

Learning Markov Network Structure using Few Independence Tests

Parichey Gandhi
Iowa State University
Ames, Iowa, USA
parichey@iastate.edu

Facundo Bromberg
Iowa State University
Ames, Iowa, USA
bromberg@iastate.edu

Dimitris Margaritis
Iowa State University
Ames, Iowa, USA
dmarg@iastate.edu

Abstract

In this paper we present the Dynamic Grow-Shrink Inference-based Markov network learning algorithm (abbreviated DGSIMN), which improves on G SIMN, the state-of-the-art algorithm for learning the structure of the Markov network of a domain from independence tests on data. DGSIMN, like other independence-based algorithms, works by conducting a series of statistical conditional independence tests toward the goal of restricting the number of possible structures to one, thus inferring that structure as the only possibly correct one. During this process, DGSIMN, like the G SIMN algorithm, uses the axioms that govern the probabilistic independence relation to avoid unnecessary tests i.e., tests that can be inferred from the results of known ones. This results in both efficiency and reliability advantages over the simple application of statistical tests. However, one weakness of G SIMN is its rigid and heuristic ordering of the execution of tests, which results in potentially inefficient execution. DGSIMN instead uses a principled strategy, dynamically selecting the locally optimal test that is expected to increase the state of our knowledge about the structure the most. This is done by calculating the expected number of independence facts that will become known (through inference) after executing a particular test (before it is actually evaluated on data), and by selecting the one that is expected to maximize the number of such inferences, thus avoiding their potentially expensive evaluation on data. As we demonstrate in our experiments, this results in an overall decrease in the computational requirements of the algorithm, sometimes dramatically, due to the decreased the number of tests required to be evaluated on data. Experiments show that DGSIMN yields savings of up to 88% on both sampled and benchmark data while achieving similar or better accuracy in most cases.

1 Introduction

Over the last few decades, with the increase in computational processing power and disk storage size and the decrease in the cost of gathering data, it has become significantly cheaper and easier to generate enormous amounts of data. To extract meaningful and useful in-

formation from this data, a number of data mining algorithms are being developed. Frequently in these algorithms, the task of using existing data to extract interesting relationships or 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 among the variables in the domain. A better alternative that makes use of this type of information is the use of a graphical model (a Bayesian or Markov network) to succinctly store the joint probability distribution. Additionally, graphical models have the advantage of clear semantics, intuitive visualization of salient domain dependencies, and are based on a sound and widely accepted theoretical foundation (probability theory).

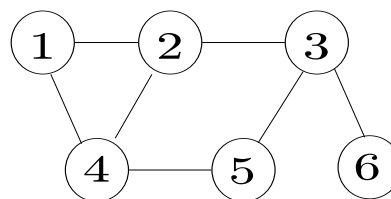


Figure 1: An example Markov network with nodes representing the variables in the domain $\mathbf{V} = \{1, 2, 3, 4, 5, 6\}$.

In this work 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 pa-

rameters. Together, the graph and the parameters can be used to represent the joint probability distribution of the random variables of the domain. The structure of an example Markov network is shown in Figure 1. 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 [19]. For example in Fig. 1, variable 1 is conditionally independent of 5 given the value of 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. In this work we concentrate on the problem of learning the structure of a Markov network, and present the **DGSIMN** algorithm (Dynamic Grow-Shrink Inference-based Markov Network structure learning algorithm) that is able to learn the structure using the outcomes of statistical conditional independence tests.

The rest of the paper is organized as follows: In the next section we review previous work related to topics in the present paper, followed by an introduction in Section 3 of some of the notation and definitions that we will use. Our main contribution, the DGSIMN algorithm, is presented in Section 4 and experimentally evaluated in Section 5. We conclude with a summary in Section 6.

2 Related Work

Markov networks have been used in 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 recent years, Markov networks have also been applied to spatial statistics, with potential applications in transportation, environmental sciences, meteorology, agronomy and others [21], as well as various research problems including analysis of gene expression pathways [12] and computer vision [11], among others. The problem with applying such models however is the fact that, except in rare situations, the true structure of the underlying Markov network is usually unknown. In most of these applications the structure of the network is provided by an expert, usually derived by connecting each node with its physically or conceptually nearest neighbors (as measured by either Euclidean or some other abstract distance). A solution to this problem is to learn the structure from data [14, 8] which, 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.

A number of algorithms for learning Markov network structure have appeared in the literature. Two common categories are *score-based* [9, 18] and *independence-based* or *constraint-based* [22, 7] 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 super-exponential in the domain size i.e., $2^{n(n-1)/2}$ for a domain that contains n variables, these algorithms frequently must resort to heuristic search. Scores that have been used include maximum likelihood, minimum description length, 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 [15] 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 statistical 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 main contribution is the DGSIMN algorithm which belongs to the latter class (independence-based). 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 our extension and generalization of the Grow-Shrink (GS) algorithm by Margaritis and Thrun [17], an independence-based algorithm originally developed for learning the

structure of Bayesian networks. The DGSIMN algorithm is also 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 [19] 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 rigid and heuristic 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 results in superior performance in most cases, as our experimental evaluation confirms.

In the next section, we present the notation that will be used throughout the rest of the paper and describe related concepts and algorithms.

