2007

Markov network structure discovery using independence tests

Facundo Bromberg
Iowa State University

Markov network structure discovery using independence tests

by

Facundo Bromberg

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Dimitris Margaritis, Major Professor
Alicia Carriquiry
Vasant Honavar
Giora Slutzki
Leigh Tesfatsion

Iowa State University

Ames, Iowa

2007

UMI Number: 3289377

UMI Microform 3289377

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

DEDICATION

To Nati, a great soul full of love and strength, for sharing these qualities with me every single day during all these years.

(*A Nati, una gran alma llena de amor y fuerza por compartir estas cualidades conmigo cada día durante todos estos años.*)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This thesis and the completion of my Ph.D. would have never been possible without the daily guidance of Dr. Dimitris Margaritis. I was at the verge of pursuing other horizons when I started working with him. With generous patience and almost daily guidance he helped me master the craft of scientific research and technical writing.

I want to also thank my committee members Drs. Alicia Carriquiry, Vasant Honavar, Giora Slutzki, and Leigh Tesfatsion. During my years at Iowa State University they influenced me in many ways other than being part of my committee. A special thanks to Dr. Honavar for giving me advice and support early in my career.

I want to show my gratitude to the Department of Computer Science and Iowa State University for giving me the opportunity to fulfill my academic goals, and for providing me continuous economical support during all these years.

During my teaching experience in the Department I had the privilege to work closely with excellent professors. Special thanks to Dr. Gary Leavens for being the first, with all the support and encouragement such experience requires, Drs. Honavar and Slutzki for giving me the opportunity to assist them during the teaching of graduate level courses, and Dr. Markus Lumpe for trusting me the design and conduction of important parts of his course, but most importantly for being a good friend.

Thanks to Melanie Eckard and Linda Dutton, graduate secretaries of the Computer Science Department, and Virginia McCallum, international students and scholars' advisor for helping me to deal with the burocratic Labyrinth of the university and government systems. And thanks to so many other people in the Department: Brian Patterson, Giorgi Ushakov, David Fernandez-Baca, Flavian Vasile, Oksana Yakhnenko, Jack Lutz.

The academic endeavor is full of ups and downs. In the down moments my very cherished friends, Adrian Silvescu, and Dr. Doina Caragea supported me with incredibly helpful and diverse advises that only an experienced Ph.D student but most importantly great human beings can give. Every interaction with them was bluntly honest and insightful.

Somebody told me once that a Ph.D. is not enough. It is just one part of the greatest game of life (I say). So I want to thank all the souls that made my days in Ames a great life, and for giving me a friendship for many years to come: Beatriz Olleta, Laura (Lali) Aguiar, Krisztina Eleki, Massiel Orellana, Valentín, Lucía, Santi and Mati Picasso, los gallegos" Fer and Rut Anton, Charlie and Joanna Courteau, Paula, Alex, Chiara and Jacob Travesset, Gina, Adi, and Ranvir Singh, and Giorgi (Giorgicus) Chighladze. Thanks also to my old friends Raúl and Guada Bonadé, Nehuen and Sole Campitelly, Irma Saffe, Mariana Caram, and to all 11milamaneceres friends, for being so close from so far away. Each one of them deserves a chapter by themselves, but not in a thesis, their friendship does not need any proof.

Finally a thanks and a promise to my family. Thanks for decades of support to this curious kid that wanted to study the atoms, the stars, the body, the brain and the soul of this universe, carrying many of them with me in this endeavor or leaving many far away. My profound gratitude to each of them, and a promise that my spatial journey is finally over. From now on my exploration will continue close to you. Let poetry help me explain myself:

We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time

T.S. Elliot

And finally to those that are beyond acknowledgement. A warmth an profound gratitude to my soulfriends, my wife Natalia and my daughter Zoe.

ABSTRACT

We investigate efficient algorithms for learning the structure of a Markov network from data using the independence-based approach. Such algorithms conduct a series of conditional independence tests on data, successively restricting the set of possible structures until there is only a single structure consistent with the outcomes of the conditional independence tests executed (if possible). As Pearl has shown, the instances of the conditional independence relation in any domain are theoretically interdependent, made explicit in his well-known conditional independence axioms. The first couple of algorithms we discuss, GSMN and GSIMN, exploit Pearl's independence axioms to reduce the number of tests required to learn a Markov network. This is useful in domains where independence tests are expensive, such as cases of very large data sets or distributed data. Subsequently, we explore how these axioms can be exploited to "correct" the outcome of unreliable statistical independence tests, such as in applications where little data is available. We show how the problem of incorrect tests can be mapped to inference in inconsistent knowledge bases, a problem studied extensively in the field of non-monotonic logic. We present an algorithm for inferring independence values based on a sub-class of non-monotonic logics: the argumentation framework. Our results show the advantage of using our approach in the learning of structures, with improvements in the accuracy of learned networks of up to 20%. As an alternative to logic-based interdependence among independence tests, we also explore probabilistic interdependence. Our algorithm, called PFMN, takes a Bayesian particle filtering approach, using a population of Markov network structures to maintain the posterior probability distribution over them given the outcomes of the tests performed. The result is an approximate algorithm (due to the use of particle filtering) that is useful in domains where independence tests are expensive.

CHAPTER 1. Introductory Remarks

In this thesis we discuss a series of contributions to the problem of learning graphical models from data, which are a special type of probabilistic model. A probabilistic model is a tool for reasoning under uncertainty that makes possible the calculation of the probability of any well-formed propositional sentence through a process called *probabilistic inference*. The propositions considered usually take the form of an assignment of values to variables, where a variable is some attribute or property of the world that can take one out of many possible values in a domain. Examples of variables are the side of a coin facing upwards, the letter grade of a student, the crop yield of a corn field, with corresponding domains {head, tails}, {A,B,C,D,F} and [0.1, 50] in kg, respectively. An important example of a probabilistic model is a *tabular probabilistic model*. A tabular model is a function, represented as a table, that assigns a probability to every possible joint event (a joint event is an assignment to every variable) such that the sum of the probabilities adds up to 1.

As exemplified in the following series of examples, the tabular model presents unsurmountable storage and computational difficulties. Graphical models propose an efficient alternative by exploiting the conditional independences that exists among the random variables in the domain. It has been shown that these conditional independencies can be represented graphically, giving the name to this family of models. (See Section 1.2.2 for a more detail discussion of these ideas.) In this thesis we present a series of contributions to the problem of learning the conditional independence structure (i.e., the graph) of a graphical model from data.

Example 1.1. *We start with a simple example of a probabilistic model for the problem of modeling the crop yield (X) in an agricultural study. We assume the modeller has at his or her disposal (usually noisy) observations of quantities such as: soil acidity (Y_1), soil humidity*

Table 1.1 Tabular probabilistic model consisting on five binary random variables: X, Y_1, Y_2, Y_3, Y_4. The left column shows the assignment of values for each of the variables as a string of bits and the left column shows the probability associated to the corresponding assignment.

#	x, y_1, y_2, y_3, y_4	$\Pr(X = x, Y_1 = y_1, Y_2 = y_2, Y_3 = y_3, Y_4 = y_4)$
1	*00000*	*0.0483*
2	*00001*	*0.0551*
3	*00010*	*0.0531*
4	*00011*	*0.0331*
5	*00100*	*0.0602*
6	*00101*	*0.0424*
7	*00110*	*0.0626*
8	*00111*	*0.0159*
9	*01000*	*0.0063*
10	*01001*	*0.0293*
11	*01010*	*0.0465*
12	*01011*	*0.0095*
13	*01100*	*0.0321*
14	*01101*	*0.0354*
15	*01110*	*0.0069*
16	*01111*	*0.0374*
17	*10000*	*0.0113*
18	*10001*	*0.0320*
19	*10010*	*0.0068*
20	*10011*	*0.0202*
21	*10100*	*0.0083*
22	*10101*	*0.0409*
23	*10110*	*0.0219*
24	*10111*	*0.0288*
25	*11000*	*0.0040*
26	*11001*	*0.0238*
27	*11010*	*0.0611*
28	*11011*	*0.0643*
29	*11100*	*0.0026*
30	*11101*	*0.0270*
31	*11110*	*0.0134*
32	*11111*	*0.0594*

(Y_2), concentration of Sodium (Y_3) and Potassium (Y_4). For simplicity, we assume the domain of these variables to be binary, with each variable taking either value high *or* low*, symbolized as* 1 *or* 0*, respectively. (Note this is a contrived example and may not necessarily correspond to current experiments in the agricultural sciences). The quantities $Y_1, \ldots, Y_4$ are commonly called observable variables, or simply observables, and quantity X, crop yield in the example, is commonly called a (hidden) target variable.*

We are frequently interested in the quantity

$$\Pr(X \mid Y_1 \ldots Y_4) = \frac{\Pr(X, Y_1, \ldots, Y_4)}{\Pr(Y_1, \ldots, Y_4)}, \tag{1.1}$$

in order to predict the value of the target variable given the knowledge of the observable variables. This is an example of probabilistic inference, where the corresponding boolean formulas are $X, Y_1, \ldots, Y_4$ *and* $Y_1, \ldots, Y_4$ *(with commas symbolizing logical conjunction). One can calculate this quantity using the tabular probabilistic model shown in Table 1.1. The numerator is obtained by simply looking up in the table of the tabular model the value of* $\Pr(X, Y_1, \ldots, Y_4)$. *To compute the denominator we apply the law of total probability, i.e.,*

$$\Pr(Y_1, \ldots, Y_4) = \sum_{x \in \{0,1\}} \Pr(X = x, Y_1, \ldots, Y_4)$$

where $X = x$ *denotes the assignment of value* x *to variable* X. *The sum is over all values* x *in the domain of* X. *Each of the terms in this summation can now be computed like the numerator by a simple look up in the table of the tabular model.*

Let us consider a concrete case to illustrate this calculation in which a farmer measured $Y_1 = 1$, $Y_2 = 0$, $Y_3 = 0$, $Y_4 = 1$, *that is, high soil acidity, low soil humidity, low Sodium concentration, and high Potassium concentration. Based on these measurements, the farmer is interested in predicting the probability of whether the crop yield will be high (*$X = 1$*) or low (*$X = 0$*). For that, it computes the conditional probability of the crop yield given the observables, i.e.,* $\Pr(X = 1 \mid Y_1 = 1, Y_2 = 0, Y_3 = 0, Y_4 = 1)$, *using Eq. (1.1). From Table 1.1, the numerator for the bit string* 11001 *(entry #26) equals* 0.0238. *The denominator equals the sum of the probabilities for* $X = 0$ *and* $X = 1$, *i.e., bit strings* 01001 *and* 11001 *corresponding to entries #10 and #26 in the table, and probabilities* 0.0293 *and* 0.0238, *respectively. We thus have that*

$$\Pr(X = 1 \mid Y_1 = 1, Y_2 = 0, Y_3 = 0, Y_4 = 1) = \frac{0.0238}{0.0293 + 0.0238} = 0.4482.$$

Since the conditional distribution is normalized, $\Pr(X = 0 \mid Y_1 = 1, Y_2 = 0, Y_3 = 0, Y_4 = 1) = 1 - \Pr(X = 1 \mid Y_1 = 1, Y_2 = 0, Y_3 = 0, Y_4 = 1) = 0.5518$. *From this result, the farmer concludes it is more likely that the crop yield will be low.*

Table 1.2 Tabular probabilistic model consisting on 64 groups of five binary random variables: $\{X_i, Y_{1,i}, Y_{2,i}, Y_{3,i}, Y_{4,i}\}_{i=1}^{64}$. The left column shows the assignment of values for each of the 320 variables as a string of bits and the left column shows the probability associated to the corresponding assignment.

#	$\{x_i, y_{1,i}, y_{2,i}, y_{3,i}, y_{4,i}\}_{i=1}^{64}$	$\Pr(\{X_i = x_i, Y_{1,i} = y_{1,i}, Y_{2,i} = y_{2,i}, Y_{3,i} = y_{3,i}, Y_{4,i} = y_{4,i}\}_{i=1}^{64})$
1	*000000...0000000*	*0.001483*
2	*000000...0000001*	*0.002551*
3	*000000...0000010*	*0.002151*
4	*000000...0000011*	*0.000331*
...	...	...
$2^{5\times64}-1$	*111111...1111110*	*0.003561*
$2^{5\times64}$	*111111...1111111*	*0.009741*

Unfortunately computing the probabilities of arbitrary boolean formulas can become extremely difficult when the number of variables is large. Let us illustrate this by extending the above example.

Example 1.2. *We extend Example 1.1 by considering a very common measurement setting in agricultural studies in which each of the above five variables, the target X, and the five observables $Y_1, \ldots, Y_4$, are measured in each location $i = 1, \ldots, m$, out of tens of locations in a field. These locations are usually obtained by sub-dividing a field into plots using a regular grid, with each square of the grid corresponding to a plot. A common case is an 8×8 grid, i.e., $m = 64$. As in the previous example, one is frequently interested in the most probable assignment of the target variables given some value for the observables (i.e., what is the most probable crop yield under certain conditions). This can be computed using the distribution of the target variables $\{X_1, \ldots, X_{64}\}$ conditioned on the observable variables, i.e.,*

$$\Pr(X_1, \ldots, X_{64} \mid \{Y_{1,i}, \ldots, Y_{4,i}\}_{i=1}^{64}) = \frac{\Pr(X_1 = x_1, \ldots, X_{64} = x_{64}, \{Y_{1,i}, \ldots, Y_{4,i}\}_{i=1}^{64})}{\Pr(\{Y_1, \ldots, Y_4\}_{i=1}^{64})}, \quad (1.2)$$

The numerator is computed by simply looking up in the table of the tabular probabilistic model shown partially in Table 1.2. Since each variable is binary, the number of entries in the complete model is $2^{(5\times64)}$. Unfortunately the storage of such a model is prohibitive in practice. To illustrate, consider recording it in paper. At 100 entries per page, we would require 1.07×10^{94} sheets of paper. To record it in a computer memory instead would require, at 4 bytes per entry, more than 2×10^{85} terabytes. To exacerbate this problem the time complexity

required to compute the denominator is also exponential. Let us compute it by applying the law of total probability:

$$\Pr(\{Y_1, \ldots, Y_4\}_{i=1}^{64})) = \sum_{\{x_1,\ldots,x_{64}\}} \Pr(\{X_i, Y_{1,i}, \ldots, Y_{4,i}\}_{i=1}^{64}))$$

where $\{x_1, \ldots, x_{64}\}$ *denotes the set of all possible assignments for the set of random variables* $\{X_1, \ldots, X_{64}\}$. *The terms in the summation can now be computed by simply looking up in the table of the tabular model. Assuming the look-up in the table takes a constant time of* 1 *nanosecond, the running time required to retrieve all* 2^{64} *entries would take 584 years.*

The example shows that for relatively simple domains the tabular probabilistic model, although simple, has prohibitive memory and time requirements. These difficulties can be overcome by the use of a graphical model. One such model is a Markov Random Field (MRF). Markov random fields are graphical probabilistic models that allow a more efficient solution of the problem of probabilistic inference. The storage and computational difficulties illustrated by Example 1.2 are a result of storing the probability for every joint event, i.e., the probability of every possible assignment of the complete set of variables. An important insight in the theory of MRF is the recognition that conditional independencies among the variables in the domain can greatly reduce the storage and computational requirements. We demonstrate the use of MRFs in the following two examples. We first demonstrate this by an extreme but simple case in which variables at different locations are non-interacting (independent). We then present an example that considers a sightly more realistic set of independencies. These examples introduce the concept of a MRF at an intuitive level. A formal definition of an MRF is given in detail below (c.f. Section 1.2.5).

Example 1.3. *Let us examine the crop yield problem under the extreme assumption that variables at different locations are non-interacting, more precisely, the crop yield* X_i *at location* i *is only affected by the soil acidity* $Y_{1,i}$, *soil humidity* $Y_{2,i}$, *concentration of Sodium* $Y_{3,i}$, *and concentration of Potassium* $Y_{4,i}$ *at location* i. *This is modelled probabilistically as conditional independence (c.f. Section 1.2.2), i.e.*

$$\Pr(X_i, Y_{1,i}, \ldots, Y_{4,i} \mid \{X_j, Y_{1,j}, \ldots, Y_{4,j}\}_{j\neq i}) = \Pr(X_i, Y_{1,i}, \ldots, Y_{4,i})$$

and thus the joint can be decomposed into a product,

$$\Pr(\{X_i, Y_{1,i}, \ldots, Y_{4,i}\}_{i=1}^{64}) = \prod_{i=1}^{64} \Pr(X_i, Y_{1,i}, \ldots, Y_{4,i}).$$

According to this decomposition we can compute the joint distribution as the product of the probability of each factor. In other words, we can model the joint distribution by a set of tabular models, one for each factor. Table 1.1 of Example 1.1 serves as an example of the tabular model of one of these factors. Therefore, since there are 64 factors, and the tabular model for each factor contains 32 *entries, the complete model is represented by only* $64 \times 32 = 2048$ *entries, an overwhelming reduction from the* 2.13×10^{96} *required for the fully dependent model.*

The above example presents overwhelming improvements in the storage requirements, showing the utility of exploiting the independences among the variables. Unfortunately, the assumption of non-interacting locations may be an oversimplification that does not reflect many dependencies that may exist in the underlying, true model.

The solution proposed by the theory of Markov random fields is to consider interactions only among nearest neighbors. For that, it defines a *neighborhood structure* $\mathcal{N}$ among the locations (i.e., $(i, j) \in \mathcal{N}$ if and only if location i is a neighbor of location j). In what follows we overload this notation denoting by $\mathcal{N}(i)$ the neighbors of location i in $\mathcal{N}$, and we may refer to the neighborhood structure $\mathcal{N}$ as simply the *structure* of the model. One example of a structure is a regular lattice, in which neighbors in the lattice correspond to neighbors in $\mathcal{N}$. A more general example of a neighborhood structure, but still constrained to spatial problems, can be defined through a radio $r \in \mathbb{R}$ and the euclidean distance $d(i, j)$ between locations i and j. We say i and j are neighbors, i.e., $(i, j) \in \mathcal{N}$, if and only if $d(i, j) \leq r$.

The structure is commonly provided by an expert, for instance, in the next example we illustrate the usage of the neighborhood structure in the crop yield problem by assuming $\mathcal{N}$ to be a regular lattice. Expert information may be inaccurate leading to incorrect structures. In this thesis we present several algorithms for automatically discovering the structure from data.

Fig. 1.1 shows the regular lattice considered for the crop yield problem. As exemplified by this figure, the neighborhood structure can be represented graphically by a *Markov network*, an

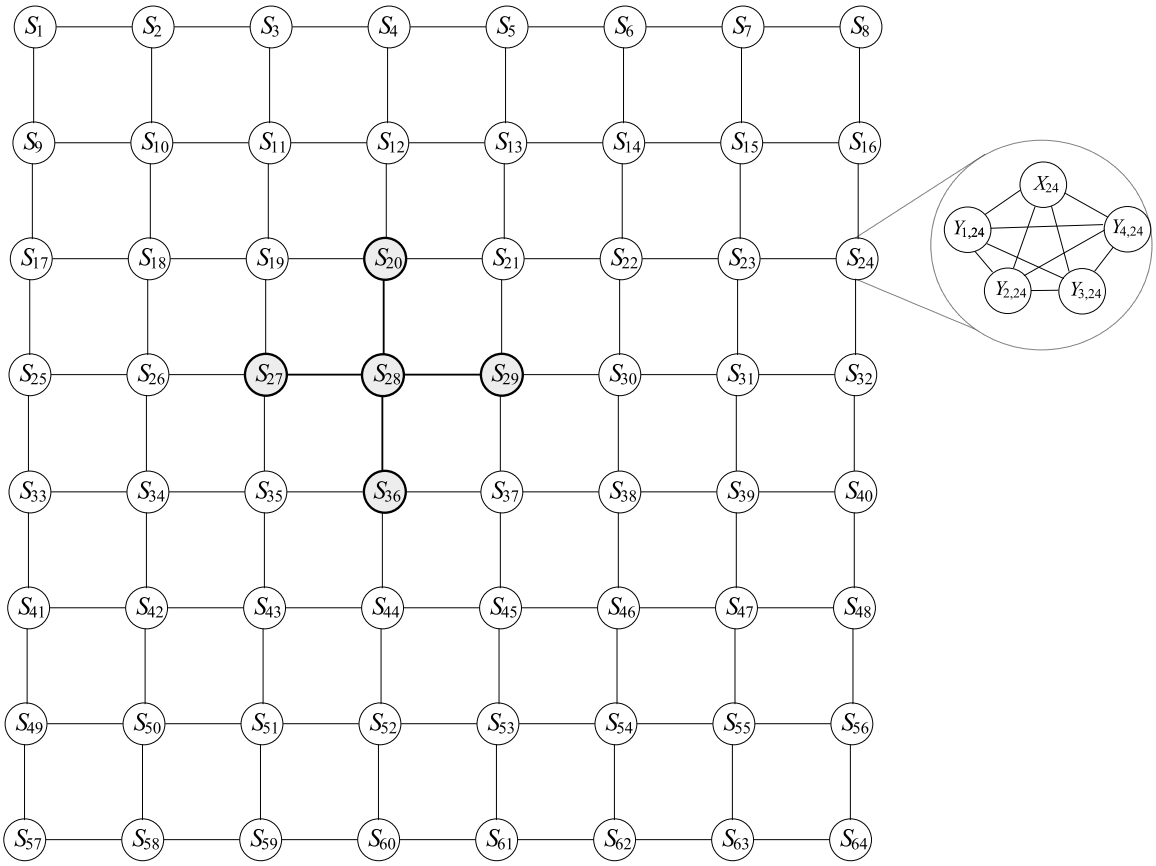


Figure 1.1 Example of a regular lattice Markov network with 64 locations. Each node $\mathbf{S}_i$ corresponds to the set of random variables $\{X_i, Y_{1,i}, Y_{2,i}, Y_{3,i}, Y_{4,i}\}$ at location i in an the crop field. The figure highlights the neighbors $20, 27, 29$, and 36 of location 28.

undirected graph whose nodes correspond to random variables, and there is an edge between any two nodes i and j such that $(i, j) \in \mathcal{N}$.

The nearest neighbors assumption proposed by the MRF formalism implies that a location i interacts with the whole system only through its neighbors. Probabilistically this is equivalent to say that location i is conditionally independent of all other locations when conditioned on its neighbors. If we denote by $\mathbf{S}_i$ the set of variables at location i, (e.g., $\mathbf{S}_i = \{X_i, Y_{1,i}, Y_{2,i}, Y_{3,i}, Y_{4,i}\}$ in the crop yield example), the nearest neighbor assumption is formalized by

$$\Pr(\mathbf{S}_i \mid \{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_{64}\} - \{\mathbf{S}_i\}) = \Pr(\mathbf{S}_i \mid \mathcal{N}(i)). \tag{1.3}$$

This is called the Markov property and gives its name to the theory of MRFs.

One important contribution of the theory of MRFs is the demonstration that under the Markov property, the joint distribution can be decomposed into a product of simpler factors, i.e.,

$$\Pr(\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n) \quad = \quad \frac{1}{Z} \prod_{i=1}^{n} f_i(\mathcal{N}(i)) \tag{1.4}$$

where n denotes the number of locations, and Z is a normalization constant historically called the *partition function*.

The above decomposition is similar to the decomposition obtained under the non-interacting assumption of the previous example, but in this case each factor is a slightly more complex function that depends on the neighbors of location i. Despite this extra complexity, the storage and computational requirements are also greatly simplified, allowing an overwhelming reduction in the number of entries in its tabular model when compared to the joint case of Example 1.2.

This is demonstrated in the following extension of Example 1.3.

Example 1.4. *Let us examine the crop yield problem by using Markov random fields as the probabilistic model instead of the tabular model used in the previous example. For this problem, we assume the neighborhood $\mathcal{N}$ to be a regular lattice, as shown in the Markov network of Fig. 1.1.*

Assuming the neighborhoods sizes are all the same i.e., $|\mathcal{N}(i)| = m$, $i = 1, \ldots, 64$, for some constant m, the above decomposition requires only $64 \times 2^{5.m}$ numbers to be stored (i.e., 2^{5m} entries per factor), resulting in an exponential reduction from the $2^{5\times 64}$ required for the joint case (of Example 1.2). For instance, for $m = 4$ the number of entries required is $2^{20} = 1.05 \times 10^6$, or 1.05 megabytes of storage (versus the 10^{85} terabytes of the joint case).

The above example assumes that both the structure of interactions and the parameters are known. The structure could be given by an expert (e.g., the regular lattice proposed for the crop yield examples), and the parameters (implicit in the functions f_i of Eq. (1.4)) are usually obtained experimentally. However, in many cases such information is not available. In

this work we present several efficient algorithms that automate the first of these tasks, namely constructing the structure of a Markov network from data. Learning the structure from data has the potential to produce structures that are closer to the structure of the underlying model, thus improving the model and in turn the applications that rely on inferences from it.

1.1 Related Work

As mentioned above, this thesis addresses the problem of learning the structure of graphical models from data. Graphical models can be divided into directed models or **Bayesian networks** and undirected models or **Markov networks**. A graphical model consists of two parts: a graph (the model structure), and a set of parameters.

The graph serves as a concise representation of the conditional independences that theoretically hold among the random variables in the domain. Bayesian and Markov networks differ in their representation power, i.e., in the type of independences their graphs can represent. From a more practical perspective, they also differ in the properties of their factor functions (see below). The factor functions of Markov networks are not normalized, requiring a costly (usually exponentially) computation of a normalization constant (commonly known as partition function) in order to obtain a fully quantified probabilistic model. Bayesian networks, instead, allow a factorization of the joint into conditional probability distributions that can be learned efficiently from data.

In this thesis we focus on Markov networks. As exemplified in the examples of the previous section, for Markov networks, the independences encoded by the graph allow a decomposition of the joint distribution into *factors* or *potential functions* (or simply potentials) consisting of only a subset of the variables. The potentials are combined as a product to recover the joint distribution, commonly known as the *Gibbs distribution*. These potential functions are quantified by the parameters of the graphical model. Markov networks have been used in numerous application domains, ranging from discovering gene expression pathways in bioinformatics (Friedman et al., 2000) to computer vision (Geman and Geman (1984); Besag et al. (1991), and more recently Isard (2003) and Anguelov et al. (2005)), where they have been

historically called Markov random fields. Bayesian networks have found more applicability lately in the representation of causal models (Pearl, 2000) mainly due to the directionality of their graphs. Markov networks have instead been used primarily to represent symmetrical spatial relationships where no variable can be singled out as the cause of others. For example, in the context of climatology, a high temperature at one geographic location, although correlated to the high temperature at a neighboring location, can hardly be considered its cause. The data mining community has also recently exhibited some interest in the use of Markov networks for spatial data mining, which has applications in geography, transportation, agriculture, climatology, ecology and others (Shekhar et al., 2004).

The problem of learning a graphical probabilistic model from data consists of first learning the structure, and given the structure, learning the parameters (Heckerman et al., 1995; Buntine, 1994). Although interesting and in certain cases difficult, we do not consider the problem of parameter learning from data in this thesis. Also, we consider only random variables with discrete domains, as this simplifies the notation and requires only rudimentary concepts from probability theory, while maintaining all the challenges of the structure learning problem.

The problem of structure learning consists on finding the graph G among the (super-exponentially many) possible candidate graphs, such that, combined with the parameters Θ, results in a probabilistic model $h = (G, \Theta)$ that fits the data. Historically, this problem has been addressed by two very distinct approaches: *score-based* and *constrained-based* (also called *independence-based*). The score-based approach is exemplified for Bayesian networks by Lam and Bacchus (1994); Heckerman (1995), and for Markov networks by Della Pietra et al. (1997); McCallum (2003). Algorithms that follow this approach conduct a search in the space of legal structures in an attempt to discover a model structure of maximum score. Due to the intractable size of the search space i.e., the space of all legal graphs, which is super-exponential in size, score-based algorithms usually resort to heuristic search. At each step of the structure search, a probabilistic inference step is necessary to evaluate the score (e.g., maximum likelihood, minimum description length (Lam and Bacchus, 1994), or pseudo-likelihood (Besag, 1974)). Probabilistic inference requires a complete model, and thus at each step of the search

the parameters of the model must be estimated from the data. For Bayesian networks the computation of the parameters is tractable and therefore several practical score-based algorithms for structure learning have been developed (Lam and Bacchus, 1994; Heckerman, 1995). In the general case, for undirected models the computation of the parameters requires the calculation of the partition function (i.e., the normalizing constant), a problem known to be NP-hard for all but a small class of Markov networks, namely the class of structures with tree-width greater than 1 (Jerrum and Sinclair, 1993; Barahona, 1982; Srebro and Karger, 2001). Srebro and Karger (2001) considered a restricted class of graphical models, the class of decomposable models, which allows an efficient computation of the parameters. An example of learning non-decomposable MNs is presented by Hofmann and Tresp (1998), which is an approach for learning structure in continuous domains with non-linear relationships among the domain attributes. Their algorithm removes edges greedily based on a leave-one-out cross validation log-likelihood score. A non-score-based approach is Abbeel et al. (2006), which introduces a new class of efficient algorithms for structure and parameter learning of factor graphs, a class of graphical models that subsumes Markov and Bayesian networks. Their approach is based on a new parametrization of the Gibbs distribution in which the potential functions are forced to be probability distributions. It is a promising and theoretically sound approach, supported by a generalization of the Hammersley-Clifford theorem for factor graphs, that may lead in the future to practical and efficient algorithms for undirected structure learning.

