfactory result for various optimization problems in different areas. For example, it is applied in data mining, image processing, pattern recognition, and signal processing [2, 20, 33, 3, 4, 23].

In essence, evolution computation is a group search algorithm with guidance. A candidate solution in the search space is called a *chromosome*. A chromosome consists of a number of *genes*, which correspond to the elements constituting a solution. At the beginning, a *population* of chromosomes is created randomly. For each chromosome, its fitness value is computed according to a predefined *fitness function*. In subsequent *generations*, new chromosomes (the *offspring*) are created by genetic operators which alter the genetic composition of the parental chromosomes. Then, selection comes into play where the weaker ones will vanish while stronger ones will have higher chance to survive onto the next generation. This process is repeated until certain termination criterion is satisfied. Because better ones will have higher chance to survive, it is expected that a good, or near optimal, solution can be obtained ultimately.

2.2.1 Cooperative Coevolution

Coevolution is the evolution of different species in the same environment, where the interactions among them affect the genetic composition of one and others. There are two kinds of coevolution: competitive and cooperative. In nature, the kind of coevolution we often see is competitive coevolution. For instance, the "arm race" between two species is a good

demonstration of competitive coevolution. In cooperative coevolution, the natural selection pressure will prefer individuals that could have good collaboration with other species.

Based on the work of Potter and DeJong [43, 44, 42], cooperative coevolution represents a problem breakdown methodology. A problem instance is divided into a number of subcomponents that correspond to different species. The analogy is that once species (i.e. solutions to subcomponents) could cooperate among themselves, the collaboration (i.e. the assembled solution) will be a good solution.

In each species population, evolutionary search is conducted separately. During fitness evaluation, an individual is assigned a fitness value so that cooperation is promoted. To achieve this, a collaborative structure $\mathcal{S}$ is first assembled from representatives of different species populations. Note that $\mathcal{S}$ now is a complete solution to the original problem. When an individual is subject to fitness evaluation, it replaces its representative in $\mathcal{S}$ and forms $\mathcal{S}'$. As such, the individual is assigned with the fitness value of $\mathcal{S}'$ which reflects, to a certain degree, how good it cooperates with other individuals in other species.

By using cooperative coevolution, a hard and complex problem could be handled in a systematic and efficient manner. For example, cooperative coevolution is applied in learning neural networks [43] and in learning of sequential decision rules [44].

3. Learning Bayesian Networks Using Evolutionary Computation

Recently, there are two approaches [35, 53] that apply evolutionary computation to tackle the problem of learning Bayesian network structures using the search-and-scoring approach. The first one uses genetic algorithms (GAs) while the later one uses evolutionary programming (EP).

3.1 Learning Bayesian Network Using Genetic Algorithms

Larrañaga et al. [35] proposed to use genetic algorithms [23, 31] to search for the optimal Bayesian network structure. In their research, the network structure (composed of n nodes) is represented by an $n \times n$ "connectivity matrix" C which is, in effect, the transpose of the adjacency matrix. Each element C_{ij} in the matrix is defined as:

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

With this representation, the i^{th} row in the matrix encodes the parent set of node N_i (i.e. Π_{N_i}). 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) was 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 [15]) which is also used in the K2 algorithm [30]. Note that since the genetic operators could generate illegal offspring structures (i.e. networks that are not directed acyclic graphs), cycles repairing is needed after an offspring is produced.

Because it is rare to have a densely connected network in real-world problems, they imposed a limit on the number of parents a node could have in their implementation. They 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.

3.2 MDLEP

Wong et al. [53] used evolutionary programming (EP) to tackle the learning problem. Since they used the Minimum Description Length metric [34] 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 [21, 20]. Unlike the restricted use of string in GAs, EP does not have any restriction in solution representation. An individual in MDLEP is simply the connectivity 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 2.

In essence, MDLEP uses four mutation operators which include simple mutation, reverse mutation, move mutation, and knowledge-guided mutation. The simple mutation operator randomly picks an edge, 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 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 (A Logical Alarm Reduction Mechanism) and PRINTD (Printing Diagnosis). 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.

3.3 Problems of the Previous Approaches

Although the EP approach outperforms its GA opponent, we observe that it often 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, [2] which is not practical for real-world applications. 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 parent) 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 not efficient enough in finding good solutions.

4. A Hybrid Approach for Learning Bayesian Networks

Since the two approaches for Bayesian network structure learning have their own advantages and drawbacks, we propose a new hybrid approach that combines the characteristics of the two existing approaches to improve learning effectiveness and efficiency. In essence,

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

information from dependency analysis is exploited in the search-and-scoring process so that the searching will be more efficient.

In this research, we assume that there is no missing values or hidden variables in the given data set and we assume no prior knowledge about the structures is provided. Similar to the previous approaches using GAs and EP, we assume that a node could not have more than a specific number of parents. In our new hybrid approach, we evaluate the quality of a candidate network using the MDL metric, although other metrics, like Bayesian score could also be used.

4.1 A Hybrid Approach

In the dependency analysis approach, CI tests are typically used to check the validity of a conditional independence assertion $I(X, Z, Y)$ of any given two nodes X, Y and a conditioning set Z. In a simplest sense, if the assertion $I(X, Z, Y)$ is valid, X and Y cannot be connected because this will violate that X and Y are being d-separated. In other words, neither the edge $X \rightarrow Y$ nor the edge $X \leftarrow Y$ will present in the resultant network G. In the SGS algorithm, the same rationale is used in constructing a Bayesian network in the initial steps where X—Y is removed from an undirected connected graph for each verified assertion $I(X, Z, Y)$.

With such observation, we formulate a general hybrid approach for Bayesian network learning which consists of two phases. In the first phase, we conduct low order CI tests so that we know which edge could be removed. As discussed in Section 2.1, a CI relation and a CI test are characterized by their order, which is the number of variables in the conditioning set. Low order CI tests are used because their results are more reliable than higher order CI tests [50] while the time complexity is bounded. In the second phase, we use a search-and-score approach together with the knowledge obtained previously. In particular, we refine the search space by excluding networks that contain the edges $X \rightarrow Y$ and $X \leftarrow Y$ for every verified assertion $I(X, Z, Y)$. Consequently, the search space is reduced which would imply a speedup for the search process because we would not waste time in adding (or removing) potentially *wrong* edges. The proposed approach is general in the sense that it does not necessitate a particular testing procedure for the CI test phase, or a particular search method in the search phase.

In this work, we propose to use cooperative coevolution for searching because the problem contains certain characteristics that would benefit a modular decomposition technique. In

Figure 3, we provide an outline of the algorithm. Because we use Cooperative Coevolution and Genetic Algorithms, we call our new algorithm CCGA for short.

4.2 CI Test Phase

Initially, we let the possible parent set of each node to contain all other nodes. Using the hybrid approach discussed before, 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 view as though the edges $X \leftarrow Y$ and $X \rightarrow Y$ are both excluded for further consideration. As described in Section 2.1.1, higher order CI tests may be unreliable, thus 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 2.1.1 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 our findings in CI test phase, it is impossible to obtain the optimal solution in the subsequent search phase. In our implementation, we choose a *moderate* α value in order to lessen the reliance on the test results.

4.3 Cooperative Coevolution Search Phase

As mentioned before, cooperative coevolution is a kind of problem breakdown. In our case, we divide the the network learning problem of n variables into n sub-problems, the goal of which is to find the "optimal" parent set for each node. Alternatively, it could be viewed as though we divide the matrix representation into rows as is illustrated in Figure 4.

