2007

Dynamic inference-based learning of Markov network structure

Parichey Gandhi

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation


https://lib.dr.iastate.edu/rtd/15030

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Dynamic inference-based learning of Markov network structure

by

Parichey Gandhi

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Dimitris Margaritis, Major Professor
Vasant Honavar
Arka P. Ghosh

Iowa State University
Ames, Iowa
2007

Copyright © Parichey Gandhi, 2007. All rights reserved.
DEDICATION

I would like to dedicate this thesis to my parents Mrs. Sneh Lata Gandhi and Mr. Ashok Kumar Gandhi without whose support I would not have been able to complete this work. I would also like to thank my friends for their loving guidance during the writing of this work.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>viii</td>
</tr>
<tr>
<td>CHAPTER 1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER 2. Notation and Preliminaries</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Notations</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Markov Blanket</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Independence Oracle Implementation</td>
<td>7</td>
</tr>
<tr>
<td>2.4 The Triangle Theorem</td>
<td>8</td>
</tr>
<tr>
<td>2.5 Fringe</td>
<td>9</td>
</tr>
<tr>
<td>2.6 Assumptions</td>
<td>11</td>
</tr>
<tr>
<td>CHAPTER 3. Review of Related Work</td>
<td>13</td>
</tr>
<tr>
<td>3.1 The Grow-Shrink Algorithm</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Grow Shrink Markov Network</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Grow Shrink Inference Markov Network</td>
<td>18</td>
</tr>
<tr>
<td>CHAPTER 4. The DGSIMN Algorithm</td>
<td>20</td>
</tr>
<tr>
<td>4.1 Multiple Columns</td>
<td>20</td>
</tr>
<tr>
<td>4.2 DGSIMN</td>
<td>23</td>
</tr>
<tr>
<td>4.3 Utility Function</td>
<td>26</td>
</tr>
<tr>
<td>4.4 Experimental Results</td>
<td>27</td>
</tr>
<tr>
<td>4.4.1 Exact Learning</td>
<td>28</td>
</tr>
</tbody>
</table>
4.4.2 Sampled Data ......................................................... 30
4.4.3 Real-world Data ..................................................... 31

CHAPTER 5. Conclusion and Future Work ......................... 37

BIBLIOGRAPHY .......................................................... 38

ACKNOWLEDGEMENTS .................................................. 41
LIST OF TABLES

Table 2.1 Markov blanket corresponding to variables in Fig. 1.1. . . . . . . . . . 6

Table 4.1 Weighted number of tests and accuracy for several real-world data sets. For each evaluation measure, the best performance between DGSIMN and GSIMN is indicated in bold. The number of variables in the domain is denoted by $n$ and the number of data points in each data set by $N$. 32
LIST OF FIGURES

Figure 1.1 An example Markov network with nodes representing the variables in the domain $V = \{1, 2, 3, 4, 5\}$. ....................................................... 2

Figure 2.1 Graphical representation of Triangle rule. Edges are labeled with sets and represent conditional dependence or independence. Solid edges in the graph represent conditional dependence, and dotted edge represents independence. ................................................................. 9

Figure 2.2 State of Markov learning procedure for Fig. 1.1 at a certain point. White boxes represent variable is in dependence column of that variable and black boxes represent that the variable is in independence column. 10

Figure 3.1 Graphical representation of the grow phase state $(D_1, I_1)$ of variable 1. The dependence column $D_1$ is shown in white and the independence column $I_1$ in black. ................................................................. 15

Figure 3.2 State of the grow phase for variable 1 after the test $(1 \not\in \{2, 5\})$ is evaluated. ................................................................. 16

Figure 3.3 State for variable 1 at the end of GS algorithm ........................................ 17

Figure 4.1 Graphical representation of the multi-column grow phase state $(D_1, I_1)$ of variable 1. ................................................................. 21

Figure 4.2 Ratio of the weighted cost DGSIMN over GSIMN for different network sizes (number of nodes) $n$ and average degrees $\tau = 1, 2, 4,$ and $8$. Ratios smaller than 1 indicate an advantage of DGSIMN over GSIMN. ........ 29
Figure 4.5  Ratio of the weighted cost of DGSIMN over GSIMN and difference between the accuracy of DGSIMN and GSIMN on real data sets. Ratios smaller that 1 and positive bars indicate an advantage of DGSIMN over GSIMN. The numbers in the $x$-axis are indices of the data sets as shown in Table 4.1. 

Figure 4.3  Ratio of weighted cost of DGSIMN over GSIMN (continuous line) and the difference of the ratio of correct edges of DGSIMN and GSIMN (bar graph) for different number of variables $n$ and average degrees $\tau = 1$, and 2. Ratios smaller that 1 and positive bars indicate an advantage of DGSIMN over GSIMN. 

Figure 4.4  Ratio of weighted cost of DGSIMN over GSIMN (continuous line) and the difference of the ratio of correct edges of DGSIMN and GSIMN (bar graph) for different number of variables $n$ and average degrees $\tau = 4$, and 8. Ratios smaller that 1 and positive bars indicate an advantage of DGSIMN over GSIMN.
ABSTRACT

In this thesis we address the problem of learning Markov network structure from data by presenting the Dynamic GSIMN or DGSIMN algorithm. DGSIMN is an extension of GSIMN algorithm, and works by conducting a series of statistical conditional independence tests on the data, and uses the axioms that govern the independence relation to avoid unnecessary tests i.e., tests that can be inferred from the results of known ones. However, DGSIMN improves on the GSIMN algorithm by dynamically selecting the locally optimal test that will increase the state of knowledge about the structure the most. This is done by estimating the number of inferences that will be obtained after executing a test (before it is actually evaluated on data), and selecting the one that is expected to maximize the number of such inferences. This helps decreasing the number of tests required to be evaluated on data, resulting in an overall decrease in the computational requirements of the algorithm. Experiments show that DGSIMN yields savings of up to 85% while achieving similar or better accuracy in most cases.
CHAPTER 1. Introduction

Over the last few decades, with the increase in computer processing power and disk storage and the decrease in the cost of gathering data, it has become significantly easier and cheaper to generate enormous amounts of data. To extract meaningful and useful information from this data, a number of data mining algorithms are being developed. Frequently, in these algorithms, the task of using existing data to help in predicting future outcomes can be made significantly easier by estimating the joint probability distribution of the domain. The probability distribution of a discrete domain can be represented in a number of ways. The simplest of these is to explicitly represent it as a table containing one entry for the probability of each possible joint value combination of the set of variables of the domain. Unfortunately, the size of this table grows exponentially with the number of random variables (e.g., it would require of the order of $2^{100}$ entries for a domain with one hundred binary variables). Frequently however this is not necessary due to numerous independences that may exist in the domain. Therefore, a better alternative is to use a graphical model (a Bayesian or Markov network) that uses domain independences to succinctly store the joint probability distribution. Additionally, graphical models have the advantage of clear semantics and a sound and widely accepted theoretical foundation (probability theory).