Contrary to the score-based approach, algorithms that take the *independence-based* approach are able to learn the structure of any Markov networks efficiently (including non-decomposable networks). These algorithms exploit directly the semantics of the graph structure, namely, the fact that it encodes the conditional independences that hold in the domain (Spirtes et al., 2000). This approach casts the problem of structure learning as an instance of the constraint satisfaction problem, where the constraints are conditional independences among the variables. Algorithms in this class perform statistical tests on data to learn about conditional independences that hold in the data (e.g. Pearson's χ^2 test of independence), assume those independences hold in the model (i.e., assume the data is a sample of the model to be

learned and that the statistical test performed on the data is correct), and discard those structures that violate these independences. Most algorithms in this class proceed by successively performing statistical tests of independence until all structures but one have been discarded (uniqueness follows under certain assumptions, more below). Algorithms exist that require a number of tests that is polynomial in the number n of variables in the domain, which, together with the fact that statistical tests can be executed in a time proportional to the number of data points N in the data set, result in a total running time that is polynomial in both n and N. Intuitively, the non-exponential running time (even though the space has super-exponential size) results from the fact that each test may discard an exponential number of structures. For instance, consider a test performed at the very beginning of the algorithm that has sufficient information to discard an edge between two variables X and Y (i.e., it discards (X, Y) from the neighborhood structure $\mathcal{N}$); such a test eliminates half of the candidate structures. Another advantage of the independence-based algorithms is that they can learn the structure without the need of parameter estimation, making them the best choice for undirected graphical models (in order to learn the complete model, parameters must be estimated, but only once, when the final structure has been discovered).

For Bayesian networks, the independence-based approach has been mainly exemplified by the SGS (Spirtes et al., 2000), PC (Spirtes et al., 2000), and the Grow-Shrink (GS) (Margaritis and Thrun, 2000) algorithms, as well as algorithms for restricted classes such as trees (Chow and Liu, 1968) and polytrees (Rebane and Pearl, 1989). An important contribution presented in this thesis is the **GSMN** algorithm (**G**row-**S**hrink **M**arkov **N**etwork structure learning algorithm) which, to the best of our knowledge, is the first independence-based algorithm (and as such, the first efficient algorithm) for structure learning of Markov networks. This algorithm is presented in Chapter 2 together with **GSIMN** algorithm (**G**row-**S**hrink **I**nference-based **M**arkov **N**etwork structure learning algorithm). GSIMN improves on GSMN by exploiting Pearl's axioms of independence (see Section 1.2) to infer unknown independences from those known so far, thus saving the need to perform the corresponding statistical test on data, an operation that can be costly for large data sets or in distributed domains.

Although very promising, the independence-based approach has the disadvantage that the quality of statistical independence tests may degrade rapidly with the number of variables involved in the test (see Section 1.2.4 for details). One can therefore think of the independence-based algorithms as exchanging run time complexity for sample complexity. In Chapter 4 we address this problem by proposing a mechanism to improve the quality of tests based on argumentation (Dung, 1995; Loui, 1987; Prakken, 1997; Prakken and Vreeswijk, 2002). We model the problem of independence-based structure discovery as a knowledge base containing a set of independences related through certain axioms (a set of relationship constraints that hold between independences that we describe in more detail in Section 1.2.2 below). Statistical tests on finite data sets may result in errors in these tests and therefore inconsistencies in the knowledge base. Our approach uses the theory of argumentation for resolving these inconsistencies (an instance of the class of defeasible logics) augmented with a preference function that is used to reason and possibly correct errors in these tests. This results in a more robust conditional independence test, which we shall call the **argumentative independence test** or **AIT**.

Graphical models and formal argumentation systems share a common goal: tackling the problem of reasoning under uncertain information. This common goal has motivated several initiatives to combine them (Vreeswijk, 2005; Saha and Sen, 2004; Kohlas, 2003). For instance, Vreeswijk et al. establish an explicit connection between formal argumentation and Bayesian networks by introducing a notion of an argument in Bayesian networks. Although we use both approaches in Chapter 4, our objective is not to combine the two frameworks but rather to use argumentation as a "service" in the structure learning algorithm. That is, we use argumentation in structure learning algorithms to better cope with the uncertainty in test outcomes.

In Chapter 3, we present the **PFMN** algorithm (**P**article **F**ilter **M**arkov **N**etwork), another independence-based algorithm for structure learning. This algorithm uses a Bayesian perspective, employing a generative model (instead of argumentation) as an alternative method to deal with uncertainty in the tests outcomes. It models the problem of independence-based struc-

ture learning a statistical model whose random variables are the Markov network structure, the conditional independence tests tests, and the data.

1.2 Notation and Preliminaries

In this section we introduce important concepts necessary for an understanding of the algorithms discussed in the following chapters. We start by defining and discussing probabilistic models, continue with important concepts such as conditional independence and graphical models, and conclude with an overview of statistical tests of independence. The presentation of these basic concepts, besides allowing us to introduce our notation, also motivates the general framework of graphical models by highlighting its purpose and strengths. We refer the interested reader to Pearl (1988) for a good self-contained discussion of the topics covered in this section.

1.2.1 Probabilistic models

A probabilistic model is a powerful tool for reasoning under uncertainty that makes possible the calculation of the probability of any well-formed propositional sentence through a process called *probabilistic inference*. Given a set of propositions $\Sigma = \{\alpha, \beta, \gamma, \ldots\}$, the set of well-formed proposition sentences consists on the Boolean formulas involving these propositions, e.g. $(\alpha \wedge \beta) \vee \neg\gamma$, where a proposition is any sentence that can take only `true` or `false` values.

The propositions considered usually take the form of an assignment of values to variables where a *variable* is some attribute or property of the world (e.g. side of the coin facing upward, the letter grade of a student, the crop yield of a corn field, tree color) that can take one out of several possible values in a domain (e.g. {head, tails}, {A,B,C,D,F}, [0.1, 50] in kg, {red, yellow, green}, respectively). Examples of propositions are therefore "side of the coin = head," "crop yield = 0.95kg," "tree color = yellow," etc. If we have probabilities assigned to the possible values that a variable may take we refer to it as a *random variable*. In this work we consider only random variables in discrete domains, as this simplifies the notation and requires only elementary concepts from probability theory, while maintaining all the challenges of the

structure learning problem

Under this ontological stance, a state of the world (or a *possible world* as it is commonly called in the literature) is described by a *joint event*, i.e., an assignment of values to all random variables.

In the rest of the thesis we denote random variables with capitals (e.g., X, Y, Z) and sets of variables with bold capitals (e.g., $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$). In particular, we denote by $\mathbf{V} = \{V_1, \ldots, V_n\}$ the set of all n variables in the system, and we use notation V_i, $i = 1, 2, \ldots, n$ to denote the i-th variable in $\mathbf{V}$. In many algorithms we name the variables directly by their indices in $\mathbf{V}$; for instance, we refer to V_3, the third variable in $\mathbf{V}$, simply by 3. We denote a value of a variable by its corresponding lowercase letter and assignments using the equal sign. For example, $X = x$ denotes the event where variable X takes value x. Slightly abusing notation we denote a set of values corresponding to a set of variables by bold lowercase letters, e.g., $\mathbf{X} = \mathbf{x}$, and we may abbreviate the probability of an assignment by omitting the variables names, e.g., $\Pr(\mathbf{x}) \equiv \Pr(\mathbf{X} = \mathbf{x})$. Under this notation, a joint event can be succinctly denoted by $\mathbf{v}$.

In all algorithms in this thesis we consider the same format for data set D, i.e., a table with n columns corresponding to the variables of the system, and N rows which correspond to *data points*, a value assignment to each of the variables. An example of a data point in a domain with n binary variables $\mathbf{V} = \{V_1, V_2, V_3, V_4, V_5\}$ is $(Y_1 = 0, Y_2 = 1, Y_3 = 1, Y_4 = 0, Y_5 = 1)$. The underlying assumption that we make is that data set D is a sample of the underlying probability distribution that the algorithm is trying to model.

An important example of a probabilistic model is the *tabular model*, which for binary variables is $\Pr : \{\texttt{true}, \texttt{false}\} \times \{\texttt{true}, \texttt{false}\} \times \cdots \longrightarrow \mathbb{R}$, a function that assigns a probability $\Pr(\mathbf{v})$ to every possible joint event $\mathbf{v}$ such that the sum of the probabilities adds up to 1.

1.2.2 Conditional Independence and Graphoids

Let P be a joint probability distribution over a set of variables $\mathbf{V}$ and let $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ denote any three disjoint subsets of $\mathbf{V}$. The sets $\mathbf{X}$ and $\mathbf{Y}$ are said to be *conditionally independent*

given $\mathbf{Z}$ if for every configuration $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ of variables $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$, respectively, it is the case that

$$P(\mathbf{x} \mid \mathbf{y}, \mathbf{z}) = P(\mathbf{x} \mid \mathbf{z}) \text{ whenever } P(\mathbf{y}, \mathbf{z}) > 0. \tag{1.5}$$

Intuitively, this means that learning the value of $\mathbf{Y}$ does not provide additional information about $\mathbf{X}$, once we know the value of $\mathbf{Z}$. One can think of $\mathbf{Z}$ as "shielding" $\mathbf{X}$ from $\mathbf{Y}$.

We will use the notation of Dawid (1979) $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})_P$ or simply $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$ to denote that in the probabilistic model P, $\mathbf{X}$ is independent of $\mathbf{Y}$ conditioned on $\mathbf{Z}$. $(\mathbf{X} \not\perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$ denotes conditional dependence. We will use $(X \perp\!\!\!\perp Y \mid \mathbf{Z})$ as shorthand for $(\{X\} \perp\!\!\!\perp \{Y\} \mid \mathbf{Z})$. Unconditional independence will be denoted by $(X \perp\!\!\!\perp Y \mid \emptyset)$ or simply $(X \perp\!\!\!\perp Y)$.

Eqs. (1.6) below contain a (partial) list of properties satisfied by the conditional independence relation $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$. (Intersection is valid in strictly positive probability distributions). The proof of these properties can be derived from Eq. (1.5) and the axioms of probability theory. We will therefore refer to them as the *general axioms* of independence. These properties were independently proposed by Dawid (1979); Spohn (1980); Pearl and Paz (1987).

$$\begin{aligned}
&\textbf{(Symmetry)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \iff (\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z}) \\
&\textbf{(Decomposition)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}) \\
&\textbf{(Weak Union)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \\
&\textbf{(Contraction)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \\
&\textbf{(Intersection)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z})
\end{aligned} \tag{1.6}$$

where $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ and $\mathbf{W}$ are all disjoint subsets of $\mathbf{V}$.

Pearl (Pearl, 1988, p. 87) describes some striking similarities between the general axioms and vertex separation in graphs. For this reason he called *graphoids* any three-place relation that satisfies the general axioms. Conditional independence is one example, but interestingly these axioms are also satisfied by vertex separation in undirected graphs, by d-separation in directed graphs (see formal definition of d-separation below), and by embedded multi-valued dependencies in relational databases (see for instance Butz (2000)).

The similarity of the general axioms with the axioms of vertex separation suggests that it may be possible to encode a set of conditional independences in the form of a graph. This encodings are called *graphical models*. Assuming a one-to-one correspondence between the set

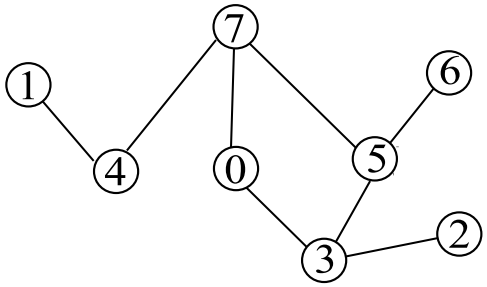


Figure 1.2 Example Markov network. The nodes represent variables in the domain $\mathbf{V} = \{0, 1, 2, 3, 4, 5, 6, 7\}$.

of random variables of the probabilistic model and the nodes in the graph, a graphical model assigns a semantics to graphs that allows them to encode correctly the independences of the probabilistic model (if possible) and allow one to know precisely which class of probabilistic models can be encoded by a given class of graphs. To fully quantify a graphical model one must also find a way to quantify the edges of the graph. That is, we consider graphical models that consists of two parts: a graph (the model structure), and a set of parameters. We distinguish between two major classes of graphical models: Markov networks and Bayesian networks, which differ by the type of their edges: undirected and directed, respectively.

In undirected models or Markov networks (MNs), the independences of a probabilistic model P are encoded in a graph G as a mapping of conditional independences into vertex separation. An example of MN is shown in Fig. 1.2. The main idea is that a subset $\mathbf{Z}$ of nodes in a graph G intercepts all paths between nodes of $\mathbf{X}$ and those of $\mathbf{Y}$ (written $\langle \mathbf{X}, \mathbf{Y} \mid \mathbf{Z} \rangle_G$) if and only if $\mathbf{X}$ is independent of $\mathbf{Y}$ given $\mathbf{Z}$ in P. Formally,

$$(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})_P \iff \langle \mathbf{X}, \mathbf{Y} \mid \mathbf{Z} \rangle_G.$$

A graph that satisfies this condition is called a *perfect map* of P and we say it is *faithful* to the probability distribution P. For instance, in the example MN of Fig. 1.2, the set $\{0, 5\}$ intercepts all paths from node 7 to node 3, and therefore, under the faithfulness assumption, it must be the case that 7 is conditionally independent of 3 given $\{0, 5\}$. As argued in Pearl (1988), not every probability distribution has a perfect map. Those models that do have a perfect map are called *graph-isomorph* and satisfy, in addition to the general axioms Eqs. (1.6),

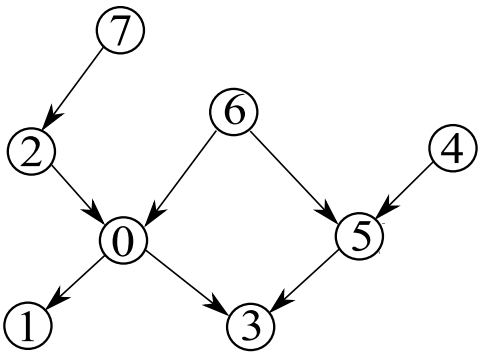


Figure 1.3 Example Bayesian network. The nodes represent variables in the domain $\mathbf{V} = \{0, 1, 2, 3, 4, 5, 6, 7\}$.

the following, more specific set of axioms:

$$\begin{aligned}
&\textbf{(Symmetry)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \iff (\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z}) \\
&\textbf{(Decomposition)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}) \\
&\textbf{(Intersection)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \\
&\textbf{(Strong Union)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \\
&\textbf{(Transitivity)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \gamma \mid \mathbf{Z}) \vee (\gamma \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})
\end{aligned} \tag{1.7}$$

where $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ and $\mathbf{W}$ are all disjoint subsets of $\mathbf{V}$, and γ stands for a single variable, not in $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$.

We refer to these axioms as the *specific undirected* axioms.

Directed graphs or Bayesian networks (BNs) use a slightly more complex separability criterion called *d-separation*, which takes into consideration the directionality of the arrows in the graph. An example BN is shown in Fig. 1.3. Intuitively, the notion of d-separation matches that of vertex-separation, where the notion of separation is replaced by the more complex notion of "blocked" paths. This is defined formally in the next two definitions.

Definition 1.1. *A set of nodes* $\mathbf{Z}$ *is said to* d-separate $\mathbf{X}$ *from* $\mathbf{Y}$*, denoted* $\langle \mathbf{X}, \mathbf{Y} \mid \mathbf{Z} \rangle_D$*, if* $\mathbf{Z}$ *"blocks" every path from a node in* $\mathbf{X}$ *to a node in* $\mathbf{Y}$*.*

Definition 1.2. *A set of nodes* $\mathbf{Z}$ *"blocks" a path from a node in set* $\mathbf{X}$ *to a node in set* $\mathbf{Y}$*, if there is a node* W *in the path satisfying one of the following two conditions: (1)* W *has converging arrows and none of* W *or its descendants are in* $\mathbf{Z}$*, or (2)* W *does not have converging arrows and* W *is in* $\mathbf{Z}$*.*

For instance, in Fig. 1.3, $\{2\}$ d-separates 7 and 0 since the only path from 7 to 0 is blocked by $\{2\}$. This is because there is a node 2 in the path that does not have converging arrows (condition (2) above) and is in $\{2\}$. As another example, consider nodes 4 and 3. These nodes are not d-separated by $\{5\}$, since the path $4-5-6-0-3$ between 4 and 3 is not blocked. This is because in that path there is a node (5) that has converging arrows (from nodes 4 and 6) and it is in $\{5\}$.

Analogously to the undirected case, we say that a directed graph is faithful to (i.e., it is a perfect map of) a probability distribution P if it satisfies

$$(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})_P \iff \langle \mathbf{X}, \mathbf{Y} \mid \mathbf{Z} \rangle_D$$

and conversely, if a probability distribution P has a faithful graph it is called *causal* and it satisfies, beside the general axioms, the following, more specific set of axioms:

$$\begin{aligned}
&\textbf{(Symmetry)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \iff (\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z}) \\
&\textbf{(Decomposition)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}) \\
&\textbf{(Composition)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \impliedby (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}) \\
&\textbf{(Intersection)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \\
&\textbf{(Weak Union)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \\
&\textbf{(Contraction)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \\
&\textbf{(Weak Transitivity)} && (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \gamma) \implies (\mathbf{X} \perp\!\!\!\perp \gamma \mid \mathbf{Z}) \vee (\gamma \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \\
&\textbf{(Chordality)} && (\alpha \perp\!\!\!\perp \beta \mid \gamma \cup \delta) \wedge (\gamma \perp\!\!\!\perp \delta \mid \alpha \cup \beta) \implies (\alpha \perp\!\!\!\perp \beta \mid \gamma) \vee (\alpha \perp\!\!\!\perp \beta \mid \delta)
\end{aligned} \tag{1.8}$$

In the above, $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ and $\mathbf{W}$ are disjoint subsets of $\mathbf{V}$, and α, β, γ and δ are distinct single variables.

In the above,

We refer to these axioms as the *specific directed* axioms.

1.2.3 Assumptions

Graphical models, as well as the independence-based approach of learning such models from data, are grounded on a sound formalism. For that reason below we explicitly present the set of assumptions that we make in all remaining chapters of this thesis.

- The input data set is an i.i.d. (independent and identically distributed) sample of the underlying probabilistic model.
- We assume Faithfulness, i.e., that the underlying probability distribution has a perfect map. This implies that the specific axioms of Eqs. (1.7) and Eqs. (1.8) hold (for undirected and directed models respectively).
- For the operation of the algorithms we assume the existence of an independence query oracle that can answer correctly statistical independence queries (see next section).
- We assume that there exist no hidden variables and no missing values. That is, the input data set contains data for all the variables in the domain, and each data point has no missing values.

These are standard assumptions that are needed for formally proving the correctness of independence-based structure learning algorithms (Spirtes et al., 2000).

1.2.4 Independence Oracle Implementation

As mentioned above, this thesis addresses the problem of independence-based structure learning of the graphical model of a domain from data. Independence-based algorithms operate by conducting a series of conditional independence queries. For these we assume that an *independence-query oracle* exists that is able to provide such information. This approach can be viewed as an instance of a statistical query oracle (Kearns and Vazirani, 1994). In practice such an oracle does not exist, but is frequently implemented approximately by a statistical test evaluated on the data set (for example, this can be Pearson's conditional independence χ^2 (chi-square) test (Agresti, 2002), Wilk's G^2 test, a mutual information test or the Bayesian independence test of Margaritis (2005) etc.). In this work we used Wilk's G^2 test (Agresti, 2002) and Margaritis' Bayesian test. To determine conditional independence between two variables X and Y given a set $\mathbf{Z}$ from data, the statistical test G^2 (and any other independence test based on hypothesis testing, e.g., the χ^2 test) returns a *p-value*, which is the probability of error in assuming that the two variables are dependent when in fact they are not. If the p-value

of a test is $p(X, Y \mid \mathbf{Z})$, the statistical test concludes independence if and only if $1-p(X, Y \mid \mathbf{Z})$ is smaller than or equal to a confidence threshold α i.e.,

$$(X \perp\!\!\!\perp Y \mid \mathbf{Z}) \iff p(X, Y \mid \mathbf{Z}) \geq 1 - \alpha. \tag{1.9}$$

Common values for α are 0.95, 0.99, and 0.90.

The computation of the p-value of some triplet $t = (\mathbf{X}, \mathbf{Y} \mid \mathbf{Z})$ requires the construction of a contingency table, which is a histogram containing a count for each possible value combination $\mathbf{w}(t)$ of the variables $\mathbf{W}(t) = \mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$ of triplet t. Each entry in the histogram contains the number of data points in the input data set D whose values for the variables in $\mathbf{W}(t)$ match the value combination $\mathbf{w}(t)$ corresponding to that entry. Let us consider an example of a contingency table. For a domain with five binary variables $\{V_1, V_2, V_3, V_4, V_5\}$, the contingency table of triplet $(V_2, V_4 \mid V_5)$ has an entry for each combination of values $(V_2 = 0, V_4 = 0, V_5 = 0)$; $(V_2 = 0, V_4 = 0, V_5 = 1)$; ... $(V_2 = 1, V_4 = 1, V_4 = 1)$. A data point $(V_1 = 0, V_2 = 1, V_3 = 1, V_4 = 0, V_5 = 1)$ i.e., a row in the data set, would increment the count of entry $(V_2 = 1, V_4 = 0, V_5 = 1)$ by one.

The number of cells in a contingency table grows exponentially with the number of variables involved in the table. Therefore, tests involving a large number of variables will produce sparse contingency tables which result in unreliable tests. This is because the number of possible configurations of the variables grows exponentially with the size of the test—for example, there are 2^n cells in a test involving n binary variables, and to fill such a table with one data point per cell we would need a data set of at least exponential size i.e., $N \geq 2^n$. Exacerbating this problem, more than one data point per cell is typically necessary for a reliable test: As recommended by Cochran (1954), if more than 20% of the cells of the contingency table have less than 5 data points the test is deemed unreliable. This exposes an important limitation of independence-based algorithms, namely, their rapid degradation in quality with the size of the tests it requires during its execution.

1.2.5 Learning the Parameters

In Section 1.2.2 we established the semantics of graphical models in terms of the purely qualitative notion of conditional independence. In this thesis we only address the problem of structure learning, and so no further understanding is necessary. However, we briefly explain the quantitative aspects of a graphical model for the sake of completeness, and also because it better motivates our work.

This section address the problem of constructing a probability distribution that preserves the dependency structure of an arbitrary graph G. For undirected models this problem has been addressed by the theory of Markov random fields (Isham, 1981; Lauritzen, 1982), which provides a method for constructing the Gibbs distribution for an arbitrary undirected graph G:

1. Identify the (maximal) cliques of G. A clique is a maximal subgraph of G whose nodes are all adjacent to each other.

2. Let $\mathcal{C}(G)$ denote the set of all maximal cliques of graph G, and $\mathbf{X}_C$ the set of variables in clique C. Then, for each clique $C \in \mathcal{C}(G)$, define a potential function $g_C(\mathbf{x}_C)$ which assigns a non-negative value to each assignment $\mathbf{x}_C$ of variables $\mathbf{X}_C$.

3. Form the product $\widetilde{P}_G(x_1, \ldots, x_n) = \prod_{C \in \mathcal{C}(G)} g_C(\mathbf{x}_C)$ of the potential functions over all cliques in $\mathcal{C}(G)$.

4. Construct the *Gibbs distribution* by normalizing the product $\widetilde{P}_G(x_1, \ldots, x_n)$

$$\Pr(x_1, \ldots, x_n) = \frac{\widetilde{P}_G(x_1, \ldots, x_n)}{Z}. \tag{1.10}$$

   The normalization constant Z is usually called the *partition function* and is computed by summing the product $\widetilde{P}_G(\mathbf{x})$ over all possible value combinations of the variables in the domain

$$Z = \sum_{x_1, \ldots, x_n} \widetilde{P}_G(x_1, \ldots, x_n).$$

The Hammersley-Clifford theorem, an important result in the theory of MRFs, proves that the general form of the Gibbs distribution of Eq. (1.10) satisfies the conditional independences

encoded in the graph $G = (\mathbf{V}, \mathbf{E})$ i.e., for every node $X \in \mathbf{V}$, the set of nodes adjacent to X (denoted by $\mathcal{N}(X)$), shields X from the rest of the variables, formally

$$\forall X \in \mathbf{V}, \Pr(X \mid \mathbf{V} - \{X\}) = \Pr(X \mid \mathcal{N}(X))$$

where Pr is defined by the Gibbs distribution of Eq. (1.10), and the equality holds for all possible assignments of X, $\mathcal{N}(X)$, and $\mathbf{V} - \mathcal{N}(X)$.

The general form of the Gibbs distribution introduced above presents some difficulties. First, the meaning of the potential functions is difficult to discern. Second, the cost of computing the partition function Z is exponential, as it requires a sum over all possible configurations (a system with n binary variables has 2^n possible configurations). Bayesian networks present an alternative procedure for quantifying the model that overcomes these difficulties. In Bayesian networks, the joint distribution $\Pr(x_1, \ldots, x_n)$ is factored into a product of conditional distributions using the well-known chain rule, i.e.

$$\begin{aligned} \Pr(x_1, \ldots, x_n) &= \Pr(x_n \mid x_{n-1}, \ldots, x_1) \Pr(x_{n-1} \mid x_{n-2}, \ldots, x_1) \cdots \Pr(x_3 \mid x_2, x_1) \Pr(x_2 \mid x_1) \Pr(x_1) \\ &= \prod_i \Pr(x_i \mid x_{i-1}, \ldots, x_1). \end{aligned}$$

It has been proved (Pearl, 1988, p. 119) that

$$\Pr(x_1, \ldots, x_n) = \prod_i \Pr(x_i \mid \mathit{parents}(x_i)). \tag{1.11}$$

where $\mathit{parents}(X)$ denotes the set of variables whose nodes are the parents of the node corresponding to variable X in the Bayesian network.

Therefore, one simply needs, for each variable X_i, the conditional probabilities $P(x_i \mid \mathit{parents}(x_i))$ in order to fully quantify the model. Unfortunately, Markov networks do not have an expression equivalent to Eq. (1.11), and thus must rely on the computationally expensive Gibbs decomposition of Eq. (1.10) in order to fully quantify the model.

CHAPTER 2. The GSIMN Algorithm

2.1 Introduction

In this chapter we introduce two related independence-based algorithms for structure learning: **GSMN** (**G**row **S**hrink **M**arkov **N**etwork learning algorithm) and and **GSIMN** (**G**row **S**hrink **I**nference-based **M**arkov **N**etwork learning algorithm) (Bromberg et al., 2006, 2007). To the best of our knowledge, the GSMN algorithm is the first independence-based structure learning algorithm for Markov networks that has appeared in the literature. GSIMN extends GSMN with an inference step that reduces the number of tests performed when compared to GSMN without adversely affecting the quality of the output network.

The GSMN algorithm is an adaptation to Markov networks of the GS algorithm by Margaritis and Thrun (2000), originally developed for learning the structure of Bayesian networks. Until very recently, algorithms for structure learning were based on maximum likelihood estimation which has been proved to be NP-hard for Markov networks. The independence-based approach does not require the computation of likelihoods, and thus both GSMN and GSIMN can compute the structure efficiently (as shown in our experiments).

GSMN is thus interesting and useful in itself, but we also use it as a point of reference of the performance with regard to time complexity and accuracy achieved by GSIMN. With GSIMN, we explored the possibility of exploiting Pearl's axiomatic characterization of conditional independences (shown in Eqs. (1.7)) to improve the running time by reducing the number of independence tests performed on data.

In the next section we present some preliminary concepts and some basic algorithms that are useful to acquire an intuitive comprehension of the GSMN algorithm. We then present the GSMN algorithm in Section 2.3, followed by the GSIMN algorithm in Section 2.5.