3 Notation and Preliminaries

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

We use the notation $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$ or $(X, Y, \mathbf{Z}) = \mathbf{true}$ to denote the fact that \mathbf{X} is independent of \mathbf{Y} conditioned on \mathbf{Z} and $(\mathbf{X} \not\perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$ or $(X, Y, \mathbf{Z}) = \mathbf{false}$ to denote that \mathbf{X} is dependent on \mathbf{Y} conditional on \mathbf{Z} . Slightly abusing notation, we use the shortcut X instead of $\{X\}$ e.g., $(X \perp\!\!\!\perp Y \mid \mathbf{Z})$ instead of $(\{X\} \perp\!\!\!\perp \{Y\} \mid \mathbf{Z})$. We denote the data set as \mathcal{D} , and its size (number of data points) by N . We use \mathbf{M}_X to denote the Markov blanket of variable X . Formally, the Markov blanket \mathbf{M}_X of variable $X \in \mathbf{V}$ is any set of variables $\mathbf{S} \subseteq \mathbf{V}$ such that

$$(X \perp\!\!\!\perp \mathbf{V} - \mathbf{S} - \{X\} \mid \mathbf{S}).$$

The set of all Markov blankets can be used to construct the Markov network by making use of the following theorem.

THEOREM 3.1. (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 1: Markov blanket corresponding to variables in Figure 1.

Variable	Markov blanket
1	2, 4
2	1, 3, 4
3	2, 5
4	1, 2, 5
5	3, 4

To illustrate, Table 1 shows the Markov blanket (direct neighbors) of each variable in the Markov network depicted in Fig. 1.

In this work we also assume the underlying probability distribution of the domain to be graph-isomorph [19]. A distribution is *graph-isomorph* if and only if it has a faithful graph G , that is, if G ’s connectivity represents exactly those dependencies and independences that hold true in the distribution. As Pearl et al [19] showed, a necessary and sufficient condition for a distribution to be graph-isomorph is for its set of independences to satisfy the properties in Eqs. (3.1), displayed framed on the following page. These properties are universally quantified over all sets of variables $\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{W}$ and single variable γ , and can thus be used as inference rules to infer unseen independences from the ones that are known. This observation is exploited by the DGSIMN algorithm, presented in the next section.

For the operation of the algorithm we also assume the existence of an *ideal oracle* that can always answer statistical independence queries (e.g., by directly querying the underlying probability distribution). A practical implementation (approximation) of such an oracle is discussed in Section 3.1 below.

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 [19].

Based on Pearl’s properties, Bromberg et al introduced in [7] a simplified set of properties, called the *Triangle rules* due to their graphical interpretation (not shown in this paper). The correctness of these rules follows from Eqs. (3.1) as the following theorem shows.

THEOREM 3.2. (TRIANGLE THEOREM) *Given Eqs. (3.1), for every variable X, Y, W and the sets \mathbf{Z}_1 and \mathbf{Z}_2 such that $\{X, Y, W\} \cap \mathbf{Z}_1 = \{X, Y, W\} \cap \mathbf{Z}_2 = \emptyset$,*

$$\begin{aligned} (X \not\perp\!\!\!\perp W \mid \mathbf{Z}_1) \wedge (W \not\perp\!\!\!\perp Y \mid \mathbf{Z}_2) &\implies (X \not\perp\!\!\!\perp Y \mid \mathbf{Z}_1 \cap \mathbf{Z}_2) \\ (X \perp\!\!\!\perp W \mid \mathbf{Z}_1) \wedge (W \not\perp\!\!\!\perp Y \mid \mathbf{Z}_1 \cup \mathbf{Z}_2) &\implies (X \perp\!\!\!\perp Y \mid \mathbf{Z}_1). \end{aligned}$$

As shown in [7], this simplified set of rules al-

Symmetry	$(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \iff (\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z})$	(3.1)
Composition/ Decomposition	$(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \iff (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z})$	
Intersection	$(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z})$	
Strong Union	$(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W})$	
Transitivity	$(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \gamma \mid \mathbf{Z}) \vee (\gamma \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$	

lows a considerable improvement in speed compared to a full-blown inference procedure using the axioms of Eqs. (3.1), while still inferring most of the useful independence relations that can be inferred by applying the complete set. Our DGSIMN algorithm uses a combination of these Triangle rules and the property of Strong Union for inferring new independence relations from known ones.

3.1 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. To show theoretical correctness, one must assume the existence of an independence-query oracle that can provide 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 such tests of independence have been introduced in the statistical literature during the last century or so e.g., Pearson’s χ^2 conditional independence test [2], a mutual information test, etc. All algorithms in this work use Pearson’s χ^2 square test, which is simple and easy to compute. To test if variable X is independent of variable Y conditioned on variable \mathbf{Z} , the χ^2 conditional independence test computes the probability of making an error if we assume that the two variables are dependent given the data when in fact they are independent, a quantity known as the *p-value* of the test. 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 confidence threshold $1 - \alpha$, with α commonly taking the value $\alpha = 0.05$. More precisely, denoting the p-value by $G(X, Y \mid \mathbf{Z})$,

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

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

$$(3.2) \quad I(X, Y, \mathbf{Z}) = \text{true} \text{ if and only if } (X \perp\!\!\!\perp Y \mid \mathbf{Z}).$$

As a statistical test conceptually constructs a contingency table of counts, one for each combination of values of the variables involved in the test, a naive implementation of such a 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 that does not explicitly store zero counts, possibly implemented using a hash table—that only needs to examine each data point once, incrementing the appropriate count of the contingency table. Using such an implementation, the time complexity of a statistical test is 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 to as *weighted cost* hereon, with each test conducted weighted by the number of variables involved in it. This is a more accurate measure of the actual time that the algorithm will take to execute.