Graphical models have been applied to various research problems in the recent years such as analysis of gene expression pathways [12], computer vision [13, 6] and more recently [11, 3], among others. Except in rare situations however, the structure of the network of a domain is unknown and has to be learned from data drawn from it. A solution to the problem of learning graphical models from data [14, 15, 8], besides being theoretically interesting in itself, also holds the potential of advancing the state-of-the-art in application domains where such
models are used.

An example Markov network with nodes representing the variables in the domain $V = \{1, 2, 3, 4, 5\}$.

In this thesis we focus on the problem of learning Markov networks from data. Markov networks are graphical models that consist of an undirected graph whose nodes represent the random variables of the domain (the model structure), and a set of numeric parameters. An example Markov network is shown in Fig. 1.1. Together, the graph and the parameters can represent the joint probability distribution of the random variables of the domain. The structure graphically encodes conditional probability independences that exist in the domain. In particular, an edge is missing between two nodes if and only if the corresponding random variables are conditionally independent given a set containing some of the remaining variables. Conditional independence among variables in a Markov network can be shown to be equivalent to vertex separation among the corresponding nodes in the graph. For example in Fig. 1.1, variable 1 is conditionally independent of 5 given variables 3 and 4. To learn a Markov network, the process consists of first learning the structure, and, given the structure, subsequently learning the parameters. Here we concentrate on the problem of learning the structure of a Markov network.

Markov networks have numerous applications. In the past they have been used in the
physics and computer vision communities [13, 6, 3] where they have been historically called Markov random fields. In most of these applications the structure of the network has been provided by an expert (as opposed to learned from the data), usually derived by connecting each node with its physically nearest neighbors (as measured by either Euclidean or some other abstract distance). Recently, Markov networks have also been applied to spatial statistics, with potential applications in transportation, environmental sciences, meteorology, agronomy and others [23].

A number of algorithms for learning Markov network structure have appeared in the literature. Two common categories are score-based [9, 19] and independence-based or constraint-based [24] algorithms. Score-based algorithms perform a search over the space of all undirected graphs in an attempt to find the graph with maximum score. As the space of all graphs has size \(2^{n(n-1)/2}\) for a domain that contains \(n\) variables, these algorithms frequently resort to heuristic search. Scores that have been used include maximum likelihood, minimum description length [17], and pseudo-likelihood [5]. However, evaluation of these scores usually requires the computation of the parameters for each candidate structure, a task that has been proved to be NP-hard for undirected models [4]. Therefore, score-based approaches are theoretically intractable regardless of the quality of the search heuristics in use. Approaches that attempt to overcome this intractability include [16] and [1]. The latter in particular introduces a new class of efficient algorithms for structure and parameter learning of factor graphs that subsume Markov and Bayesian networks. It is a promising and theoretically sound approach that may lead in the future to practical efficient algorithms for Markov networks structure learning.

The second category of algorithms are independence-based. A major advantage of members of this class is that they do not require computation of the parameters of the model during the structure discovery process, and thus are efficient. The main idea behind these algorithms is to exploit the independence semantics of the graphs, i.e., the fact that the structure implies that a set of independences exist in the distribution of the domain, and therefore in the data set provided as input to the algorithm (under assumptions, see below). They work by conducting a set of conditional independence tests on data, successively restricting the number of possible
structures consistent with the results of those tests to a singleton (if possible), and inferring
that structure as the only possible one.

Our contribution here is an independence-based algorithm, called **DGSIMN** (Dynamic
Grow-Shrink Inference-based Markov Network structure learning algorithm). To learn the
structure, DGSIMN learns the set of direct neighbors of each variable in the domain (also called
the *Markov blanket* of the variable), which collectively uniquely determine the structure of the
undirected graph. The Markov blankets are learned using the Grow-Shrink (GS) algorithm by
Margaritis and Thrun [18], an independence-based algorithm originally developed for learning
the structure of Bayesian networks. The DGSIMN algorithm is an improvement over the
GSIMN algorithm of Bromberg et al [7]. Both GSIMN and DGSIMN use Pearl’s theorems on
the properties of conditional independence relation [20] to infer additional dependencies and
independences from the set of already known ones resulting from statistical tests and previous
inferences, thus avoiding the execution of these tests on data and therefore speeding up the
structure learning process. One shortcoming of the GSIMN algorithm however is the fixed
ordering in which it performs its tests, that introduces potential limitations in the number
of tests it can infer. DGSIMN instead greedily and dynamically selects, at each iteration
in the algorithm, the test that maximizes the expected number of inferences resulting from
performing the test on data. This provides it with superior performance in most cases, as our
experimental evaluation confirms.
CHAPTER 2. Notation and Preliminaries

In this chapter we formally define the notations used throughout this thesis. Then we introduce related information and concepts required for understanding of the work done as part of this research.

2.1 Notations

In this thesis we use capital letters to denote random variables (e.g., \(X, Y, Z\)). Bold capital letters are used to represent a set of random variables (e.g., \(\mathbf{X}, \mathbf{Y}, \mathbf{Z}\)). Small letters are used to represent values of these variables (e.g., \(x, y, z\)). In particular, we denote by \(V = \{1, \ldots, n\}\) the set of all \(n\) variables in the domain. We assume that the joint probability distribution over \(V\) is positive i.e., every value combination of the variables in \(V\) has some non-zero probability of occurring.

We use the notation \((X \perp Y \mid Z)\) or \((X, Y, Z) = \text{true}\) to denote the fact that \(X\) is independent of \(Y\) conditioned on \(Z\). Similarly we use the notation \((X \not\perp Y \mid Z)\) or \((X, Y, Z) = \text{false}\) to denote that \(X\) is dependent on \(Y\) conditional on \(Z\). Slightly abusing notation, we use the shortcut \(X\) instead of \(\{X\}\) e.g., \((X \perp Y \mid Z)\) instead of \((\{X\} \perp \{Y\} \mid Z)\). We denote the data set as \(D\), and its size (number of data points) by \(N\). We use \(M_X\) to denote the Markov blanket of variable \(X\).

2.2 Markov Blanket

The Markov blanket \(M_X\) of variable \(X \in V\) is any set of variables \(S \subseteq V\) such that

\[(X \perp V - S - \{X\} \mid S).\]
That is, all variables in the domain $\mathbf{V}$, excluding variables in $\mathbf{M}_X$ and $X$, are independent of $X$ conditional to Markov blanket $\mathbf{M}_X$. The set of all Markov blankets can be used to construct the Markov network by making use of the following theorem.

**Theorem 1 (Pearl and Paz, 1985)** *The Markov network of any strictly positive distribution can be constructed by connecting each variable $X$ to all members of its Markov blanket $\mathbf{M}_X$.*

<table>
<thead>
<tr>
<th>Variable</th>
<th>Markov blanket</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 4</td>
</tr>
<tr>
<td>2</td>
<td>1, 3, 4</td>
</tr>
<tr>
<td>3</td>
<td>2, 5</td>
</tr>
<tr>
<td>4</td>
<td>1, 2, 5</td>
</tr>
<tr>
<td>5</td>
<td>3, 4</td>
</tr>
</tbody>
</table>

To illustrate, Table 2.1 shows the Markov blanket (direct neighbors) of each variable in the Markov network depicted in Fig. 1.1.