2.2 Preliminaries

The concept of Markov blanket is central to many independence-based algorithms as it gives a theoretically sound and straightforward procedure for learning the undirected structure of a Markov network. Given the domain variables $\mathbf{V}$, Pearl (Pearl, 1988, p. 97) defines the *Markov blanket* $\mathbf{MB}(X)$ of a variable $X \in \mathbf{V}$ as the minimal subset $\mathbf{S} \subseteq \mathbf{V}$ such that

$$(X \perp\!\!\!\perp \mathbf{V} - \mathbf{S} - \{X\} \mid \mathbf{S}) \text{ and } X \notin \mathbf{S} \tag{2.1}$$

that is, none of the proper subsets of $\mathbf{S}$ satisfy Eq. (2.1).[1]

The Markov blanket can be interpreted as the smallest set of variables that shields X from the influence of all other variables. Note that $\mathbf{MB}(X)$ always exists because $(X \perp\!\!\!\perp \mathbf{S}' \mid X)$, which holds trivially for every set $\mathbf{S}' \subseteq \mathbf{V}$, guaranteeing that the set $\mathbf{S} = \{X\}$ satisfies Eq. (2.1).

The relation between Markov blankets and the structure of a Markov network is explained by the following theorem (Pearl, 1988; Pearl and Paz, 1987).

Theorem 2.1. *Every variable $X \in \mathbf{V}$ in a dependency model M satisfying Symmetry, Decomposition, Intersection, and Weak Union (see Eqs. (1.6)) has a unique Markov blanket $\mathbf{MB}(X)$. Moreover, $\mathbf{MB}(X)$ coincides with the set of vertices $\mathbf{B}_M(X)$ adjacent to X in the model M.*

In particular, the theorem is satisfied by every strictly positive distribution, since the probabilistic dependence relation in any such distribution satisfies Symmetry, Decomposition, Intersection, and Weak Union.

The theorem suggests a very simple algorithm for learning the structure of a Markov network, shown in Alg. (1). This algorithm learns the Markov blanket of every variable in the probabilistic model M, and links each of these variables with every member of its Markov blanket.

Algorithm 1 relies on a subroutine for discovering the Markov blanket of the variables in the domain. In Margaritis and Thrun (2000), Margaritis et. al. introduced the GS algorithm

[1] Pearl distinguishes between those subsets that are minimal, calling them *Markov boundaries*, and those that are not necessarily minimal, calling them *Markov blankets*. Here we simplify our nomenclature by omitting this distinction.

Algorithm 1 **Markov Blanket Structure Learning**$(\mathbf{V}, M)$.

1: $G \longleftarrow (\mathbf{V}, \mathbf{E})$ *// Initialize structure of Markov network.*
2: **for all** $X \in \mathbf{V}$ **do**
3: $\quad$ $\mathbf{MB}(X) \longleftarrow$ Discover Markov blanket of X in probabilistic model M (e.g., using GS algorithm Alg.(2)).
4: $\quad$ **for all** $Y \in \mathbf{MB}(X)$ **do**
5: $\quad\quad$ add edge (X, Y) to $\mathbf{E}$
6: **return** G

Algorithm 2 **GS**$(X, \mathbf{V}, M)$.

1: $\mathbf{S} \longleftarrow \emptyset$.
2: **while** $\exists Y \in \mathbf{V} - \{X\}$ such that $(Y \not\perp\!\!\!\perp X \mid \mathbf{S})$ **do** $\mathbf{S} \longleftarrow \mathbf{S} \cup \{Y\}$. **[Growing Phase]**
3: **while** $\exists Y \in \mathbf{S}$such that $(Y \perp\!\!\!\perp X \mid \mathbf{S} - \{Y\})$ **do** $\mathbf{S} \longleftarrow \mathbf{S} - \{Y\}$. **[Shrinking Phase]**
4: **return** $\mathbf{S}$

shown in Alg. (2), an independence-based algorithm for learning the Markov blanket of any input variable in the domain.

The GS algorithm maintains a set $\mathbf{S}$, initialized empty, that will contain the Markov blanket after the end of the shrink phase (line 4). The algorithm then proceeds in two stages, the **grow** and **shrink** phases. During the grow phase (line 2) the algorithm augments $\mathbf{S}$ with every variable Y that is found dependent on X conditioned on the current state of $\mathbf{S}$. This process guarantees that by the end of the grow phase, the set $\mathbf{S}$ (denoted $\mathbf{S}^G$) contains all members of the Markov blanket. However, it may include some false positives that are non-members, i.e., $\mathbf{S}^G \supseteq \mathbf{MB}(X)$. We can argue for the correctness of this stage in two parts. First we see that $Y \in \mathbf{MB}(X) \implies Y \in \mathbf{S}^G$: From Theorem 2.1, there must be an edge (X, Y) so no set of vertices can separate X from Y. In particular, the set $\mathbf{S}$ at the time Y is tested against X during the grow phase, regardless of its content, cannot separate X from Y, and thus Y is added to $\mathbf{S}$ (and not removed from $\mathbf{S}$ during the grow phase). The converse i.e., $Y \in \mathbf{S}^G \implies Y \in \mathbf{MB}(X)$, is not generally true, and for that reason there may be some false positives added to $\mathbf{S}$. These false positives are removed during the shrink phase. We must convince ourselves that every false positive, and no true positive, is removed from $\mathbf{S}^G$, i.e., $Y \notin \mathbf{MB}(X)$ if and only if $(Y \perp\!\!\!\perp X \mid \mathbf{S} - \{Y\})$ (the condition tested during the shrink

phase). The "if" direction follows from the definition of Markov blanket and the fact that $\mathbf{S}^G$ is a superset of the Markov blanket. The "only if" direction follows directly from the definition of Markov blanket.

Algorithm 1, with the GS algorithm as the subroutine for discovering Markov blankets, forms the outline of GSMN, described in detail in the next section. The extra complexity of GSMN comes from an initialization stage that computes a heuristics ordering of variables for the grow and shrink phases, and some straightforward savings of tests caused by the relationship between the Markov blanket of any two variables. Since the GS algorithm is not our main contribution, we only sketched a proof of correctness above with the goal of helping the reader gain a good intuition of the GS algorithm. Appendices A and B contains a detailed proof for the correctness of GSMN and GSIMN, our main contributions.

2.3 The GSMN Algorithm

In this section we discuss our first algorithm, called GSMN (Grow-Shrink Markov Network learning algorithm), for learning the structure of a MN. This algorithm extends the simple algorithm (Alg. 1) described in the previous section with an initialization phase that computes a heuristic for the ordering in which variables are tested in the grow and shrink phases, and an extra phase, the *propagation* phase, during the Markov blanket discovery. Given as input a data set D and a set of variables $\mathbf{V}$, GSMN returns the set of nodes (variables) $\mathbf{B}^X$ that are adjacent to each variable $X \in \mathbf{V}$; these completely determine the structure of the domain MN. The algorithm is shown in two parts, the main part (Algorithm 3) and the independence test (Algorithm 4).

The algorithm starts with an initialization phase. For reasons of clarity of presentation, we explain the initialization phase near the end of this section. GSMN then executes its main loop, in which it examines each variable in $\mathbf{V}$ (lines 8–24) according to the **visit order** π, determined during the initialization phase. Each iteration of the main loop (lines 8–24) includes three phases: the *propagation phase* (lines 11–14), the *grow phase* (lines 15–21), and the *shrink phase* (line 22). The order that variables are examined during the grow phase of variable X is

Algorithm 3 **GSMN**$(\mathbf{V}, D)$.

```
1: /* Initialization. */
2: for all X, Y ∈ V, X ≠ Y do p_XY ← G(X, Y | ∅)
3: initialize π such that i < i' iff avg_j log(p_{π_i j}) < avg_j log(p_{π_{i'} j})
4: for all X ∈ V do
5:     B^X ← ∅
6:     initialize λ^X such that j < j' iff p_{X λ^X_j} < p_{X λ^X_{j'}}
7:     remove X from λ^X
8: /* Main loop. */
9: while π not empty do
10:    X ← dequeue(π)
11:    /* Propagation phase. */
12:    T ⟵ {Y : Y was visited and X ∈ B^Y}
13:    F ⟵ {Y : Y was visited and X ∉ B^Y}
14:    for all Y ∈ T ∪ F do remove Y from λ^X
15:    /* Grow phase. */
16:    S ← ∅
17:    while λ^X not empty do
18:        Y ← dequeue(λ^X)
19:        if p_XY < (1 − α) then
20:            if ¬I(X, Y | S ∪ T) then
21:                S ← S ∪ {Y}
22:    /* Shrink phase. */
23:    Y ∈ S such that I(X, Y | S ∪ T − {Y}) do S ← S − {Y}
24:    B^X ← S ∪ T
25: return {B^X : X ∈ V}
```

Algorithm 4 $I(X, Y \mid \mathbf{S})$: Statistical test.

```
1: return (G(X, Y | S) ≥ 1 − α)
```

called the **grow order** λ_X of variable X, also determined during the initialization phase.

During the propagation phase, all variables Y for which $\mathbf{B}^Y$ has already been computed, i.e., all variables Y already visited, are removed from λ^X. From those, the ones whose set $\mathbf{B}^Y$ contains X (set $\mathbf{T}$), will be added to the final $\mathbf{B}^X$ at the end. This is justified by the fact that in undirected graphs, Y is in the Markov blanket of X if and only if X is in the Markov blanket of Y. The remaining variables (set $\mathbf{F}$) cannot be members of $\mathbf{B}^X$ because there exists some set of variables that has rendered them conditionally independent of X in a previous step, and they are therefore omitted from consideration (removed from λ^X).

The grow phase of X proceeds by attempting to add each variable Y to the current set of

hypothesized neighbors of X, contained in $\mathbf{S} \cup \mathbf{T}$. $\mathbf{T}$ does not change during the whole visit of X, while $\mathbf{S}$ is initially empty but grows by some variable Y during each iteration of the grow loop of X if and only if Y is found dependent with X given the current set of hypothesized neighbors $\mathbf{S} \cup \mathbf{T}$ (line 20). The condition $p_{XY} \leq 1 - \alpha$ (line 19) avoids an independence test in the case that X and Y were found (unconditionally) independent during the initialization phase, since by the axiom of Strong Union this implies X and Y are independent given any conditioning set.

Due to the heuristic ordering that the variables are examined (determined by the priority queue λ_X), at the end of the grow phase some of the variables in $\mathbf{S}$ might not be true neighbors of X in the underlying MN—these are called *false positives*. This justifies the shrink phase of the algorithm, which removes each false positive Y in $\mathbf{S}$ by testing for independence with X conditioned on $\mathbf{S} \cup \mathbf{T} - \{Y\}$. If Y is found independent of X, it cannot be a true neighbor (i.e., there cannot be an edge $X - Y$), and GSMN removes it from $\mathbf{S}$. Assuming faithfulness and correctness of the independence query results, by the end of the shrink phase $\mathbf{B}^X$ contains exactly the neighbors of X. A proof of correctness of GSMN is presented in Appendix A.

We now describe the initialization phase. As mentioned above, the order that variables are examined in the main loop and the grow phase is completely determined by the visit order π and grow orders λ_X, calculated during the initialization phase. These are implemented as priority queues and are initially permutations of $\mathbf{V}$ (λ_X is a permutation of $\mathbf{V} - \{X\}$) such that the position of a variable in the queue denotes its priority e.g., $\pi = [2, 0, 1]$ means that variable 2 has the highest priority (will be visited first), followed by 0 and finally by 1. Similarly, the position of a variable in λ_X determines the order it will be examined during the grow phase of X.

During the initialization phase the algorithm computes the strength of the dependence between each pair of variable X and Y, as given by the unconditional p-value $G(X, Y \mid \varnothing)$ for each pair of variables $X \neq Y$, denoted by p_{XY} in the algorithm. (In practice, the logarithm of the p-values is computed, which allows greater precision in domains where some dependencies may be very strong or very weak.) In particular, for the visit order π, the algorithm gives higher

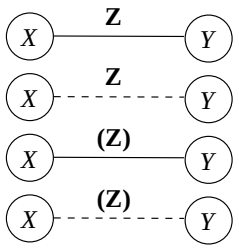
priority to (visits earlier) those variables with a lower average log p-value (line 3), indicating stronger dependence. This average is defined as:

$$\operatorname*{avg}_{Y} \log(p_{XY}) = \frac{1}{|\mathbf{V}| - 1} \sum_{Y \neq X} \log(p_{XY}).$$

For the growing order λ_X of variable X, the algorithm gives higher priority to those variables Y whose p-value (or equivalently the log of the p-value) with variable X is smaller (line 6). This ordering is a heuristic, justified by the intuition of a well-known "folk-theorem" (as Koller and Sahami (1996) puts it) that states that probabilistic influence or association between attributes tends to attenuate over distance in a graphical model. This suggests that a pair of variables X and Y with low unconditional p-value are less likely to be directly linked. Note that the computational cost for the calculation of p_{XY} is low due to the empty conditioning set.

2.3.1 Independence Graphs

We can demonstrate the operation of GSMN graphically by the concept of the *independence graph*, which we now introduce. We define an independence graph to be an undirected graph in which conditional independences and dependencies between single variables are represented by one or more annotated edges between them. A solid (dotted) edge between variables X and Y annotated by $\mathbf{Z}$ represents the fact that X and Y have been found dependent (independent) given $\mathbf{Z}$. If the conditioning set $\mathbf{Z}$ is enclosed in parentheses then this edge represents an independence or dependence that was *inferred* from Eqs. (1.7) (as opposed to computed from statistical tests). Shown graphically:



$(X \not\perp\!\!\!\perp Y \mid \mathbf{Z})$

$(X \perp\!\!\!\perp Y \mid \mathbf{Z})$

$(X \not\perp\!\!\!\perp Y \mid \mathbf{Z})$ (inferred)

$(X \perp\!\!\!\perp Y \mid \mathbf{Z})$ (inferred)

For instance, in Figure 2.2, the dotted edge between 5 and 1 annotated with $3, 4$ represents the fact that $(5 \perp\!\!\!\perp 1 \mid \{3, 4\})$. The absence of an edge between two variables indicates the absence of information about the independence or dependence between these variables under any conditioning set.

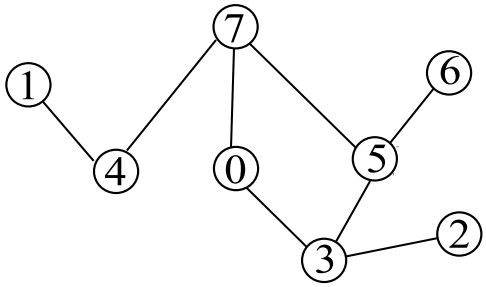


Figure 2.1 Example Markov network. The nodes represent variables in the domain $\mathbf{V} = \{0, 1, 2, 3, 4, 5, 6, 7\}$.

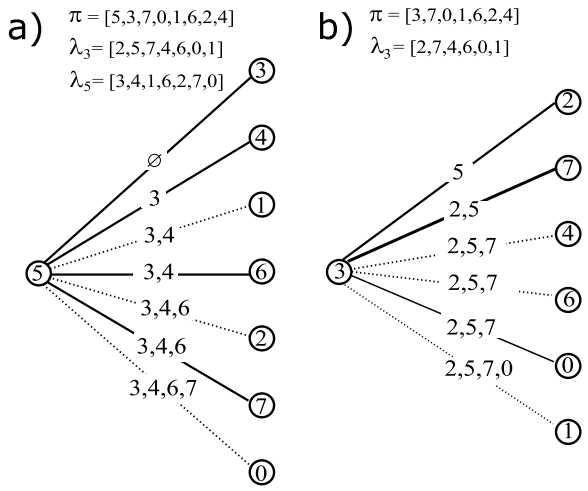


Figure 2.2 Illustration of the operation of GSMN. The figure shows the growing phase of two consecutively visited variables 5 and 3 as dictated by visit ordering π. Note that the the growing order of variable 3, which was $\lambda_3 = [3, 4, 1, 6, 2, 7, 0]$ before the propagation phase, has changed to $\lambda_3 = [2, 4, 7, 6, 0, 1]$ (see text).

Example 2.1. *This example illustrates the operation of GSMN in the domain whose underlying Markov network is shown in Figure 2.1. Figure 2.2 shows the independence graph at the end of the grow phase of the first two variables in the visit order* π *(*5 *and* 3*). We do not discuss in this example the initialization phase of GSMN. Instead, we assume that the visit (*π*) and grow (*λ*) orders are as shown in the figure.*

Variable 5 *is examined first by the algorithm (i.e., first in queue* π*). According to vertex separation on the underlying network (Figure 2.1), variables* 3*,* 4*,* 6*, and* 7 *are found dependent*

with 5 *during the growing phase i.e.,*

$$\neg I(5, 3 \mid \varnothing),$$
$$\neg I(5, 4 \mid \{3\}),$$
$$\neg I(5, 6 \mid \{3, 4\}),$$
$$\neg I(5, 7 \mid \{3, 4, 6\})$$

and are therefore successively added to $\mathbf{S}$*. Note here* $\mathbf{T}$ *is empty since* 5 *is the first variable to be visited. Variables* 1*,* 2*, and* 0 *are found independent i.e.,*

$$I(5, 1 \mid \{3, 4\}),$$
$$I(5, 2 \mid \{3, 4, 6\}),$$
$$I(5, 0 \mid \{3, 4, 6, 7\}),$$

and are not incorporated into $\mathbf{S}$*.*

The final value of $\mathbf{S}$ *at the end of the growing phase (and beginning of the shrink phase) of variable* 5 *is* $\mathbf{S} = \{3, 4, 6, 7\}$*. Among these, variables* 3*,* 6 *and* 7 *are found dependent with* 5 *during the shrink phase, i.e.,*

$$\neg I(5, 7 \mid \{3, 4, 6\})$$
$$\neg I(5, 3 \mid \{4, 6, 7\})$$
$$\neg I(5, 6 \mid \{3, 4, 7\}),$$

and are therefore not removed from $\mathbf{S}$ *(the shrink phase is not shown in the independence graph of Figure 2.2). Variable* 4 *(a false positive) is found independent with* 5 *i.e.,*

$$I(5, 4 \mid \{3, 6, 7\}),$$

and is therefore removed from $\mathbf{S}$*. By the end of the shrink phase, the set* $\mathbf{B}^X$ *of variable* 5 *is set to the value of* $\mathbf{S} \cup \mathbf{T}$*, which equals* $\{3, 6, 7\}$ *in our example. We can verify that it matches the correct Markov blanket (the set of neighbors) of variable* 5 *in the underlying domain considered in this example, shown in Figure 2.1.*

According to the visit order π*, variable* 3 *is the next one to be visited. During its propagation phase, variable* 5 *is removed from* λ_3 *and placed in* $\mathbf{T}$ *since* $3 \in \mathbf{B}^5$*, making* $\mathbf{T}$ *equal to* $\{5\}$*. Thus, during the growing phase of* 3*, all tests are conditioned on* $\mathbf{S} \cup \{5\}$*, and during the shrink phase all tests on variable* Y *are conditioned on* $\mathbf{S} \cup \{5\} - \{Y\}$*.*

During the growing phase of variable 3*, variables* 2*,* 7*, and* 0 *are found dependent with* 3*, i.e.,*

$$\neg I(3, 2 \mid \varnothing \cup \{5\}),$$
$$\neg I(3, 7 \mid \{2\} \cup \{5\}),$$
$$\neg I(3, 0 \mid \{2, 7\} \cup \{5\}),$$

and thus are added to $\mathbf{S}$*. Variables* 4*,* 6*, and* 1 *are found independent, i.e.,*

$$I(3, 4 \mid \{2, 7\} \cup \{5\}),$$
$$I(3, 6 \mid \{2, 7\} \cup \{5\}),$$
$$I(3, 1 \mid \{2, 7, 0\} \cup \{5\}),$$

and are therefore not added to $\mathbf{S}$*.*

The final value of $\mathbf{S}$ *at the end of the growing phase (and beginning of the shrink phase) of variable* 3 *is* $\mathbf{S} = \{2, 7, 0\}$*. During the shrink phase of variable* 3*, variables* 0 *and* 2 *are found dependent of* 3*, i.e.,*

$$\neg I(3, 0 \mid \{2, 5, 7\})$$
$$\neg I(3, 2 \mid \{0, 5, 7\}),$$

and are therefore not removed from $\mathbf{S}$*. Variable* 7 *is a false positive and it is found independent with* 3*, i.e.,*

$$I(3, 7 \mid \{0, 2, 5\}),$$

and is therefore removed from $\mathbf{S}$*. By the end of the shrink phase,* $\mathbf{B}^X = \mathbf{S} \cup \mathbf{T} = \{0, 2, 5\}$*, which matches the neighborhood of variable* 3 *in the underlying domain of Figure 2.1.*

2.4 The Triangle Theorem

In this section we present a theorem that is used in the subsequent GSIMN algorithm. As will be seen, the main idea behind the GSIMN algorithm is to attempt to decrease the number of tests done by exploiting the properties of the conditional independence relation, i.e., Eqs. (1.7). These properties can be seen as inference rules that can be used to derive new independences from ones that we know to be true. A careful study of these axioms suggests that only two simple inference rules, stated in the *Triangle theorem* below, are sufficient for inferring most of the useful independence information that can be inferred by a systematic application of the inference rules. This was confirmed by our experiments in Section 2.6.

Theorem 2.2 (Triangle theorem)**.** *Given Eqs. (1.7), for every variable X, Y, W and sets $\mathbf{Z}_1$ and $\mathbf{Z}_2$ such that $\{X, Y, W\} \cap \mathbf{Z}_1 = \{X, Y, W\} \cap \mathbf{Z}_2 = \varnothing$,*

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

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

Proof. We are using the Strong Union and Transitivity of Eqs. (1.7) as shown or in contrapositive form.

(Proof of D-triangle rule):

- From Strong Union and $(X \not\perp\!\!\!\perp W \mid \mathbf{Z}_1)$ we get $(X \not\perp\!\!\!\perp W \mid \mathbf{Z}_1 \cap \mathbf{Z}_2)$.
- From Strong Union and $(W \not\perp\!\!\!\perp Y \mid \mathbf{Z}_1)$ we get $(W \not\perp\!\!\!\perp Y \mid \mathbf{Z}_1 \cap \mathbf{Z}_2)$.
- From Transitivity, $(X \not\perp\!\!\!\perp W \mid \mathbf{Z}_1 \cap \mathbf{Z}_2)$, and $(W \not\perp\!\!\!\perp Y \mid \mathbf{Z}_1 \cap \mathbf{Z}_2)$, we get $(X \not\perp\!\!\!\perp Y \mid \mathbf{Z}_1 \cap \mathbf{Z}_2)$.

(Proof of I-triangle rule):

- From Strong Union and $(W \not\perp\!\!\!\perp Y \mid \mathbf{Z}_1 \cup \mathbf{Z}_2)$ we get $(W \not\perp\!\!\!\perp Y \mid \mathbf{Z}_1)$.
- From Transitivity, $(X \perp\!\!\!\perp W \mid \mathbf{Z}_1)$ and $(W \not\perp\!\!\!\perp Y \mid \mathbf{Z}_1)$ we get $(X \perp\!\!\!\perp Y \mid \mathbf{Z}_1)$.

□

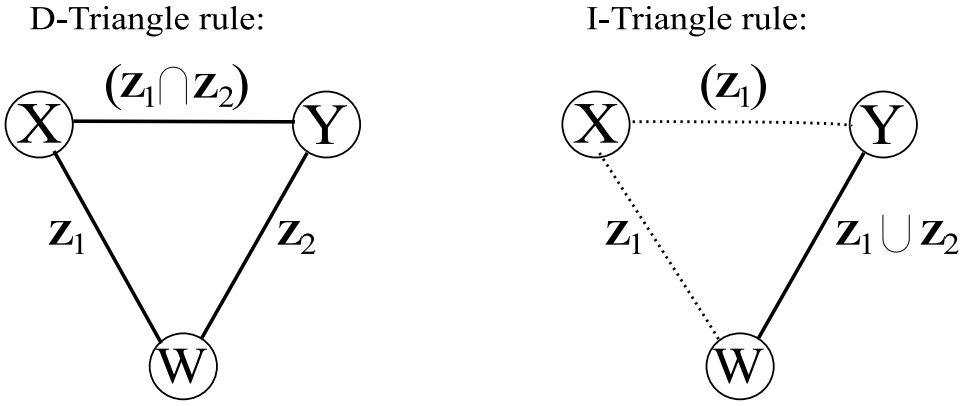


Figure 2.3 Independence graph depicting the Triangle theorem. Edges in the graph are labeled by sets and represent conditional independences or dependencies. A solid (dotted) edge between X and Y labeled by $\mathbf{Z}$ means that X and Y are dependent (independent) given $\mathbf{Z}$. A set label enclosed in parentheses means the edge was inferred by the theorem.

We can represent the Triangle theorem graphically using the independence graph construct of Section 2.3. Figure 2.3 depicts the two rules of the Triangle theorem using two independence graphs.

The Triangle theorem can be used to infer additional conditional independences from tests conducted during the operation of GSMN. An example of this is shown in Figure 2.4(a), which illustrates the application of the Triangle theorem to the example presented in Figure 2.2(a). The independence information inferred from the Triangle theorem is shown by curved edges (note that the conditioning set of each such edge is enclosed in parentheses). For example, independence edge $(4, 7)$ can be inferred by the D-triangle rule from the adjacent edges $(5, 4)$ and $(5, 7)$, annotated by $\{3\}$ and $\{3, 4, 6\}$ respectively. The annotation for this inferred edge is $\{3\}$, which is the intersection of the annotations $\{3\}$ and $\{3, 4, 6\}$. An example application of the I-triangle rule is edge $(1, 7)$, which is inferred from edges $(5, 1)$ and $(5, 7)$ with annotations $\{3, 4\}$ and $\{3, 4, 6\}$ respectively. The annotation for this inferred edge is $\{3, 4\}$, which is the intersection of the annotations $\{3, 4, 6\}$ and $\{3, 4\}$.

2.5 The GSIMN Algorithm

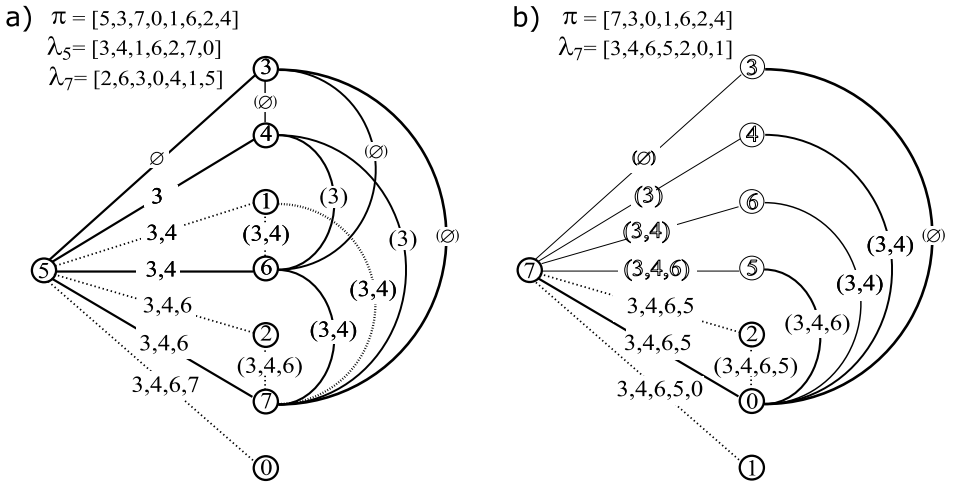


Figure 2.4 Illustration of the operation of GSIMN. The figure shows the grow phase of two consecutively visited variables 5 and 7. Contrary to GSMN (Figure 2.2), the variable visited second is not 3 but 7, according to the change in the visit order π in line 31. The set of variables enclosed in parentheses correspond to tests inferred by the Triangle theorem using two adjacent edges as antecedents. For example, the results $(7 \not\perp\!\!\!\perp 3 \mid \varnothing)$, $(7 \not\perp\!\!\!\perp 4 \mid \{3\})$, $(7 \not\perp\!\!\!\perp 6 \mid \{3,4\})$, and $(7 \not\perp\!\!\!\perp 5 \mid \{3,4,6\})$ in (b) were not executed but inferred from the tests done in (a).

In the previous section we saw the possibility of using the two rules of the Triangle theorem to infer the result of novel tests during the grow phase. The GSIMN algorithm (Grow-Shrink Inference-based Markov Network learning algorithm), introduced in this section, uses the Triangle theorem in a similar fashion to extend GSMN by inferring the value of a number of tests that GSMN executes, making their evaluation unnecessary.