Consequently, each candidate solution to the sub-problems is represented as a bit-string, which readily suggests the use of genetic algorithms for solving the problems. Following

the convention, we call the search population of each sub-problem *a species population*. Suppose that there are n variables, there will be n different species populations. Inside each population, we use a simple GA to search for the optimal solution.

Although such problem breakdown seems plausible, it is required that the composite network must be acyclic. Obviously, illegal solution could be avoided only if each species population has the knowledge of others and work cooperatively to prevent cycles formation. To realize this idea, we propose an approach which makes use of the topological ordering of a graph as a guidance. In the following sub-sections, we shall discuss our algorithm in detail.

4.3.1 A Feedback Mechanism

Noting that every legal network (i.e. an acyclic graph) conforms to a topological ordering, it follows that we could use an ordering as constraints for each species population so as to avoid cycles formation. In particular, the possible parent set of a node, which defines the search space of the corresponding species population, could only consist of nodes preceding it in the given ordering. Consequently, a composite of the solutions from the populations will be acyclic (as it conforms with the given ordering). Alternatively, it could be viewed as if we are to search the optimal network for a given ordering.

Using this idea, we propose a feedback mechanism. Essentially, we use the node ordering implied by the collaborative structure, $\mathcal{S}$, to produce constraints for each species population such that each candidate solution will conform with the ordering. Next, we update $\mathcal{S}$ with results from the species populations and start another cycle. We picture this idea in Figure 5.

Nevertheless, there is a problem in the feedback mechanism, namely, $\mathcal{S}$ will conform to the same ordering for whatever update is made. Eventually, this will drive the search process to return the optimal network for an initial ordering, which is fine only if the ordering is optimal.

To tackle this problem, we, therefore, use $\mathcal{S}$ only to *approximate* an ordering. In particular, every directed edge $X \rightarrow Y$ in $\mathcal{S}$ is associated with certain degree of belief (i.e. by throwing a dice) that it also appears in the optimal structure. If our degree of belief is less than a fixed threshold, the *belief factor*, it suggests doubt on the correctness of the ordering imposed by the edge. Hence, we allow Y to appear in the parent set of X, which is otherwise forbidden. As a result, a new $\mathcal{S}$ could exhibit an ordering which is different from the original one. However, the drawback is that we should watch out for possible cycles formation.

4.3.2 Initialization

At the beginning, the chromosomes in the species populations are randomly initialized. After the populations are initialized, we assemble $\mathcal{S}$ using the best individual from each population. Note that, in this way, $\mathcal{S}$ will probably contain cycle(s). Next, for each node N_i in the network, we create a new network $\mathcal{S}'$ which copies $\mathcal{S}$ except that N_i is a root node in $\mathcal{S}'$ (i.e. its parent set is empty). Then, the network $\mathcal{S}'$ is repaired for cycles. Using $\mathcal{S}'$ as a reference, we produce constraints on the species population of node N_i such that we assure each candidate solution, when substituted to $\mathcal{S}'$, will create a network that is still acyclic.

4.3.3 Searching inside the Species Populations

For every species population, the search space is equivalent to the possible parent set of the corresponding node. As mentioned before, the possible parent set of a node is subject to different changes during the course of searching. Consequently, the corresponding search process of the node faces both *permanent* and *temporary* constraints. For the permanent constraints, we refer to the reduction of the possible parent set due to the result from CI test. For the temporary constraints, we refer to the changes due to the aforementioned ordering implied by $\mathcal{S}$. As a result of these constraints, the length of the bit-string and the mapping (i.e. which bit corresponds to which parent) are varying.

With the varying bit-string representation, a simple GA with crossover and mutation is used to create a new population. However, instead of using one-point or two-point crossover, we devise a modified crossover operator. Since we have a limit k on the size of the parent set, large part of the bit-string are '0's. As an illustration, suppose that the number of all possible parents of a node is 30, and that k equals to five, the bit-string will contain many '0's and a few '1's. As a result, one-point or two-point crossover will be very likely to exchange segments of '0's and create nothing new.

To circumvent this problem, we use an approach similar to uniform crossover [5]. For the two bit-strings that take part in crossover, we create two different masks for each of them. As an illustration, Figure 6 shows the mask defined and the bit-strings. Here, suppose that the number of all possible parents of a node is six and hence the bit-strings are six-bit long. A mask of equal length is created for each bit-string. At position where the original string is '0', the mask is undefined. At position where the original string is '1', the mask has a value of either '1' or '0'. Consider the upper bit-string in Figure 6, it has '1's only at the

second, the fourth, and the sixth bits. Hence, the mask value is undefined for the first, the third, and the fifth bits. If the mask value is '0', the corresponding bit is copied to its offspring. Otherwise, if the mask value is '1', the corresponding bit is copied to the offspring of the other partner. Referring to Figure 6, the upper mask have the value '1' at fourth bit position, thus the corresponding bit is copied to the offspring of the other partner. The action is indicated by the arrow in the figure.

With such crossover operator, we hope to have a uniform combination of two given parent sets. From our experience, this modified crossover operator indeed improves over a two-point crossover operator.

4.3.4 Update of $\mathcal{S}$

After the new population is created and evaluated at each species population, individual that has a better score than its correspondent in $\mathcal{S}$ will be used to update $\mathcal{S}$. Assume a better parent set is found for node N_i, it will replace the parent set of node N_i in $\mathcal{S}$. However, due to the relaxation of the ordering constraint, such substitution may create an edge conflict such that $X \leftarrow Y$ and $X \rightarrow Y$ coexist in $\mathcal{S}$. Hence, we differentiate those edge additions that will create a conflict from those will not. For those that are not in conflict, they are incorporated directly to $\mathcal{S}$.

To determine which edge ($X \leftarrow Y$ and $X \rightarrow Y$) should be kept is equivalent to determine the proper orientation of the undirected edge X—Y. We consider the rest of the network to determine how to orient the undirected edges. Simply put, with the remaining part of the network known and fixed, we try all possible orientations of the undirected edges. Finally, the best configuration is incorporated into $\mathcal{S}$.

For the set of conflicting edges, *related edges* are first grouped together. By related edges, we refer to edges that share a common set of nodes, W. Next, different configurations of orientating the group of related edges are tried and evaluated. Since the MDL metric is node-decomposable, it suffices to evaluate the total MDL score of W. Note that while the present configuration is tried, the parent set of each node contains every other known parents. Since each conflicting edge has two orientation possibilities, there are 2^m configurations to try for a group of m edges (m is usually small). Finally, the configuration that gives the best score is used to update $\mathcal{S}$. This process is then repeated for the other groups. Since the best configuration of a group may contain cycles, the resultant $\mathcal{S}$ is repaired for cycles. Although it is possible to check for cycles when trying different orientation possibilities, we

suggest to avoid it as the involved cost will be great.

4.3.5 Update of the Best-so-far Structure

To obtain the best structure during the course of searching, we have maintained a best-so-far structure separately. In each generation, we try to *merge* the currently best-so-far structure with $\mathcal{S}$. Because $\mathcal{S}$ may contain some good partial structures, it is expected that a good solution can be obtained by accumulating the essential components of $\mathcal{S}$ into the best-so-far structure. If a better structure is created, the new structure will become the best-so-far structure.

To perform the recombination, we invent an heuristics which attempts to exchange parent sets between the two network structures. Because the score of a network structure is simply the summation of the scores of the parent sets, it enables us to perform greedy operations so that better ones (i.e. parent sets) will substitute the worse. Furthermore, our heuristics will perform cycles checking for each substitution so that the outcome will be a legal structure.