Algorithm 1 GS(X, \mathbf{V})

```

1:  $\mathbf{M} \leftarrow \emptyset$ 
2: /* Grow Phase. */
3: while  $\exists Y \in \mathbf{V} - \{X\}$  such that  $(X \not\perp\!\!\!\perp Y \mid \mathbf{M})$  do
4:    $\mathbf{M} \leftarrow \mathbf{M} \cup \{Y\}$ 
5: /* Shrink Phase. */
6: while  $\exists Y \in \mathbf{M}$  such that  $(X \perp\!\!\!\perp Y \mid \mathbf{M} - \{Y\})$  do
7:    $\mathbf{M} \leftarrow \mathbf{M} - \{Y\}$ 
8: return  $\mathbf{M}$ 

```

3.2 The Grow-Shrink Algorithm. As mentioned above, the general approach of the DGSIMN algorithm is to learn the structure by learning the Markov blanket of each variable in the domain. The Markov blanket \mathbf{M}_X of variable $X \in \mathbf{V}$ is learned by an extension of the Grow-Shrink (GS) algorithm [17]. (The extension is described in the next section.) In the following,

through the operation of the GS algorithm, we define and illustrate some constructs that we will use later in the description of our DGSIMN algorithm.

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 of variable X , contained in \mathbf{M} . The set \mathbf{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 \mathbf{M} . Note here that the GS algorithm does not require the use of any specific ordering that the variables are examined for inclusion in the grow phase or exclusion in the shrink phase—*any* such ordering will produce the correct Markov blanket \mathbf{M}_X . 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 \mathbf{M} may contain variables that are not in the true Markov blanket. This justifies the shrink phase, which removes each false member Y from \mathbf{M} if and only if Y is conditionally independent of X given $\mathbf{M} - \{Y\}$. 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 \mathbf{M} .

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 \mathbf{M} (line 4) and not removed from \mathbf{M} during the shrink phase. I_X contains the list of variables found independent of X (again in order) and thus not added to \mathbf{M} during the grow phase, or added but later removed from \mathbf{M} during the shrink phase (line 7).

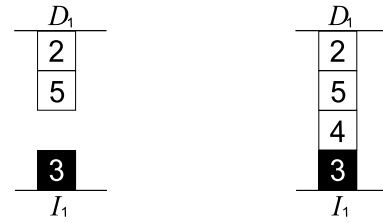
Let us illustrate these two lists by an example.

EXAMPLE 1. *Let us examine the execution of the algorithm on the example Markov network of Fig. 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*

$$\begin{aligned} & (1 \not\perp 2 \mid \emptyset) \\ & (1 \not\perp 5 \mid \{2\}) \\ & (1 \perp 3 \mid \{2, 5\}). \end{aligned}$$

After these tests, the values of D_1 and I_1 are $D_1 = [2, 5]$ and $I_1 = [3]$. The lists D_1 and I_1 can be represented

graphically as two columns growing in opposite directions, as shown in Fig. 2(a). In this figure, D_X is shown consisting of white squares (each corresponding to a variable) and growing downwards while I_X consists of dark squares and grows upwards.



(a) After tests $(1 \not\perp 2 \mid \emptyset)$, $(1 \not\perp 5 \mid \{2\})$ and $(1 \perp 3 \mid \{2, 5\})$. (b) After the additional test $(1 \not\perp 4 \mid \{2, 5\})$.

Figure 2: Graphical representation of the grow phase state (D_1, I_1) of variable 1, corresponding to Example 1. The dependence column D_1 is shown in white and the independence column I_1 in black.

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 **false**, indicating dependence. In this case D_1 will be extended by variable 4, resulting in the state shown in Fig. 2(b).

We see that during the operation of the GS algorithm, each successive test is of the form $(X, Y \mid D_X)$. Note that I_X does not affect future tests directly, but indirectly it is used to remember which tests are unnecessary. At each stage of the algorithm, we can then 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

$$F(D_X) = \left\{ \text{test } (X, Y \mid D_X) \mid Y \neq X \text{ and } Y \notin D_X \right\}.$$

In the example of Fig. 2(a), the fringe is $F_1 = \{(1, 4, \{2, 5\})\}$.

4 The DGSIMN Algorithm

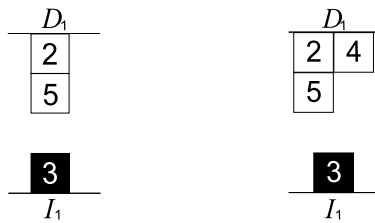
In this section we present our main contribution, the DGSIMN algorithm (Dynamic Grow-Shrink Inference-based Markov Network learning algorithm). We first extend and generalize the GS algorithm in a way that makes it more flexible in the choice of tests that it is allowed to do during its growing phase. Following that, we describe the DGSIMN algorithm, which uses this extended GS algorithm but also introduces the dynamic selection of the best next test to perform with respect to its expected future cost.

4.1 A Generalization of the GS Algorithm. 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. (As we will see in the next section, the source of these test results will be a process of inference using the axioms of Eqs. (3.1).)