GSIMN is shown in Algorithm 5. It works in a fashion similar to GSMN, but differs in three important ways, tailored to maximize the number of inferences that are made possible through the use of the Triangle theorem. First, the grow test conditions on $\mathbf{S}$ instead of $\mathbf{S} \cup \mathbf{T}$. Second, while visiting variable X, GSIMN updates the visit order and the grow order of every variable $Y \in \lambda_X$ (the π and λ_Y queues respectively). (Since $X \notin \lambda_X$, this excludes the grow order of X itself.) Third, and most importantly, it uses a new test procedure I' (shown in Algorithm 6), that attempts to infer the value of the independence test that is provided as its input by either Strong Union or the Triangle theorem. If this succeeds, I' returns the value

Algorithm 5 $GSIMN(\mathbf{V}, D)$.

```
1: /* Initialization. */
2: for all X, Y ∈ V, X ≠ Y do
3:     p_XY ← G(X, Y | ∅)
4:     t ← (p_XY > α)
5:     K_XY ← (∅, t)
6:     K_YX ← (∅, t)
7: initialize π such that i < i' ⟺ avg_j log(p_{π_i j}) < avg_j log(p_{π_{i'} j})
8: for all X ∈ V do
9:     B^X ⟵ ∅
10:    initialize λ^X such that j < j' ⟺ p_{X λ^X_j} < p_{X λ^X_{j'}}
11:    remove X from λ^X
12: /* Main loop. */
13: while π not empty do
14:    X ← dequeue(π)
15:    /* Propagation phase. */
16:    T ⟵ {Y : Y was visited and X ∈ B^Y}
17:    F ⟵ {Y : Y was visited and X ∉ B^Y}
18:    for all Y ∈ T ∪ F remove Y from λ^X
19:    /* Grow phase. */
20:    S ← ∅
21:    while λ^X not empty do
22:        Y ← dequeue(λ^X)
23:        if p_XY < (1 − α) then
24:            if ¬I'(X, Y | S) then
25:                S ← S ∪ {Y}
26:                /* Change grow order of Y. */
27:                changepos(λ_Y, X, 0)
28:                for W = S_{|S|−2} to S_0 do
29:                    changepos(λ_Y, W, 0)
30:    /* Change visit order. */
31:    for W = S_{|S|−1} to S_0 do
32:        if W ∈ π then
33:            changepos(π, W, 0)
34:            goto 35
35:    /* Shrink phase. */
36:    for Y = S_{|S|−1} to S_0 do
37:        if I(X, Y | S ∪ T − {Y}) then
38:            S ← S − {Y}
39:    B^X ← S ∪ T
40: return {B^X : X ∈ V}
```

inferred, otherwise it defaults to a statistical test on the data set (as I does). For the purpose of assisting in the inference process, the GSIMN and I' algorithms maintain a knowledge base K_{XY} for each pair of variables X and Y, containing the outcomes of all tests done so far between X and Y (from data or inferred). Each knowledge base is created empty during the

Algorithm 6 $I'(X, Y \mid \mathbf{S})$: Calculate independence test result by inference, if possible. Record test result in the knowledge base.

```
1: /* Attempt to infer dependence by Strong Union. */
2: if ∃(A, false) ∈ K_XY such that A ⊇ S then
3:     return false
4: for all W ∈ S do
5:     if ∃(A, false) ∈ K_XW such that A ⊇ S ∧ ∃(B, false) ∈ K_WY such that B ⊇ S then
6:         /* Infer dependence by the D-triangle rule. */
7:         add (A ∩ B, false) to K_XY and K_YX
8:         return false
9: /* Attempt to infer independence by Strong Union. */
10: if ∃(A, true) ∈ K_XY such that A ⊆ S then
11:     return true
12: for all W ∈ S do
13:     if ∃(A, true) ∈ K_XW such that A ⊆ S ∧ ∃(B, false) ∈ K_WY such that B ⊇ A then
14:         /* Infer independence by the I-triangle rule. */
15:         add (A, true) to K_XY and K_YX
16:         return true
17: /* Else do statistical test on data. */
18: t ← G(X, Y | S)
19: add (S, t) to K_XY and K_YX
20: return t
```

initialization phase of GSIMN (lines 5 and 6) and is maintained within the test procedure I'.

The first difference between GSMN and GSIMN, namely the conditioning on $\mathbf{S}$ instead of $\mathbf{S} \cup \mathbf{T}$, is straightforward. We now describe the second difference, namely the new ordering used in GSIMN. After the end of the grow phase for variable X, the new visit order π (set in lines 31–34) dictates that the next variable to be visited after X is the last to be added to $\mathbf{S}$ during the growing phase that has yet to be visited (i.e., still in π). For example, in Figure 2.4, the variable visited after 5 is 7 instead of 3 (as was dictated by the initial π and as would have been done in GSMN). The change in order is conducted by the subroutine *changepos*(q, Y, j) which moves Y from its current position in queue q to position j. For example, if $q = [5, 3, 7, 0, 1, 6, 2, 4]$, then after applying *changepos*$(q, 7, 0)$ the queue becomes $q = [7, 5, 3, 0, 1, 6, 2, 4]$.

For every $Y \in \lambda^X$, the change in the growing order λ_Y occurs inside the grow phase of the currently visited variable X (lines 26–29). If, for some variable Y, the algorithm reaches line 25, i.e., $p_{XY} < (1 - \alpha)$ and $I'(X, Y, \mathbf{S}) = \texttt{false}$, then X and all the variables that were found dependent with X before Y (i.e., all variables currently in $\mathbf{S}$) are promoted to the beginning of

the grow order λ_Y. This is illustrated in Figure 2.4 for variable 7, in which the grow order of 7 changes from $\lambda_7 = [2, 6, 3, 0, 4, 1, 5]$ to $\lambda_7 = [3, 4, 6, 5, 2, 0, 1]$ after the grow phase of variable 5 is complete; the variables 5, 6, 4 and 3 were promoted (in that order) to the beginning of the queue. The rationale for this is the observation that this maximizes the number of tests inferred at a future step—we explain this in more detail below.

The third difference between GSMN and GSIMN is the function I', shown in Algorithm 6, which replaces function I of GSMN. I' attempts to infer the independence value of its input triplet $(X, Y \mid \mathbf{S})$ by applying a single step of backward chaining using the Strong Union and Triangle rules i.e., it searches the knowledge base K for antecedents of instances of rules that have the input triplet $(X, Y \mid \mathbf{S})$ as consequent. The Strong Union rule is used in its direct from as shown in Eqs. (1.7) and also in its contrapositive form. The direct form can be used to infer independences, and therefore we refer to it as the I-SU rule hereon. In its contrapositive form, the I-SU rule becomes $(X \not\perp\!\!\!\perp Y \mid \mathbf{S} \cup \mathbf{W}) \implies (X \not\perp\!\!\!\perp Y \mid \mathbf{S})$, referred to as the D-SU rule since it can be used to infer dependencies. According to the D-Triangle and D-SU rules, the dependence $(X \not\perp\!\!\!\perp Y \mid \mathbf{S})$ can be inferred if the knowledge base $\{K_{XY} : X, Y \in \mathbf{V}\}$ contains

1. a test $(X \not\perp\!\!\!\perp Y \mid \mathbf{A})$ with $\mathbf{A} \supseteq \mathbf{S}$, or

2. tests $(X \not\perp\!\!\!\perp W \mid \mathbf{A})$ and $(W \not\perp\!\!\!\perp Y \mid \mathbf{B})$ for some variable W, with sets $\mathbf{A}$ and $\mathbf{B}$ both supersets of $\mathbf{S}$,

respectively. According to the I-Triangle and I-SU rules, the independence $(X \perp\!\!\!\perp Y \mid \mathbf{S})$ can be inferred if the knowledge base contains

3. a test $(X \perp\!\!\!\perp Y \mid \mathbf{A})$ with $\mathbf{A} \subseteq \mathbf{S}$, or

4. tests $(X \perp\!\!\!\perp W \mid \mathbf{A})$ and $(W \not\perp\!\!\!\perp Y \mid \mathbf{B})$ for some variable W, with $\mathbf{A} \subseteq \mathbf{S}$ and $\mathbf{B} \supseteq \mathbf{A}$,

respectively.

A subtle but important issue that must be discussed is the order of application of the I-SU, D-SU, I-Triangle and D-Triangle rules within the function I'. Given an independence-query oracle, the order of application should not matter—assuming there are more than one rules or

inferring the value of an independence, all of them are guaranteed to produce the same value due to the soundness of the axioms of Eqs. (1.7) (Pearl, 1988). As we mentioned however, in practice the oracle is implemented by statistical tests conducted on data, which can be incorrect. Of particular importance is the observation that false independences are more likely to occur than false dependencies. One example of this is the case where the data sets small or dependencies are weak—in this case any pair of variables connected in the underlying true network structure may be incorrectly deemed independent if the paths between them are long enough. While this is not always true when multiple paths between the variables exist (that can cancel the influence of each other) in practice this only happens rarely and weakly dependent variables are common, especially in large domains (large numbers of variables) with sparse structure. On the other hand, false dependencies are much more rare—the confidence threshold of $\alpha = 0.95$ of a statistical test tells us that the probability of a false dependence by chance is only 5%. Assuming i.i.d. data for each test, the chance of multiple false dependencies is even lower, decreasing exponentially fast. This practical observation i.e., that dependencies are typically more reliable than independences, provide the rationale for the way the I' algorithm works. In particular, I' prioritizes the application of rules whose antecedents contain dependencies first i.e., the D-Triangle and D-SU rules, followed by the I-Triangle and I-SU rules. In effect, this uses statistical results that are typically known with greater confidence before ones that are usually less reliable.

Similarly to GSMN, it can be shown that under the same assumptions each set $\mathbf{B}^X$ returned by the GSIMN algorithm contains exactly the neighbors of X. The proof of correctness of GSIMN is based on correctness of GSMN and is presented in detail in Bromberg et al. (2007).

2.6 Experimental Results

We evaluated the GSMN and GSIMN algorithms on both artificial and real-world data sets. Through the experimental results presented below we show that both algorithms require a polynomial number of tests (w.r.t. the number of variables in the domain) to learn the structure. Also, that a simple application of Pearl's inference rules in GSIMN algorithm

results in a significant reduction in the number of tests performed when compared to GSMN without adversely affecting the quality of the output network.

We report the following quantities:

- **Weighted number of tests.** The weighted number of tests executed by GSMN and GSIMN reflects the time complexity of the algorithm. A statistical test on triplet $(X, Y \mid \mathbf{Z})$ is proportional to the size N of the data set and the number of variables involved in it i.e., $O(N(|\mathbf{Z}| + 2))$ (and is not exponential in the number of variables involved as a naïve implementation might assume). Therefore, a polynomial number of tests implies a polynomial time complexity.

- **Quality of the resulting network.** We measure quality in two ways.

  - **Accuracy.** We compare the result (`true` or `false`) of a number of conditional independence tests on the network output (using vertex separation), to the same tests performed either on the structure of the underlying model (when that is known i.e., in the artificial data experiments) or on the data (when the underlying model is unknown, i.e., for real-world data sets).

  - **Hamming distance.** The Hamming distance between the output network and the structure of the underlying model is another measure of the quality of the output network, when the actual network that was used to generate the data is known. The Hamming distance is defined as the number of "flipped" edges between these two network structures.

In the next section we present results for domains in which the underlying probabilistic model was known. This is followed by real-world data experiments where no model structure was available.

2.6.1 Known-Model Experiments

We first evaluated our algorithm in artificial domains in which the structure of the underlying model, called *true network* or *true model*, is known. This allowed (i) a systematic study

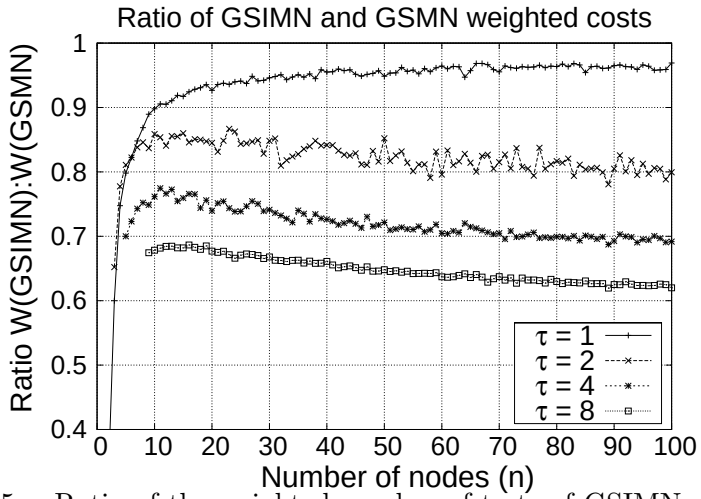


Figure 2.5 Ratio of the weighted number of tests of GSIMN over GSMN for network sizes (number of nodes) $n = 1$ to $n = 100$ and average degrees $\tau = 1$, 2, 4, and 8.

of its behavior under varying conditions of domain sizes n and amount of dependencies (reflect in the number of edges m in the true network), and, (ii) a better evaluation of quality of the output networks because the true model is known. True networks were generated randomly by selecting the first $m = \tau\binom{n}{2}$ pairs in a random permutation of all possible edges, τ being a *connectivity* parameter.

We conducted two types of experiments using known network structure: *Exact learning* experiments and *sample-based* experiments, presented in this order below.

2.6.1.1 Exact Learning Experiments

In this section we show the results of experiments for which conditional independence tests were conducted directly on the true network through vertex-separation. This presents two benefits: (i) it ensure faithfulness and correctness of the independence tests, which allows the evaluation of the algorithms under their assumptions for correctness; and (ii) these tests can be performed much faster than actual statistical tests on data, allowing the evaluation of our algorithms in large networks (up to 100 variables).

We first report the weighted number of tests executed by GSMN and GSIMN. Our results are summarized in Figure 2.5, which shows the ratio between the weighted number of tests

of GSIMN and GSMN. One hundred true networks were generated randomly for each pair (n, τ), and the figure shows the mean value. For large domains ($n \geq 20$), we see that the reduction in weighted number of tests stays approximately constant with increasing domain size n, depending only on the average degree parameter τ. The reductions for large n are approximately 5%, 20%, 30%, and 38% for $\tau = 1$, 2, 4, and 8 respectively, demonstrating the benefit of GSIMN vs. GSMN in terms of number of tests executed.

2.6.1.2 Sample-based Experiments

In these sets of experiments we evaluated GSMN and GSIMN on data sampled from the true model. This allows a more realistic assessment of the performance of our algorithm in terms of number of tests required. We therefore performed experiments on data sets sampled from known Markov networks using Gibbs sampling.

In the exact learning experiments only the structure of the true network was required, generated randomly in the fashion described above. For sampled data experiments however, we also need to specify the network parameters. For each random network, the parameters determine the strength of dependencies among connected variables in the graph. Following Agresti (2002), the strength of the probabilistic influence between two binary variables X and Y can be measured by the *log-odds ratio* 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 every pair of variables connected by an edge in the graph has a specified value. In this set of experiments, we chose $\theta = 2$ for every such pair.

In our experiments we compare the weighted number of tests, accuracy and Hamming distance of the network structures output by GSIMN or GSMN vs. the true network structure.

Figure 2.6 shows the weighted number of tests of GSIMN vs. GSMN for $|D| = 20000$ and domains $n = 20$, 50, and 75 and average degree parameters $\tau = 1$, 2, 4, and 8. In all cases GSIMN shows a reduced weighted number of tests. For sparse networks i.e., $\tau = 1$, this reduction is larger than 50% for all three domain sizes, a reduction much larger than the one

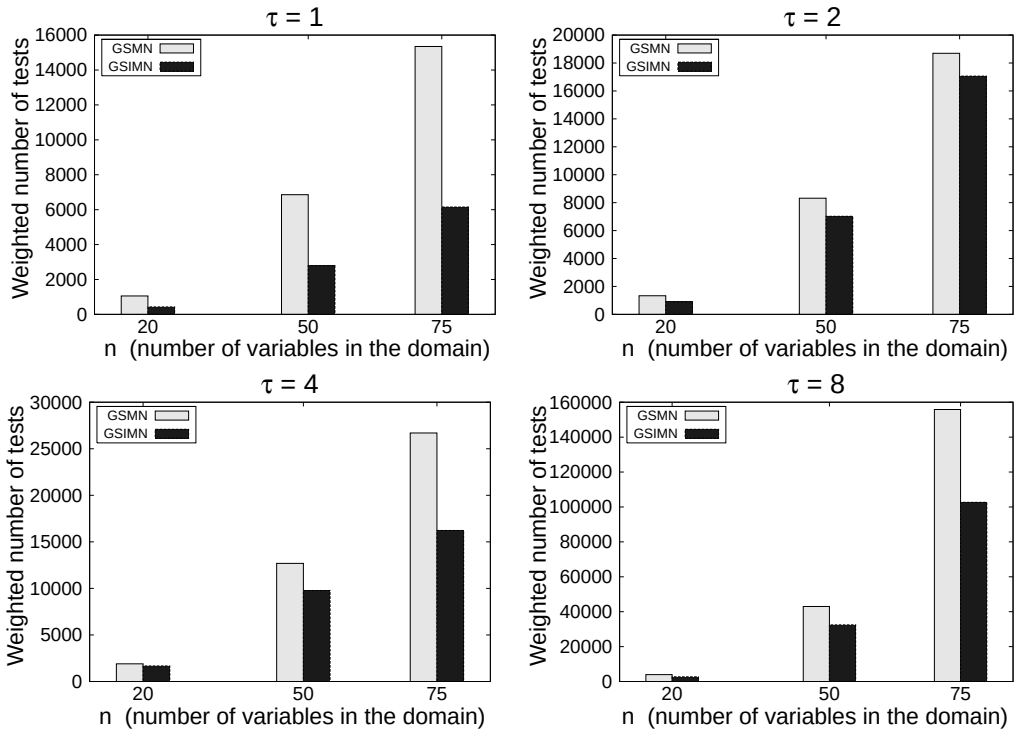


Figure 2.6 Weighted number of tests of GSIMN vs GSMN at $|D| = 20000$, for domains sizes $n = 20$, 50, and 75 and average degree parameters $\tau = 1$, 2, 4, and 8.

observed for the exact learning experiments. For $\tau = 4$ and $\tau = 8$ the reduction observed for $n = 20$ and $n = 50$ is smaller than the one observed for exact learning, but substantial nonetheless.

Testing for conditional independence in these experiments was conducted using χ^2 statistical tests on data sampled from the true model using a Gibbs sampler. For small data sets, these statistical tests may be unreliable, and thus the recovered network may differ from the true one. We therefore report the estimated accuracy of the output network on independence tests and its Hamming distance to the true network. The estimated independence accuracy of a network produced by GSMN or GSIMN was calculated by comparing the result (`true` or `false`) of a number of conditional independence tests on the network (using vertex separation) to the same tests performed on the true network (also using vertex separation). This approach is similar to estimating accuracy in a classification task over unseen instances but with inputs here being triplets $(X, Y, \mathbf{Z})$ and the class attribute being the value of the corresponding con-

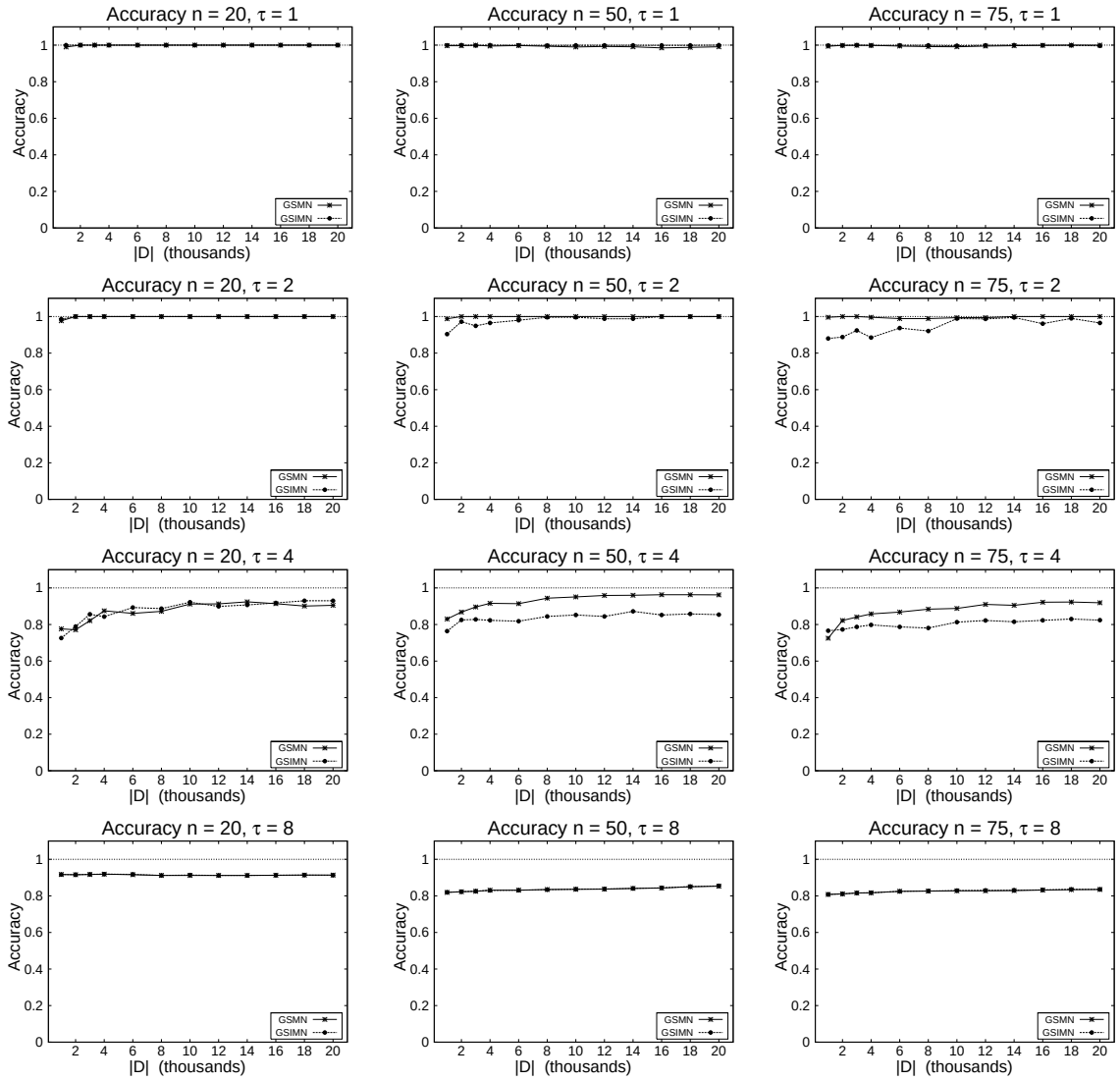


Figure 2.7 Estimated independence accuracy of GSIMN and GSMN for domain sizes $n = 20, 50, 75$ and average degrees $\tau = 1, 2, 4, 8$.

ditional independence test. Since the number of possible tests is exponential, we estimated the independence accuracy by sampling 10,000 triplets $(X, Y, \mathbf{Z})$ randomly, evenly distributed among all possible conditioning set sizes $m \in \{0, \ldots, n-2\}$ (i.e., $10000/(n-1)$ tests for each m). Each of these triplets was constructed as follows: First, two variables X and Y were drawn randomly from $\mathbf{V}$. Second, the conditioning set was determined by picking the first m variables from a random permutation of $\mathbf{V} - \{X, Y\}$. Denoting by $\mathcal{T}$ this set of 10,000 triplets, by $t \in \mathcal{T}$ a triplet, by $I_{\text{true}}(t)$ the result of a test performed on the true network, and by $I_{\text{out}}(t)$

the result of a test performed on the output network produced by either GSMN or GSIMN, the estimated accuracy is defined as:

$$\widehat{accuracy} = \frac{1}{|\mathcal{T}|}\left|\left\{t \in \mathcal{T} \mid I_{\text{out}}(t) = I_{\text{true}}(t)\right\}\right|.$$

Figure 2.7 shows plots of the accuracy of GSIMN and GSMN vs. the data set size $|D|$. We see that the accuracy of GSIMN approaches that of GSMN for large enough data sets, for all domain sizes $n = 20, 50, 75$ and average degrees $\tau = 1, 2, 4, 8$ with the exception of $(n = 50, \tau = 4)$ and $(n = 75, \tau = 4)$ which exhibit some modest discrepancies. These results indicate that the savings in the weighted number of tests obtained by using GSIMN over GSMN is not at the expense of accuracy in most scenarios.

Figure 2.8 shows plots of the normalized Hamming distance between the output network of GSIMN and GSMN and the true network. The normalization factor is $\binom{n}{2}$, the total number of node pairs. These plots show that the Hamming distance of GSIMN is comparable to the one of GSMN for all domain sizes $n = 20, 50, 75$ and all average degrees $\tau = 1, 2, 4, 8$ with the exception of $(n = 50, \tau = 4)$, in which approximately 7% more edges are incorrect in GSIMN than in GSMN, and $(n = 75, \tau = 4)$, where this difference is less than 4%. This reinforces the claim that inference done by GSIMN has a small impact on the reduction in quality of the output networks.

2.6.2 Real-World Data Experiments

While artificial data set studies have the advantage allowing a more controlled and systematic study of the performance of the algorithms (see previous section), experiments on real-world data are necessary for a more realistic assessment of their performance. Real data are more challenging because they may come from non-random topologies (e.g., a possibly irregular lattice in many cases of spatial data) and the underlying probability distribution may not be faithful.

We conducted experiments on a number of data sets obtained from the UCI machine learning data set repository (D.J. Newman and Merz, 1998a). For each data set and each

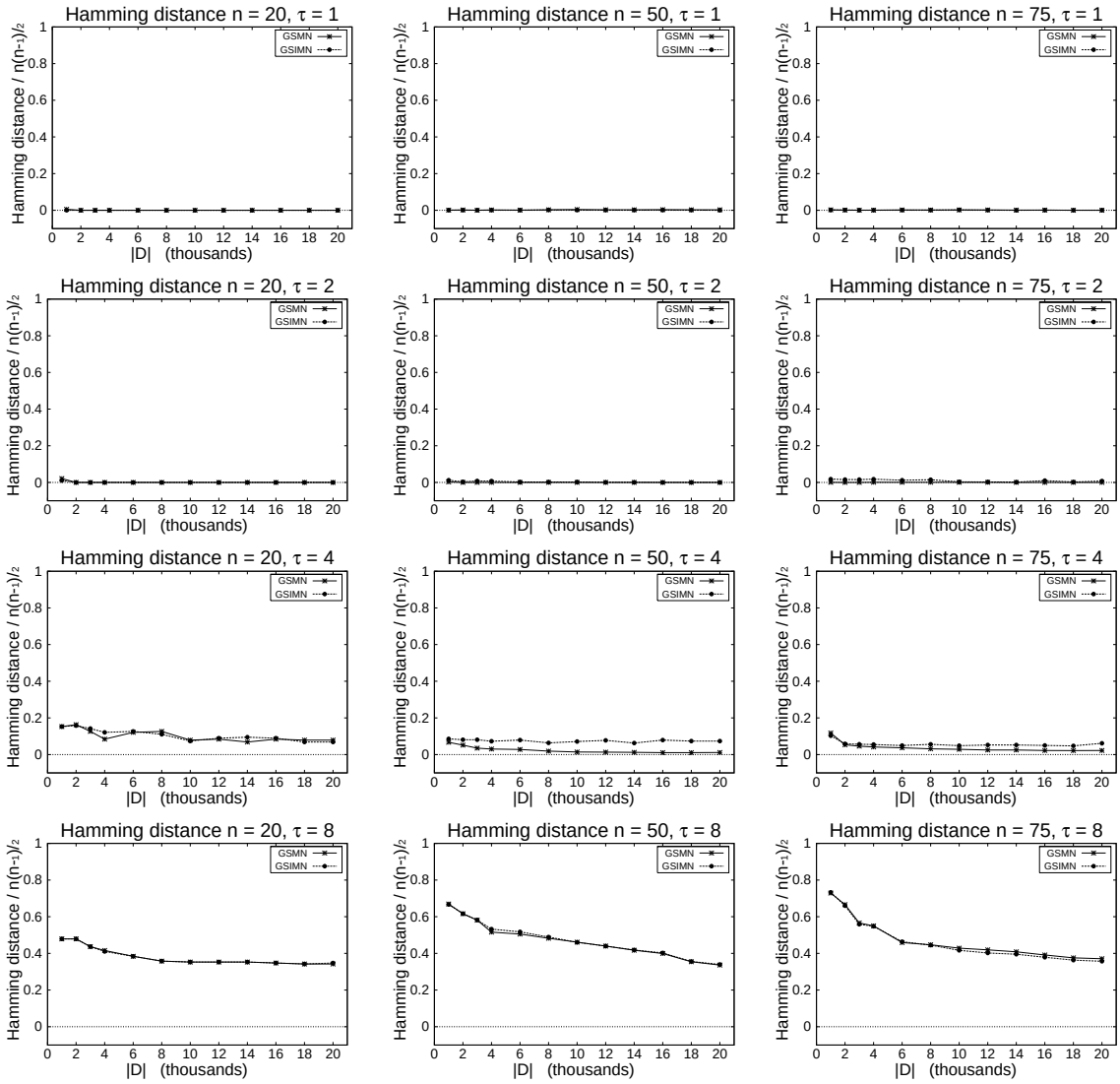


Figure 2.8 Ratio of the Hamming distance between the output network of GSIMN and GSMN and the true network, normalized by $\binom{n}{2}$, for domain sizes $n = 20, 50, 75$ and average degrees $\tau = 1, 2, 4, 8$.

algorithm, we report the weighted number of conditional independence tests conducted to discover the network and the accuracy, as defined below.