5. Performance of CCGA

5.1 Experimental Methodology

A common practice to access the performance of a Bayesian network learning algorithm is to test the algorithm on data sets generated from known network structures by probabilistic logic sampling [29]. Here, we follow the practice and test our algorithm on six different data sets. All of the data sets are generated from well-known benchmark Bayesian networks which include the ALARM (A Logical Alarm Reduction Mechanism) network and the PRINTD (Printing Diagnosis) network. Table 1 gives a summary of the data sets used in our experiments.

Five of the data sets are generated from the ALARM network obtained from different sources. Originally, the ALARM network is used in the medical domain for potential anesthesia diagnosis in the operating room [7]. Because the network, with 37 nodes and 46 directed edges, has a complex structure, it is widely used for evaluating the performance of a learning algorithm. The PRINTD network is primarily constructed for troubleshooting printer problems in the Windows$^{\text{TM}}$ operating system [27]. It has 26 nodes and 26 edges.

In our experiment, we compare the performance of our algorithm with MDLEP. All algorithms are implemented in the C++ language and are compiled using the same compiler.

Besides, the same MDL metric evaluation routine is used so that the difference among implementations are minimized. For all of the algorithms, the maximum size of a parent set is five. Since the algorithms are stochastic in nature, they are executed 40 times for each testing instance. All our experiments are conducted on the Sun Ultra-5 workstations.

We estimate the performance of the learning 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),
- the average structural difference, i.e. number of edges added, omitted, and reversed, between the final solution and the original network (ASD).

In the comparison between CCGA and MDLEP, the average MDL score of the final solutions (AFS), the average execution time (AET), and the average structural difference (ASD) are more important, because AFS and ASD estimate the quality of the solutions obtained by different algorithms, while AET indicates the efficiency of them.

Recalls that the algorithms are executed forty times for each data set, the figures are, therefore, an average of forty trials. Without any fine tuning, we adopt the following parameter values as the default setting:

- For MDLEP, we adopt the same parameter settings that appear in the original publication [53]: the population size is 50, the tournament size is seven, and the maximum number of generation is 5,000.

- For CCGA, the cutoff value for the CI test phase is 0.3. For each species population in search phase, the population size is 20. The crossover and the mutation rates are 0.7 and 0.2 respectively. The belief factor is 0.2. We use 5,000 generations as the termination criterion.

5.2 Comparing CCGA with MDLEP

We provide a summary of the results in Table 2. In the table, the MDL score of the original network is shown under the name of the data set for reference. Besides the averaged

measures, we also include the standard deviations of the respective measure which appear in parentheses.

We can observe from the AFS measure that CCGA is able to find better or equally good network comparing with MDLEP. For two out of the six cases, the difference is statistically significant at the 0.05 level using the Mann-Whitney test [6]. Using the MDL score of the original network as a reference, we observe that although MDLEP performs well (competitive with CCGA) for smaller data sets, it clearly needs longer running time to compete with our algorithm for larger data sets. For the ALARM-O data set, we regard it as a harder problem instance. Even the size of data set is relatively large, both algorithms fail to approximate the score of the original network. However, CCGA has a better performance. For the PRINTD-5000 data set, both algorithms could recover the original network structure and hence the two have identical performance.

Regarding the structural difference measure (i.e. ASD), we observe that CCGA consistently performs better than MDLEP. There are two possibilities which account for the observation: one directly relates with the CI tests, another relates indirectly. On the one hand, the CI test phase possibly helps to focus the search on dealing with only the *correct* edges (that appears in the original structure) so that the result returned will be similar to the original one. Although MDLEP may be successful in finding structures with low scores, such structures may contain some *wrong* edges. Hence, it will be the merit of the entire hybrid learning algorithm which helps to recover structures that closely resemble the original one. On the other hand, the observation could also be explained by that better results are obtained because searching is effective. In this regard, we assume that the MDL metric directly relates with the structural difference measure. Hence, networks with good scores also will resemble the original network. Because the searching is made effective as a consequence of the reduction of search space by CI tests, we can often find these good solutions that make ASD small.

Apart from the quality of the final solutions, we observe that CCGA has a speedup over MDLEP. In general, the gain is more than three-fold (varies from 3.4 to 5.0). We conjecture that there are two important reasons: (1) Due to the hybrid approach, the search space is reduced. Therefore, CCGA issues much less MDL metric evaluations (i.e. AME) than MDLEP, which is crucial as to evaluate the MDL metric is a time-consuming operation. Despite that less evaluation is made, CCGA is still effective in finding good solutions; (2) Because CCGA requires much less cycles repairing operations (only for the collaborative

structure $\mathcal{S}$ and the merged best-so-far structure) than MDLEP, it could also have saved much time.

Because CCGA executes faster while the results can still rival MDLEP, we conclude that CCGA is more efficient and effective than MDLEP.

5.3 Performance Analysis of CCGA

In this sub-section, we study the performance of CCGA under different settings that have various α values, population sizes, crossover and mutation probabilities, and belief factors. For all the experiments, we use the ALARM-O data set for testing as it is a difficult problem instance and a difference in performance could readily be observed. To focus on the performance differences caused by different settings, we use the same set of random seeds for the forty trials which guarantee the initial populations are the same. To compare the performance, we use the same measures mentioned in Section 5.1. Furthermore, we use the following basic setting: the cutoff value for the CI test phase is 0.3; the termination criterion is 1,000 generations; the population size for each species is 20; the belief factor is 0.2; the crossover and the mutation rates are 0.7 and 0.2 respectively.

5.3.1 Effect of Different α

In this experiment, our objective is to demonstrate the effect of using different values of α. In particular, we test the values of 0.02, 0.05, 0.3, 0.5, and 1.0. Since a larger α value implies that the null hypothesis is rejected more easily, it is expected that less reduction in search space will be achieved for increasing α value. In other words, the benefit from the CI test phase is diminishing. For the extreme case that α equals to 1.0, it is equivalent to omitting the CI test phase from the hybrid approach. We summarize our findings in Table 3.

These results coincide with our expectation that smaller value of α will be more effective. By using a smaller value, the running time is significantly shorter. As an example, the time used for $\alpha = 0.02$ is only one-third of the time used for $\alpha = 0.5$. By using a smaller value, we focus on a smaller set of solutions in the entire search space. Thus the search could be more efficient. The AME statistics, which measures how many different solutions are examined, is supportive of our argument.

Although we expect that a smaller value of α will be prone to erroneous CI test result, we do not observe the problem to be consequential in the experiment. It can be observed that the final solutions (AFS) are similar for the choice of α ranging between 0.05 to 0.5. The

Kruskal-Wallis test [6] suggests that the differences are not statistically significant (p-value equals to 0.43). Hence, CCGA seems to be robust when our choice of α is moderate.

For the extreme case that $\alpha = 1.0$, the performance of CCGA is poor. Without the help from CI test, CCGA fails to find good solutions with the same termination criterion. Beside, CCGA is also extremely slow, which is a result of extensive exploration of the search space. Thus, it is evident that the hybrid approach plays an important role in CCGA.

5.3.2 Effect of Different Population Sizes

Often, population size plays an important role in evolutionary computation. In general, by using a larger population size, it is more likely that a good solution could be encountered early. However, more computations will be performed for each generation. Thus, if we can strike the balance, the performance of our algorithm can be optimized.

In this experiment, we compare the performance of CCGA for different population sizes. With other parameters fixed, we increase the population size, m, in step of ten, from 20 up to 50. We put on our findings in Table 4.

The overhead of using a larger population is manifest in the findings: the average execution time (AET) increases with increasing population size. As a consequence of having more search points, the average number of MDL metric evaluations (AME) also increases. Since the initial structure is generated by assembling representatives from species populations, having larger populations results in a higher chance of getting better representatives. Hence, the initial structure is observed to have a better score for larger populations (AIS).