<table>
<thead>
<tr>
<th>Symmetry</th>
<th>$(X \Perp Y \mid Z) \iff (Y \Perp X \mid Z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition</td>
<td>$(X \Perp Y \cup W \mid Z) \iff (X \Perp Y \mid Z) \land (X \Perp W \mid Z)$</td>
</tr>
<tr>
<td>Intersection</td>
<td>$(X \Perp Y \mid Z \cup W) \land (X \Perp W \mid Z \cup Y) \implies (X \Perp Y \cup W \mid Z)$</td>
</tr>
<tr>
<td>Strong Union</td>
<td>$(X \Perp Y \mid Z) \implies (X \Perp Y \mid Z \cup W)$</td>
</tr>
<tr>
<td>Transitivity</td>
<td>$(X \Perp Y \mid Z) \implies (X \Perp \gamma \mid Z) \lor (\gamma \Perp Y \mid Z)$</td>
</tr>
</tbody>
</table>
2.3 Independence Oracle Implementation

As mentioned above, DGSIMN belongs to the category of independence-based algorithms for learning the structure of a Markov network. These algorithms work by evaluating a number of conditional independence tests, successively restricting the set of candidate structures consistent with these tests to a singleton (if possible), and inferring this structure as the only possibly correct one. To show correctness, we assume theoretically the existence of an independence-query oracle that can reply the true value of any conditional independence. In practice however, such an independence-query oracle does not exist; instead it can be approximated by a statistical independence test evaluated on data. Many statistical tests of independence have been introduced during the last century or so e.g., Pearson’s $\chi^2$ conditional independence test [2], a mutual information test, etc. All algorithms in this work uses Pearson’s $\chi^2$ square test, which is simple and easy to compute. To test if variable $X$ is independent of variable $Y$ conditioned on variable $Z$, the $\chi^2$ conditional independence test computes the probability of incorrectly claiming that the two variables are dependent when they in fact are independent, a quantity known as the $p$-value of the test. That is, a large p-value implies a large probability of incorrectly claiming dependence, and thus independence must follow. In practice, the test compares the p-value to a threshold $1 - \alpha$, with $\alpha$ usually taking a value $\alpha = 0.05$. More precisely, denoting the p-value by $G(X,Y \mid Z)$,

$$(X \perp Y \mid Z) \iff G(X,Y \mid Z) \geq 1 - \alpha.$$  

In our algorithm we represent the result of an independence test as the result of Boolean function $I(X,Y,Z)$, defined as

$$I(X,Y,Z) = \text{true} \text{ if and only if } (X \perp Y \mid Z).$$  

As a statistical test conceptually constructs a contingency table of counts, a naive implementation of the test would have a cost exponential in the number of variables involved. However, empty cells (containing zero counts) in the contingency table do not really need to be explicitly represented. A more efficient representation of the contingency table therefore is
possible (for example, using a sparse representation implemented by a hash table, that does not include zero counts) that would only examine all data points once, incrementing the appropriate count of the contingency table. Using such an implementation, the time complexity of a statistical test would be proportional to the size of data set $N$ and the number of variables involved. For example, a conditional test between variables 1 and 2 given \{4,5\} has time complexity proportional to $4N$. Therefore, in all our time complexity results, we report the weighted number of tests (and not simply the number of tests conducted), referred as weighted cost hereon, where each test conducted is weighted by the number of variables involved in the corresponding test. This is a more accurate measure of the actual time that the algorithm will take to execute.

2.4 The Triangle Theorem

Triangle theorem was originally presented and proved by Bromberg et al [7]. Main idea behind the DGSIMN algorithm is to reduce the number of tests to be done on data by extracting information from already done tests using Pearl’s theorem shown in Eqn. 2.1. Bromberg in [7] proved that Triangle theorem was enough to infer most of the independence relations that can be inferred using Pearl’s theorems.

**Theorem 2 (Triangle Theorem)** Given Eqs. 2.1, for every variable $X$, $Y$, $W$ and the sets $Z_1$ and $Z_2$ such that \{X,Y,W\} \cap Z_1 = \{X,Y,W\} \cap Z_2 = \emptyset,

\[
(X \perp W \mid Z_1) \land (W \perp Y \mid Z_2) \implies (X \perp Y \mid Z_1 \cap Z_2)
\]
\[
(X \perp W \mid Z_1) \land (X \perp Y \mid Z_1 \cup Z_2) \implies (X \perp Y \mid Z_1)
\]

We call the first relation the “D-triangle rule” and the second the “I-triangle rule”.

Triangle theorem is represented graphically in Fig. 2.1. Triangle theorem can be used efficiently to infer new independence relation between variables using the tests already done on data. DGSIMN algorithm presented as the main contribution of this work uses Triangle rules along with strong union from Pearl’s theorems to infer new independence and dependence relations, in an attempt to reduce the number of tests done on data.
2.5 Fringe

The state of the Markov network learning procedure at any point can be represented using two lists for each variable $X$: $D_X$ and $I_X$. As more and more tests of independence is done on data more and more variables are found to be dependent or independent of each other. $D_X$ contains the variables found to be dependent on $X$. $I_X$ contains the list of variables found independent of $X$. The lists $D_X$ and $I_X$ can be represented graphically as two columns growing in opposite directions, as shown in Fig. 2.2. In this figure $D_X$ grows downwards, and $I_X$ grows upward.

At each stage, we can define the set of tests that can be used to extend a column. We call this set the fringe of $X$, and it depends on the value of $D_X$. The fringe of $X$ is denoted by $F(D_X)$ and is defined as:

$$F(D_X) = \left\{ \text{test } (X,Y' \mid D_X) \mid Y \neq X \text{ and } Y \notin D_X \right\}.$$
Figure 2.2 State of Markov learning procedure for Fig. 1.1 at a certain point. White boxes represent variable is in dependence column of that variable and black boxes represent that the variable is in independence column.

The Fig. 2.2 displays the state at the certain point. The variables in the dependent list $D_X$ of any variable $X$ are arranged in a particular order. In the figure shown for variable 1, $D_1$ has variables 2 and 4, which represents the following knowledge:
Therefore as the ordering of variables in the dependent list is important, the dependent list for any variable can have multiple columns, each of them having a different ordering. This concept of multiple columns is explained in detail in later chapter of this thesis. When the two columns, $D_X$ and $I_X$ meet, the columns can not be extended any more, therefore fringe for that variable is empty. In this example the fringe of all the variables will be:

\[
\begin{align*}
F_1 &= \{(\emptyset)\} \\
F_2 &= \{(2, 3, \{1, 4\})\} \\
F_3 &= \{(3, 2, \emptyset), (3, 4, \emptyset), (3, 5, \emptyset)\} \\
F_4 &= \{(4, 3, \{1, 2\}), (4, 5, \{1, 2\})\} \\
F_5 &= \{(5, 3, \emptyset), (5, 4, \emptyset)\}.
\end{align*}
\]

2.6 Assumptions

There were certain assumption made during the development of DGSIMN algorithm. These assumptions are same as that of competing algorithms First we assume the underlying distribution to be graph-isomorph [20]. A distribution is graph-isomorph if and only if it has a faithful graph $G$, that is, if $G$’s connectivity represents only those dependencies and independences existent in the distribution. Pearl et al. showed in [20] that a necessary and sufficient condition for a distribution to be graph-isomorph is for its set of independences to satisfy the set of properties shown in Eqs. (2.1). These properties can also be used as inference rules in order to infer unseen independences from a set of already seen ones. This observation is exploited by the DGSIMN algorithm, presented in the next chapter.