Accuracy for real data is defined similarly to the case for sampled data. The main difference is that we compare the result (`true` or `false`) of a number of conditional independence tests on the network (using vertex separation) to the result of the same tests performed on the data set (using a χ^2 test). This is necessary because for real data the true model is unknown. Denoting

Table 2.1 Weighted number of tests and accuracy for several real-world data sets. For each evaluation measure, the best performance between GSMN 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 #(tests)		Accuracy	
#	name	n	N	GSMN	GSIMN	GSMN	GSIMN
1	echocardiogram	14	61	1152	**524**	0.646	**0.665**
2	ecoli	9	336	186	**101**	**0.719**	0.694
3	lenses	5	24	59	**45**	0.500	0.500
4	hayes-roth	6	132	105	**60**	0.584	0.584
5	hepatitis	20	80	1001	**416**	0.604	**0.606**
6	cmc	10	1473	296	**232**	**0.779**	0.757
7	balance-scale	5	625	80	**59**	1	1
9	flag	29	194	2316	**1209**	0.535	**0.550**
10	tic-tac-toe	10	958	344	**184**	0.473	**0.683**
11	bridges	12	70	563	**253**	0.482	**0.706**
12	car	7	1728	174	**161**	**0.699**	0.666
13	monks-1	7	556	96	**42**	0.919	0.919
14	haberman	5	306	68	**45**	0.691	0.691
15	nursery	9	12960	362	**342**	**0.627**	0.599
16	crx	16	653	1118	**561**	0.436	**0.679**
17	imports-85	25	193	2409	**1631**	**0.424**	0.371
18	dermatology	35	358	5004	**2723**	**0.535**	0.502
19	adult	10	32561	578	**438**	0.604	0.604
20	alarm	37	20001	5164	**2667**	0.704	**0.774**
21	bands	38	277	11237	**2687**	**0.508**	0.504
22	connect-4	43	65535	15275	**8359**	0.443	**0.473**

by $\mathcal{T}$ the set of 10,000 triplets, by $t \in \mathcal{T}$ a triplet, by $I_{\text{data}}(t)$ the result of a test performed on the data, and by $I_{\text{out}}(t)$ the result of a test performed on the output network produced by either GSMN or GSIMN, the accuracy is defined as:

$$\widehat{accuracy} = \frac{1}{|\mathcal{T}|}\left|\left\{t \in \mathcal{T} \mid I_{\text{network}}(t) = I_{\text{data}}(t)\right\}\right|. \tag{2.2}$$

For each of the data sets, Table 2.1 shows the detailed results for accuracy and the weighted number of tests for GSMN and GSIMN. It also serves as key for each data set index appearing in Figure 2.9, which plots two quantities in the same graph for these real-world data sets: the ratio of the weighted number of tests of GSIMN versus GSMN and the difference of the

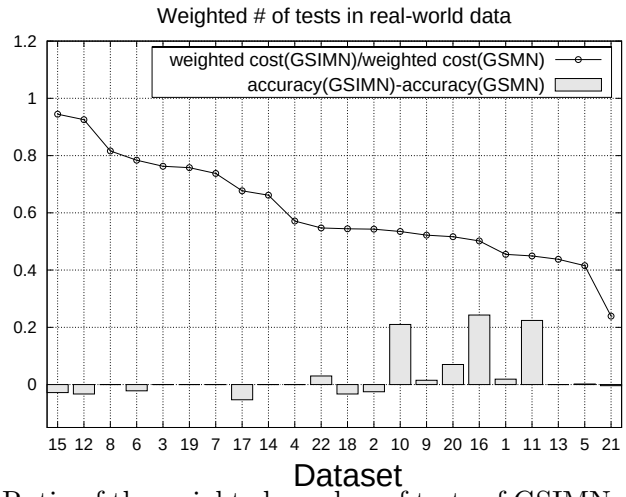


Figure 2.9 Ratio of the weighted number of tests of GSIMN versus GSMN and difference between the accuracy of GSIMN and GSMN on real data sets. Ratios smaller that 1 and positive bars indicate an advantage of GSIMN over GSMN. The numbers in the x-axis are indices of the data sets as shown in Table 2.1.

accuracy between GSIMN and GSMN. The numbers in the x-axis are indices to the data sets as shown in Table 2.1. For each data set, an improvement of GSIMN over GSMN corresponds to a number smaller than 1 for the ratios and a positive histogram bar for the accuracy differences. We can observe that GSIMN reduced the weighted number of tests on every data set, with maximum savings of 75%. Moreover, in 8 out of 22 data sets GSIMN resulted in improved accuracy, with 3 of these (11, 12, and 17) showing a considerable improvement (close to 20%) in addition to an approximate average savings of 50% in the weighted number of tests. Among the 14 remaining data sets, 7 resulted in the same accuracy, and the other 7 resulted in decrease in accuracy of less than 5% in the worst case.

2.7 Summary

In this chapter we presented two algorithms, GSMN and GSIMN, for learning the structure of a Markov network of a domain from data using the independence-based approach. We evaluated their performance through measurement of the weighted number of tests they require to learn the structure of the network and the quality of the networks learned from both artificial and real-world data sets. GSIMN showed a decrease in the weighted number of tests of up

to 38% for difficult (large) artificial domains and up to 75% for real-world data, and output network quality comparable to that of GSMN, with some cases showing improvement. In addition, GSIMN was shown to be nearly optimal in the number of tests executed compared to GSIMN-FCH, which uses an exhaustive search to produce all independence information that can inferred from Pearl's axioms. Some directions of future research include an investigation into the way the topology of the underlying Markov network affects the number of tests required and quality of the resulting network, especially for commonly occurring topologies such as grids. Another research topic is the impact on number of tests of other visit and grow orderings of the variables.

CHAPTER 3. PFMN algorithm

As in the previous chapter, we focus here on the task of learning the structure of Markov networks (**MNs**), a subclass of graphical models, from data in discrete domains. In this chapter we introduce **PFMN** (**P**article **F**ilter **M**arkov **N**etwork) (Bromberg and Margaritis, 2007), a novel independence-based algorithm for the induction of Markov network structures. Existing independence-based algorithms such as the GSMN and GSIMN algorithms discussed in the previous chapter present the disadvantage of being potential inefficient with regard to the number of tests required to learn the structure due to the relatively rigid (predefined) order in which tests are performed.

We address this problem through a Bayesian particle filtering approach, which uses a population of Markov network structures to maintain the posterior probability distribution over them, given the outcomes of the tests performed so far. We avoid the inefficiencies of previous approaches by greedily selecting, at each step, the optimally informative from a pool of candidate tests according to information gain. As such the approach can be seen as an instance of active learning (Tong and Koller, 2001).

The rest of the chapter is organized as follows: In the next section we present some notation, followed by a description of a generative model considered for our problem. Following that, we present our approach including a detailed explanation of PFMN, our main algorithm. We then present experimental results and conclude with a summary of our approach.

3.1 Notation

In this section we modify our notation slightly to facilitate the exposition of new concepts introduced in this chapter. We denote by $\mathbf{V}$ the set of $n = |\mathbf{V}|$ random variable in the domain.

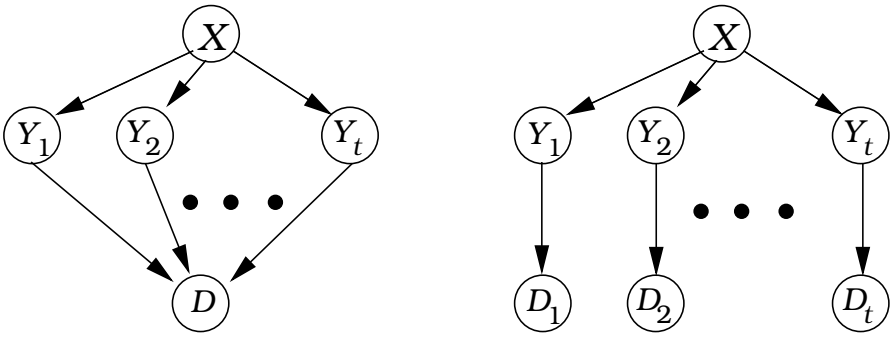


Figure 3.1 Generative model of domain. **Left:** Correct. **Right:** Assumed.

We use capital letters $A, B, \ldots$ to denote domain random variables and bold letters for sets of variables (e.g., $\mathbf{S}$). The space of all structures (given $\mathbf{V}$) is denoted by $\mathcal{X}$ and the space of all conditional independence tests by $\mathcal{Y}$. The structure of a Markov network consists of an undirected graph $x = (\mathbf{V}, E)$ whose nodes represent the random variables in $\mathbf{V}$ and its set of edges E encodes the set of conditional independences in the domain through vertex separation. As usual, conditional independence of A and B given $\mathbf{S}$ is denoted by $(A \perp\!\!\!\perp B \mid \mathbf{S})$. In this chapter we also assume Faithfulness. Therefore, the set of conditional independences implied by the structure of a MN (through vertex separation) are exactly those that hold in the actual probability distribution of the domain i.e., $(A \perp\!\!\!\perp B \mid \mathbf{S})$ if and only if A and B are separated in the MN graph after removing all nodes in $\mathbf{S}$ (and all edges adjacent to them). Faithfulness excludes certain distributions that are unlikely to happen in practice, and is needed for proofs of correctness. It is therefore a common assumption of independence-based algorithms for graphical model discovery, and as such we also assume it here.

As described later in our main algorithm, we maintain a populations of structures at each time step t; slightly abusing our notation we denote these populations as $\mathcal{X}_t$. We denote a sequence of t tests $Y_1, \ldots, Y_t$ by $Y_{1:t}$ and a sequence of value assignments to these tests (independence or dependence, corresponding to `true` and `false` respectively) by $y_{1:t}$.

3.2 Generative Model over Structures and Tests

For an input data set d, our approach uses the posterior probability over structures $\Pr(X \mid d), X \in \mathcal{X}$, to learn the structure of the underlying model as explained in more detail in the next section. For that probability to be calculated in a principled way, we use a generative model that involves random variables $X, Y_1, Y_2, \ldots, Y_t$ and D. Variable X models the structures and its domain is $\mathcal{X}$, the set of all possible structures. Variable Y_i models a single conditional independence test and its domain is binary, with $Y_i = \texttt{true}$ ($Y_i = \texttt{false}$) corresponding to the state of the world in which test Y_i evaluates to independence (dependence). Finally, variable D models the data sets, whose domain is the set of all possible data sets. However, in all our calculations we assume it is already assigned a value, the actual input data set d, e.g., $\Pr(X \mid D = d)$ or $\Pr(X \mid d)$.

Our assumption is that the tests are the *sufficient statistics* for the structure i.e., there is no information in the data set d beyond the value of the tests as far as structure is concerned. This stems from the fact that the structure X faithfully encodes the independences in the domain. Note that this assumption is not particular to our approach, but is implicit in any independence-based approach. Our generative model only formalizes it and makes it explicit.

The generative model that encodes this assumption is shown in Fig. 3.1 (left), where X and D are d-separated by the tests $Y_1, \ldots, Y_t$. However, one problem with this model is that the posterior over structures $\Pr(X \mid d)$ cannot be computed because, according to the model, $\Pr(X \mid d) = \sum_{y_{1:t}} \Pr(X \mid Y_{1:t} = y_{1:t}, d) \Pr(Y_{1:t} = y_{1:t} \mid d)$, which requires the computation of $\Pr(Y_{1:t} = y_{1:t} \mid d)$, the joint outcome of a *set* of tests, currently an unsolved problem (i.e., we can only compute the probability of the outcome of a single test). We therefore assume the model shown in Fig. 3.1 (right), which contains multiple data sets $D_1, \ldots, D_t$ (abbreviated $D_{1:t}$). In this model the data sets D_1 through D_t are independent given tests $Y_{1:t}$. This allows the model to be solved analytically because now $\Pr(Y_{1:t} = y_{1:t} \mid d) \propto \Pr(d \mid Y_{1:t} = y_{1:t}) \Pr(Y_{1:t} = y_{1:t})$, and the first factor decomposes as the product $\prod_{i=1}^{t} \Pr(d_i \mid Y_i = y_i)$. Furthermore, the factors $\Pr(d_i \mid Y_i = y_i)$ can be computed by known procedures such as the discrete version of the Bayesian test of Margaritis (2005), which computes $\Pr(Y_i = y_i \mid d_i)$ by analytically calculating

the data likelihood $\Pr(d_i \mid Y_i = y_i)$ of two competing multinomial models corresponding to $y_i = \texttt{true}$ and $y_i = \texttt{false}$ (the independent and the dependent one), i.e., $\Pr(d_i \mid Y_i = \texttt{true})$ and $\Pr(d_i \mid Y_i = \texttt{false})$. A detailed derivation of an expression for computing the posterior $\Pr(X \mid d)$ is presented in the next section.

In practice we do not have more than one data set, and therefore we use the same data set for all tests i.e., $d_i = d_j$, $i \neq j$. Thus, the model depicted on Fig. 3.1 (right) is used only as an approximation to overcome the lack of an exact solution described above. As we will show in the experiments section, this approximation works well in both artificial and real world data sets.

Under this model, the posterior probability over structures must now be computed given data sets $d_{1:t}$, i.e., $\Pr(X \mid d_{1:t})$, which we abbreviate as $\Pr_t(X)$. This calculation is presented in the next section. Also, in our calculations below we use the conditional entropy of X given the set of tests $Y_{1:t}$ and the data sets $d_{1:t}$, $H(X \,|\, Y_{1:t}, d_{1:t})$, similarly abbreviated as $H_t(X \,|\, Y_{1:t})$.

3.2.1 Posterior Distribution over Structures

To learn the MN structure of a domain from data, we employ a Bayesian approach, calculating the posterior probability $\Pr_t(x) = \Pr(X = x \mid d_{1:t})$ of a structure $x \in \mathcal{X}$ given data sets $d_{1:t}$. We now derive an expression for this posterior.

By Bayes' law we have:

$$\Pr(x \mid d_{1:t}) \propto \Pr(x) \Pr(d_{1:t} \mid x)$$

which, by law of total probability over variables $Y_{1:t}$ equals

$$\Pr(x \mid d_{1:t}) \propto \Pr(x) \sum_{y_{1:t}} \Pr(d_{1:t} \mid y_{1:t}, x) \Pr(y_{1:t} \mid x). \tag{3.1}$$

According to our Faithfulness assumption, the structure x encodes exactly those independences that hold in the underlying probability distribution, i.e., under this assumption the state of a set of conditional independence tests $Y_{1:t}$ is completely determined by x. In terms of probabilities, this is equivalent to saying that $\Pr(Y_{1:t} \mid X = x) \in \{0, 1\}$. That is, there is a single assignment $y_{1:t}$ for the set of tests $Y_{1:t}$ whose probability, conditioned on x, is non-zero.

We denote this assignment by $y^x_{1:t}$, and we say that x is *consistent* with an assignment $y_{1:t}$ when this assignment equals $y^x_{1:t}$. We can summarize these ideas by the following expression,

$$\Pr(Y_{1:t} = y_{1:t} \mid X = x) = \delta(y_{1:t}, y^x_{1:t}), \tag{3.2}$$

where $\delta(a, b)$ is the Kronecker delta function that equals 1 if and only if a is equal to b, and 0 otherwise.

Even though a structure determines the assignment $y^x_{1:t}$ of a sequence of tests $Y_{1:t}$, the reverse does not hold in general i.e., given an assignment $y_{1:t}$, there may be more than one structure consistent with it. If two structures x and x' are consistent with the same assignment $y_{1:t}$ we say they are *equivalent* and we denote this fact by $x \sim_{y_{1:t}} x'$. This equivalence relation partitions the space $\mathcal{X}$ into equivalence classes, one for each assignment $y_{1:t}$, which we denote by $\{y_{1:t}\}$, and we define it formally by

$$\{y_{1:t}\} = \big\{x \in \mathcal{X} \big| x \text{ is consistent with } y_{1:t}\big\}. \tag{3.3}$$

In what follows, whenever we need to emphasize that structure x belongs to some equivalence class we denote it by $\{y^x_{1:t}\}$.

Returning to the derivation of the posterior distribution, we apply Eq. (3.2) to Eq. (3.1) to obtain $\Pr(y_{1:t} \mid x) = \delta(y_{1:t}, y^x_{1:t})$ and thus only the term $y^x_{1:t}$ survives in the sum i.e.,

$$\Pr(x \mid d_{1:t}) \propto \Pr(x) \Pr(d_{1:t} \mid y^x_{1:t}, x).$$

Finally, using the generative model of the previous section, we obtain

$$\Pr(x \mid d_{1:t}) \propto \Pr(x) \prod_{i=1}^{t} \Pr(d_i \mid y^x_i). \tag{3.4}$$

The constant of proportionality in the above equation is independent of x and thus can be calculated by a sum over all structures in $\mathcal{X}$. Since this space is super-exponential in size, this requires an approximation. In the next section we present a principled procedure to approximate this and other quantities that require a summation over $\mathcal{X}$.

One can partition the space $\mathcal{X}$ of all structures in $\binom{n}{2}$ subspaces $\mathcal{X}_h, h = 1, \ldots, \binom{n}{2}$, each containing structures of exactly h edges. The prior $\Pr(X)$ is assumed to be uniform over each

subspace $\mathcal{X}_h$ of $\mathcal{X}$, where $\mathcal{X}_h = \left\{ x = (\mathbf{V}, E) \in \mathcal{X} \;\middle|\; |E| = h \right\}$ consists of all structures with exactly h edges. Therefore, the prior probability of structure $x = (\mathbf{V}, E)$ is

$$\Pr(X = x) = \frac{1}{|\mathcal{X}_{|E|}|}. \tag{3.5}$$

As the size $|\mathcal{X}_h|$ of a subspace $\mathcal{X}_h$ is equal to $\binom{\binom{n}{2}}{h}$, some subspaces may be exponentially larger than others e.g. $\mathcal{X}_1$ vs. $\mathcal{X}_{\binom{n}{2}/2}$. This choice for the prior is made to facilitate the sufficient exploration of small subspaces such as $\mathcal{X}_1$ or $\mathcal{X}_{\binom{n}{2}}$.

The quantity $\Pr(d_i \mid y_i^x)$ in Eq. (3.4) is the likelihood of the data given the assignment y_i^x to test Y_i. To compute these quantities we use the Bayesian test described in Margaritis (2005). This test calculates and compares the likelihoods of two competing multinomial models (with different numbers of parameters), namely $\Pr(d_t \mid Y_t = y_t)$, for $y_t \in \{\texttt{true}, \texttt{false}\}$.

We describe now our approach for solving the structure learning problem.

3.3 Particle Filtering Approach

Traditionally, the problem of learning the structure of a Markov network consists on finding a sequence of triplets $Y^\star_{1:t^\star}$ of minimum cost (defined later in the chapter), such that only a single structure $x^\star$ is consistent with the outcome $y_{1:t^\star}$ of performing an independence test on these triplets. This is always possible due to our assumption of Faithfulness, which guarantees the existence of a single structure consistent with the results of all possible tests in the domain. In our probabilistic framework however, both outcomes `true` or `false` of a test may be assigned a non-zero probability, and therefore structures consistent with both assignments are maintained. The situation in which a single structure $x^\star$ is consistent with the tests can be represented in this framework with an extreme form for the posterior that concentrates all the probability mass in $x^\star$, i.e., $\Pr_{t^\star}(X = x^\star) = 1$ and $\Pr_{t^\star}(X \neq x^\star) = 0$. This state of the posterior distribution has entropy 0, i.e. $H(X \mid Y^\star_{1:t^\star}, d_{1:t^\star}) = 0$. We can therefore summarize the problem of learning a structure in this framework by the following two steps:

1. Finding a sequence of tests $Y^\star_{1:t^\star}$ of minimum cost such that $H(X \mid Y^\star_{1:t^\star}, d_{1:t^\star}) = 0$, and
2. Finding the (unique) structure $x^\star$ such that $\Pr(X = x^\star \mid d_{1:t^\star}) = 1$.

In practice, the above procedure presents considerable difficulties because:

- The space of structures $\mathcal{X}$ is super-exponential: $|\mathcal{X}| = 2^{\binom{n}{2}}$. Thus, the exact computation of the entropy $H_t(X \mid Y_{1:t})$, a sum over all $x \in \mathcal{X}$, is intractable.
- The space of candidate tests $\mathcal{Y}$ is also at least exponential in size: there are $\binom{n}{2}\binom{n-2}{m}$ tests $(A \perp\!\!\!\perp B \mid \mathbf{S})$ with $|\mathbf{S}| = m$, and m ranges from 0 to $n-2$. Moreover, for a given number of tests t, there exist $\binom{|\mathcal{Y}|}{t}$ possible candidate test sequences $Y_{1:t}$ to consider.

We address the first issue using a particle filtering approach. At each step t, we maintain a population $\mathcal{X}_t$ of candidate MN structures (called *particles*) for the purpose of representing the posterior probability distribution over structures given the outcomes of tests performed so far. In this way, all required quantities, such as posterior probability $\Pr_t(x)$ or conditional entropy $H_t(X \mid Y_{1:t})$, can be estimated by simple averaging over the particle population $\mathcal{X}_t$. The theory of Monte Carlo sampling provides a principled procedure for approximating these summations using a sample $\{x^{(i)}\}_{i=1}^N$ of the space $\mathcal{X}$. In our equations, we used the particle population $\mathcal{X}_t$ for such a sample.

Formally, the Monte Carlo principle (Andrieu et al., 2003, p. 5) states that, given an i.i.d. sample $\{x^{(i)}\}_{i=1}^N$ of probability distribution $p(x)$, it is the case that,

$$I_N(f) = \frac{1}{N}\sum_{i=1}^{N} f(x^{(i)}) \xrightarrow[N\longrightarrow\infty]{a.s.} I(f) = \sum_{x\in\mathcal{X}} f(x)p(x). \tag{3.6}$$

The estimate $I_N(f)$ is unbiased and, by the strong law of large numbers, almost surely $(a.s.)$ converges to $I(f)$. To illustrate, let us approximate the normalization constant k of Eq. (3.4) (i.e., $\Pr(x \mid d_{1:t}) = \frac{\Pr(x)\Pr(d_{1:t}|x)}{k}$), a summation over all structures in $\mathcal{X}$, using the set of particles $\mathcal{X}_t = \{x^{(i)}\}_{i=1}^N$ at time t as follows:

$$k = \sum_{x\in\mathcal{X}} \Pr(x)\Pr(d_{1:t} \mid x) \approx \frac{1}{N}\sum_{i=1}^{N} \Pr(d_{1:t} \mid x^{(i)}).$$

The second issue (choosing the next test to perform) is addressed using a greedy search approach. At each step $t+1$ of our algorithm, we choose as the next test to perform the member Y_{t+1} of $\mathcal{Y}$ that minimizes the expected entropy $H_t(X \mid Y_{1:t}, Y_{t+1})$, penalized by a

factor proportional to its cost. Since $\mathcal{Y}$ is exponential in size, the minimization is performed through a heuristic search approach. The next section explains our main algorithm, called PFMN, and following that we present in detail the test selection algorithm.

3.3.1 The PFMN Algorithm

Our algorithm is called **PFMN** (Particle Filter Markov Network structure learner), and is shown in Algorithm 7. At each time step t, the algorithm maintains a set $\mathcal{X}_t$ containing N structure particles.

Initially (line 1), each structure in $\mathcal{X}_0$ is generated by sampling from the prior distribution $\Pr(x)$ of Eq. (3.5). This is done by first generating N structures randomly, and then "moving" each structures M_0 times using the Metropolis-Hastings algorithm, shown in Algorithm 10 and explained in detail in Section 3.3.4 (we used $M_0 = 500$ in our experiments). As explained later, this algorithm requires that a *proposal* distribution $q(x^\star \mid x)$ be provided as parameter. As proposal we used the same proposal used during the particle filter moves of line 9, explained later in Section 3.3.4.

Algorithm 7 Particle Filter Markov Network (PFMN) algorithm.
$x = PFMN(N, M, q(X^\star \mid X))$

1: $\mathcal{X}_0 \longleftarrow$ sample N independent particles distributed according to prior $\Pr(x)$ (c.f. Eq.(3.5)) using the Metropolis-Hastings algorithm (Algorithm 10).
2: $t \longleftarrow 0$
3: **loop**
4: $\quad Y_{t+1} \longleftarrow$ compute $\arg\max_{(A,B)} \arg\max_{\mathbf{S}} score_t(A, B \mid \mathbf{S})$ using Algorithm 8
5: $\quad Y_{1:t+1} \longleftarrow Y_{1:t} \cup \{Y_{t+1}\}$
6: $\quad p_T \longleftarrow \Pr(d_{t+1} \mid Y_{t+1} = \mathtt{t})$ /* *Perform test on data.* */
7: $\quad p_F \longleftarrow \Pr(d_{t+1} \mid Y_{t+1} = \mathtt{f})$ /* *Perform test on data.* */
8: $\quad$ Update $\Pr_{t+1}(X)$ from p_T and p_F using Eq. (3.4).
9: $\quad \mathcal{X}_{t+1} \longleftarrow PF(\mathcal{X}_t, M, \Pr_{t+1}(X), q(X^\star \mid X))$
10: $\quad$ **if** $H_{t+1}(X \mid Y_{1:t+1}) = 0$ **then**
11: $\quad\quad$ **return** $\arg\max_{x \in \mathcal{X}_t} \Pr_t(x)$ /* *Return most probable particle.* */
12: $\quad t \longleftarrow t + 1$

At each time t during the main loop of the algorithm (lines 3–12), the test $Y_{t+1} = (A^\star, B^\star \mid \mathbf{S}^\star) \in \mathcal{Y}$ that optimizes a *score function* is selected. Since for each pair of variables $(A, B) \in \mathbf{V} \times \mathbf{V}$ the space of possible conditioning sets is exponential (equaling the power set of $\mathbf{V} - \{A, B\}$),

this optimization is performed by heuristic search. Both the score function and the optimization algorithm are described in detail in the next section.

The PFMN algorithm continues by computing the data likelihoods of the optimal test Y_{t+1} in lines 6 and 7, which are used to update the posterior over structures and obtain the new posterior $\Pr_{t+1}(X)$ using Eq. (3.4). With this updated distribution, the new set of particles $\mathcal{X}_{t+1}$ is computed in line 9, using the particle filter algorithm described later in section 3.3.4.

The PFMN algorithm terminates when the entropy $H_t(X \mid Y_{1:t})$ (estimated over the population $\mathcal{X}_t$) is zero. This occurs when each equivalence class contains at most one particle. The algorithm then returns the structure x in $\mathcal{X}_t$ with the highest posterior probability.

3.3.2 Optimal Test Selection Algorithm

We now explain the algorithm for selecting the next test to perform used in line 4 of the PFMN algorithm. The algorithm is shown in Alg. 8. It takes as input the set $\mathbf{V}$ of variables in the domain and a score function *score*. Since the space of tests $\mathcal{Y}$ is super-exponential in size, i.e., for each pair of variables $(A, B) \in \mathbf{V} \times \mathbf{V}$ the space of possible conditioning sets is exponential (equal to $\wp(\mathbf{V} - \{A, B\})$, the power set of $\mathbf{V} - \{A, B\}$), this optimization is performed by the heuristic search shown in Algorithm 8, which selects the test $Y_{t+1} = (A^\star, B^\star \mid \mathbf{S}^\star) \in \mathcal{Y}$ that optimizes (locally) the *score function*.

In the main loop (line 3) the algorithm iterates over all pairs (A, B) such that $A, B \in \mathbf{V}$ and $A \neq B$. For each such pair it performs a hill-climbing search in the space $\wp(\mathbf{V} - \{A, B\})$ (lines 8–14) starting with $\mathbf{S} = \varnothing$ and iteratively moving to the "neighbor" set with maximum score (loop of lines 12-13), until no such neighbor exists (i.e., until a global optimum is reached in line 14). During this procedure, the "neighbors" Σ_k of a current point $\mathbf{S}$ are defined to be all sets $\mathbf{S}' \subseteq \mathbf{V} - \{A, B\}$ that are Hamming distance k from $\mathbf{S}$. We can represent a subset $\mathbf{S}$ of $\mathbf{V} - \{A, B\}$ as a binary string $\mathbf{b} = [b_1, \ldots, b_{|\mathbf{V}-\{A,B\}|}]$ of length $|\mathbf{V} - \{A, B\}|$, with $b_i = 1$ if and only if the i-th element of $\mathbf{V} - \{A, B\}$ is in $\mathbf{S}$. Under this representation of a set, the Hamming distance between two sets is the number of bits that differ among their respective binary string representations. The set Σ_k of such neighbors of $\mathbf{S}$ is constructed in line 10.