Although we expect that by increasing the population size, we could also accelerate the searching (in terms of number of generations), the result does not support this claim. If we compare ANG for different population sizes, the largest difference is still quite small (i.e. about 70 generations). The Kruskal-Wallis test also suggests that the differences are not statistically significant (p-value equals to 0.451). In other words, larger population size does not help to obtain the final solution earlier. Besides, the final results (AFS) are also similar. Again, the Kruskal-Wallis test suggests that the differences are not statistically significant (p-value equals to 0.490). Thus, while the advantage of using a larger population size is not apparent, we recommend the choice of a smaller population size.

5.3.3 Effect of Varying Crossover and Mutation Probabilities

Apart from the population size, we also investigate the effect of various crossover and mutation probabilities combinations. In particular, we test the performance of CCGA with the crossover probability, denoted by p_c, taking the values of 0.5, 0.7, and 0.9 while the mutation probability, p_m, is varied among 0.05, 0.2, and 0.5. Hence, we have tried a total of nine combinations. We present our findings in Table 5. Since we are using the same set of random seeds, the average initial scores (AIS) are the same and are therefore omitted from the table.

From the table, we can observe that the average number of MDL metric evaluations (AME) increases with increasing mutation probability. On the other hand, the differences obtained by changing the crossover probability are order of magnitude less than those obtained by modifying the mutation probability. As AME reflects the number of distinct solutions examined, this coincides with our general expectation that more mutations will lead to the creation of more new solutions. As a consequence of performing more MDL metric evaluations, the execution time also increases. This is supported by the result we obtained. We illustrate these observations in Figure 7.

Although the choice of the crossover probability seems to exhibit little impact on the performance, it is observed that higher crossover probability consistently decreases the number of MDL metric evaluations and hence the running time. To account for this, we notice that a higher crossover probability also implies more individuals of the species population undergo gene exchange. In such case, a super-fit individual will dominate the population quickly. Thus, more individuals are alike and the population converges in a few generations. As an example, suppose that a super-fit individual, which encodes a parent set that is sub-optimal, appears in the population. The sub-optimal parent set will spread around in the population quickly as a result of more crossover. Thus, most individuals will be similar and essentially encode the similar parent sets. As a result, there will be less distinct individuals produced in the long run in compared with the case for lower crossover probability where diversity is more likely to be preserved for a longer time.

If we look at the result (AFS and ANG) obtained in different runs, we notice that there are no significant differences among them. Furthermore, there is no observable trend that we could follow. For the seemingly superior combination of $p_c = 0.5$ and $p_m = 0.05$, the superiority seems to be vulnerable: for the three runs using $p_m = 0.05$, the Kruskal-Wallis

test suggests that the differences in the final score obtained are not statistically signficiant (p-value equals 0.394). In conclusion, the experimental result favors the choice of using a low mutation probability while the crossover probability should not be too high.

5.3.4 Effect of Varying Belief Factor

In essence, the belief factor is the minimum threshold that we believe an edge, which appears in the collaborative structure $\mathcal{S}$, is correct (See Section 4.3). In this experiment, we test CCGA using different values of belief factor, which include 0.0, 0.1, 0.2, 0.5, 0.7, and 1.0. We present the result in Table 6.

In the extreme case that the belief factor equals to 1.0, we believe that the topological ordering specified by $\mathcal{S}$ is correct. Thus, the ordering and the constraints are not changed. CCGA will therefore evolve a structure for an initial ordering which is likely to be a sub-optimal solution. This explains the poor performance of CCGA when the belief factor equals to 1.0.

As the belief factor decreases, the average number of MDL metric evaluations (AME) increases. This is reasonable as a smaller belief factor means we have more doubts about the placement of an edge. If more edges seem to be wrong, we have fewer constraints. Therefore, the search space is larger when compared to the one using a larger belief factor. Consequently, this also explains the increase in the execution time when the belief factor decreases.

Meanwhile, a larger belief factor seems to deliver poorer final solution (evidence can be observed when the belief factor varies from 0.2 to 0.5). Hence, despite that a larger belief factor would quicken up CCGA, it has the adverse effect of leading us to a sub-optimal solution. This states the trade off involved in choosing a large belief factor. Nevertheless, since there is no statistically significant degradation of the final solution obtained when the belief factor increases from 0.0 to 0.2 (p-value equals 0.246), the adopted value of 0.2 seems to be a fairly good choice.

6. Application in Direct Marketing

In this section, we investigate the feasibility of applying Bayesian networks on a real-world data mining problem. The problem relates with direct marketing in which the objective 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 problem requires ranking the customer list by the likelihood of purchase [57, 8]. Given that Bayesian networks 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 a typical scenario, we often have a huge list of customers. But among the huge list, there are usually few real buyers which amount to a few percents [13]. 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.

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. The 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 desired because it allows decision makers to select the portion of customer list to roll out to [57]. For instance, out of the 200,000 customers on the list, we might wish to send out catalogs or brochures to the most promising 20% of customers so that the advertising campaign is cost-effective [8]. 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 paragraphs.

In the recency-frequency-monetary model (RFM) [41], the profitability of a customer is estimated by three factors including the recency of buying, frequency of buying, and the amount of money one spent. Hence, only individuals that are profitable will be the targets of the campaign.

The Automatic Interaction Detection (AID) system uses tree analysis to divide consumers into different segments [41]. Later, the system was modified and became the Chi-Squared Automatic Interaction Detector (CHAID). The logistic regression model assumes that the logarithm of the *odd ratio* (*logit*) of the dependent variable (active or inactive respondents) is

a linear function of the independent variables. The odd ratio is the ratio of the probabilities of the event happening to not happening. Because the approach is popular, newly proposed models are often compared with the logistic regression model as the baseline comparison [8, 9, 56].

Zahavi and Levin [56] 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.

Ling and Li [36] combined the naïve Bayesian classifier and C4.5 to construct the response model. They evaluated their response model across three different real-life data sets, the result illustrated that their approach are effective for solving the problem.

Bhattacharyya formulated the direct marketing problem as a multi-objective optimization problem [8, 9]. He suggested that the evaluation criterion should include the performance of the model at a given depth-of-file. In an early attempt [8], he used a genetic algorithm (GA) to learn the weights of a linear response model while the fitness evaluation function was 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 indicated a superior performance. Recently, he applied genetic programming (GP) to learn a tree-structured symbolic rule form as the response model [9]. Instead of using a weighted average criterion function, the 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 [9].

6.2 Experiment

Because Bayesian networks can estimate the probability of 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 networks. Specifically, we compare the performance of the Bayesian networks evolved by CCGA with the back-propagation neural networks and the logistic regression models.

6.2.1 Experimental Methodology

The response models are evaluated on a real-life data set. The data set contains records of customers of a specialty catalog company, which mails catalogs to potential customers on a regular basis. There is a total of 106,284 customers in the data set and each entry is described by 278 attributes. The average amount of money spent by a real buyer is US$115.4.

Typical 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* selection process or a *filter* selection process [48]. 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, 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 use the forward selection criteria of logistic regression to select nine attributes, which are recency of buying, frequency of buying, the amount of money spent, use of house credit card, average order size, life time contacts, life time orders, cash payment, and telephone order. As we can see, these attributes include the three factors identified by the recency-frequency-monetary model [41].

To compare the performance of different response models, we use decile analysis which estimates the enhancement of the response rate 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 [46]:

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

Percentage of Active 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 The cumulative percentage of active respondents from decile 0 to this decile.

Actives The number of active respondents in the decile.