Suppose that a new test result $(X \not\perp\!\!\!\perp Y \mid \mathbf{S})$ becomes available to the algorithm during the grow phase of X , with $\mathbf{S} \neq D_X$. If $\mathbf{S} \supseteq D_X$, then $(X \not\perp\!\!\!\perp Y \mid D_X)$ follows by Strong Union and we can therefore extend the D_X column with Y . If $\mathbf{S} \not\supseteq D_X$ the test cannot be used by the GS algorithm. However, our key idea here is that one can actually 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 (it returns the correct Markov blanket for any order), 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 \mathbf{D}_X .

Let us clarify this idea through an example.

EXAMPLE 2. *Let us revisit the situation described in Example 1(a), reproduced here for convenience in Fig. 3(a). In this example the state for variable 1 consists of a single column i.e., $\mathbf{D}_1 = \{\{2, 5\}\}$, $I_1 = [3]$. Suppose now that we learn that $(1 \not\perp\!\!\!\perp 4 \mid 6)$. Since $\{6\} \not\supseteq D_1$, this test cannot be used to grow the existing dependence column. However, through Strong Union, we can infer that $(1 \perp\!\!\!\perp 4 \mid \emptyset)$. 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., $\mathbf{D}_1 = \{\{2, 5\}, [4]\}$. The new state of variable X is shown in Fig. 3(b).*



(a) After tests $(1 \not\perp\!\!\!\perp 2 \mid \emptyset)$, $(1 \not\perp\!\!\!\perp 5 \mid \{2\})$ and $(1 \perp\!\!\!\perp 3 \mid \{2, 5\})$. (b) After the additional test $(1 \perp\!\!\!\perp 4 \mid 6)$.

Figure 3: Graphical representation of the grow phase state (D_1, I_1) of variable 1, corresponding to Example 2.

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 \mathbf{D}_X , the grow phase terminates if and only if $D_X^* \cup I_X = \mathbf{V} - \{X\}$. 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 .

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. The new fringe of X , denoted \mathbf{F}_X , is now defined for a set of columns \mathbf{D}_X as (4.3)

$$\mathbf{F}_X = \begin{cases} \emptyset & \text{if } D_X^* \cup I_X = \mathbf{V} - \{X\} \\ \bigcup_{D_X \in \mathbf{D}_X} F(D_X) & \text{otherwise.} \end{cases}$$

Algorithm 2 *updateColumns* $((X, Y, \mathbf{S}), t)$

```

1: if  $t = \text{false}$  then
2:   if  $\exists D \in \mathbf{D}_X$  such that  $\mathbf{S} \supseteq D$  then
3:      $D \leftarrow \text{concatenate}(D, Y)$ 
4:   else
5:      $\mathbf{D}_X \leftarrow \mathbf{D}_X \cup \{[Y]\}$ 
6:   Update the fringe  $\mathbf{F}_X$  of  $X$  using Eq. (4.3).
7: else
8:    $I_X \leftarrow \text{concatenate}(I_X, Y)$ 
9:   for each  $D \in \mathbf{D}_X$  do
10:    remove  $Y$  from  $D$ 

```

Alg. 2 summarizes these ideas, showing precisely how \mathbf{D}_X , I_X and \mathbf{F}_X are updated after test $(X, Y \mid \mathbf{S})$ evaluates to t , where $t \in \{\text{true}, \text{false}\}$. Given as input (X, Y, \mathbf{S}) and t , the *updateColumns* procedure tries to advance the grow phase of the GS algorithm of variable X one step. If $t = \text{false}$ (dependence) and there exists some element $D \in \mathbf{D}_X$ for which $\mathbf{S} \supseteq D$, it extends D by appending Y to its end; otherwise it creates a new column containing only Y i.e., $[Y]$ is added to \mathbf{D}_X . If $t = \text{true}$ (independence) then Y is appended to I_X and removed from every column that contains it.

4.2 DGSIMN Algorithm Description. We are now ready to describe the DGSIMN algorithm, shown in Alg. 3. 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 the output network.

The algorithm maintains a propositional knowledge base K that contains the independence and dependence propositions that have either been evaluated on data or inferred (see below). During the initialization phase (lines 1–6), the algorithm creates K initially empty. It also initializes the set of dependence columns for

Algorithm 3 DGSMN(\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:    $\mathbf{M}_X \leftarrow nil$ 
7:
8: /* Main loop. */
9: while  $\mathbf{F} \neq \emptyset$  do
10:  /* Select test with maximum utility (cf. §4.3). */
11:   $(X, Y, \mathbf{S}) \leftarrow \arg \max_{T \in \mathbf{F}} U(T)$ 
12:  /* Do the test. */
13:   $t \leftarrow I(X, Y, \mathbf{S})$ 
14:   $applyTest((X, Y, \mathbf{S}), t)$ 
15:
16:  /* Shrink complete columns not yet shrunk. */
17:  for each  $Z \in \mathbf{V}$  s.t.  $\mathbf{F}_Z = \emptyset$  and  $\mathbf{M}_Z = nil$  do
18:    /* Shrink longest column  $D_Z^*$ . */
19:    for each  $W \in D_Z^*$  do
20:      if  $I(Z, W, D_Z^* - \{W\}) = \mathbf{true}$  then
21:         $applyTest((Z, W, D_Z^* - \{W\}), \mathbf{true})$ 
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:  $G \leftarrow$  graph with nodes  $\mathbf{V}$  and set of edges  $\mathbf{E}$ 
30: return  $G$ 
```

each variable to a single empty column and sets its independence column to be empty. Finally, it initializes its Markov blanket to the special value *nil* (indicating that its Markov blanket has not been created yet).

The Markov blanket of each variable in \mathbf{V} is learned in the main loop (lines 9–22) by a simultaneous application of the generalized GS algorithm that maintains multiple columns (see Section 4.1 above). An important component of the algorithm is the method that it employs to select the next test to perform: the next test is chosen dynamically during each iteration of the main loop, selected greedily from the pool of useful tests contained in the *global fringe* \mathbf{F} (or simply *fringe*). The fringe is defined as

$$\mathbf{F} = \bigcup_{X \in \mathbf{V}} \mathbf{F}_X.$$

The algorithm selects the test in \mathbf{F} that is expected to produce the largest number of *useful* inferences i.e., those that can be used to extend one or more columns.

Whether a test is useful or not—or, more precisely, how useful it is—is computed using a *utility* function, discussed in detail in Section 4.3 below.

Algorithm 4 $applyTest((X, Y, \mathbf{S}), t)$