Initially, the hill-climbing procedure is run for $k = 1$, and k is incremented by 1 (up to a maximum of $k = 4$) while the initial test (i.e., $(A, B \mid \varnothing)$) is a local maximum. The rationale for this extra loop is to improve the chances of getting out of the initial state, when that state is a local maximum. At the end of the iteration for (A, B), the algorithm compares the score of test $(A, B \mid \mathbf{S})$ against $(A^\star, B^\star \mid \mathbf{S}^\star)$, the optimal test so far, and chooses the highest-scoring one. Therefore, at the end of the main loop, $(A^\star, B^\star \mid \mathbf{S}^\star)$ is optimal among all pairs, and the algorithm returns this triplet as the optimal one.

Algorithm 8 Computes test with optimal score. $(A^\star, B^\star \mid \mathbf{S}^\star) =$ optimalTest($\mathbf{V}$, $\mathcal{X}_t$, *score*)

1: $(A^\star, B^\star \mid \mathbf{S}^\star) \leftarrow$ `nil`.
2: */* Find overall optimal test by performing a search over all pairs of variables. */*
3: **for all** $A, B \in \mathbf{V}, A \neq B$ **do**
4: */* Search for conditioning set that optimizes score*$(A, B \mid \mathbf{S})$*. */*
5: $k \leftarrow 0$
6: **repeat** */* while* $k \leq 4$ **/*
7: $k \leftarrow k + 1$
8: $\mathbf{S} \leftarrow \varnothing$
9: **repeat** */* Find local maximum for sets* k *"flips" away. */*
10: $\Sigma_k \leftarrow \{\mathbf{S}' \in \wp(\mathbf{V} - \{A, B\}) \mid hamming(\mathbf{S}', \mathbf{S}) = k\}$ */* Σ_k is set of neighbors of* $\mathbf{S}$*. */*
11: $\mathbf{S}_0 \leftarrow \mathbf{S}$
12: **for all** $\mathbf{S}' \in \Sigma_k$ **do** */* Find optimal neighbor */*
13: **if** $score(A, B \mid \mathbf{S}') > score(A, B \mid \mathbf{S})$ **then** $\mathbf{S} \leftarrow \mathbf{S}'$
14: **until** $(\mathbf{S} = \mathbf{S}_0)$ */* If set* $\mathbf{S}$ *has not changed, its a local maximum, stop. */*
15: **until** $(k > 4 \vee \mathbf{S} \neq \varnothing)$ */* Continue while the initial set* $(\varnothing)$ *is a local maximum. */*
16: */* Record current test* $(A, B \mid \mathbf{S})$ *if its better than optimal */*
17: **if** $\big($ $score(A, B \mid \mathbf{S}) > score(A^\star, B^\star \mid \mathbf{S}^\star)$ $\big)$ **then** $(A^\star, B^\star \mid \mathbf{S}^\star) \leftarrow (A, B \mid \mathbf{S})$
18: **return** $(A^\star, B^\star \mid \mathbf{S}^\star)$

3.3.3 Score Function

We now proceed to define the score function given as input to the test optimization algorithm. The main idea behind our choice of the score function is to select greedily the test that brings us closer to the termination condition of zero entropy. We therefore define the score of

a candidate test Y_{t+1} as follows:

$$score_t(Y_{t+1}) = -\frac{H_t(X \mid Y_{1:t}, Y_{t+1})}{[W(Y_{t+1})]^\beta} \tag{3.7}$$

where the factor $W(Y)$ denotes the cost of Y, which we take to be proportional to the number of variables involved in the test, and β is a constant parameter provided by the user. This factor is used to discourage expensive tests. $H_t(X \mid Y_{1:t}, Y_{t+1})$ is the entropy over structures given the (not yet performed) test Y_{t+1}, and is equal to:

$$H_t(X \mid Y_{1:t}, Y_{t+1}) = H_t(X \mid Y_{1:t}) - IG_t(Y_{t+1}). \tag{3.8}$$

In this expression $IG_t(Y_{t+1})$ denotes the *information gain* of candidate test Y_{t+1}. The derivation of Eq. (3.8) proceeds as follows. By the definition of conditional entropy we have that:

$$H_t(X \mid Y_{1:t}, Y_{t+1}) = -\sum_x \sum_{y_{1:t}} \sum_{y_{t+1}} \Pr_t(x, y_{1:t}, y_{t+1}) \log \left[\Pr(x \mid y_{1:t}, y_{t+1})\right].$$

Applying the chain rule to the first factor, and the Bayes rule to the input of the logarithm, the above equals

$$-\sum_x \sum_{y_{1:t}} \sum_{y_{t+1}} \Pr_t(y_{t+1} \mid x, y_{1:t}) \Pr_t(x, y_{1:t}) \log \left[\frac{\Pr(y_{t+1} \mid y_{1:t}, x) \Pr(x \mid y_{1:t})}{\Pr_t(y_{t+1} \mid y_{1:t})}\right]. \tag{3.9}$$

According to Eq. (3.2), the factor $\Pr_t(y_{t+1} \mid x, y_{1:t})$ of Eq. (3.9) equals $\delta(y^x_{t+1}, y_{t+1})$, therefore

$$H_t(X \mid Y_{1:t+1}) = -\sum_x \sum_{y_{1:t}} \Pr_t(x, y_{1:t}) \log \left[\frac{\Pr_t(x \mid y_{1:t})}{\Pr_t(y^x_{t+1} \mid y_{1:t})}\right]. \tag{3.10}$$

We now split the logarithm of the quotient into a difference of logarithms to obtain

$$-\sum_x \sum_{y_{1:t}} \Pr_t(x, y_{1:t}) \log \left[\Pr_t(x \mid y_{1:t})\right] + \sum_x \sum_{y_{1:t}} \Pr_t(x, y_{1:t}) \log \left[\Pr_t(y^x_{t+1} \mid y_{1:t})\right].$$

According to the definition of conditional entropy, the first term is equivalent to $H_t(X \mid Y_{1:t})$ i.e.,

$$H_t(X \mid Y_{1:t+1}) = H_t(X \mid Y_{1:t}) + \sum_x \sum_{y_{1:t}} \Pr_t(x, y_{1:t}) \log \left[\Pr_t(y^x_{t+1} \mid y_{1:t})\right].$$

The second term is the *information gain* of candidate test Y_{t+1} given all information at time t, and we denote it by $IG_t(Y_{t+1})$ i.e.,

$$IG_t(Y_{t+1}) = -\sum_x \sum_{y_{1:t}} \Pr_t(x, y_{1:t}) \log \left[\Pr_t(y^x_{t+1} \mid y_{1:t})\right].$$

We can simplify this further by applying the chain rule to $\Pr_t(x, y_{1:t})$ to obtain $\Pr_t(y_{1:t} \mid x) \Pr_t(x)$. The above expression then becomes

$$IG_t(Y_{t+1}) = -\sum_{x \in \mathcal{X}} \Pr_t(x) \log \Pr_t(y^x_{t+1} \mid y^x_{1:t}). \tag{3.11}$$

since only the term $y_{1:t} = y^x_{1:t}$ survives in the summation over tests, all others being equal to zero according to Eq. (3.2).

As with all other quantities that requires a summation over the space of particles $\mathcal{X}$, we can approximate the above expression for the information gain using Eq. (3.6), i.e.,

$$IG_t(Y_{t+1}) \approx -\frac{1}{N} \sum_{i=1}^{N} \log f(\Pr_t(x^{(i)}))$$

where $f(\Pr_t(x))$ denotes the quantity $\Pr_t(y^x_{t+1} \mid y^x_{1:t})$. This quantity is a function that depends solely on $\Pr_t(X)$ as the following calculation demonstrates:

$$\Pr_t(y^x_{t+1} \mid y^x_{1:t}) = \frac{\Pr_t(y^x_{t+1}, y^x_{1:t})}{\Pr_t(y^x_{1:t})} = \frac{\sum_{x' \in \{y^x_{t+1}, y^x_{1:t}\}} \Pr_t(x')}{\sum_{x' \in \{y^x_{1:t}\}} \Pr_t(x')} \tag{3.12}$$

obtained by applying the law of total probability over x and keeping only those structures in the summation that are consistent with the assignments $y^x_{1:t}$ and y^x_{t+1} of $Y_{1:t}$ and Y_{t+1} respectively (again by Eq. (3.2)).

Eq. (3.12) has an interesting and intuitive interpretation: at each time t, the posterior probability of a test Y being true (false), given some assignment $y_{1:t}$ of tests $Y_{1:t}$, equals the total posterior probability mass of the structures in the equivalence class $\{y_{1:t}\}$ (c.f. Eq.(3.3)) that are consistent with $Y = \texttt{true}$ ($Y = \texttt{false}$), normalized by the posterior probability mass of the class $\{y_{1:t}\}$. Under this interpretation, it is not hard to prove that information gain is maximized by a test that splits evenly the mass of each equivalence class.

3.3.4 Particle filter for structures

We explain now the particle filter algorithm used in PFMN. A particle filter (Andrieu et al., 2003) is a sequential Markov-Chain Monte-Carlo (**MCMC**) method that uses a set of samples, called *particles*, to represent a probability distribution that may change after each observation in a sequence of observations. At each step, an observation is performed and a new set of

particles is obtained from the set of particles at the previous step. The new set represents the new (posterior) distribution given the sequence of observations so far. One of the advantages of this sequential approach is the often drastic reduction of the cost of sampling from the new distribution, relative to alternative (non-sequential) sampling approaches. In our case, the domain is $\mathcal{X}$ and particles represent structures. Observations correspond to the evaluation of a single conditional independence test on data.

Algorithm 9 Particle filter algorithm. $\mathcal{X}'' = PF(\mathcal{X}, M, f, q)$

1: $\Pr_t(X) \longleftarrow f(X)$
2: **for** $m = 1$ to M **do**
3: **for all** particles $x \in \mathcal{X}$ **do**
4: $w(x) \longleftarrow \Pr_t(x)$ */* Compute weights. */*
5: **for all** particles $x \in \mathcal{X}$ **do**
6: $\widetilde{w}(x) \longleftarrow \frac{w(x)}{\sum_{x' \in \mathcal{X}} w(x')}$ */* Normalize weights. */*
7: */* Resample particles in $\mathcal{X}$ using $\widetilde{w}$ as the sampling probabilities. */*
8: $\mathcal{X}' \longleftarrow$ *resample*$(\mathcal{X}, \widetilde{w})$
9: */* Move each particle in $\mathcal{X}'$ once using Metropolis-Hastings and distr.* $\Pr_t(X)$. **/*
10: $\mathcal{X}'' \longleftarrow \varnothing$
11: **for all** $x \in \mathcal{X}'$ **do**
12: $\mathcal{X}'' \longleftarrow \mathcal{X}'' \cup M\text{-}H(x, 1, \Pr_t, q)$.
13: **return** $\mathcal{X}''$

Algorithm 10 Metropolis-Hastings algorithm. $x' = M\text{-}H(x, M, p, q)$

1: $x^{(0)} \longleftarrow x$
2: **for** $i = 0$ to $M - 1$ **do**
3: $u \sim \mathcal{U}_{[0,1]}$.
4: $x^\star \sim q(x^\star \mid x^{(i)})$.
5: **if** $u < \mathcal{A}(x^{(i)}, x^\star) = \min\left\{1, \frac{p(x^\star)q(x^{(i)}|x^\star)}{p(x^i)q(x^\star|x^{(i)})}\right\}$ **then**
6: $x^{(i+1)} \longleftarrow x^\star$
7: **else**
8: $x^{(i+1)} \longleftarrow x^{(i)}$
9: $x' \longleftarrow x^{(M)}$
10: **return** x'

The particle filter algorithm, shown in Algorithm 9, is used in line 9 of PFMN to transform population $\mathcal{X}_t$ into $\mathcal{X}_{t+1}$. This is done in a sequence of two steps, *resampling* (lines 3–8) and *moves* (lines 10-12), that are repeated M times in the main loop (lines 2–12) (in our experiments

we used $M = 20$). The resampling step quickly re-distributes the existing particles to better reflect the new distribution, but it may result in a sample that represents only a small fraction of the space of structures, a problem that is exacerbated at each new iteration. The MCMC step address this problem by allowing particles to explore novel regions of the space using Metropolis-Hastings "moves" (more later).

The change in the probability distribution from $\Pr_t(X)$ to $\Pr_{t+1}(X)$ is reflected in a "weight." Intuitively, this weight measures the amount of probability mass represented by the particle. Let us explain this in more detail. A representative sample generated from distribution $\Pr_t(X)$ has, on average, a number of particles per x proportional to $\Pr_t(X)$. It is therefore the number of particles at structure x that models the amount of probability mass of x. In this case the weights are the same for all particles, and thus are not considered when sampling from a single distribution. In sequential MCMC, the goal is to use the sample $\mathcal{X}_t$ as a starting point in the search for $\mathcal{X}_{t+1}$. In this case, the sample of $\Pr_t(X)$ may not be a representative sample of $\Pr_{t+1}(X)$. We therefore assign weights to particles to accommodate this difference. The procedure assigns a weight $w(x)$ to each particle $x \in \mathcal{X}_t$ equal to $\Pr_{t+1}(x)$, the probability mass of x in the new distribution (line 4). The normalized weights (line 6) are used as a probability in the resampling step (line 8), which generates a set of particles distributed according to the new distribution by selecting $|\mathcal{X}_t|$ particles randomly and with replacement from $\mathcal{X}_t$, proportional to the value of $\widetilde{w}(x)$.

During the move phase, all particles are moved (lines 10–12) through one pair of proposal/acceptance steps (i.e., $M = 1$) using the Metropolis-Hastings (**M-H**) algorithm, an instance of the class of MCMC algorithms, shown in Alg. 10 (Andrieu et al., 2003). The M-H algorithm requires that $\Pr_t(X)$ and a *proposal* distribution $q(X^\star \mid X)$ be provided as parameters. $\Pr_t(X)$ has already been addressed above in Eq. (3.4). We now explain the proposal distribution used in our algorithm. We consider a "random walk" proposal $q(x^\star \mid x)$ that generates a sample $x^\star$ from x by iteratively inverting each edge of x with probability α. Thus, if h denotes the Hamming distance between structures x and $x^\star$, and $m = n(n-1)/2$, where n is the number of nodes (same for both x and $x^\star$), we have that $q(x^\star \mid x) = q(x \mid x^\star) = \alpha^h(1-\alpha)^{m-h}$.

With this proposal, the quotient $\frac{q(x^{(i)}|x^\star)}{q(x^\star|x^{(i)})}$ in line 5 of the Metropolis-Hastings algorithms is equal to one for all particles $x^{(i)}$. Also, the probability of inverting h edges follows a binomial distribution, and thus the expected number of inversions is αm. In our experiments, we use a value for α such that on average, only one edge is inverted per move, i.e., $\alpha = 1/m$.

A well-designed proposal $q(X^\star \mid X)$ i.e., one that ensures convergence to the posterior distribution $\Pr_t(X)$, must satisfy the requirements of aperiodicity and irreducibility (Andrieu et al., 2003). The above proposal q satisfies both requirements: it is aperiodic since it always allows for rejection, and irreducible since its support is the entire set of structures $\mathcal{X}$.

This concludes the explanation of our approach. In the following section we present and discuss results that demonstrate its practical value.

3.4 Experiments

Our main objective in this chapter is to prove the viability of PFMN as an alternative to GSIMN for GSMN by showing it requires a smaller weighted number of tests to learn the networks of Markov networks while maintaining or improving their accuracy. As such, we compared PFMN against GSIMN as well as GSMN. We conducted two group of experiments. In the first group we conducted experiments on domains for which the true model was generated randomly, and thus it was known. In the second group the experiments were conducted on real-world data sets, for which the underlying true model was unknown. The results confirm PFMN outperforms both GSMN and GSIMN while producing networks with comparable accuracy, with few exceptions. Results for the known-model and real-world experiments are described below in Sections 3.4.1 and 3.4.2, respectively.

In all these experiments we report the *weighted number of tests* and their *accuracy*. The weight of each test is used to account for the execution times of tests with different conditioning set sizes, and is taken to be the number of variables involved in each test. To obtain the quality of the recovered network, we measure accuracy as the fraction of *unseen* conditional independence tests that are correct i.e., we compare the result of each test on the resulting structure (using vertex separation) with the true value of the test (calculated using vertex

separation in the true model for artificial domains or statistical tests on the data for real-world domains). We say more on this below.

3.4.1 Known-model Experiments

We first evaluated our algorithm in artificial domains in which the structure of the underlying model, called *true network*, is known. This allowed (i) a systematic study of its behavior under varying conditions of domain sizes n and amount of dependencies (reflected in the number of edges m in the true network) and, (ii) a better evaluation of quality of the output networks because the true model is known. True networks were generated randomly by selecting the first $m_\tau = \tau \frac{n}{2}$ pairs in a random permutation of all possible edges. The factor $1/2$ is necessary because each edge contributes to the degree of two nodes.

Two types of known-model experiments were conducted. In one, information on underlying independences were obtained by vertex separation on the true network and thus are 100% accurate. Results of these experiments are presented in the next section. In the second type of experiments information of the underlying independences were obtained through statistical tests performed on data sampled from the true network (using a Gibbs sampler). We used the Bayesian test of Margaritis (2005) for all three algorithms. Also, in all known-model experiments we considered the following values for parameters of PFMN: $M = 20$ for the number of Metropolis-Hasting moves, $\alpha = 1/\binom{n}{2}$ for the probability of edge reversal in the proposal distribution (as explained in earlier in the chapter, this value of α was chosen so that on average the state of a single edge is inverted per move), and $\beta = 2$ for the penalizing factor of the score (c.f. Eq. (3.7)). The remaining parameters vary for different experiments.

3.4.1.1 Exact Learning Experiments

Figures 3.2 and 3.3 show the results of the exact learning experiments, where the ground truth comes from conditional independence tests conducted directly on the true network through vertex separation i.e., the outcomes of tests are always correct. Figure 3.2 shows the absolute value of the weighted number of tests conducted by PFMN compared to both

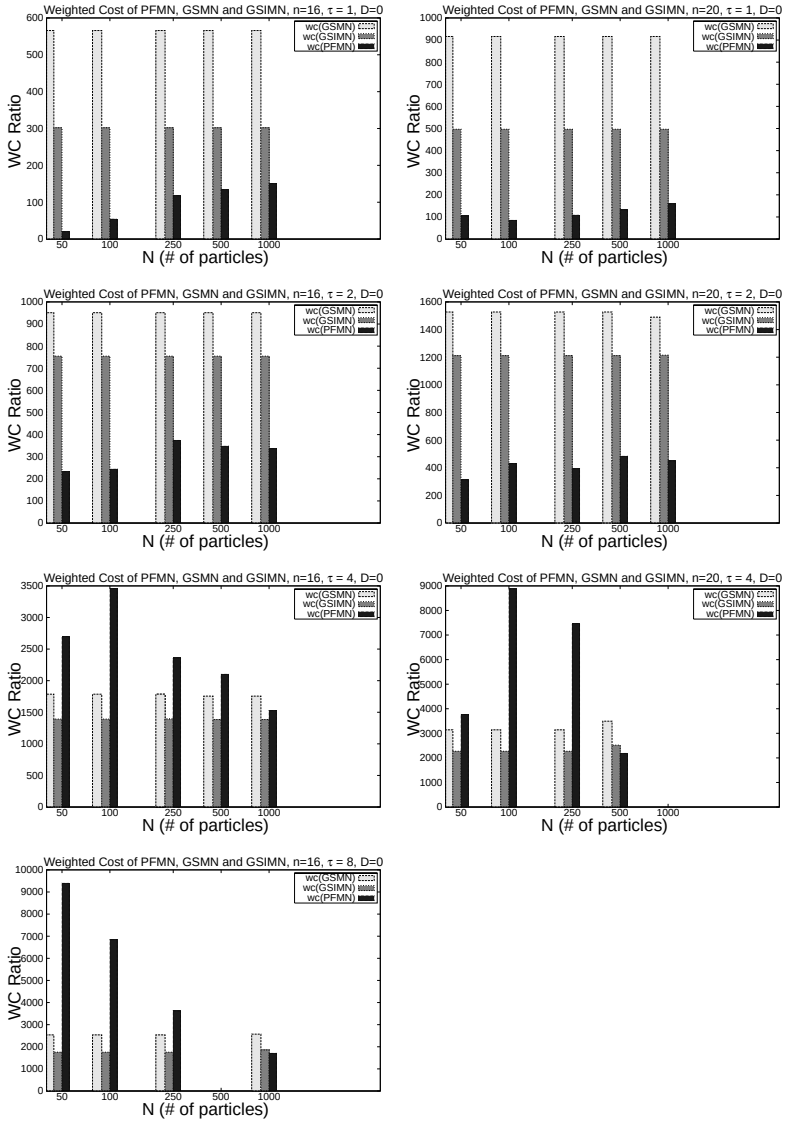


Figure 3.2 Comparisons of the weighted number of tests required by GSMN, GSIMN, and PFMN to learn a Markov networks in the exact learning case for varying number of particles $N = 50, 100, 250, 500$, and 1000. The two columns correspond to true networks with $n = 16$ and 20 variables and the different rows to varying connectivity parameters $\tau = 1, 2, 4$, and 8.

GSMN and GSIMN for domain sizes $n = 16$ (left column) and $n = 20$ (right column), a number of connectivities ($\tau = 1, 2, 4$, and 8), and an increasing number of structure particles N. Up to ten true networks were generated randomly for each pair (n, τ), and the figure shows the mean value.

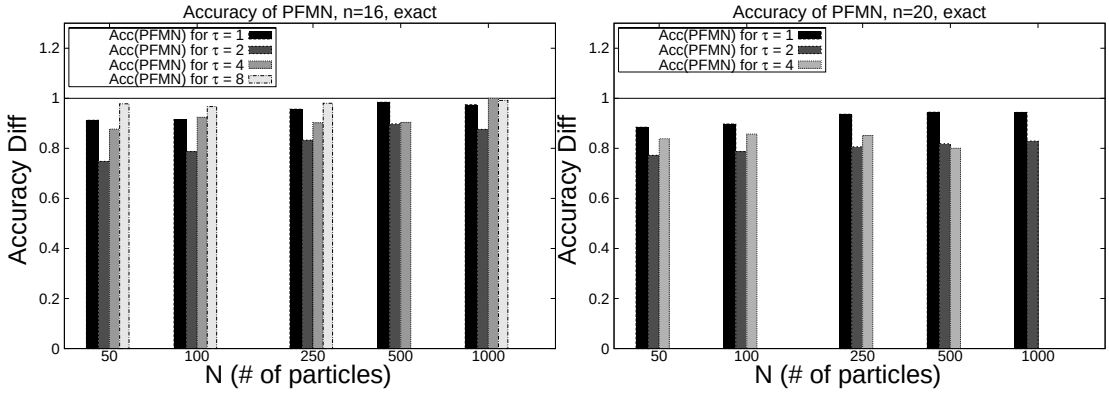


Figure 3.3 Accuracy of the Markov networks output by PFMN in the exact learning case for varying number of particles $N = 50, 100, 250, 500$, and 1000 and varying connectivity parameters $\tau = 1, 2, 3, 4, 8$.

We can see in Figure 3.2 that for large N the weighted cost of PFMN is lower than that of GSIMN and GSMN for all domain sizes and connectivities, reaching savings of up to 85% for $n = 20$ and $\tau = 1$. The cases of $\tau = 4$ and $\tau = 8$ shows the difficulty of PFMN to perform better than GSIMN and GSMN for small N. This can be explained by the fact that the number of structures with connectivity τ grows exponentially as the number of edges $m_\tau = \tau\frac{n}{2}$ in the network approach $\frac{1}{2}\binom{n}{2}$ (from either side), i.e., the middle point between a totally sparse and totally connected network, making the search task in PFMN more difficult. For $n = 16$, $\frac{1}{2}\binom{n}{2} = 60$, and the number of edges for $\tau = 4$ and 8 are $m_4 = 32$ and $m_8 = 64$, respectively. For $n = 20$, $\frac{1}{2}\binom{n}{2} = 85$, and the number of edges for $\tau = 4$ and 8 are $m_4 = 40$ and $m_8 = 80$.

PFMN is an approximate algorithm, and thus even for the case of exact tests it may produce a network different that the true one. We thus measure the estimated accuracy of the networks produced by PFMN. This accuracy was calculated using Eq. (2.2), reproduced here for convenience:

$$\widehat{accuracy}_{PFMN} = \frac{1}{|\mathcal{T}|}\left|\left\{t \in \mathcal{T} \mid I_{\text{PFMN}}(t) = I_{\text{true}}(t)\right\}\right|.$$

where $t \in \mathcal{T}$ denotes a triplet, $I_{\text{true}}(t)$ denotes the result of a test performed on the true network, and $I_{\text{PFMN}}(t)$ denotes the result of a test performed on the output network produced by PFMN. This accuracy definition compares the result (`true` or `false`) of a number of

conditional independence tests on the network (using vertex separation) to the same tests performed on the true network (also using vertex separation). For this calculation, we sampled randomly a set $\mathcal{T}$ of 2,000 triplets $(A, B, \mathbf{S})$ evenly distributed among all possible conditioning set sizes $m \in \{0, \ldots, n-2\}$ (i.e., $2000/(n-1)$ tests for each m). Each of these triplets was constructed as follows: First, two variables A and B were drawn randomly from $\mathbf{V}$. Second, the conditioning set was determined by picking the first m variables from a random permutation of $\mathbf{V} - \{A, B\}$.

Fig. 3.3 shows that even though the accuracies $\widehat{accuracy}_{\text{PFMN}}$ of PFMN are not 1, in most cases they show an increasing trend for large values of N, surpassing 90% in all but few cases ($n = 20$, $\tau = 2, 4$).

3.4.1.2 Sampled data experiments

In this set of experiments we evaluated PFMN on data sampled from the true model. This allows a more realistic assessment of the performance of our algorithm in terms of number of tests required and accuracy of its output network. We therefore performed experiments on data sets sampled from known Markov networks using Gibbs sampling.

In the exact learning experiments only the structure of the true network was required, generated randomly in the fashion described above. For sampled data experiments however, we also need to specify the network parameters. The network parameters were generated randomly with the strength of dependencies among connected variables regulated by a user input parameter θ as explained in detail in Section 2.6.1.2 of the previous chapter. The data sets used in the PFMN experiments were sampled using $\theta = 1$.