For the operation of the algorithm we also assume the existence of an oracle that can always
answer statistical independence queries (e.g., by directly querying the underlying probability distribution).

Graph-isomorphism, strict positivity, and existence of an oracle (i.e., reliable tests) are standard assumptions necessary to assure uniqueness, i.e., that there exists a single structure consistent with the tests performed, and correctness, i.e., that the algorithm outputs this unique structure [20].
CHAPTER 3. Review of Related Work

3.1 The Grow-Shrink Algorithm

The Grow-Shrink (GS) algorithm was presented by Margaritis and Thrun in [18]. It was originally developed for learning structure of Bayesian networks. As mentioned earlier the DGSIMN algorithm learns the structure of Markov network by learning the Markov blanket of each of the variable of the domain. To learn the Markov network it uses an extension of the GS algorithm. In this section we explain a generalized version of GS algorithm that will give an idea on how Markov blanket of a variable is learned.

Algorithm 1 GS(X,V)

1: $M \leftarrow \emptyset$
2: /* Grow Phase. */
3: while $\exists Y \in V - \{X\}$ such that $(X \perp Y \mid M)$ do
4: $M \leftarrow M \cup \{Y\}$
5: /* Shrink Phase. */
6: while $\exists Y \in M$ such that $(X \perp Y \mid M - \{Y\})$ do
7: $M \leftarrow M - \{Y\}$
8: return $M$

The GS algorithm, shown in Alg. 1, consists of two phases: The grow phase (lines 3–4) and the shrink phase (lines 6–7). The grow phase proceeds by attempting to add variables to the current set of hypothesized members of the Markov blanket $M_X$, contained in $M$. The set $M$ starts empty and is iteratively grown by some variable $Y$ if and only if $Y$ is found dependent of $X$ when conditioned on the current hypothesized Markov blanket $M$. Due to arbitrariness of the order that variables are examined in the grow loop of the GS algorithm, by the end of the grow phase $M$ may contain variables that are not in the true Markov blanket. This justifies the shrink phase, which removes each false member $Y$ from $M$ if and only if $Y$ is conditionally...
independent of \( X \) given \( M - \{Y\} \). This is because if \( Y \) is found independent of \( X \) it cannot possibly be in its Markov blanket (as there cannot be an edge between \( X \) and \( Y \)), and GS therefore removes it from \( M \).

As previously mentioned, the algorithm does not specify the order in which the variables are grown and shrunk (it returns the correct Markov blanket for any order). In the case of exact independence tests (e.g., by querying an independence-oracle), one can show uniqueness, and thus the order is irrelevant. Otherwise, different orderings may lead to different outcomes.

As mentioned in Section 2.5, one can represent the state of the algorithm during its execution using two lists of variables for each variable \( X \): \( D_X \) and \( I_X \). At each point of the algorithm, \( D_X \) contains the variables found dependent of \( X \) (listed in order) during the grow phase and thus added to the set \( M \) (line 4). \( I_X \) contains the list of variables found independent of \( X \) and thus not added to \( M \) during the grow phase, or added but later removed from \( M \) during the shrink phase (line 7).

Let us illustrate these two lists by the execution of the algorithm on the example Markov network of Fig. 1.1. Let \( X = 1 \) and suppose that the order that variables are examined during the grow phase is \([2, 5, 3, 4]\). The GS algorithm has no knowledge of the underlying structure, but instead conducts a number of independence test queries. In this case, the results of the sequence of these tests during the grow phase are

\[
\begin{align*}
(1 \perp 2 | \emptyset) \\
(1 \perp 5 | \{2\}) \\
(1 \perp 3 | \{2, 5\}).
\end{align*}
\]
Figure 3.1 Graphical representation of the grow phase state \((D_1, I_1)\) of variable 1. The dependence column \(D_1\) is shown in white and the independence column \(I_1\) in black.

After these tests, the values of \(D_1\) and \(I_1\) are \(D_1 = [2, 5]\) and \(I_1 = [3]\). In the example of Fig. 3.1, the fringe is \(F_1 = \{(1, 4, \{2, 5\})\}\).
Let us now assume that an additional test is evaluated. According to the GS algorithm, this test is \((1, 4, \{2, 5\})\), which evaluates to \texttt{false}, indicating dependence. In this case \(D_1\) will be extended by variable 4, resulting in the state shown in Fig. 3.2. Now since the two columns meet the fringe is empty and this concludes the grow phase for this variable.

The algorithm now enters the shrink phase where it tries to remove variables from dependent column \(D_1\) and add it to independent column \(I_1\). In this case, the results of the sequence of these tests during the shrink phase are

\[
(1 \perp 2 \mid \{5, 4\})
\]
\[
(1 \perp 5 \mid \{2, 4\})
\]
\[
(1 \perp 4 \mid \{2\}).
\]

Since variable 1 is independent of variable 5 conditioned on \(2, 4\) it is removed from \(D_1\) and added to \(I_1\). State for this variable is now shown in Fig. 3.3. Now the variables in the

Figure 3.2 State of the grow phase for variable 1 after the test 
\((1 \perp 4 \mid \{2, 5\})\) is evaluated.
dependent column $D_1$ are the variables that will constitute the Markov blanket of variable 1. Now if we take a look at the Fig. 1.1 we see that this is correct.

![Diagram](image)

Figure 3.3 State for variable 1 at the end of GS algorithm

### 3.2 Grow Shrink Markov Network

The Grow-Shrink Markov Network (GSMN) algorithm was developed by Bromberg in [7]. The GSMN algorithm is an adaptation of GS algorithm presented in the Section 3.1. GSMN works by first learning the local neighborhood or the Markov blanket of each variable in the domain by applying GS algorithm. It then learns the underlying Markov network by connecting each variable to all the variables in its Markov blanket.

The GSMN algorithm uses a fixed ordering of variables to learn the Markov network of each variable. The ordering in which variables are visited is determined by a visit ordering or grow ordering computed by the algorithm initially. Before beginning to learn the Markov network of any variable in the domain, the GSMN algorithm computes the strength of dependence between each pair of variable X and Y in the domain. This is done by calculating the unconditional
p-value for each pair of variables $X \neq Y$. This ordering is a heuristic that states that variables $X$ and $Y$ with low unconditional p-value are less likely to be directly connected in the Markov network.