Percentage of Total Actives This calculates the ratio of the number of active respondents in the decile to the number of all active respondents.

Cumulative Actives 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 Lift 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 The cumulative lift 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.

Without any fine tuning of the parameter values of CCGA, the cutoff value for the conditional independence (CI) test phase is 0.3, the termination criterion is 1,000 generations, the population size for each species is 20, the belief factor is 0.2, and the crossover and the mutation rates are 0.7 and 0.2 respectively.

A visual neural data mining system called Tiberius [12] is used to learn back-propagation neural networks. The default parameter values of the system are adopted and the learning process terminates after 10,000 epoches have been performed. Different neural network structures are attempted and it is found that the one with 1 hidden layer and 5 hidden neurons has the best performance. Thus, we only report the performance of that neural network structure in the following sub-section.

6.2.2 Cross-Validation Result

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 7, 8 and 9, the experimental results for the back-propagation neural networks, the logistic regression models and the Bayesian networks obtained by CCGA are shown respectively. We tabulate the statistics at each decile averaged over the ten runs. Table 7 indicates the first two deciles have cumulative lifts of 336.33 and 232.08 respectively, suggesting that by mailing to the top two deciles alone, the back-propagation neural networks generate over twice as many respondents as a random mailing without a model. From Table 8, the cumulative lifts in the first two deciles for the logistic regression models are respectively 342.27 and 249.20.

Table 9 shows that the Bayesian networks have cumulative lifts of 358.20 and 274.20 in the first two deciles respectively, which are significantly higher than those of the back-propagation neural networks (p-values are 0.00439 and 0.0000311 respectively) and the logistic regression models (p-values are 0.013 and 0.000023 respectively). Overall, the Bayesian networks perform significantly better than the back-propagation neural networks and the logistic regression models in predicting consumer response to direct mail promotions.

In Figure 8, the models are compared in a gains chart that depicts the performance of the response models by plotting the cumulative percentage of total actives in the ten deciles. A diagonal line on the chart shows the performance of a random approach. It can be observed from the chart that the models outperform the random approach by discerning a larger percentage of active respondents for different depth-of-file. When comparing the Bayesian networks and the logistic regression models, it can be observed that the former perform better than the latter in the first three deciles. At later deciles, however, the models deliver similar performance. On the other hand, the Bayesian networks outperform the back-propagation neural networks in nearly all deciles.

Since 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 and increase profit. For example, if the Bayesian networks are applied to select 20% of customers from the data set used in this experiment, about US$364,100.7 can be earned. On the other hand, about US$307,321.7, US$330,864.2, and US$132,484.2 can be earned respectively if the back-propagation neural networks, the logistic regression models, and random models are used. In other words, additional US$56,779.0, US$33,236.5, and US$231,616.5 can be obtained respectively by using the Bayesain networks, when compare with the back-propagation neural networks, the logistic regression models, and random models. Consequently, it seems that the Bayesian networks are more desirable than the other

models.

7. Conclusion

In this paper, we describe a new algorithm, CCGA, for learning Bayesian networks effectively and efficiently. CCGA combines the characteristics of the dependency analysis and the search-and-scoring approaches. It consists of the conditional independence (CI) test and the search phases. Dependency analysis is conducted in the first phase to reduce the size of the search space, while a cooperative coevolution genetic algorithm is used in the second phase to generate good Bayesian networks.

The algorithm has been tested on a number of Bayesian networks learning problems to show that it can discover good Bayesian networks. We have compared CCGA with MDLEP and found that CCGA is much more effective and efficient. With an effective and efficient algorithm, it enables us to explore interesting applications of Bayesian networks on real-world data mining problems. We have also performed a parameter study to investigate the performance of CCGA under different parameter settings.

We have employed CCGA to a real-world data set of direct marketing and compared the Bayesian networks obtained by CCGA with the back-propagation neural networks and the logistic regression models. For this data set, the Bayesian networks outperform the other models. This study shows that CCGA can potentially become a powerful and efficient data mining tool.

Acknowledgment

This research was supported by the Earmarked Grant LU 3012/01E from the Research Grant Council of the Hong Kong Special Administrative Region.

References

[1] Agresti, A. *Categorical Data Analysis*. Wiley, New York, circa 1990.

[2] Bäck, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Agorithms*. Oxford University Press, 1996.

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

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

[5] Beasley, D., Bull, D. R., and Martin, R. R. An overview of genetic algorithms: Part 1, fundamentals. *University Computing 15*, 2 (1993), 58–69. ftp://ralph.cs.cf.ac.uk/pub/papers/GAs/ga_overview1.ps.

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

[7] Beinlinch, I., Suermondt, H., Chavez, R., and Cooper, G. 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* (1989), pp. 247–256.

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

[9] Bhattacharyya, S. 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* (August 2000), pp. 465–473.

[10] Bayesian network tools in java (bnj). http://bndev.sourceforge.net.

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

[12] Brierley, P. Tiberius: Viaual neural data mining. http://www.philbrierley.com.

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

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

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

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

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

[18] de Campos, L. M., and Huete, J. F. 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.

[19] Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery in Data Mining*. AAAI Press, Menlo Park, CA, 1996, pp. 1–34.

[20] Fogel, D. B. *Evolutionary Computation: Toward a New Philosohpy*. IEEE Press, 2000.

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

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

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

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

[25] Heckerman, D. Bayesian networks for data mining. *Data Mining and Knowledge Discovery 1* (1997), 79–119.

[26] Heckerman, D., and Horvitz, E. Inferring informational goals from free-text queries: A Bayesian approach. In *Proceedings of Fourteenth Conference of Uncertainty*

in Artificial Intelligence (Wisconsin, USA, July 1998), G. F. Cooper and S. Moral, Eds., Morgan Kaufmann, pp. 230–237.

[27] Heckerman, D., and Wellman, M. P. Bayesian networks. *Communications of the ACM 38*, 3 (March 1995), 27–30.

[28] Heckerman, D. E., Horvitz, E. J., and Nathwani, B. N. Toward normative expert sysems: Part i. the pathfinder project. *Methods of Information in Medicine 31*, 2 (1992), 90–105.

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

[30] Herskovits, E., and Cooper, G. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning 9*, 4 (1992), 309–347.

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

[32] Hugin. http://www.hugin.com.

[33] Koza, J. R. *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[34] Lam, W., and Bacchus, F. Learning Bayesian belief networks-an approach based on the MDL principle. *Computational Intelligence 10*, 4 (1994), 269–293.

[35] Larrañaga, P., Poza, M., Yurramendi, Y., Murga, R., and Kuijpers, C. Structural learning of Bayesian network by genetic algorithms: A performance analysis of control parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 9 (September 1996), 912–926.

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

[37] Msbnx: Microsoft belief network tools. http://www.research.microsoft.com/adapt/MSBNx.

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

[39] Olesen, K. G., Kjærulff, U., Jensen, F., Jense, F. V., Falck, B., Andreassen, S., and Andersen, S. K. A MUNIN network for the median nerve - a case study in loops. *Applied Artificial Intelligence 3* (1989), 385–404.

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

[41] Petrison, L. A., Blattberg, R. C., and Wang, P. Database marketing: Past present, and future. *Journal of Direct Marketing 11*, 4 (1997), 109–125.

[42] Potter, M. A. *The Design and Analysis of a Computational Model of Cooperative Coevolution.* PhD thesis, George Mason University, 1997.

[43] Potter, M. A., and DeJong, K. A. Evolving neural networks with collaborative species. In *Proceedings of the 1995 Summer Computer Simulation Conference* (Ottawa, Canada, 1995).