```
1:  $updateColumns((X, Y, \mathbf{S}), t)$ 
2:  $updateColumns((Y, X, \mathbf{S}), t)$  /* For symmetry. */
3: /* Do inference using new test. */
4:  $K \leftarrow$  run forward chaining on  $K \cup \{(X, Y, \mathbf{S}) = t\}$ 
5: for every newly inferred test  $t' = (X', Y', \mathbf{S}')$  do
6:    $updateColumns((X', Y', \mathbf{S}'), t')$ 
7:    $updateColumns((Y', X', \mathbf{S}'), t')$ 
```

The optimal test $T^* = (X, Y, \mathbf{S})$ selected is then performed on data using the boolean function I (cf. Eq. (3.2)). Its outcome (**true** or **false**) is then *applied* to the state of the algorithm using the *applyTest* procedure of Alg. 4, an operation that consists of three steps: First, the new test is used to update the columns of X and Y using Alg. 2, 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 and a *forward-chaining inference procedure*¹ [20] is applied on $K \cup \{T^*\}$, updating it to contain all conducted and inferred independences inferable using the Triangle rules (cf. Theorem 3.2). Finally, each of the newly inferred tests (X', Y', \mathbf{S}') is used to update the columns of X' and Y' using Alg. 2.

After applying the optimal test T^* , the column of one or more variables may have been completed, and thus that variable may proceed to its shrink phase. Let denote by Z such a variable (if any) and D_Z^* its longest (completed) column. The algorithm detects if a column of variable Z is complete by checking if its fringe is empty i.e., $\mathbf{F}_Z = \emptyset$ (see Eq. (4.3)). However, some variables may have already completed its shrinking phase and thus also have an empty fringe. To avoid attempting to execute the shrinking phase for these variables again, the algorithm confirms that its Markov blanket \mathbf{M}_Z has not been initialized i.e., checks that $\mathbf{M}_Z = nil$. As in the growing phase, every test performed on data during the shrink phase (line 20) is applied to the global state using Alg. 4 (line 21). The shrink phase of a variable Z terminates by setting its

¹The forward-chaining procedure is a standard inference algorithm that computes the *closure* of a set of facts and if-then rules. It works by iteratively applying each rule, instantiated to every possible antecedent formed by the current facts in the knowledge base, and adding each inferred new fact back into the knowledge base. This procedure repeats until all newly inferred facts are already contained in the knowledge base.

Markov blanket \mathbf{M}_Z to D_Z^* , which contains at this point exactly those variables in the Markov blanket.

The main loop proceeds until the grow and shrink phases of all variables has been completed, a condition satisfied when the fringe of every 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 3.1.

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. As mentioned above, 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 with respect to a column is therefore proportional to the (negative of its) distance to termination i.e., the negative of its cost, as measured by the weighted number of tests required to complete the column. Its utility with respect to a variable is the negative of the cost required to complete exactly one column of that variable.

Unfortunately, exact calculation of this utility (cost) is impossible in the general case, as that would require prior knowledge of the result of future tests. Instead we calculate the *expected* cost of each column. (This is consistent with the principle of selecting an action of maximum expected utility, a standard approach in decision theory for domains that contain uncertainty [20].) To calculate the expected cost of tests remaining for the grow phase of a column of a variable we need to average over all possible events, weighing each by its probability of occurrence. Lacking prior knowledge of the results of future tests, we use the *principle of indifference* according to which, in the absence of any information, the events considered (i.e., the next test outcome being **true** or **false**, corresponding to independence or dependence respectively) are equiprobable. 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, which are presented in the next section.

Let (D_X, I_X) be the state of a column of some variable X still in its grow phase during the execution of the algorithm, with D_X and I_X its dependence and independence lists respectively, and let $d = |D_X|$ and $i = |I_X|$. Then that column is $m = n - (d + i + 1)$ tests