**Algorithm 2** GSMN($\mathbf{V}, \mathcal{D}$)

1: /* Initialization */
2: for all $X \in \mathbf{V}$ do
3: Learn ordering $\pi_X$
4: $M_X \leftarrow \text{nil}$
5: /* Main Algorithm */
6: for all $X \in \mathbf{V}$ do
7: $M_X \leftarrow \text{GS}(X, \mathbf{V})$ using the ordering $\pi_X$
8: /* Construct and return the output network. */
9: $E \leftarrow \emptyset$
10: for all $X \in \mathbf{V}$ do
11: for all $Y \in M_X$ do
12: $E \leftarrow E \cup (X, Y)$
13: $G \leftarrow$ graph with nodes $\mathbf{V}$ and set of edges $E$
14: return $G$

The Alg. 2 represents a simplified version of GSMN algorithm. The *Initialization phase* (lines 1–6) starts by first learning the grow ordering $\pi_X$ for each variable $X \in \mathbf{V}$. It then initializes Markov blanket $M_X$ for each variable $X \in \mathbf{V}$ to empty. Then the GSMN algorithm learns the Markov blanket $M_X$ of each variable using the GS algorithm shown in Section 3.1 and ordering the variables according to $\pi_X$. After that the GSMN algorithm learns the structure of the Markov network by connecting each variable to all the variables in its Markov blanket.

### 3.3 Grow Shrink Inference Markov Network

The Grow Shrink Inference Markov Network (GSIMN) algorithm was presented by Bromberg in [7]. This algorithm works in a similar fashion to that of GSMN algorithm. There are a few differences between GSIMN and GSMN algorithm. First GSIMN uses triangle rules explained in Section 2.4 and strong union to infer independence relations. Every time the algorithm needs to do a test on data, it first applies triangle rules and strong union to the tests already done, and checks if the test that needs to be done can be inferred. If the test can not be
inferred then the test is done on data. Secondly the algorithm updates the visit ordering in an attempt to maximize the use of inferences.

How these inferences work and how they lead to saving in number of tests can be better seen with an example. If we see the example shown Section 3.1, while growing variable 1, the following tests were done on data:

\[
(1 \perp 2 \mid \emptyset)
\]
\[
(1 \perp 5 \mid \{2\})
\]
\[
(1 \perp 3 \mid \{2, 5\})
\]
\[
(1 \perp 4 \mid \{2, 5\}).
\]

Now if we apply the triangle rules to the tests done here we can infer the following:

\[
(3 \perp 4 \mid \{2, 5\}).
\]

Now from this we can see that 4 will never be in the Markov blanket of 3 and vice-versa. Therefore while growing variables 3 or 4 we saved a test. These changes make GSIMN perform much better than GSMN. The GSIMN algorithm showed an improvement of up to 75% over GSMN algorithm. However we believe that having a fixed ordering initially and how GSIMN algorithm updated the visit ordering were not enough to use the inferences efficiently. By using a more dynamic algorithm the saving can be even more.
CHAPTER 4. The DGSIMN Algorithm

In this chapter we present to you the DGSIMN algorithm, our main contribution to this work. Before presenting the algorithm we present to you Multiple columns, a way to represent the state of the algorithm at any point in Section 4.1. We explain the DGSIMN algorithm in detail in Section 4.2. In Section 4.3 we discuss how DGSIMN algorithm dynamically chooses the next test to be done on data. Section 4.4 discusses the various experiments conducted as part of this thesis and the comparison between DGSIMN algorithm vs GSIMN algorithm.

4.1 Multiple Columns

We now show that the GS algorithm can be extended to exploit an independence test coming from an external source, even if this test cannot be used to extend a column. The source of these test results is a combination of Triangle theorem shown in Section 2.4 and using the axioms of Eqs. (2.1). This is achieved by considering several alternative growing phases, i.e., by maintaining multiple columns.

Suppose that a new test result \((X \perp Y \mid S)\) becomes available to the algorithm, with \(S \neq D_X\). If \(S \supset D_X\), then \((X \perp Y \mid D_X)\) follows by Strong Union and we can therefore extend the column with \(Y\). Otherwise, a way to exploit this test is by considering it as a step in an alternative grow phase. This is possible because the GS algorithm does not require any particular ordering for the grow phase, and thus more than one alternative ones may exist simultaneously. To represent the state of the algorithm we now must be able to maintain more than one dependence list \(D_X\), one for each alternative. We denote the set of dependence lists of variable \(X\) by \(D_X\).

Let us clarify this idea through the example of Fig. 4.1. In this example the state for
variable 1 consists of a single column i.e., $D_1 = \{[2, 5]\}$, $I_1 = \{\}$. Suppose now that we learn that $(1, 4 | 3)$. Since $\{3\} \not\subseteq D_1$, this test cannot be used to grow the existing dependence column. However, through Strong Union, we can infer that $(1, 4 | \emptyset)$. This new fact can be considered as the beginning of a new (alternative) growing phase, and we can therefore create a new column $[4]$, i.e., $D_1 = \{[2, 5], [4]\}$. The state of the new growing phase is shown in Fig. 4.1.

![Figure 4.1](image)

Figure 4.1 Graphical representation of the multi-column grow phase state $(D_1, I_1)$ of variable 1.

The benefit of maintaining several columns is that some columns may complete the grow phase before others. More precisely, if $D_X^*$ is the longest column in $D_X$, the grow phase terminates if and only if $D_X^* \cup I_X = V - \{X\}$. Completing the grow phase allows the GS algorithm to proceed to the shrink phase sooner, speeding up the learning of the Markov blanket of $X$. In this case we say the the grow phase is completed.

The extension to multiple columns requires a generalization of the concept of fringe, since now there may exist more than one tests that may be useful to evaluate, i.e., one for each
column. Therefore the fringe of $X$, denoted $F_X$, is now defined for a set of columns $D_X$ as

$$F_X = \begin{cases} \emptyset & \text{if } D_X \cup I_X = V - \{X\} \\ \bigcup_{D_X \in D_X} F(D_X) & \text{otherwise.} \end{cases} \quad (4.1)$$

where we set it to the empty set when the longest column is complete, i.e., when the grow phase of variable $X$ is finished. Technically, columns can still be grown after one of its columns completed, but for things to be consistent later we wanted to make explicit the fact that the grow phase is over. In the example shown in Fig 4.1, the multi-column fringe of variable 1 is,

$$F_1 = F([2,5]) \cup F([4]) = \{(1,4,\{2,5\}), (1,2,\{4\}), (1,5,\{4\})\}$$

**Algorithm 3** update-cols($(X,Y,S),t$)

1. if $t = \text{false}$ then
2. \quad if $\exists D \in D_X$ such that $S \supseteq D$ then
3. \quad \quad $D \leftarrow \text{concatenate}(D,Y)$
4. \quad else
5. \quad \quad $D_X \leftarrow D_X \cup \{[Y]\}$
6. else
7. \quad $I_X \leftarrow \text{concatenate}(I_X,Y)$
8. for each $D \in D_X$ do
9. \quad remove $Y$ from $D$
10. Update the fringe $F_X$ of $X$ using Eq. (4.1).