[44] Potter, M. A., DeJong, K. A., and Grefenstette, J. A coevolutionary approach to learning sequential decision rules. In *Proceedings of the Sixth International Conference (ICGA'95)* (July 1995), pp. 366–372.

[45] Rissanen, J. Modelling by shortest data description. *Automatica 14* (1978), 465–471.

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

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

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

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

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

[51] Suzuki, J. Learning Bayesian belief networks based on the minimum description length principle: Basic properties. In *IEICE Transactions Fundamentals* (November 1999), vol. E82-A.

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

[53] Wong, M. L., Lam, W., and Leung, K. S. Using evolutionary programming and minimum description length principle for data mining of Bayesian networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence 21*, 2 (February 1999), 174–178.

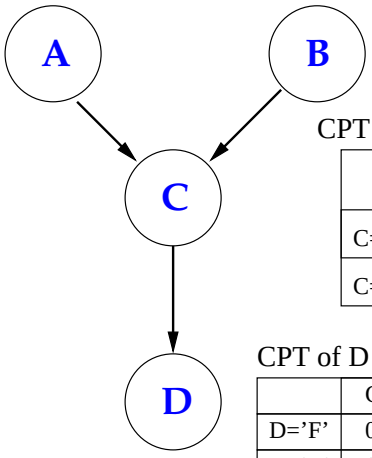
[54] Wong, M. L., Lam, W., Leung, K. S., Ngan, P. S., and Cheng, J. C. Y. Discovering knowledge from medical databases using evolutionary algorithms. *IEEE Engineering in Medicine and Biology Magazine 19*, 4 (2000), 45–55.

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

[56] Zahavi, J., and Levin, N. Applying neural computing to target marketing. *Journal of Direct Marketing 11*, 4 (1997), 76–93.

[57] Zahavi, J., and Levin, N. Issues and problems in applying neural computing to target marketing. *Journal of Direct Marketing 11*, 4 (1997), 63–75.

Figures



CPT of C

	A=’F’, B=’F’	A=’F’, B=’T’	A=’T’, B=’F’	A=’T’, B=’T’
C=’F’	0.65	0.54	0.78	0.01
C=’T’	0.35	0.46	0.22	0.99

CPT of D

	C=’F’	C=’T’
D=’F’	0.23	0.6
D=’T’	0.77	0.4

Figure 1: A Bayesian network example.

1. Set t, the generation count, to 0.
2. Create an initial population, Pop(t) of m random directed acyclic graphs (m is the population size).
3. Each directed acyclic graph in the population is evaluated using the MDL metric.
4. While t is less than the maximum number of generations,
   - Each directed acyclic graph in Pop(t) produces one offspring by performing a number of mutation operations. If there are cycles, a randomly picked edge in each cycle is removed.
   - The directed acyclic graphs 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 directed acyclic graphs 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 2: The MDLEP algorithm.

- For each node, initialize the possible parent set to contain every other nodes.
- CI Test Phase
  - Perform CI test (up to order-1) between all pairs of nodes.
  - If the nodes are found to be conditionally independent, remove each other from their possible parent sets.
- Cooperative Coevolution Search Phase
  1. Set t, the generation count, to 0.
  2. For each species population,
     - Randomly initialize each chromosome in accordance with the possible parent set.
     - Evaluate the fitness of each chromosome by using the MDL metric.
  3. Compose $\mathcal{S}$ by combining the best chromosome from each species population.
  4. Pass the constraints from $\mathcal{S}$ to each species population.
  5. While t is less than the maximum number of generations,
     - Inside each species population,
       * Temporarily change the possible parent set with the given constraints.
       * Perform selection.
       * Evolve new offspring using crossover and mutation
       * Evaluate the fitness of each chromosome.
     - Update $\mathcal{S}$.
     - Produce a node ordering from $\mathcal{S}$ and pass the constraints to each species population.
     - Update the best-so-far structure.

Figure 3: The CCGA algorithm.

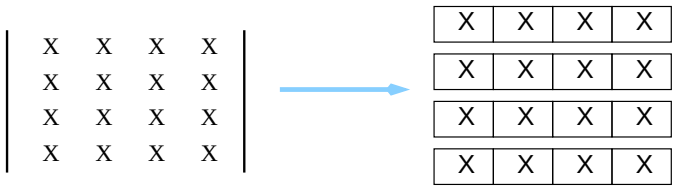


Figure 4: Decomposition of the matrix representation into rows.

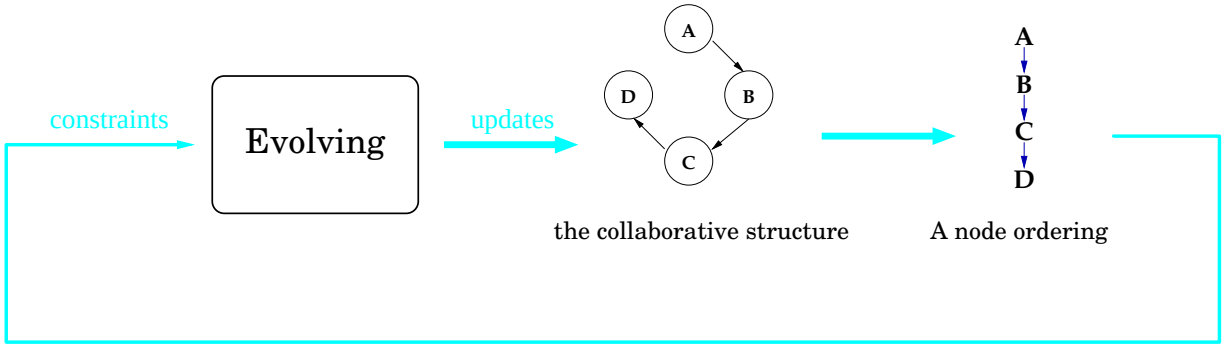


Figure 5: The feedback mechanism.

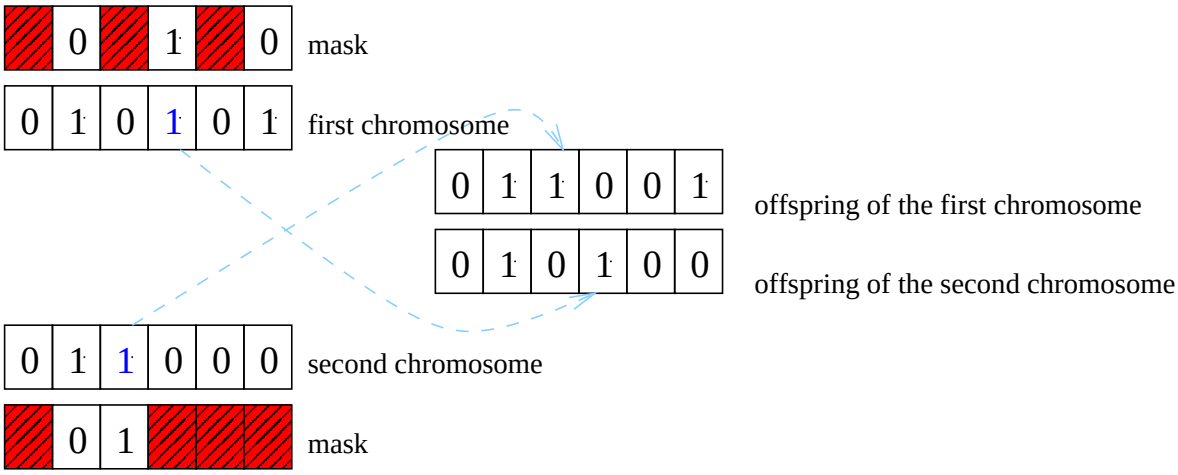


Figure 6: The modified crossover operator.