away from completion of its grow phase, where $n = |\mathbf{V}|$, because this is the number of variables remaining to be examined for inclusion to either D_X or I_X (depending on the result of the corresponding test). Using the principle of indifference mentioned above, the expected weighted cost $f(d, i)$ of the remaining tests required to complete column (D_X, I_X) is

$$(4.4) \quad f(d, i) = \frac{1}{2^m} \sum_{s \in \{\mathbf{true}, \mathbf{false}\}^m} w(s),$$

where $\{\mathbf{true}, \mathbf{false}\}^m$ denotes the set of all sequences of length m containing the values **true** or **false**, and $w(s)$ denotes the cost of test sequence s , which can be regarded as a binary vector of length m .

Note that even though the calculation of the value of $f(d, i)$ appears to require exponential time in m , that is not true—it can be computed in closed form recursively as follows: Each test $(X, Y \mid D_X)$ that returns dependence grows the D_X column, thus determining (enlarging by one) the conditioning set of subsequent tests during the grow phase, while independence results grow the I_X column which does not affect their conditioning set (and therefore their cost). This logic allows the calculation of $f(d, i)$ as the solution of the following recurrence:

$$f(d, i) = \begin{cases} d + 2 & \text{if } d + i + 2 = n \\ \frac{1}{2}f(d + 1, i) + \frac{1}{2}f(d, i + 1) & \text{otherwise.} \end{cases}$$

The first case of the recurrence corresponds to the base case where there is only one variable remaining to be tested in the growing phase. The second case corresponds to the general situation and is the average of two alternatives for the outcome of the next test to be done: if the next test returns dependence (with assumed probability $1/2$), then D_X will be enlarged by Y and the expected cost of subsequent tests will be $f(d + 1, i)$; otherwise Y will be added to I_X and the expected cost of subsequent tests will be $f(d, i + 1)$. The solution to the recurrence is

$$f(d, i) = d + 2 + \frac{n - (d + i + 2)}{2}.$$

Finally, the total distance of the algorithm from termination is the sum of the distances for each variable in the domain:

$$f = \sum_{X \in \mathbf{V}} f(|D_X^*|, |I_X|)$$

where D_X^* is X 's longest column.

We can now define the utility $U(T)$ of some test T to be done (i.e., $T = T^*$) as the average distance

to the goal of the resulting state of the algorithm (the states of \mathbf{D}_X and I_X of every variable X in the domain after updating the knowledge base with the new test result, conducting forward chaining, and updating the columns accordingly using Alg. 4) if its value on data is **true**, denoted by $f_{T=\text{true}}$ or the state if its value is **false**, denoted by $f_{T=\text{false}}$. The utility of test T is then

$$U(T) = -\frac{f_{T=\text{true}} + f_{T=\text{false}}}{2}.$$

4.4 Correctness of DGSIMN. As proved in [16], the GS algorithm correctly recovers the Markov blanket of a variable for an arbitrary ordering of examination of the variables during the grow phase.

THEOREM 4.1. ([16]) *Assuming Contraction, Composition and Decomposition, at the end of the GS algorithm \mathbf{M} is a Markov boundary of X .*

(A Markov boundary is a minimal Markov blanket. Also, we do not show Contraction here due to lack of space but it is guaranteed to hold in every probability distribution—see [19].) The proof of theoretical correctness of DGSIMN (under our stated assumptions) therefore follows directly from the correctness of GS, as DGSIMN faithfully follows the execution of the GS algorithm (its grow and shrink phases), albeit being significantly more efficient by inferring the results of some of the tests involved and maintaining several alternative grow phases in parallel.

5 Experimental Results

We conducted experiments on both artificial and benchmark data sets. As we 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 explained in section 4.3, the running time of the algorithm in the limit of very expensive tests, such as scenarios with large data sets or distributed data, is proportional to the total weighted cost of all tests conducted.
- **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 are either present in both networks or absent from both networks.

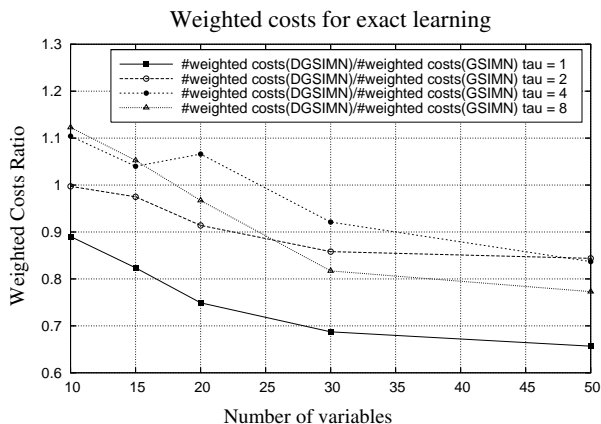


Figure 4: Ratio of the weighted cost DGSIMN over GSIMN for different network sizes (number of nodes) n and average degrees $\tau = 1, 2, 4,$ and 8 . Ratios smaller than 1 indicate an advantage of DGSIMN over GSIMN.