Alg. 3 summarizes these ideas, showing precisely how $D_X$, $I_X$ and $F_X$ are updated after test $(X,Y \mid S)$ evaluates to $t$, where $t \in \{\text{true, false}\}$. Given as input $(X,Y,S)$ and $t$, the update-cols subroutine tries to advance the grow phase of the GS algorithm of variable $X$ one step. If $t = \text{false}$ and there exists some element $D \in D_X$ for which $S \supseteq D$, it extends $D$ by appending $Y$ to its end, otherwise it creates a new column containing only $Y$ i.e., $[Y]$ is added to $D_X$. If $t = \text{true}$ then $Y$ is appended to $I_X$ and removed from every column in $D_X$ that contains it. If variable $X$ and $Y$ are independent conditioned on some set of variables, then $X$ and $Y$ will never be in the Markov blanket of each other. Therefore it is reasonable to remove it from all the dependent columns and add it to the independent list. Also since ordering does not matter in the independent list $I_X$, therefore $I_X$ will only have one column per variable.
4.2 DGSIMN

In this section we present our main contribution, the DGSIMN algorithm (Dynamic Grow-Shrink Inference-based Markov Network learning algorithm). The algorithm is shown in Alg. 4; it takes as input a data set $\mathcal{D}$ and a set of random variables $\mathbf{V}$, and outputs an undirected graph $G$, with set of nodes $\mathbf{V}$ and set of edges $\mathbf{E}$, that best represents the underlying Markov network. The algorithm consists on three phases: initialization, main loop, and construction of output network.

Algorithm 4 DGSIMN($\mathbf{V}, \mathcal{D}$)

1: /* Initialization. */
2: $K \leftarrow \emptyset$
3: for all $X \in \mathbf{V}$ do
4: $\mathbf{D}_X \leftarrow \{\emptyset\}$
5: $I_X \leftarrow \emptyset$
6: $\mathbf{M}_X \leftarrow \text{nil}$
7:
8: /* Main loop. */
9: while $\mathbf{F} \neq \emptyset$ do
10: /* Select test with maximum utility (cf. §4.3). */
11: $(X, Y, S) \leftarrow \arg \max_{T \in \mathbf{F}} U(T)$
12: /* Do the test. */
13: $t \leftarrow I(X,Y,S)$
14: applyTest$((X,Y,S), t)$
15:
16: /* Shrink complete columns not yet shrunk. */
17: for each $Z \in \mathbf{V}$ s.t. $\mathbf{F}_Z = \emptyset$ AND $\mathbf{M}_Z = \text{nil}$ do
18: /* Shrink longest column $D^*_Z$. */
19: for each $W \in D^*_Z$ do
20: if $I(Z,W,D^*_Z - \{W\}) = \text{true}$ then
21: applyTest$((Z,W,D^*_Z - \{W\}), \text{true})$
22: $\mathbf{M}_Z \leftarrow D^*_Z$
23: 24: /* Construct and return the output network. */
25: $\mathbf{E} \leftarrow \emptyset$
26: for all $X \in \mathbf{V}$ do
27: for all $Y \in \mathbf{M}_X$ do
28: $\mathbf{E} \leftarrow \mathbf{E} \cup (X,Y)$
29: $G \leftarrow \text{graph with nodes } \mathbf{V} \text{ and set of edges } \mathbf{E}$
30: return $G$
During the initialization phase (lines 1–6), the algorithm creates a knowledge base \( K \) initially with no propositions. (\( K \) maintains the independence and dependence propositions that are either evaluated on data or inferred.) It also initializes the set of dependence columns of each variable to a single empty column and its independence column to be empty. Finally, it initializes its Markov blanket as \( \text{nil} \) (indicating that its Markov blanket has not been created yet).

The Markov blankets of all the variables in \( V \) are learned in the main loop (lines 9–22) by a simultaneous application of the generalized GS algorithm presented in Section 3.1. The order of the tests performed is generated dynamically during the execution of the main loop, where at each iteration the test to be performed next is selected greedily from the pool of useful tests, contained in the global fringe \( F \) (or simply fringe) defined as

\[
F = \bigcup_{X \in V} F_X.
\]

The main loop continues until the Fringe is not empty, i.e until the grow phase for all variables in the domain has not ended. The algorithm selects the test in \( F \) that is expected to produce the larger number of useful inferences (i.e., those that can be used to extend one or more columns). This is computed by the utility function \( U \), discussed in detail in Section 4.3 below.

The optimal test \( T^* = (X,Y,S) \) selected is then performed on data using the boolean function \( I \) (cf. Eq. (2.2)). Its outcome (true or false) is applied to the state of the algorithm using Alg. 5, an operation that consists of three steps: First, the new test is used to update the columns of \( X \) and \( Y \) using Alg. 3, as explained in detail in the previous section. Second, \( T^* \) is added to a knowledge base \( K \) of independence and dependence propositions maintained throughout the algorithm, which is populated with all inferred independences resulting from running a forward chaining inference algorithm [21] on \( K \cup \{T^*\} \) using the Triangle rules (cf. Theorem 2). Third, each of the newly inferred tests \( (X',Y',S') \) are used to update the columns of \( X' \) and \( Y' \) using Alg. 3.

After applying the optimal test \( T^* \), the column of one or more variables may have been completed, and thus that variable must proceed to its shrink phase. Let's denote by \( Z \) such a variable (if any). The algorithm detects if a column of variable \( Z \) is complete by checking if
its fringe is empty ($F_Z = \emptyset$). To avoid shrinking a variable twice, the algorithm confirms that its Markov blanket $M_Z$ has not been initialized ($M_Z = \text{nil}$). As in the growing phase, every test performed on data during the shrink phase (line 20) is applied to the global state using Alg. 5 (line 21). The shrink phase of a variable $Z$ terminates by setting its Markov blanket $M_Z$ to $D^*_Z$, which after the shrink phase contains exactly those variables in the Markov blanket.

It is important that as soon as a column for any variable gets completed i.e. reaches the end of grow phase, it proceeds into Shrink phase to improve the performance of the algorithm. Let us assume that the variable $X$ contains another variable $Y$ in its longest column at the end of grow phase, which is not in the Markov blanket of $X$. The shrink phase will remove any such variable $Y$ from column of $X$, and will also remove variable $X$ from columns of all such variables $Y$. Therefore when next test is done on data involving variable $Y$, we might be able to do the test with a conditioning set of smaller size, hence saving on the weighted number of test. Also once the Markov blanket of a variable is known, no further changes are made to the columns of this variable.

The main loop proceeds until the grow and shrink phases of all variables has been completed, a condition satisfied when the fringe of each variable is empty. At this point, the Markov blanket of all variables has been learned. The algorithm concludes with the construction of the output network $G$. This proceeds by connecting each variable to all the variables in its Markov blanket (lines 25–29), as indicated by Theorem 1.

**Algorithm 5** applyTest($\langle X, Y, S \rangle, t$)

1: $\text{update-cols}((X, Y, S), t)$
2: $\text{update-cols}((Y, X, S), t)$ /* For symmetry. */
3: /* Do inference on new test. */
4: $K \leftarrow \text{run forward chaining on } K \cup \{(X, Y, S) = t\}$
5: for every newly inferred test $t' = (X', Y', S')$ do
6: $\text{update-cols}((X', Y', S'), t')$
7: $\text{update-cols}((Y', X', S'), t')$ /* For symmetry. */
4.3 Utility Function

The dynamic aspect of the DGSIMN algorithm refers to its ability to choose the next test to evaluate on the fly. This test is selected greedily from the fringe of all variables and consists of the test that has the maximum expected utility. Intuitively, the best choice of test to execute next is the one that will move the state of the algorithm closer to termination. The utility of the test is therefore the expected distance to termination, as measured by the expected number of weighted tests required to complete exactly one column of each random variable.