As in the exact case, we conducted experiments to compare the performance of PFMN against both GSMN and GSIMN. We report the weighted number of tests required by these algorithms and the accuracy of the output network. In the sample experiments, the accuracy of GSMN and GSIMN is no longer 1 because testing for conditional independence in these experiments was conducted using statistical tests on data. For small data sets, these statistical tests may produce incorrect outcomes, and thus the recovered network may differ from the true

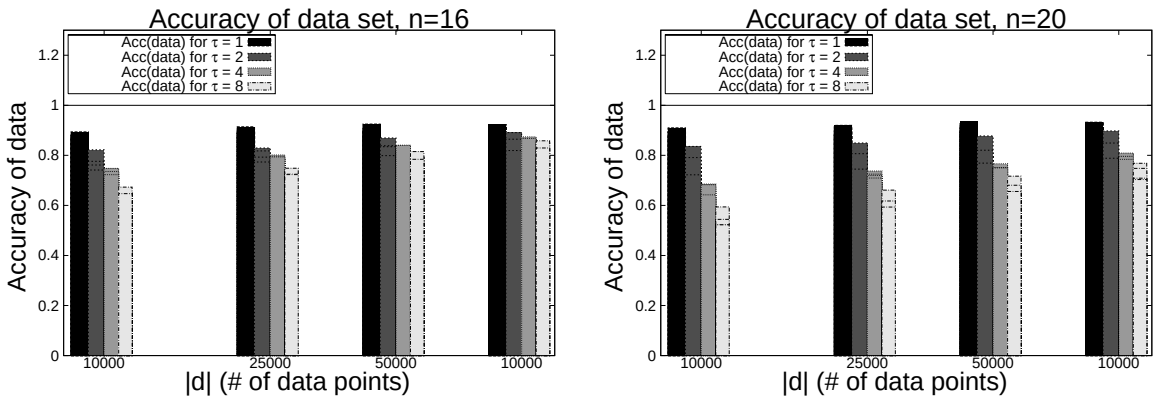


Figure 3.4 Faithfulness of sampled data sets. The figure shows plots of $\widehat{accuracy}^{\text{data}}_{\text{true}}$ for the data sets used in our experiments, i.e., sampled from randomly generated MNs with domain sizes $n = 16$ (left) and 20 (right) and connectivities $\tau = 1, 2, 4, 8$ (shown as histogram bars).

one. We thus compare the estimated accuracy of the networks output by all three algorithms. This accuracy was calculated by a procedure similar to that used in the exact case. That is, we sampled randomly a set $\mathcal{T}$ of 2,000 triplets $(A, B, \mathbf{Z})$ evenly distributed among all possible conditioning set sizes $m \in \{0, \ldots, n-2\}$. The result (`true` or `false`) of performing a conditional independence test for each triplet t on the network output by the algorithms, denoted $I_{\text{ALG}}(t)$, $\text{ALG} \in \{\text{PFMN}, \text{GSMN}, \text{GSIMN}\}$, was compared to the outcome of the test on the true model. This time, however, instead of querying for independence directly in the true network using vertex separation, we queried for independence by performing statistical tests on the data set containing all possible data points $|d|$. In the data sets used in our experiments, this value is $|d| = 100,000$. If we denote by $I_{\text{data}}(t)$ the result of performing a statistical independence test on the complete data set, the accuracy of algorithm ALG is as follows:

$$\widehat{accuracy}^{data}_{\text{ALG}} = \frac{1}{|\mathcal{T}|}\left|\left\{t \in \mathcal{T} \mid I_{\text{ALG}}(t) = I_{\text{data}}(t)\right\}\right|. \tag{3.13}$$

This modification (using the independencies in the complete data set as the ground truth) attempts to account for deficiencies of the sampling algorithm which produced unfaithful data sets, that is, data sets whose conditional independences does not match the underlying model (i.e., the model used to sampled the data). To assess the amount by which the data sets differ from the true model we measured the accuracy of the data against the true model using a set

$\mathcal{T}$ of $2,000$ randomly generated triplets, i.e.,

$$\widehat{accuracy}^{data}_{true} = \frac{1}{|\mathcal{T}|}\left|\left\{t \in \mathcal{T} \mid I_{\text{true}}(t) = I_{\text{data}}(t)\right\}\right|.$$

Figure 3.4 shows plots of $\widehat{accuracy}^{\text{data}}_{\text{true}}$ for subsets of several data sets with increasing number of data points $|d|$. The data sets considered are those used in the experiments of this section, i.e., data sets sampled from randomly generated MNs with domain sizes $n = 16$ (left) and 20 (right) and connectivities $\tau = 1, 2, 4, 8$ (shown as histogram bars). The figure shows that even though the accuracy of the data sets increases with the number of data points $|d|$ for all domains sizes and connectivities, it barely surpasses 80% in most cases. In the worst case, $n = 20$, $\tau = 8$, the accuracy for $|d| = 100,000$ is as low as 65%. This confirms that the data sets used in our experiments are not faithful, and justifies the use of the data sets themselves as ground truth (as opposed to using the true network.)

Figure 3.5 shows the weighted number of tests of PFMN vs. GSMN for different domain sizes $n = 16$ (left column) and $n = 20$ (right column), a number of connectivities ($\tau = 1, 2, 4,$ and 8), and a fixed number of particles $N = 250$. Each of the eight plots shows the absolute value of the weighted number of tests as histogram bars, and their differences as line-curves (with a positive value corresponding to an improvement of PFMN with respect to GSMN) for data sets with increasing number of data points $|d|$. The x-axis is plotted using a log-scale. Again, up to ten true networks were generated randomly for each pair (n, τ), and the figure shows the mean values of the weighted cost. In all cases with the exception of $\tau = 8$, PFMN demonstrate a considerable reduction in the weighted number of tests, reaching savings of up to 95% for $n = 20$, $\tau = 1$, $|d| = 500$. These results improve slightly over those of the exact case, which also showed improvements (of up to 85%) in all cases but $\tau = 4$ and $\tau = 8$.

The estimated accuracies of PFMN vs. GSMN (i.e., $\widehat{accuracy}^{\text{data}}_{\text{PFMN}}$ and $\widehat{accuracy}^{\text{data}}_{\text{GSMN}}$, respectively) are shown in Figure 3.6 for the same domain sizes $n = 16, 20$ (columns) and connectivity parameters $\tau = 1, 2, 4, 8$ (rows). In all eight plots, the mean value over up to ten runs of the absolute accuracy are shown in the plots as histogram bars, and their difference as a line-curve (with a positive value corresponding to an improvement of PFMN with respect to GSMN). Again, the x-axis is plotted using a log-scale. The results in this figure are ambiguous,

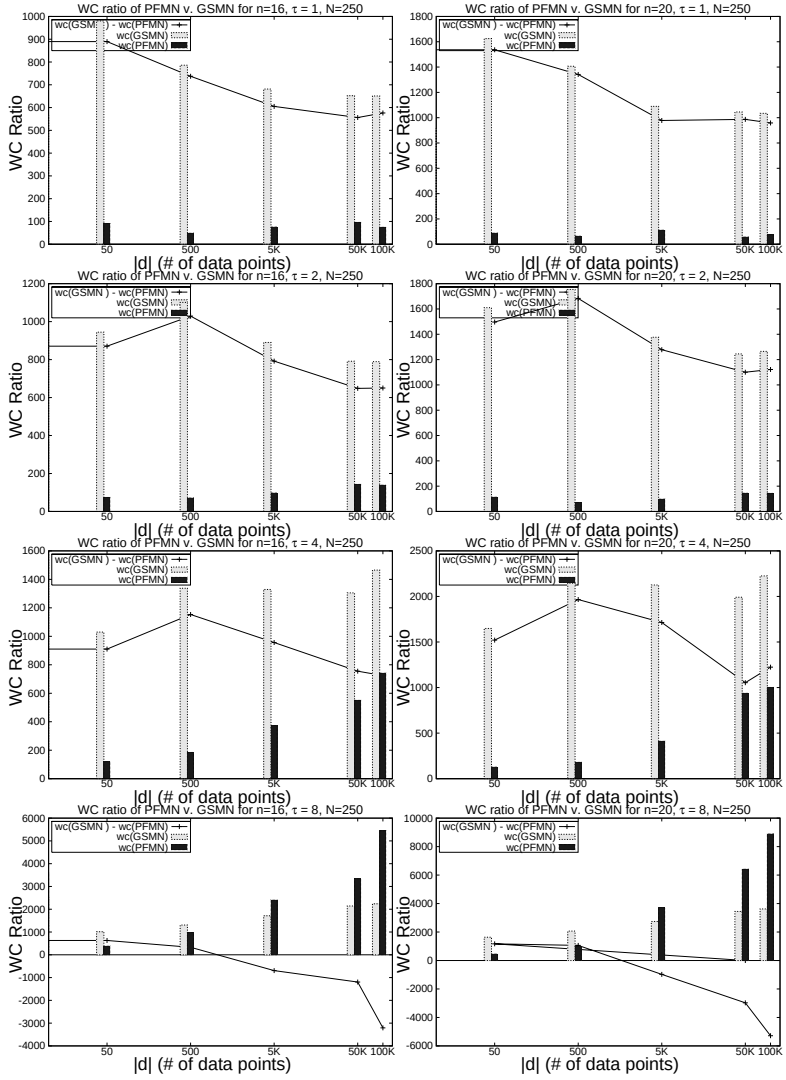


Figure 3.5 Comparisons of the weighted number of tests of PFMN vs. GSMN for sampled data sets with increasing number of data points $|d|$. The x-axis is plotted in log-scale. The figure shows eight plots for different domain sizes $n = 16$ (left column) and $n = 20$ (right column), a number of connectivities ($\tau = 1, 2, 4$, and 8), and a fix number of particles $N = 250$.

showing considerable improvements of PFMN over GSMN for $\tau = 1, 2$ and 8 in both domain sizes $n = 16$ and 20 and small data set sizes $|d|$, with improvements of up to 40% for $\tau = 1$ and $|d| = 50$, and showing a decline in accuracy of PFMN with respect to GSMN for $\tau = 4$ and both domain sizes with difference of up to 20% for $|d| = 50$.

We also compared PFMN against GSIMN for $N = 250$, using the same networks and their

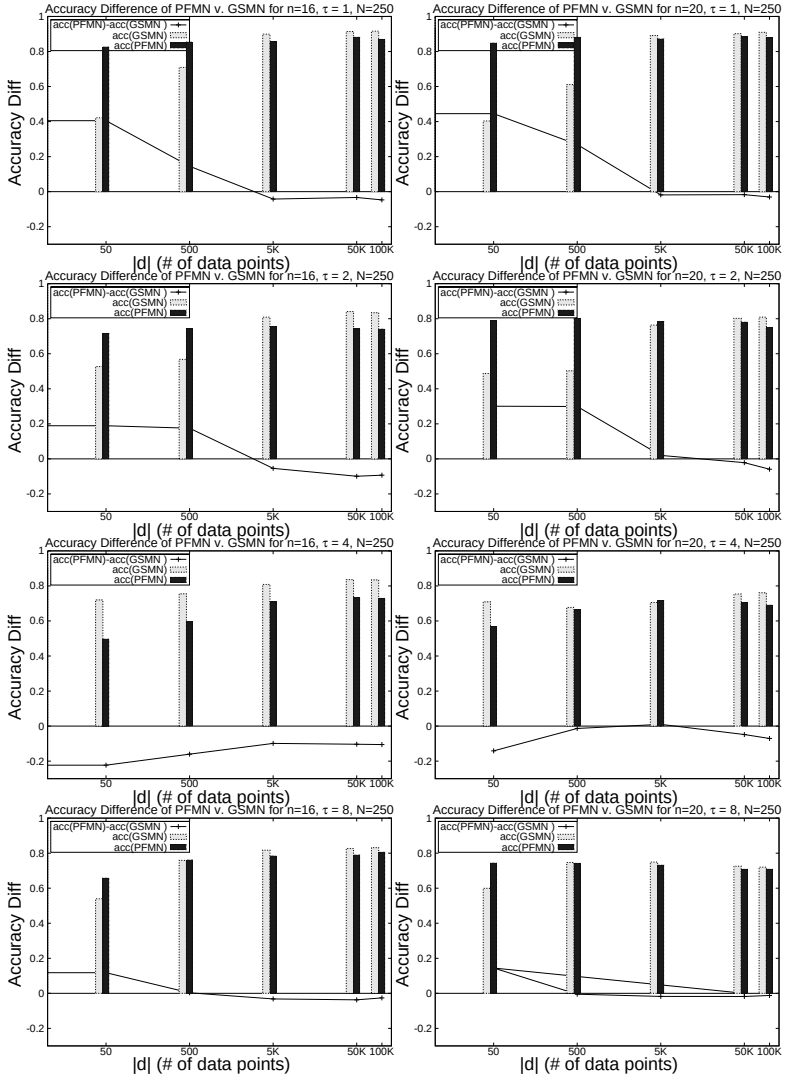


Figure 3.6 Comparisons of the accuracies of PFMN vs. GSMN for sampled data sets with increasing number of data points $|d|$. The x-axis is plotted in log-scale. The figure shows eight plots for different domain sizes $n = 16$ (left column) and $n = 20$ (right column), a number of connectivities ($\tau = 1, 2, 4,$ and 8), and a fixed number of particles $N = 250$.

corresponding sampled data of domain sizes $n = 16, 20$ and connectivities $\tau = 1, 2, 4, 8$ used for the comparison against GSMN. Figure 3.7 shows the weighted number of tests of PFMN vs. GSIMN with results for $n = 16$ and $n = 20$ (columns) and connectivities $\tau = 1, 2, 4,$ and 8 (rows). Once again, the figure shows the mean values over the same group of random networks generated over each pair (n, τ). Qualitatively the results are similar to the comparison against

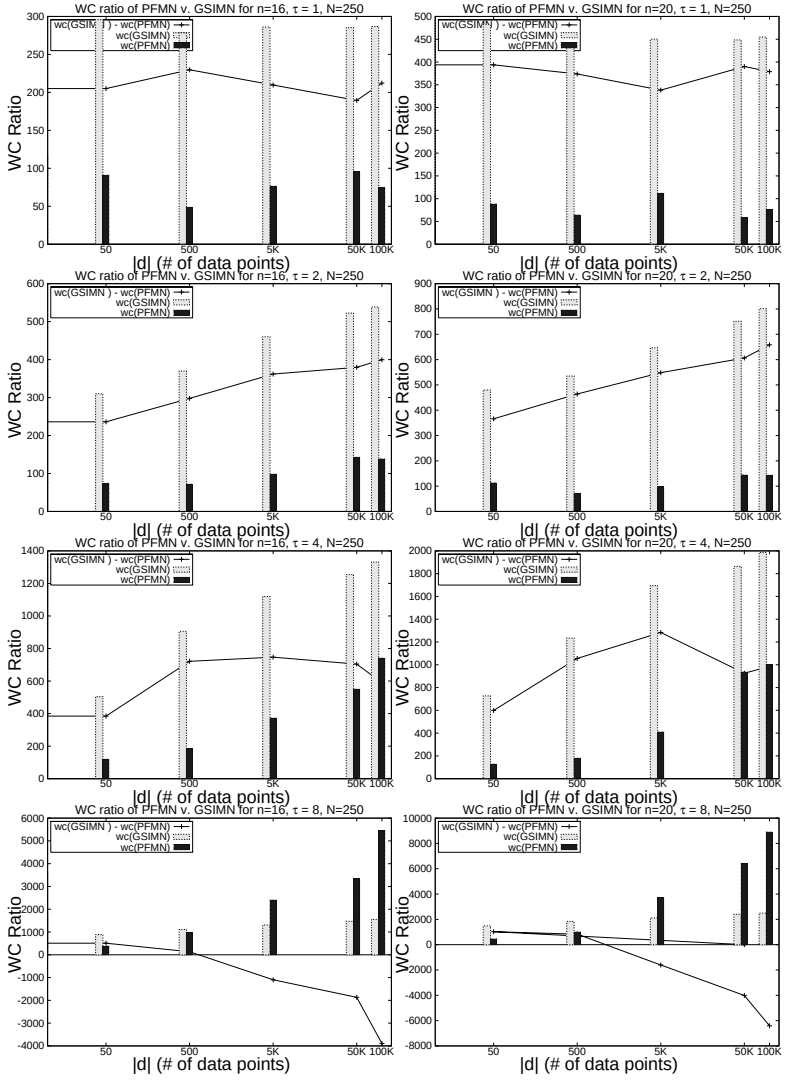


Figure 3.7 Comparisons of the weighted number of tests of PFMN vs. GSIMN for sampled data sets with increasing number of data points $|d|$. The x-axis is plotted in log-scale. The figure shows eight plots for different domain sizes $n = 16$ (left column) and $n = 20$ (right column), a number of connectivities ($\tau = 1, 2, 4,$ and 8), and a fixed number of particles $N = 250$.

GSMN but this time with smaller difference between the two weighted costs. This is expected because —as showed in the previous chapter— GSIMN outperforms GSMN in the weighted number of tests required. In all cases, with the exception of $\tau = 8$, PFMN demonstrates a considerable reduction in the weighted number of tests, reaching savings of up to 90% for $n = 20$, $\tau = 2$, $|d| = 500$. These results improve slightly over those of the exact case, which

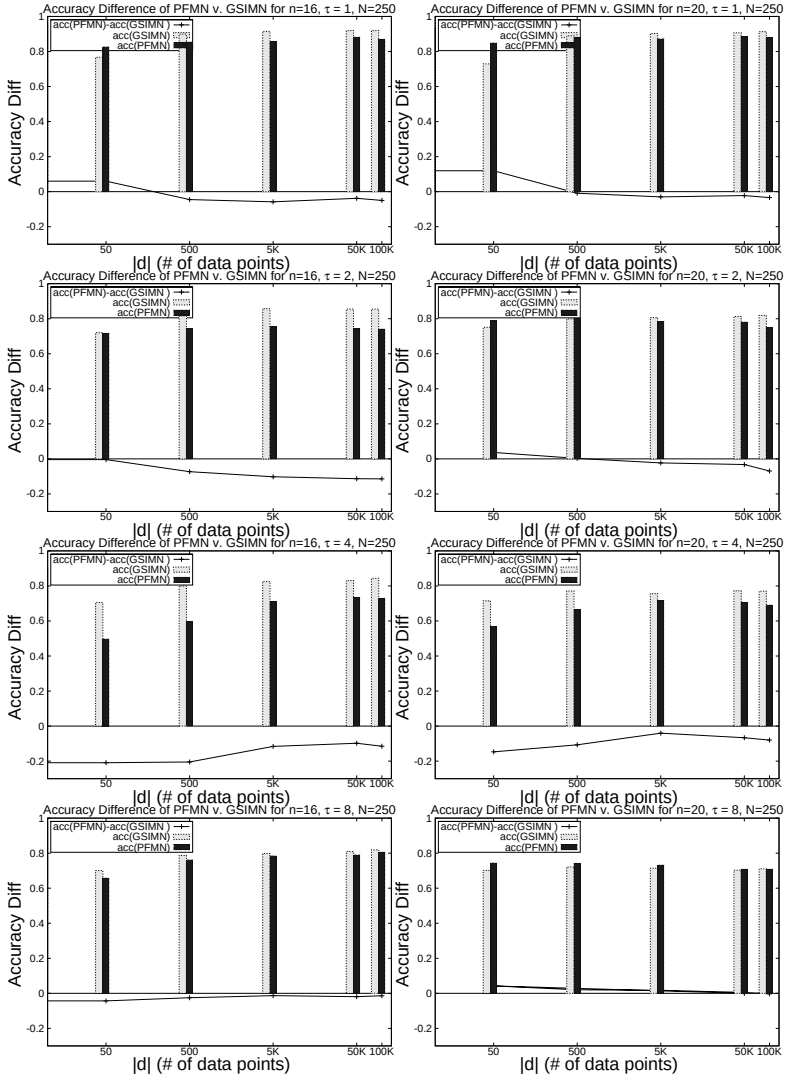


Figure 3.8 Comparisons of the accuracies of PFMN vs. GSIMN for sampled data sets with increasing number of data points $|d|$. The x-axis is plotted in log-scale. The figure shows eight plots for different domain sizes $n = 16$ (left column) and $n = 20$ (right column), a number of connectivities ($\tau = 1, 2, 4$, and 8), and a fixed number of particles $N = 250$.

also show improvements in all cases but $\tau = 4$ and $\tau = 8$.

The estimated accuracies of PFMN vs. GSIMN (i.e., $\widehat{accuracy}^{\text{data}}_{\text{PFMN}}$ and $\widehat{accuracy}^{\text{data}}_{\text{GSIMN}}$, respectively) are shown in Figure 3.8 for the same domain sizes $n = 16, 20$ (columns) and connectivity parameters $\tau = 1, 2, 4, 8$ (rows). The plots show the absolute values of the accuracy as histogram bars, and their difference as a line-curve (with a positive value corresponding to

an improvement of PFMN with respect to GSIMN). Once again the plots show the mean value over up to ten runs, and the x-axis is plotted using a log-scale. The results in this figure show the accuracy of PFMN is comparable to that of GSIMN in all but $\tau = 4$, with a difference in accuracy of up to 20% for small number of data points ($|d| \leq 500$) but a recovery to a less than 10% for large number of data points.

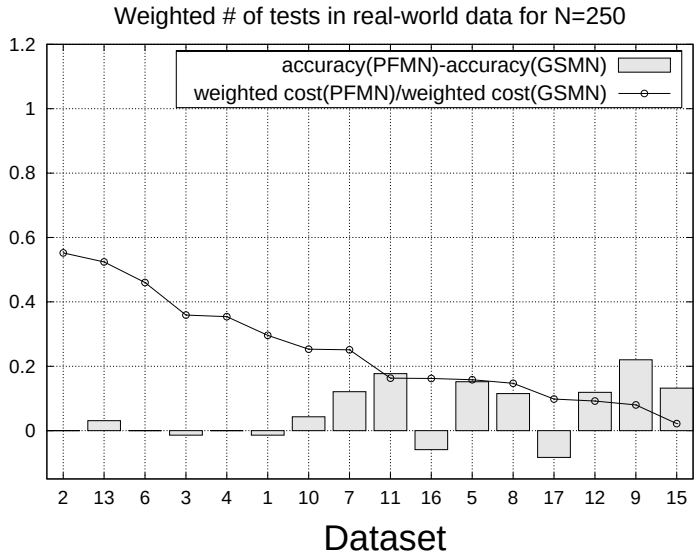


Figure 3.9 Performance comparison of PFMN vs. GSMN on real-world data sets. The line curve shows the ratio of weighted cost of PFMN vs. GSMN and the histogram bars show the difference between accuracy of PFMN and GSMN on real-world data sets. The numbers on the x-axis correspond to data set indices in Table 3.1.

3.4.2 Real-World Data Experiments

We also conducted experiments on a substantial number of data sets obtained from the UCI ML and KDD archives (D.J. Newman and Merz, 1998a,b). As above, we compare the weighted cost and accuracy of PFMN vs. GSMN and GSIMN. While known-model experiments have the advantage of allowing a more controlled and systematic study of their performance, experiments on real-world data are necessary for a more realistic assessment of their performance. Real data sets are more challenging because their underlying probability distribution may not be faithful

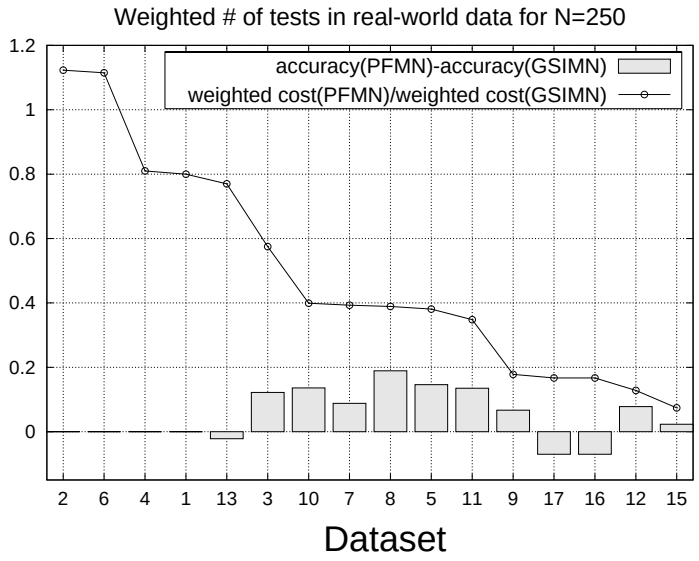


Figure 3.10 Performance comparison of PFMN vs. GSIMN on real-world data sets. The line curve shows the ratio of weighted cost of PFMN vs. GSIMN and the histogram bars show the difference between accuracy of PFMN and GSIMN on real-world data sets. The numbers on the x-axis correspond to data set indices in Table 3.2.

to any Markov network, and its structure may be non-random (e.g., a possibly irregular lattice in many cases of spatial data, or small-worlds networks in many cases of social networks).

Figure. 3.9 shows the ratio of the weighted cost of PFMN vs. GSMN (with a number smaller than 1 shows improvement of PFMN vs. GSMN) and the difference in the accuracies of PFMN and GSMN (with a positive histogram bar shows an improvement of PFMN vs. GSMN). In these experiments, PFMN used $N = 250$. The numbers in the x-axis are indices to the data sets shown in Table 3.1. In all 17 data sets PFMN required lower weighted cost than GSMN, reaching ratios as low as 0.02 (i.e., a 98% reduction). Moreover, for 12 out 17 data sets PFMN achieves this reduction in cost with better or no reduction in accuracy compared to GSMN, the rest exhibit only a modest reduction of less than 8.3% (the largest difference being for the dermatology data set).

Figure 3.10 shows the ratio of the weighted cost of PFMN vs. GSIMN (with a number smaller than 1 showing an improvement of PFMN vs. GSIMN) and the difference in the

Table 3.1 Comparison of weighted number of tests and accuracy of PFMN vs. GSMN. for several real-world data sets. For each evaluation measure, the best performance between PFMN and GSMN is indicated in bold. The table also reports n, the number of variables in the domain, and N, the number of data points in each data set.

Data set				Weighted #(tests)		Accuracy	
#	name	n	N	PFMN	GSMN	PFMN	GSMN
1	haberman	5	306	**16**	54	0.730	**0.743**
2	hayes-roth	6	132	**64**	116	0.794	0.794
3	balance-scale	5	625	**23**	64	0.770	**0.784**
4	monks-1	7	556	**34**	96	0.899	0.899
5	car	7	1728	**32**	202	**0.733**	0.581
6	baloons	5	20	**29**	63	0.946	0.946
7	crx	16	653	**337**	1344	**0.723**	0.602
8	nursery	9	12960	**58**	394	**0.723**	0.608
9	hepatitis	20	80	**127**	1588	**0.797**	0.577
10	cmc	10	1473	**114**	450	**0.689**	0.646
11	tic-tac-toe	10	958	**65**	399	**0.656**	0.479
12	flag	29	194	**365**	3979	**0.461**	0.343
13	bridges	12	70	**282**	538	**0.600**	0.570
15	alarm	37	20001	**258**	11906	**0.562**	0.430
16	imports-85	25	193	**465**	2877	0.364	**0.423**
17	dermatology	35	358	**753**	7651	0.312	**0.395**

Table 3.2 Comparison of weighted number of tests and accuracy of PFMN vs. GSIMN. for several real-world datasets. For each evaluation measure, the best performance between PFMN and GSMN is indicated in bold. The table also reports n, the number of variables in the domain, and N, the number of data points in each data set.

Data set				Weighted #(tests)		Accuracy	
#	name	n	N	PFMN	GSIMN	PFMN	GSIMN
1	haberman	5	306	**16**	20	0.730	0.730
2	hayes-roth	6	132	64	**57**	0.794	0.794
3	balance-scale	5	625	**23**	40	**0.770**	0.649
4	monks-1	7	556	**34**	42	0.899	0.899
5	car	7	1728	**32**	84	**0.733**	0.586
6	baloons	5	20	29	**26**	0.946	0.946
7	crx	16	653	**337**	858	**0.723**	0.635
8	nursery	9	12960	**58**	149	**0.723**	0.534
9	hepatitis	20	80	**127**	714	**0.797**	0.731
10	cmc	10	1473	**114**	286	**0.689**	0.552
11	tic-tac-toe	10	958	**65**	187	**0.656**	0.521
12	flag	29	194	**365**	2853	**0.461**	0.383
13	bridges	12	70	**282**	366	0.600	**0.622**
15	alarm	37	20001	**258**	3482	**0.562**	0.539
16	imports-85	25	193	**465**	2777	0.364	**0.434**
17	dermatology	35	358	**753**	4512	0.312	**0.383**

accuracies of PFMN and GSIMN (again with a positive histogram bar showing an improvement of PFMN vs. GSIMN). As in the experiments for GSMN, PFMN used $N = 250$. The numbers in the x-axis are indices to the data sets shown in Table 3.2. In 15 out of 17 data sets PFMN

required lower weighted cost than GSIMN, reaching ratios as low as 0.08 (i.e., a 92% reduction). As in the GSMN case, in most cases (14 out of 17) PFMN achieves this reduction in cost with better or no reduction in accuracy compared to GSIMN. The remaining cases exhibit only a modest reduction of less than 7% (the largest difference being for the imports-84 data set).

3.5 Summary

In this chapter we presented PFMN, an independence-based algorithm for learning the structure of a Markov network from data. We presented an analysis of the domain of structures and independences and a number of interesting relations using an explicit generative model. We also showed experimentally that, compared to existing independence-based algorithms, PFMN executes fewer tests in domain with sparse networks with little or no reduction in accuracy. This helps in domains where data are scarce and tests uncertain or data are abundant and/or distributed over a potentially slow network, making the execution of tests expensive.

CHAPTER 4. Argumentative Independence Tests

4.1 Introduction and Motivation

In this chapter we present a framework that address one of the most important problems of the independence-based approach: the problem of the unreliability of statistical independence tests on finite data sets, especially small ones.

It is well-known that independence-based algorithms have several shortcomings. A major one has to do with the effect that unreliable independence information has on their output. In general such independence information comes from two sources: (a) a domain expert that can provide his or her opinion on the validity of certain conditional independences among some of the variables, usually with a degree of confidence attached to them, and/or (b) statistical tests of independence, conducted on data gathered from the domain. As expert information is often costly and difficult to obtain, the latter is the most commonly used option in practice. A problem that occurs frequently however is that the data set available may be small. This may happen for various reasons: lack of subjects to observe (e.g., in medical domains), expensive data-gathering process, privacy concerns and others. Unfortunately, the reliability of statistical tests significantly diminishes on small data sets. For example, Cochran (1954) recommends that Pearson's χ^2 test be deemed unreliable if more than 20% of the cells of the test's contingency table have an expected count of less than 5 data points. Unreliable tests, besides producing errors in the resulting model structure, may also produce cascading errors due the way that independence-based algorithms work: their operation, including which test to evaluate next, typically depends on the outcomes of previous ones. Therefore, an error in a previous test may have large (negative) consequences in the resulting structure, a property that is called algorithm *instability* in Spirtes et al. (2000).