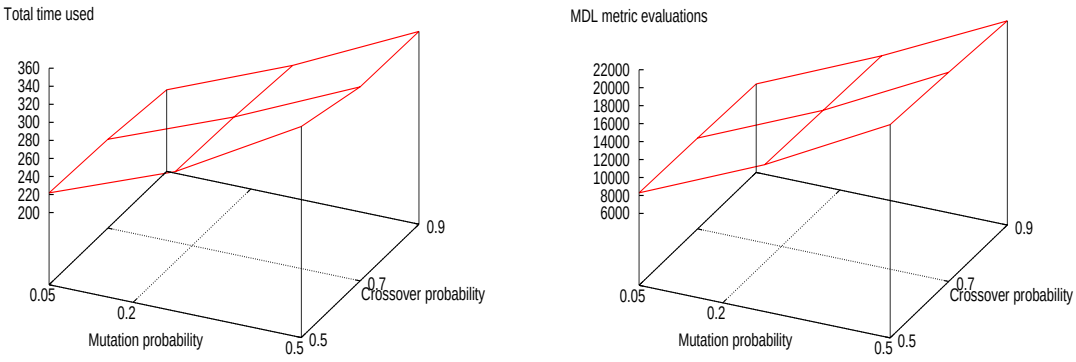


Figure 7: Effects on time used and number of evaluations.

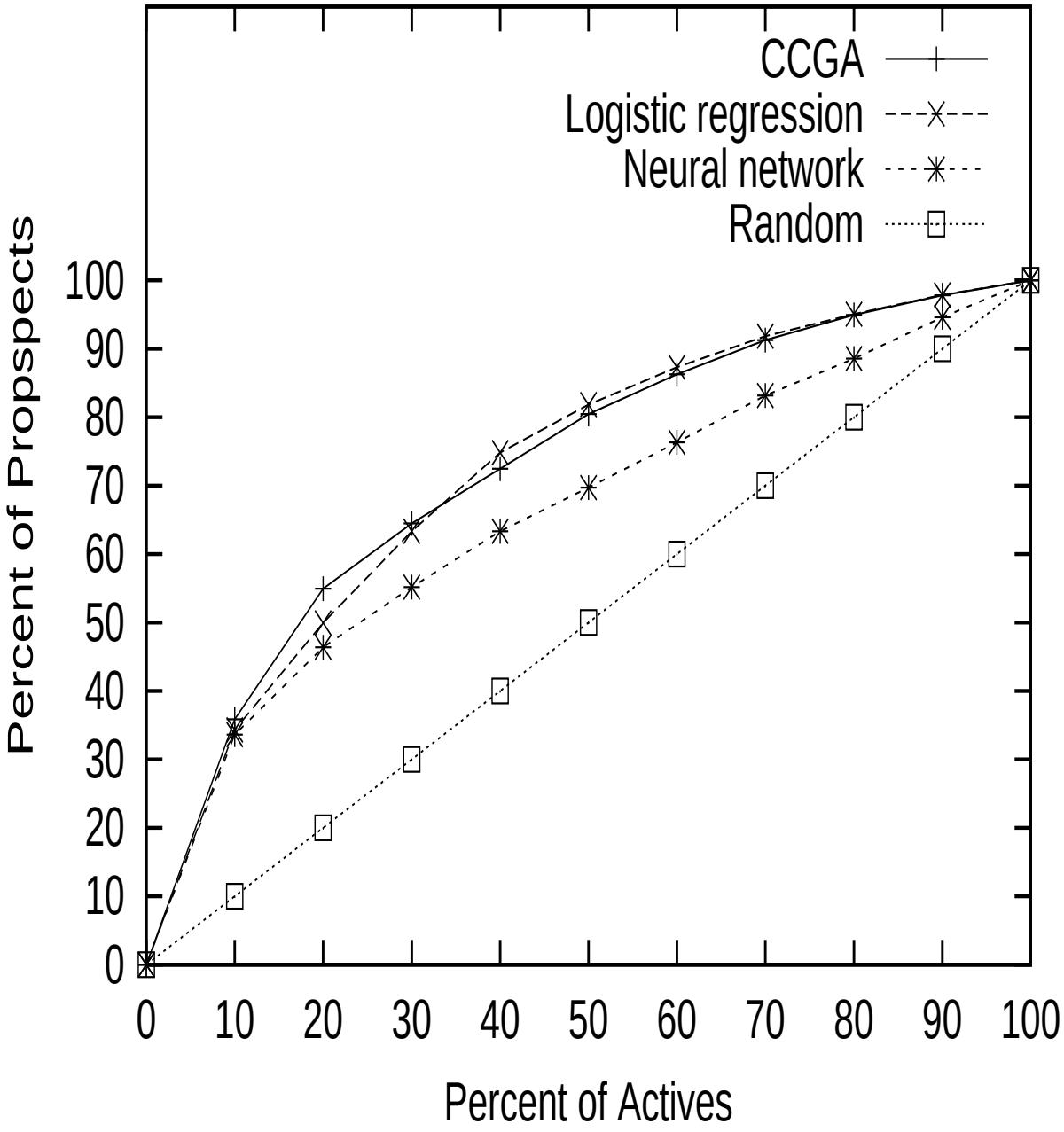


Figure 8: Model comparison gains chart.

Tables

Data set	Original Network	Size	MDL Score of Original Network
ALARM-1000	ALARM	1,000	18,533.5
ALARM-2000	ALARM	2,000	34,287.9
ALARM-5000	ALARM	5,000	81,223.4
ALARM-10000	ALARM	10,000	15,8497.0
ALARM-O	ALARM	10,000	138,455.0
PRINTD-5000	PRINTD	5,000	106,541.6

Table 1: Data sets used in the experiments.

Data Set		AFS	AIS	AET	ANG	AME	ASD
ALARM-1000 (18533.5)	CCGA	17,866.1 (37.3)	23,827.8 (1,059.6)	199.8 (4.6)	1,181.1 (1,031.6)	10,461.4 (115.1)	12.8 (2.5)
	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)
ALARM-2000 (34287.9)	CCGA	33,777.6 (21.6)	45,757.6 (1,405.3)	258.6 (5.2)	1,094.8 (1,005.9)	16,711.5 (159.7)	7.7 (0.9)
	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)
ALARM-5000 (81233.4)	CCGA	81,004.0 (0.0)	112,379.5 (4,832.3)	362.9 (8.3)	583.4 (387.8)	17,804.7 (124.2)	6.2 (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)
ALARM-10000 (158497.0)	CCGA	158,420.6 (3.4)	223,957.7 (8,120.7)	547.3 (20.3)	1,402.3 (1,074.7)	20,453.9 (138.6)	3.0 (0.7)
	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)
ALARM-O (138455.0)	CCGA	138,760.1 (562.3)	217,291.5 (9,336.4)	827.8 (59.4)	1,012.7 (960.9)	28,828.6 (215.2)	11.6 (3.6)
	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)
PRINTD-5000 (106541.6)	CCGA	106,541.6 (0.0)	114,764.0 (1,066.9)	207.4 (5.3)	10.1 (3.8)	3,721.1 (48.8)	0.0 (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)

Table 2: Performance comparison between CCGA and MDLEP

	AFS	AIS	AET	ANG	AME	ASD
$\alpha = 0.02$	138,712.4 (356.7)	203,667.3 (11,446.4)	173.0 (23.0)	448.9 (216.1)	4,096.4 (62.5)	11.0 (4.3)
$\alpha = 0.05$	138,866.5 (528.2)	201,723.3 (11,651.8)	191.6 (23.9)	512.4 (272.9)	5,189.3 (85.3)	12.4 (5.1)
$\alpha = 0.3$	138,854.3 (564.1)	217,386.1 (12,898.4)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
$\alpha = 0.5$	138,858.4 (642.8)	223,194.1 (13,842.5)	497.7 (37.5)	523.8 (293.2)	37,160.7 (878.5)	12.7 (4.3)
$\alpha = 1.0$	141,456.6 (893.4)	215,187.5 (11,235.0)	6,664.5 (68.7)	904.7 (108.8)	814,931.6 (3,397.9)	39.1 (4.8)

Table 3: Performance of CCGA with different α.