To calculate the expected number of tests remaining for the grow phase of a column of a variable, let \((D, I)\) be its state in the algorithm, with \(D\) and \(I\) its dependence and independence lists respectively and \(d = |D|\) and \(i = |I|\). Then that column is \(m = n - d - i - 1\) tests away from completion of its grow phase, where \(n = |V|\). Unfortunately, it is not possible to calculate the exact remaining cost for that column because each of the remaining tests may result in dependence or independence, producing a difference in the cost of subsequent tests (dependences grow the \(D\) column which determines the conditioning set of subsequent tests, while independences grow the \(I\) column which does not affect the cost of the next test). Instead we must therefore consider all possible values for the sequence of the remaining tests of the column.

Lacking prior knowledge of the results of future tests, we assume that the events of the test being true or false (i.e., returning independence or dependence) are equiprobable. Under this assumption, the expected number of weighted tests \(f(d, i)\) required to complete a column \((D, I)\) is

\[
f(d, i) = \frac{1}{2^m} \sum_{s \in \{t, f\}^m} w(s),
\]

where \(\{t, f\}^m\) denotes the set of all sequences of \(t\)'s and \(f\)'s of length \(m\), and \(w(s)\) denotes the cost of test sequence \(s\). For a given variable \(X\), we estimate that the expected number of tests required to terminate its grow phase to be the expected number of tests required to terminate the longest column \(D_X^*\). Even though such an estimate is a heuristic, it has proved to work well in our experiments, presented in the next section.
Finally, the total distance of the algorithm from termination is the sum of the distances for each variable in the domain:

\[ f = \sum_{x \in V} f(|D_X|, |I_X|). \]

We can now define the utility \( U(T) \) of some test \( T \) as the distance to the goal, averaged over its value of dependence or independence. In other words, if \( f_{T=r}, r \in \{t, f\} \) is the distance to the goal after applying the test result \( T = r \) to our current state of the algorithm (i.e., running forward-chaining and updating the columns accordingly using Alg. 5), the utility of test \( T \) is

\[ U(T) = -\frac{f_{T=t} + f_{T=f}}{2}. \]

4.4 Experimental Results

We conducted experiments on both artificial and real-world data sets. As we will show below, the dynamic selection of tests of DGSIMN results in a reduction of the weighted cost required to learn the Markov structure, without significantly affecting the quality of the output network in most cases. We report the following quantities:

- **Weighted cost**: As argued in section 4.3, the weighted cost is proportional to the running time of the algorithm in the limit of very expensive tests, such as scenarios with large data sets or distributed data.

- **Ratio of correct edges**: In cases where the network used to generate the data is known, we measured the ratio of correct edges between the output network and the network of the underlying model. This is defined as the number of “matching” edges between the two networks normalized by \( n(n-1)/2 \), the total number of possible edges. The matching edges are those edges that exist in both networks, in addition to the number of edges that are missing in both networks.

- **Accuracy**: For real-world data the underlying model is unknown. We thus assess the quality of the output network by comparing a number of conditional independences and
dependences represented in the network (using vertex separation) with those represented in the data (using statistical independence tests).

4.4.1 Exact Learning

Exact structure learning occurs when the learning algorithm has access to an oracle that can accurately answer any independence query. In cases where the structure of the underlying model is known (called the true network hereon), we can simulate such an oracle by vertex separation on the true structure. Exact learning allows an evaluation of the algorithm under ideal conditions of reliable tests and graph-isomorphic domains, which guarantee the correct output, i.e., the output network matches exactly the true network. We thus report only the weighted cost in this set of experiments.

Experiments were conducted on randomly generated true networks of different sizes (number of nodes) and average degrees per node. Each true network containing $n$ variables was generated randomly as follows: the network was initialized with $n$ nodes and no edges. The average degree is determined by a user-specified parameter $\tau$ that equals the average number of direct neighbors per node. Given $\tau$, edges were selected uniformly by selecting the first $\frac{\tau n^2}{2}$ edges from a random permutation of the set of all pairs of nodes. The factor $1/2$ is needed to account for each edge contributing to the degree of two nodes.
Figure 4.2 Ratio of the weighted cost DGSIMN over GSIMN for different network sizes (number of nodes) \( n \) and average degrees \( \tau = 1, 2, 4, \) and 8. Ratios smaller than 1 indicate an advantage of DGSIMN over GSIMN.

Figure 4.2 depicts the ratio of weighted cost of DGSIMN over GSIMN for true networks of size up to 50 variables. The different line plots in the graph represent the average degree \( \tau = 1, 2, 4, 8 \). From the plots we observe that for \( \tau = 1 \) weighted cost ratio is always less than 1, which means that DGSIMN algorithm always performs better than GSIMN. For networks of 10 variables there is 10% improvement and the improvement improves as the number of variables in the domain increase. For networks with 50 variables, there is an improvement of around 35% for \( \tau = 1 \). For \( \tau = 2 \) we observe similar behavior, and DGSIMN shows improvement of around 22% in networks with 50 variables. For \( \tau = 4, 8 \), and networks with less variables the DGSIMN
does worse than GSIMN algorithm around 10% in terms of weighted number of test. But a pattern similar to $\tau = 1$, and 2 is seen and as number of variables in the domain increases, DGSIMN performs better than GSIMN. Also for these networks as the independence tests are absolutely accurate, therefore both the networks learn the exact graph. Also from this we can infer that DGSIMN performs better when the underlying Markov network is less dense.

4.4.2 Sampled Data

Experiments in this section compare the performance of DGSIMN and GSIMN on data sampled from a known network, using a Gibbs sampler. Unlike the exact learning experiments of the previous section, independence queries are now performed on data using Pearson’s $\chi^2$ statistical test. The utility of this set of experiments is two-fold: First, they provide a more realistic measure of the behavior of the algorithms as statistical tests may be unreliable. Second, this quality can be measured more accurately through a direct comparison with the true network.

Data sets containing 10,000 data points were sampled with random networks of sizes $n = 4$, 8, 12, 20, 30, 50, 75 and average degrees $\tau = 1, 2, 4, 8$, generated randomly using the procedure described in Section 4.4.1.

To fully describe the true distribution we must specify not only its network, but also its parameters. These parameters determine the strength of the dependencies among variables connected in the graph. Agresti [2] proposes the log-odds ratio $\theta_{XY}$ as a measure of the strength between random variables $X$ and $Y$, defined as

$$\theta_{XY} = \log \frac{\Pr(X = 0, Y = 0) \Pr(X = 1, Y = 1)}{\Pr(X = 0, Y = 1) \Pr(X = 1, Y = 0)}.$$ 

The network parameters were generated randomly so that the log-odds ratio between variables directly connected by an edge has a specific value. In our experiments we chose $\theta = 1$ for each of these pairs.