- **Accuracy:** For benchmark 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 resulting network (using vertex separation) with those represented in the data (using statistical independence tests).

5.1 Exact Learning Experiments. 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 size (number of nodes) and average degree 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 τ that equals the average number of direct neighbors per node. Given τ , edges were selected uniformly by selecting the first $\tau \frac{n}{2}$ 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 depicts the ratio of weighted cost of

DGSIMN over GSIMN for true networks of size up to 50 and average degree $\tau = 1, 2, 4, 8$. We can observe that for $\tau = 1, 2$, DGSIMN always performs better than GSIMN in terms of weighted cost. Also, while for $\tau = 4, 8$, and small n , GSIMN performs better, as the number of variables increases DGSIMN significantly outperforms GSIMN—for data sets with 50 variables we see a savings of 15%–35% in weighted cost.

5.2 Sampled Data Experiments. 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 χ^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 degree of $\tau = 1, 2, 4, 8$, whose structure was generated randomly using the procedure described in Section 5.1. Given a structure, to fully describe a probability distribution we must also specify its parameters. These parameters determine the strength of the dependencies among variables connected in the graph. Agresti [2] proposes the *log-odds ratio* θ_{XY} as a measure of the strength between two 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 used $\theta = 1$ for each of these pairs.

We depict the ratio of the weighted cost and the ratio of the number of correct edges of DGSIMN versus GSIMN in Fig. 5 for each τ . Each graph contains two plots, a histogram representing the difference in accuracy between DGSIMN and GSIMN and a line plot showing the ratio of weighted costs. The relative behavior of the two algorithms in terms of weighted costs is similar to the case of exact learning. We can see that overall DGSIMN improves over GSIMN in weighted cost without sacrificing accuracy by a great amount in many cases, and actually improving accuracy in most cases. For less connected networks ($\tau = 1, 2$), DGSIMN always requires less weighted cost. For $\tau = 4, 8$, DGSIMN starts outperforming GSIMN as the number of variables in the domain increases. Most importantly,

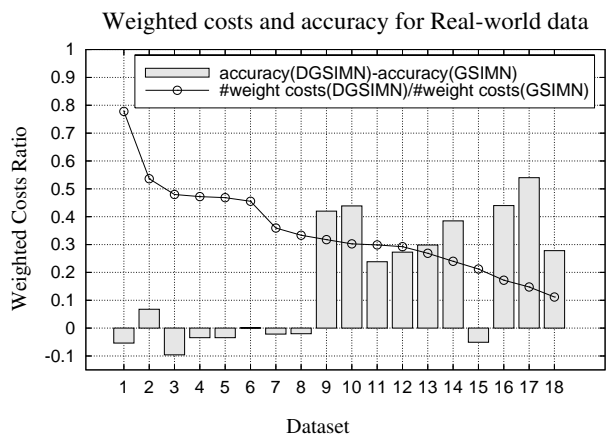


Figure 6: Ratio of the weighted cost of DGSIMN over GSIMN and difference between the accuracy of DGSIMN and GSIMN on real data sets. Ratios smaller than 1 and positive bars indicate an advantage of DGSIMN over GSIMN. The numbers in the x -axis are indices of the data sets as shown in Table 2.

DGSIMN has the most impact (improvement) with large numbers of variables, which is frequently the most interesting and useful case in which to apply it.

5.3 Benchmark Data Experiments. We also conducted a number of experiments on benchmark data sets, shown in Table 2, from the UCI machine learning data repository [10]. Experiments on benchmark data are useful for providing us a more realistic 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 low due to the usually small data set size; and the structure of the underlying model may have a non-random topology.

Since the structure of the underlying network is unknown for virtually all benchmark 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 χ^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 \mathcal{T} composed of 100 randomly sampled triplets (X, Y, \mathbf{S}) per conditioning set size $m = |\mathbf{S}|$, for $m \in \{0, \dots, n - 2\}$. Denoting by $I_{out}(t) \in \{\mathbf{true}, \mathbf{false}\}$ the result of the test performed on the output network for triplet $t \in \mathcal{T}$ and by $I_{data}(t) \in \{\mathbf{true}, \mathbf{false}\}$ the result of the test performed on data, the accuracy is defined

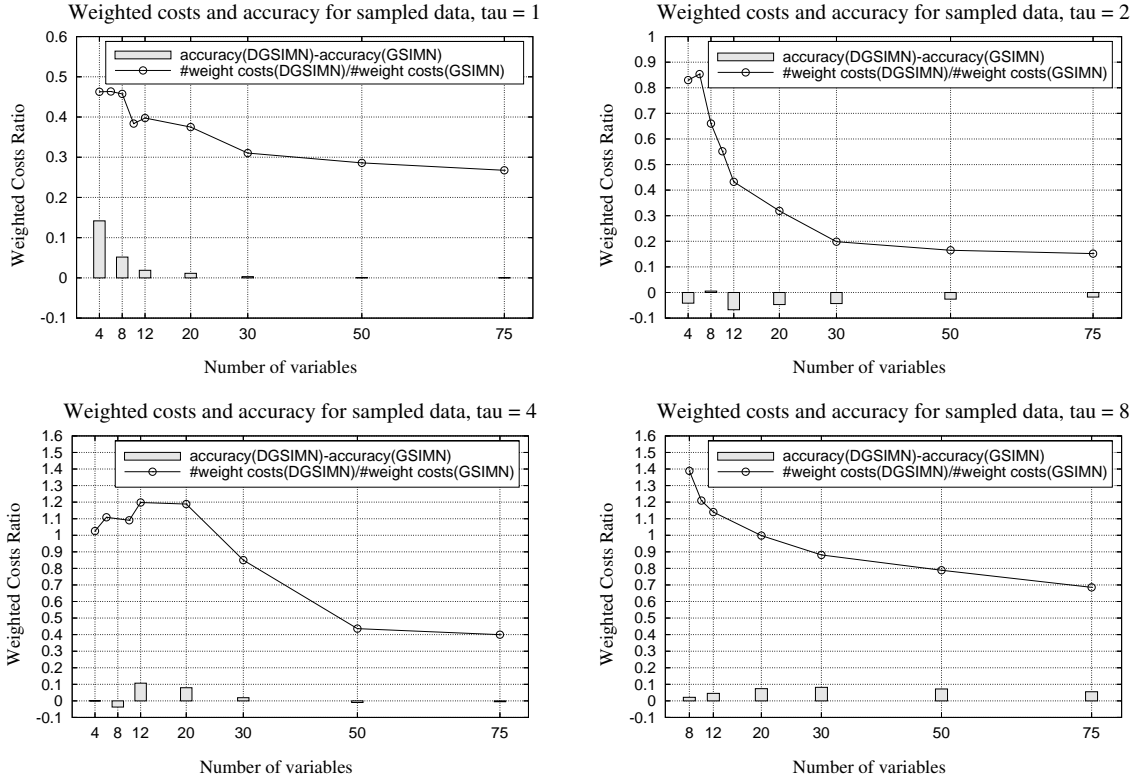


Figure 5: Ratio of weighted cost of DGSIMN over GSIMN (continuous line) and the difference of the ratio of correct edges of DGSIMN and GSIMN (bar graph) for different number of variables n and average degrees $\tau = 1, 2, 4,$ and 8 . Ratios smaller than 1 and positive bars indicate an advantage of DGSIMN over GSIMN.

as

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