	AFS	AIS	AET	ANG	AME	ASD
$m = 20$	138,854.3 (564.1)	217,386.1 (12,898.4)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
$m = 30$	138,631.1 (162.3)	214,091.9 (10,703.8)	294.3 (31.6)	466.5 (260.1)	16,034.3 (325.6)	10.3 (4.3)
$m = 40$	138,710.8 (452.6)	213,138.4 (12,088.9)	310.7 (28.4)	396.3 (218.8)	17,887.4 (310.2)	11.6 (4.0)
$m = 50$	138,676.0 (427.4)	212,518.9 (10,641.6)	331.1 (28.3)	406.5 (268.5)	19,682.3 (373.9)	10.5 (4.2)

Table 4: Performance of CCGA under different population sizes.

	AFS	AET	ANG	AME	ASD
$p_c = 0.5$	138,716.8 (453.1)	221.8 (23.6)	419.5 (250.0)	8,288.0 (163.6)	10.6 (4.3)
$p_c = 0.7$	138,925.9 (737.7)	218.4 (27.5)	471.0 (238.8)	8,106.2 (152.7)	11.7 (4.5)
$p_c = 0.9$	138,815.9 (567.8)	210.4 (24.5)	559.4 (253.3)	7,857.7 (140.9)	11.5 (4.5)

(a) $p_m = 0.05$

	AFS	AET	ANG	AME	ASD
$p_c = 0.5$	138,952.9 (662.1)	270.5 (19.0)	547.0 (277.3)	14,194.0 (174.4)	12.9 (4.4)
$p_c = 0.7$	138,854.3 (564.1)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
$p_c = 0.9$	139,059.7 (796.3)	253.3 (20.0)	485.3 (247.2)	13,144.2 (213.5)	13.3 (3.2)

(b) $p_m = 0.2$

	AFS	AET	ANG	AME	ASD
$p_c = 0.5$	139,136.3 (647.7)	353.9 (31.7)	563.1 (231.3)	21,737.3 (455.9)	14.5 (4.0)
$p_c = 0.7$	138,978.3 (560.1)	335.1 (28.4)	496.3 (264.4)	21,292.4 (396.1)	13.1 (4.5)
$p_c = 0.9$	138,953.1 (527.8)	333.8 (30.3)	557.2 (244.3)	20,729.6 (371.8)	13.1 (4.1)

(c) $p_m = 0.5$

Table 5: Performance of CCGA with different p_c and p_m.

	AFS	AET	ANG	AME	ASD
factor = 0.0	138,761.9 (461.7)	559.5 (106.8)	498.1 (250.0)	12,519.5 (255.2)	11.6 (4.0)
factor = 0.1	138,761.1 (467.7)	352.8 (52.0)	497.2 (291.4)	13,688.9 (212.9)	11.9 (3.9)
factor = 0.2	138,854.3 (564.1)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
factor = 0.5	139,286.0 (731.9)	231.0 (8.5)	590.0 (265.2)	12,004.1 (255.2)	15.5 (3.7)
factor = 0.7	140,052.6 (826.2)	233.3 (11.9)	626.3 (281.1)	10,160.9 (293.8)	19.4 (3.8)
factor = 1.0	164,003.6 (5,972.5)	107.7 (14.9)	12.7 (9.8)	1,799.4 (705.2)	41.5 (4.3)

Table 6: Performance comparison of different belief factor.

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	81.81%	18.16%	18.16%	192.90	33.61%	192.90	33.61%	336.33	336.33
1	1063	41.62%	6.91%	12.54%	73.40	12.77%	266.30	46.38%	127.83	232.08
2	1063	24.58%	4.79%	9.96%	50.90	8.78%	317.20	55.16%	87.91	184.02
3	1063	18.14%	4.42%	8.57%	46.90	8.17%	364.10	63.33%	81.72	158.45
4	1063	16.20%	3.44%	7.54%	36.50	6.39%	400.60	69.73%	64.01	139.56
5	1062	16.19%	3.55%	6.88%	37.70	6.61%	438.30	76.34%	66.16	127.33
6	1063	13.99%	3.69%	6.42%	39.20	6.83%	477.50	83.17%	68.41	118.91
7	1063	13.93%	2.91%	5.98%	30.90	5.37%	508.40	88.54%	53.73	110.76
8	1063	8.51%	3.25%	5.68%	34.50	6.03%	542.90	94.57%	60.33	105.16
9	1062	8.01%	2.91%	5.40%	31.10	5.43%	574.00	100.00%	53.93	100.00
Total	10,628				574.00					

Table 7: Gains table of the result from back-propagation neural networks.

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%	18.53%	18.53%	197.00	34.32%	197.00	34.32%	342.70	342.70
1	1063	15.59%	8.44%	13.49%	89.70	15.60%	286.70	49.93%	155.40	249.20
2	1063	12.67%	7.25%	11.41%	77.00	13.41%	363.7	63.33%	133.70	210.40
3	1063	10.45%	6.20%	10.10%	65.90	11.50%	429.60	74.83%	114.40	186.70
4	1063	8.50%	3.76%	8.84%	40.00	6.99%	469.60	81.82%	69.40	163.10
5	1062	6.63%	2.97%	7.86%	31.60	5.50%	501.20	87.32%	54.60	144.90
6	1063	4.89%	2.43%	7.08%	25.80	4.50%	527.00	91.82%	44.50	130.70
7	1063	3.45%	1.74%	6.41%	18.50	3.23%	545.50	95.05%	31.90	118.40
8	1063	2.36%	1.53%	5.87%	16.30	2.83%	561.80	97.88%	27.70	108.20
9	1062	1.38%	1.15%	5.40%	12.20	2.12%	574.00	100.00%	20.60	100.00
Total	10,628				574.00					

Table 8: Gains table of the result from logistic regression models.

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	28.38%	19.37%	19.37%	205.90	35.87%	205.90	35.87%	358.20	358.20
1	1063	10.41%	10.31%	14.84%	109.60	19.09%	315.50	54.97%	190.30	274.20
2	1063	6.30%	5.17%	11.62%	54.90	9.54%	370.40	64.50%	94.80	214.60
3	1063	3.87%	4.29%	9.78%	45.60	7.97%	416.00	72.48%	79.30	180.70
4	1063	2.61%	4.30%	8.69%	45.70	7.98%	461.70	80.46%	79.50	160.40
5	1062	1.82%	3.15%	7.76%	33.50	5.83%	495.20	86.28%	57.70	143.40
6	1063	1.23%	2.69%	7.04%	28.60	4.97%	523.80	91.26%	49.30	129.80
7	1063	0.78%	1.99%	6.41%	21.10	3.67%	544.90	94.92%	36.20	118.20
8	1063	0.43%	1.55%	5.87%	16.40	2.86%	561.30	97.79%	28.20	108.10
9	1062	0.16%	1.20%	5.40%	12.70	2.21%	574.00	100.00%	21.60	100.00
Total	10,628				574.00					

Table 9: Gains table of the result from Bayesian networks.