Figures 4.3 and 4.4, depict the result obtained from experiments on sampled data for $\tau$ ranging from 1 to 8 for different number of variables in the domain. Each of the graphs here contain two plots. A line plot showing the ratio of number of weighted test done by DGSIMN vs.
GSIMN. Therefore a ratio of less than one shows that DGSIMN performs better than GSIMN in that particular case. The bar plot shows the difference in accuracy between DGSIMN and GSIMN. A positive number means that the network learned by DGSIMN was closer to the true underlying Markov network. The relative behavior of the two algorithms in terms of weighted costs is similar to the case of exact learning. For less connected networks ($\tau = 1, 2$), DGSIMN always requires less weighted cost. In case of graphs with $\tau = 1, 2$ DGSIMN shows an improvement of up to 85% in some cases. For $\tau = 4, 8$, DGSIMN starts outperforming GSIMN as the number of variables in the domain increases. We see an improvement of up to 60% for $\tau = 4$ and improvement of 30% for $\tau = 8$.

In terms of accuracy we see that DGSIMN improves accuracy for cases of $\tau = 1, 4$ and $\tau = 8$. Also for $\tau = 4$ the accuracy is still within 10% of GSIMN. Overall DGSIMN improves by a great amount over GSIMN in terms of weighted cost while still maintaining and improving accuracy in most cases.

### 4.4.3 Real-world Data

We also conducted a number of experiments on real-world data sets from the UCI machine learning data repository [10]. Experiments on real-world data provide us a better assessment of the performance of the algorithm due to the several factors: the underlying model may not be graph-isomorph, violating one of our assumptions; the reliability of the tests may be lower due to the usually small data set size; and the structure of the underlying model may have a non-random topology. In this thesis we address discrete data only, so continuous variables in the data sets were discretized. We used a widely recommended discretization method [22] that dictates that the optimal number of equally-spaced discretization bins for each continuous variable is $k = 1 + \log_2 N$, where $N$ is the number of data points in the data set.

Since the structure of the underlying network is unknown for virtually all real-world data sets, the output network cannot be compared with the true one. Instead we measured quality by comparing the result (true or false) of a number of conditional independence tests performed on the output network (using vertex separation) and on the data (using the $\chi^2$ test). Ideally we
would compare all possible tests, but this is usually impossible as there exist an exponential
number of them. We thus estimate the accuracy over a set $T$ composed of 100 randomly
sampled triplets $(X, Y, S)$ per conditioning set size $m = |S|$, for $m \in \{0, \ldots, n - 2\}$. Denoting
by $I_{out}(t) \in \{\text{true, false}\}$ the result of the test performed on the output network for triplet
$t \in T$ and by $I_{data}(t) \in \{\text{true, false}\}$ the result of the test performed on data, the accuracy
is defined as

$$\text{accuracy} = \frac{1}{|T|} \{ t \in T | I_{out}(t) = I_{data}(t) \}.$$

Table 4.1   Weighted number of tests and accuracy for several real-world
data sets. For each evaluation measure, the best performance
between DGSIMN and GSIMN is indicated in bold. The number
of variables in the domain is denoted by $n$ and the number of
data points in each data set by $N$.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Weighted cost</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>name</td>
<td>$n$</td>
</tr>
<tr>
<td>1</td>
<td>cmc</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>flare2</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>bridges</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>hepatitis</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>hayes-roth</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>crx</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>flag</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>monks-1</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>haberman</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>echocardiogram</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>lenses</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>imports-85</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>balance-scale</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>baloons</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>tic-tac-toe</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>car</td>
<td>7</td>
</tr>
<tr>
<td>17</td>
<td>nursery</td>
<td>9</td>
</tr>
<tr>
<td>18</td>
<td>dermatology</td>
<td>35</td>
</tr>
</tbody>
</table>
Table 4.1 displays the comparison between DGSIMN and GSIMN on a set of real-world data sets. From the table we can see that DGSIMN requires less number of weighted test compared to GSIMN for every dataset. The improvement in number of weighted number of test range from 20% to 88%. Also at the same time we can all infer from the table that the DGSIMN is more accurate than GSIMN algorithm in 8 cases. It does similar in accuracy in 2 cases and worse in the other 8 cases.

![Weighted costs and accuracy for Real-world data](image.jpg)

**Figure 4.5** Ratio of the weighted cost of DGSIMN over GSIMN and difference between the accuracy of DGSIMN and GSIMN on real data sets. Ratios smaller that 1 and positive bars indicate an advantage of DGSIMN over GSIMN. The numbers in the x-axis are indices of the data sets as shown in Table 4.1.
graph and the ratio of weighted cost for each file in Table 4.1. This figure is similar to one shown for sampled data. From the graph we can clearly see that when DGSIMN is less accurate the accuracy of DGSIMN is within 10% of GSIMN. But in the cases where DGSIMN improves in accuracy the accuracy of DGSIMN is up to 54% better than GSIMN. Therefore from the graph and table we can infer that DGSIMN reduces the required number of weighted tests over GSIMN while achieving similar or better accuracy in most cases.
Figure 4.3  Ratio of weighted cost of DGSIMN over GSIMN (continuous line) and the difference of the ratio of correct edges of DGSIMN and GSIMN (bar graph) for different number of variables \( n \) and average degrees \( \tau = 1 \), and 2. Ratios smaller that 1 and positive bars indicate an advantage of DGSIMN over GSIMN.
Figure 4.4  Ratio of weighted cost of DGSIMN over GSIMN (continuous line) and the difference of the ratio of correct edges of DGSIMN and GSIMN (bar graph) for different number of variables $n$ and average degrees $\tau = 4$, and 8. Ratios smaller than 1 and positive bars indicate an advantage of DGSIMN over GSIMN.
CHAPTER 5. Conclusion and Future Work

In this thesis we presented the DGSIMN algorithm that improves upon the GSIMN algorithm in the task of learning the structure of the Markov network of a domain from data using an independence-based approach. DGSIMN works by conducting a series of statistical conditional independence tests on the data, and uses the axioms that govern the independence relation to avoid unnecessary tests i.e., tests that can be inferred from the results of known ones. DGSIMN improves on the GSIMN algorithm by dynamically selecting the locally optimal test that will increase the state of knowledge about the structure the most. This is done by estimating the number of inferences will be obtained by executing a test before it is done on data, and selecting the one that is expected to maximize the number of such inferences. Experiments show that DGSIMN yields savings of up to 85% in both sampled data real-world data while achieving similar or better accuracy in most cases.

Some directions of future work in this area would be to explore the idea of speculative growing of Markov blanket in the grow phase. This can be achieved by doing set of all unconditional tests initially and using p-value to approximate which all variables will constitute the Markov blanket. Another direction of future research can be directed toward investigating an more extensive utility function which would employ a multi-step look ahead. Also a specific ordering like more dependent or more independent variables first can be applied in the cases where two or more tests have the same utility.
BIBLIOGRAPHY


ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Dimitris Margaritis for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education.

I would like to thank Facundo Bromberg for his help, guidance and insights that helped me complete my research.

I would also like to thank my committee members Dr. Vasant Honavar and Arka P. Ghosh for their support, helpful discussions and comments.