Table 2 displays the comparison between DGSIMN and GSIMN on a set of benchmark data sets and in Figure 6 we plot the difference in accuracy between DGSIMN and GSIMN as bar graph and the ratio of weighted cost for each file in Table 2. From the table and the graph we can see that DGSIMN clearly improves on weighted cost for every data set, producing savings of up to 88%. Results also show that DGSIMN performs slightly worse in 8 out of 18 cases, similar in 2 cases, and much better the remaining 8 cases. Overall for benchmark data sets DGSIMN improves by a great amount in weighted tests while achieving similar or better accuracy on average.

6 Conclusion

In this paper we presented the DGSIMN algorithm that improves on GSIMN, the state-of-the-art 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 88% on both sampled and benchmark data while achieving similar or better accuracy in most cases.

References

- [1] P. Abbeel, D. Koller, and A. Y. Ng. Learning factor graphs in polynomial time and sample complexity. *Journal of Machine Learning Research*, 7:1743–1788, 2006.

Table 2: Weighted number of tests and accuracy for several benchmark 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 .

Data set				Weighted cost			Accuracy	
Index	Name	n	N	GSIMN	DGSIMN	Improvement	GSIMN	DGSIMN
1	cmc	10	1473	261	203	22.2%	0.693	0.64
2	flare2	13	1400	386	207	46.4%	0.633	0.7
3	bridges	12	70	294	141	52.0%	0.882	0.786
4	hepatitis	20	80	845	399	52.8%	0.934	0.899
5	hayes-roth	6	132	79	37	53.2%	0.7	0.666
6	crx	16	653	685	312	54.5%	0.757	0.758
7	flag	29	194	2482	892	64.1%	0.901	0.879
8	monks-1	7	556	96	32	66.7%	0.823	0.803
9	haberman	5	306	63	20	68.3%	0.407	0.827
10	echocardiogram	14	61	562	170	69.8%	0.467	0.905
11	lenses	5	24	67	20	70.1%	0.502	0.74
12	imports-85	25	193	2912	851	70.8%	0.501	0.774
13	balance-scale	5	625	67	18	73.1%	0.332	0.63
14	baloons	5	20	75	18	76.0%	0.338	0.723
15	tic-tac-toe	10	958	231	49	78.8%	0.742	0.691
16	car	7	1728	186	32	82.8%	0.287	0.721
17	nursery	9	12960	393	58	85.2%	0.229	0.769
18	dermatology	35	358	7238	808	88.8%	0.597	0.875

- [2] A. Agresti. *Categorical Data Analysis*. Wiley, 2nd edition.
- [3] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng. Discriminative learning of Markov random fields for segmentation of 3D range data. 2005.
- [4] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A*, 15(10):3241–3253, 1982.
- [5] J. Besag. Spacial interaction and the statistical analysis of lattice systems. *Journal of Royal Statistical Society, Series B*, 1974.
- [6] J. Besag, J. York, and A. Mollie. Bayesian image restoration with two applications in spatial statistics. *Annals of the Institute of Statistical Mathematics*, 43:1–59, 1991.
- [7] F. Bromberg, D. Margaritis, and V. Honavar. Efficient markov network structure discovery using independence tests. In *Proceedings of SIAM Conf on Data Mining (SDM)*, 2006.
- [8] W. L. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.
- [9] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on PAMI*, 19(4):390–393, 1997.
- [10] C. B. D.J. Newman, S. Hettich and C. Merz. UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Science, 1998.
- [11] W. T. Freeman and E. Pasztor. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.
- [12] N. Friedman and M. Linial. Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7(3–4):601–620, August 2000.
- [13] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian relation of images. *IEEE Transactions on PAMI*, 6:721–741, 1984.
- [14] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 1995.
- [15] R. Hoffman and V. Tresp. Nonlinear markov networks for continuous variables. volume 10, pages 521–529, 1998.
- [16] D. Margaritis. On theoretically optimal variable selection. *Submitted to the Journal of Machine Learning Research*, 2007.
- [17] D. Margaritis and S. Thrun. Bayesian network induction via local neighbourhoods. volume 12, pages 505–511. MIT Press, 2000.
- [18] A. McCallum. Efficiently inducing features of conditional random fields. *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, 2003.
- [19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [20] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.
- [21] S. Shekhar, P. Zhang, Y. Huang, and R. R. Vatsavai. *Trends in Spatial Data Mining*, chapter 19, pages 357–379. AAAI Press / The MIT Press, 2004.
- [22] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. Adaptive Computation and Machine Learning Series. MIT Press, 2nd edition, 2000.