In this chapter we introduce a framework for increasing reliability of independence tests for small data sets and, as a result, the reliability of independence-based algorithms when they use them to discovery the structure of graphical models (both directed and undirected). The framework introduces a general-purpose independence test, the **argumentative independence test** or **AIT**, that produces more accurate conditional independence decisions that its statistical counterpart by "correcting" their outcome. The correction is based on the information of other independencies that hold in the domain and the relations that holds between them as dictated by Pearl's independence axioms.

We model this setting as a propositional knowledge base whose contents are conditional independences that are potentially inconsistent. Our main insight is to recognize that the outcomes of independence tests are not themselves independent but are constrained by the outcomes of other tests through Pearl's well-known properties of the conditional independence relation (Pearl, 1988). Therefore, such constraints can be sometimes used to correct certain inconsistent test outcomes, choosing instead the outcome that can be inferred by other tests that are not involved in contradictions. We illustrate this by an example.

Example 4.1. *Consider an independence-based knowledge base that contains the following propositions, obtained through statistical tests on data.*

$$(0 \perp\!\!\!\perp 1 \mid \{2,3\}) \tag{4.1}$$

$$(0 \perp\!\!\!\perp 4 \mid \{2,3\}) \tag{4.2}$$

$$(0 \not\perp\!\!\!\perp \{1,4\} \mid \{2,3\}) \tag{4.3}$$

where $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$ *denotes conditional independence of the set of variables* $\mathbf{X}$ *with* $\mathbf{Y}$ *conditional on set* $\mathbf{Z}$*, and* $(\mathbf{X} \not\perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$ *denotes conditional dependence. Suppose that (4.3) is in fact wrong. Such an error can be avoided if there exists some constraint involving these independence propositions. For example, suppose that we also know that the following rule holds in the domain (this is an instance of the Composition axiom, one of Pearl's directed axioms introduced in Eqs. (4.6)).*

$$(0 \perp\!\!\!\perp 1 \mid \{2,3\}) \wedge (0 \perp\!\!\!\perp 4 \mid \{2,3\}) \implies (0 \perp\!\!\!\perp \{1,4\} \mid \{2,3\}). \tag{4.4}$$

$$
\begin{array}{rl}
\textbf{(Symmetry)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \iff (\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z}) \\
\textbf{(Decomposition)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}) \\
\textbf{(Weak Union)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \\
\textbf{(Contraction)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \\
\textbf{(Intersection)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z})
\end{array}
\tag{4.5}
$$

Rule (4.4) and dependence proposition (4.3) contradict each other, resulting in an inconsistent knowledge base. Therefore proposition (4.3) can no longer be accepted. The incorrect independence of proposition (4.3) could be rejected (and the error corrected) if it was possible to resolve the inconsistency in favor of implication (4.4). The framework presented in the rest of this chapter provides a principled approach for resolving such inconsistencies.

The situation described in the previous example, while simple, demonstrates the general idea that we will use in the rest of the chapter: the set of independences and dependences used in an independence-based discovery algorithm form a potentially inconsistent knowledge base, and making use of general rules that we know hold in the domain helps us correct certain outcomes of statistical tests from other ones (frequently more than one). In this way we can improve the reliability of independence-based discovery algorithms that use them to derive graphical models. To accomplish this we will use the framework of *argumentation*, which provides a sound and elegant way of resolving inconsistencies in such knowledge bases, including ones that contain independences.

The rest of the chapter is organized as follows. The next section introduces our notation and definitions. Section 4.3 presents the argumentation framework and its extension with preferences, and describes our approach for applying it to represent and reason in potentially inconsistent knowledge bases that contain independence information. We present our experimental evaluation in Section 4.7, and conclude with a summary of our approach in Section 4.8.

4.2 Notation and Preliminaries

Bayesian and Markov networks (**BN** and **MN**) are directed and undirected graphical models (respectively) which represents the joint probability distribution over $\mathbf{V}$. Each node in the graph represents one of the random variables in the domain. The structure of the network

$$
\begin{array}{rl}
\textbf{(Symmetry)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \iff (\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z}) \\
\textbf{(Decomposition)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}) \\
\textbf{(Composition)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \impliedby (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}) \\
\textbf{(Intersection)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \\
\textbf{(Weak Union)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \\
\textbf{(Contraction)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \\
\textbf{(Weak Transitivity)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \gamma) \implies (\mathbf{X} \perp\!\!\!\perp \gamma \mid \mathbf{Z}) \vee (\gamma \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \\
\textbf{(Chordality)} & (\alpha \perp\!\!\!\perp \beta \mid \gamma \cup \delta) \wedge (\gamma \perp\!\!\!\perp \delta \mid \alpha \cup \beta) \implies (\alpha \perp\!\!\!\perp \beta \mid \gamma) \vee (\alpha \perp\!\!\!\perp \beta \mid \delta)
\end{array}
\tag{4.6}
$$

$$
\begin{array}{rl}
\textbf{(Symmetry)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \iff (\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z}) \\
\textbf{(Decomposition)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}) \\
\textbf{(Intersection)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z} \cup \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}) \\
\textbf{(Strong Union)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}) \\
\textbf{(Transitivity)} & (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \gamma \mid \mathbf{Z}) \vee (\gamma \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})
\end{array}
\tag{4.7}
$$

represents a set of conditional independences on the domain variables. Given the structure of a network, the set of independences implied by it can be identified by a process called *d-separation* for BNs, and *vertex-separation* for MNs.

The structure of BNs and MNs can be discovered from data using independence-based algorithms, which operate by conducting a series of conditional independence queries. In practice conditional independence information is implemented approximately by a statistical test evaluated on the data set (for example, this can be Pearson's conditional independence χ^2 (chi-square) test (Agresti, 2002), Wilk's G^2 test, a mutual information test etc.). In this chapter we used Wilk's G^2 test (Agresti, 2002). To determine conditional independence between two variables X and Y given a set $\mathbf{Z}$ from data, the statistical test G^2 (and any other independence test based on hypothesis testing, e.g., the χ^2 test) returns a *p-value*, which is the probability of error given the data in assuming that the two variables are dependent when in fact they are not. If the p-value of a test is $p(X, Y \mid \mathbf{Z})$, the statistical test concludes independence if and only if $1 - p(X, Y \mid \mathbf{Z})$ is smaller than or equal to a confidence threshold α i.e.,

$$(X \perp\!\!\!\perp Y \mid \mathbf{Z}) \iff p(X, Y \mid \mathbf{Z}) \geq 1 - \alpha. \tag{4.8}$$

Common values for α are 0.95, 0.99, and 0.90.

The conditional independences and dependences of a domain are connected through the set of Pearl axioms of Eqs. (1.6), (1.8), and (1.7) presented in Chapter 1. For convenience, we

reproduce them here in Eqs. (4.5), (4.6), and (4.7), respectively. The general rules of Eqs. (4.5) apply for every probability distribution over $\mathbf{V}$. For domains for which there exists a faithful Bayesian network the set of directed axioms of Eqs. (4.6) hold. If instead a Markov network structure exists that is faithful to the domain, the undirected axioms of Eqs. (4.7) hold.

In the next section we describe the argumentation framework in general, followed by its application to our problem of answering independence queries from knowledge bases that contain sets of potentially inconsistent independence propositions.

4.3 The Argumentation Framework

As we mentioned previously, we model the framework of learning a causal model through independence queries as a set of rules (Eqs. (4.5) or (4.6)) and a knowledge base (**KB**) that contains independence propositions that may be inconsistent.

There exist two major approaches for reasoning with inconsistent knowledge that correspond to two different attitudes: One is to resolve the inconsistencies by removing a subset of propositions such that the resulting KB becomes consistent; this is called *belief revision* in the literature (Gärdenforst, 1992; Gärdenforst and Rott, 1995; Shapiro, 1998; Martins, 1992). A known shortcoming (Shapiro, 1998) of belief revision stems from the fact that it removes propositions, which, besides discarding potentially valuable information, has the same potential problem as the problem that we are trying to solve: an erroneous modification of the KB may have unintended negative consequences if later more propositions are inserted in the KB. A second approach to inconsistent KBs is to allow inconsistencies but to use rules that may be possibly contained in it to deduce which truth value of a proposition query is "preferred" in some way. One instance of this approach is *argumentation* (Dung, 1995; Loui, 1987; Prakken, 1997; Prakken and Vreeswijk, 2002), a sound approach that allows inconsistencies but uses a proof procedure that is able to deduce (if possible) that one of the truth values of certain propositions is preferred over its negation; this may happen because the latter is contradicted by other rules and/or propositions in the KB (a more precise definition is given below). Argumentation is a reasoning model that belongs to the broader class of defeasi-

ble logics (Pollock, 1992; Prakken, 1997). Our approach uses the argumentation framework of Amgoud and Cayrol (2002) that considers preferences over arguments, extending Dung's more fundamental framework (Dung, 1995). Preference relations give an extra level of specificity for comparing arguments, allowing a more refined form of selection between conflicting propositions. Preference-based argumentation is presented in more detail in the Section 4.3.2.

We proceed now to describe the argumentation framework.

Definition 4.1. *An* argumentation framework *is a pair $\langle \mathcal{A}, \mathcal{R} \rangle$, where $\mathcal{A}$ is a set of arguments and $\mathcal{R}$ is a binary relation representing a defeasibility relationship between arguments, i.e., $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$. $(a, b) \in \mathcal{R}$ or equivalently "a $\mathcal{R}$ b" means that argument a defeats the argument b. We also say that a and a are in conflict.*

An example of the defeat relation $\mathcal{R}$ is *logical defeat*, which occurs when an argument contradicts another logically.

The elements of the argumentation framework are not propositions but *arguments*. Given an inconsistent knowledge base $\mathcal{K} = \langle \Sigma, \Psi \rangle$ with a set of propositions Σ and a set of inference rules Ψ, arguments are defined formally as follows.

Definition 4.2. *An* argument *over knowledge base $\langle \Sigma, \Psi \rangle$ is a pair (H, h) where $H \subseteq \Sigma$ such that:*

- *H is consistent,*
- *$H \vdash_\Psi h$,*
- *H is minimal (with respect to set inclusion).*

H is called the support *and h the* conclusion *or* head *of the argument.*

In the above definition $\vdash_\Psi$ stands for classical logical inference over the set of inference rules Ψ. Intuitively an argument (H, h) can be thought as an "if-then" rule i.e., "if H then h". In inconsistent knowledge bases two arguments may contradict or *defeat* each other. The defeat relation is defined through the *rebut* and *undercut* relations, defined below.

Definition 4.3. *Let* (H_1, h_1), (H_2, h_2) *be two arguments.*

- (H_1, h_1) rebuts (H_2, h_2) *iff* $h_1 \equiv \neg h_2$.
- (H_1, h_1) undercuts (H_2, h_2) *iff* $\exists h \in H_2$ *such that* $h \equiv \neg h_1$.

(The symbol "$\equiv$" stands for logical equivalence.) In other words, $(H_1, h_1)\ \mathcal{R}\ (H_2, h_2)$ if and only if (H_1, h_1) *rebuts* or *undercuts* (H_2, h_2).

The objective of argumentation is to decide on the acceptability of each argument. There are three possibilities: an argument can be accepted, rejected, or neither. This partitions the space of arguments $\mathcal{A}$ in three classes:

- The class $Acc_{\mathcal{R}}$ of *acceptable arguments*. Intuitively, these are the "good" arguments. In the case of an inconsistent knowledge base, these will be inferred from the base.
- The class $Rej_{\mathcal{R}}$ of *rejected arguments*. These are the arguments defeated by acceptable arguments. When applied to an inconsistent knowledge base, these will not be inferred from it.
- The class $Ab_{\mathcal{R}}$ of arguments *in abeyance*. These arguments are neither acceptable nor rejected.

The semantics of acceptability proposed by Dung dictates that an argument should be accepted if it is not defeated, or if it is defended by acceptable arguments i.e., each of its defeaters is itself defeated by an acceptable argument. This is formalized in the following definitions.

Definition 4.4. *Let* $\langle \mathcal{A}, \mathcal{R} \rangle$ *be an argumentation framework, and* $S \subseteq \mathcal{A}$. *An argument* a *is* defended *by* S *if and only if* $\forall b$, *if* $b\ \mathcal{R}\ a$ *then* $\exists c \in S$ *such that* $c\ \mathcal{R}\ b$.

Dung characterizes the set of acceptable arguments by a monotonic function $\mathcal{F}$, i.e., $\mathcal{F}(S) \subseteq \mathcal{F}(S \cup T)$ for some S and T. Given a set of arguments $S \subseteq \mathcal{A}$ as input, $\mathcal{F}$ returns the set of all arguments defended by S:

Definition 4.5. *Let* $S \subseteq \mathcal{A}$. *Then* $\mathcal{F}(S) = \{a \in \mathcal{A} \mid a \text{ is defended by } S\}$.

1: $S' \longleftarrow S \cup \{a \in \mathcal{A} \mid a \text{ is defended by } S\}$
2: **if** $S = S'$ **then**
3: **return** S'
4: **else**
5: **return** $\mathcal{F}(\mathcal{A}, \mathcal{R}, S')$

Algorithm 11 Recursive computation of acceptable arguments: $Acc_{\mathcal{R}} = \mathcal{F}(\mathcal{A}, \mathcal{R}, S)$

Slightly overloading our notation, we define $\mathcal{F}(\varnothing)$ to contain the set of arguments that are not defeated, i.e., defend themselves.

Definition 4.6. *Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework, and let $a \in \mathcal{A}$ be some argument. We say a* defends itself *if it is not defeated by any other argument, i.e. $\forall b \neq a \in \mathcal{A}$, $\neg(b \; \mathcal{R} \; a)$.*

Definition 4.7. *$\mathcal{F}(\varnothing) = \{a \in \mathcal{A} \mid a \text{ defends itself}\}$.*

Dung proved that the set of acceptable arguments is the least fix-point of $\mathcal{F}$, i.e., the smallest set S such that $\mathcal{F}(S) = S$.

Theorem 4.8 (Dung (1995))**.** *Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework. The set of acceptable arguments $Acc_{\mathcal{R}}$ is the least fix-point of the function $\mathcal{F}$.*

Dung also showed that if the argumentation framework $\langle \mathcal{A}, \mathcal{R} \rangle$ is finitary i.e., for each argument A there are finitely many arguments that defeat A, the least fix-point of function $\mathcal{F}$ can be obtained by iterative application of $\mathcal{F}$ to the empty set. We can understand this intuitively: From our semantics of acceptability it follows that all arguments in $\mathcal{F}(\varnothing)$ are accepted. Also, every argument in $\mathcal{F}(\mathcal{F}(\varnothing))$ must be acceptable as well since each of its arguments is defended by acceptable arguments. This reasoning can be applied recursively until a fix-point is reached. The fix-point S is the set of arguments that cannot defend any other argument not in S i.e., no other argument is accepted. This suggests a simple algorithm for computing the set of acceptable arguments. Algorithm 11 shows a recursive procedure for this, based on the above definition. The algorithm takes as input an argumentation framework $\langle \mathcal{A}, \mathcal{R} \rangle$ and the set S of arguments found acceptable so far i.e., $S = \varnothing$.

Let us illustrate these ideas with an example.

Example 4.2. *Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework defined by $\mathcal{A} = \{a, b, c\}$ and $\mathcal{R} = \{(a, b), (b, c)\}$. The only argument that is not defeated (i.e., defends itself) is a, and therefore $\mathcal{F}(\varnothing) = \{a\}$. Argument b is defeated by the acceptable argument a, so b cannot be defended and is therefore rejected i.e., $b \in Rej_{\mathcal{R}}$. Argument c, though defeated by b, is defended by (acceptable argument) a which defeats b, so c is acceptable. The set of acceptable arguments is therefore $Acc_{\mathcal{R}} = \{a, c\}$ and the set of rejected arguments is $Rej_{\mathcal{R}} = \{b\}$.*

The *bottom-up* approach of Algorithm 11 has the disadvantage that it requires the computation of all acceptable arguments to answer the acceptability status of a single one. In practice, and in particular in the application of argumentation to independence tests, the entire set of acceptable arguments is rarely needed. Below we present an alternative algorithm for deciding the acceptability of an input argument that is adapted from the *dialog tree* algorithm of Amgoud and Cayrol (2002). This algorithm is provably equivalent to algorithm (11) (whenever it is given the same input it is guaranteed to produce the same output), but it is considerably more efficient (as shown in Section 4.5.2). We sketch the algorithm here and complete its presentation after the presentation of the preference-based argumentation framework in Section 4.3.2.

The algorithm employs a *top-down* approach to answer whether an input argument a is accepted or not. We consider the same acceptability semantics used for Algorithm 11 and proposed by Dung: an argument is accepted if it is not defeated, or if it is defended by acceptable arguments, and an argument is rejected if it is defeated by an accepted argument. Putting these two together we see that an argument is accepted if it is not defeated, or if its

---

1: $defeaters \leftarrow$ set of arguments in $\mathcal{A}$ that defeat a according to $\mathcal{R}$.
2: **for** $d \in defeaters$ **do**
3: **if** top-down$(\mathcal{A}, \mathcal{R}, a)$ = `accepted` **then**
4: **return** `rejected`
5: **return** `accepted`

Algorithm 12 Top-down computation of acceptable arguments: $top - down(\mathcal{A}, \mathcal{R}, a)$

---

defeaters are rejected. In summary:

[**Acceptance**] A node is accepted if it has no defeaters or if all its defeaters are rejected, i.e.

$$A(a) \iff \forall b \in defeaters(a),\ R(b)$$

[**Rejection**] A node is rejected if at least one of its defeaters is accepted, i.e.

$$R(a) \iff \exists b \in defeaters(a),\ A(b) \tag{4.9}$$

[**Abeyance**] A node is in abeyance if its not accepted nor rejected, i.e.

$$Ab(a) \iff \neg A(a) \wedge \neg R(a)$$

The logic of these equations can be easily implemented with a recursive algorithm. This is shown in Algorithm 12. The algorithm loops over all defeaters of some input argument a and responds `rejected` if any of its defeaters is accepted (line 4). If it reached the end of the loop at line 5 it means none of its defeaters was accepted, and thus it accepts the input a. We can represent the execution of the top-down algorithm graphically by a tree that contains a at the root node, and all the defeaters of a node as its children. A leaf is reached when a node has no defeaters. In that case the loop contains no iterations and line 5 is reached trivially.

Unfortunately, the top-down algorithm, as presented in Algorithm 12, will fail to terminate when a node is in abeyance. This is clear from the following theorem (proved formally in Appendix C):

Lemma 4.9. *For every argument a it holds that*

$$Ab(a) \implies \exists b \in attackers(a), Ab(b).$$

From this Lemma, if an argument is in abeyance, its set of defeaters contains an argument in abeyance and thus it will recursively call the algorithm. The recursive call never ends, since there is always another defeater in abeyance. While there are ways to overcome this difficulty in the general case, for the special case of argumentation of independence propositions we will prove that no argument can be in abeyance, and thus the algorithm always terminates. A formal proof is presented below in Section 4.5, together with a time complexity analysis.

We conclude the section by proving that the top-down algorithm is provably equivalent to the bottom-up algorithm of Algorithm 11, i.e., whenever it is given the same input than Algorithm 11, is guaranteed to produce the same output. The proof assumes no argument is in abeyance. This assumption is satisfied for argumentation in independence knowledge bases (c.f. Theorem 4.19, presented later). Formally,

Theorem 4.10. *Let a be an argument in the argumentation framework $\langle \mathcal{A}, \mathcal{R} \rangle$, and let $\mathcal{F}$ be the set of acceptable arguments output by Algorithm 11. Assuming a is not in abeyance,*

$$\textit{top-down}(\mathcal{A}, \mathcal{R}, a) = \texttt{accepted} \iff a \in \mathcal{F} \tag{4.10}$$

$$\textit{top-down}(\mathcal{A}, \mathcal{R}, a) = \texttt{rejected} \iff a \notin \mathcal{F} \tag{4.11}$$

Proof. According to Theorem 4.8, the fix point of function $\mathcal{F}$ returned by Algorithm 11 contains the set of arguments considered acceptable by the acceptability semantics of Dung. On the other hand, the top-down algorithm is a direct implementation of the same acceptability semantics, and thus the double implication of Eq. (4.10) must follow. To prove the second double-implication of Eq. (4.11) we can prove the equivalent expression with both sides negated, i.e.,

$$\text{top-down}(\mathcal{A}, \mathcal{R}, a) \neq \texttt{rejected} \iff a \in \mathcal{F}.$$

Since a is not in abeyance, if the top-down algorithm does not return `rejected` it must return `accepted`. The double implication is thus equivalent to Eq. (4.10), which was proved true. □

4.3.1 Argumentation in Independence Knowledge Bases

We can apply the argumentation framework to our problem of answering queries from knowledge bases that contain a number of potentially inconsistent independences and dependencies and a set of rules that express relations among them.

Definition 4.11. *An* independence knowledge base ***(IKB)*** *is a knowledge base $\langle \Sigma, \Psi \rangle$ such that its proposition set Σ contains only independence propositions of the form $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$ or $(\mathbf{X} \not\perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$, for $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ three disjoint subsets of $\mathbf{V}$, and its set of inference rules Ψ is*

either the general set of axioms shown in Eqs. (4.5), or the specific set of axioms of undirected models shown Eqs. (4.7), or the specific set of axioms of directed models shown in Eqs. (4.6).

For IKBs, the set of arguments $\mathcal{A}$ is constructed in two steps. First, for each proposition $\sigma \in \Sigma$ (independence or dependence) we add to $\mathcal{A}$ the argument $(\{\sigma\}, \sigma)$. This is a valid argument according to Definition 4.2 since its support $\{\sigma\}$ is (trivially) consistent, it (trivially) implies the head σ, and it is minimal (the pair $(\{\varnothing\}, \sigma)$ is not a valid argument since $\varnothing$ is equivalent to the proposition `true` which does not entail σ in general. Arguments of the form $(\{\sigma\}, \sigma)$ are called *propositional arguments* since they correspond to single propositions. The second step in the construction of the set of arguments $\mathcal{A}$ concerns rules and proceeds as follows: for each inference rule $(\Phi_1 \wedge \Phi_2 \ldots \wedge \Phi_n \implies \varphi) \in \Psi$, and each subset of Σ that matches exactly the set of antecedents i.e., each subset $\{\varphi_1, \varphi_2 \ldots, \varphi_n\}$ of Σ such that $\Phi_1 \equiv \varphi_1, \Phi_2 \equiv \varphi_2 \ldots \Phi_n \equiv \varphi_n$, we add argument $(\{\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n\}, \varphi)$ to $\mathcal{A}$.[1]

IKBs can be augmented with a set of preferences that allow one to take into account the reliability of tests when deciding on the truth value of independence queries. This is described in the next section.

4.3.2 Preference-based Argumentation Framework

Following Amgoud and Cayrol (2002), we now refine the argumentation framework of Dung (1995) for cases where it is possible to define a preference order Π over arguments.

Definition 4.12. *A* preference-based argumentation framework ***(PAF)*** *is a triplet* $\langle \mathcal{A}, \mathcal{R}, \Pi \rangle$ *where* $\mathcal{A}$ *is a set of arguments,* $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ *is a binary relation representing a defeat relationship between pairs of arguments, and* Π *is a (partial or complete) ordering over* $\mathcal{A} \times \mathcal{A}$*.*

For the case of inconsistent knowledge bases, preference Π over arguments follows the preference π over their support i.e., stronger support implies a stronger argument, which is given as a partial or total order over sets of propositions. Formally:

[1] This is equivalent to propositionalizing the set of rules, some of which may be first-order (the rules of Eqs. (4.5) and (4.6) are, as they are universally quantified over all sets of variables). As this may be expensive (exponential in the number of propositions), in practice it may not be implemented in this way, instead matching appropriate rules on the fly during the argumentation inference process.

Definition 4.13. *Let $\mathcal{K} = \langle \Sigma, \Psi \rangle$ be a knowledge base, π be a (partial or total) ordering on subsets of Σ and (H, h), (H', h') two arguments over $\mathcal{K}$. Argument (H, h) is* π-preferred *to (H', h') (denoted $(H, h) \gg_\pi (H', h')$) if and only if H is preferred to H' with respect to π.*

In what follows we overload our notation by using π to denote either the ordering over arguments or over their supports.

An important sub class of preference relations is the *strict* preference relation defined as follows.

Definition 4.14. *We say preference relation π over arguments is* strict *if the ordering of arguments induced by it is a strict total ordering, i.e., for every pair a and b of arguments,*

$$a \gg_\pi b \iff \neg\big(b \gg_\pi a\big). \tag{4.12}$$

The importance of the property of strictness will become clear later when we argue the correctness of the argumentative independence test (defined in Section 4.4).

We now introduce the concept of *attack* relation, a combination of the defeat and preference relations.

Definition 4.15. *Let $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ be a PAF, and a, $b \in \mathcal{A}$ be two arguments. We say b* attacks *a if and only if $b \mathcal{R} a$ and $\neg(a \gg_\pi b)$.*

We can see that a preference-based argumentation framework is a special case of the more general argumentation framework, having a more refined defeat relation. Therefore the same conclusions apply, in particular Theorem 4.8, which allows us to compute the set of acceptable arguments of a PAF using Alg. 11 or Alg. 12.

In Sections 4.3.3 and 4.4 below, we apply these ideas to construct a more reliable approximation to the independence-query oracle.

4.3.3 Preference-based Argumentation in Independence Knowledge Bases

In this section we describe how to apply the preference-based argumentation framework of Section 4.3.2 to improve the reliability of conditional independence tests conducted on (possibly small) data sets.

A preference-based argumentation framework has three components. The first two, namely $\mathcal{A}$ and $\mathcal{R}$ are identical to general argumentation frameworks. We now describe how we construct the third component, namely the preference ordering π over subsets H of Σ, in IKBs. We define it using the probability $\nu(H)$ that all propositions in H are correct, that is

$$H \gg_\pi H' \iff \nu(H) > \nu(H'). \tag{4.13}$$

Before we explain how we compute $\nu(H)$, let us prove that π, as defined by the above equation, is strict (c.f. Definition 4.14).

Lemma 4.16. *The preference relation for independence knowledge bases defined by Equation 4.13 is strict.*

Proof.

$$\begin{aligned} H \gg_\pi H' &\iff \nu(H) > \nu(H') && \text{by Eq. (4.13)} \\ &\iff \neg(\nu(H') > \nu(H)) \\ &\iff \neg(H' \gg_\pi H') && \text{again by Eq. (4.13)} \end{aligned}$$

□

Note that the second step would not be true if we allow equality in Eq. (4.13), i.e., if our definition for the preference states that H is preferred to H' when $\nu(H) = \nu(H')$.

We compute the probability $\nu(H)$ that a set of propositions H is correct by assuming independence among these propositions. Overloading notation and denoting by $\nu(h)$ the probability of an individual proposition h being correct, the probability of all elements in H being correct under this assumption of independence is

$$\nu(H) = \prod_{h \in H} \nu(h). \tag{4.14}$$

In our case we have independence propositions. The probability that an independence proposition is correct can be computed in different ways, depending on the particular choice of independence oracle chosen. In this chapter we use Wilk's G^2 test. As discussed in Section 4.2,

the p-value $p(X, Y \mid \mathbf{Z})$ computed by this test is the probability of error in assuming that X and Y are dependent when in fact they are not. Therefore, the probability of a test returning dependence of being correct is

$$\nu_D(X \not\perp\!\!\!\perp Y \mid \mathbf{Z}) = 1 - p(X, Y \mid \mathbf{Z}) \tag{4.15}$$

where the subscript D indicates that this expression is valid only for dependencies.

The probability of correctly reporting an independence is defined in terms of the β-value, the probability of incorrectly reporting independence when in fact the variables are dependent:

$$\nu_I(X \perp\!\!\!\perp Y \mid \mathbf{Z}) = 1 - \beta(X, Y \mid \mathbf{Z}) \tag{4.16}$$

where again the subscript I indicates that it is valid only for independences.

To the best of our knowledge, the general computation of the β-value is an open problem. While computing the p-value involves evaluating the probability of a statistic under the distribution generated by the independence model, i.e., a model under which the variables are independent, which for discrete domains is unique, computing β is difficult because there are infinitely many possible models in which the variables are dependent. In statistical applications, the β value is commonly approximated by assuming one particular dependence model if some prior knowledge is available. In the absence of such information however we take an alternative approach of approximating the β-value from the p-value. We estimate the β-value of a test on triplet $(X, Y \mid \mathbf{Z})$ from the p-value assuming the following heuristic constraints on β:

$$\beta(p(X, Y \mid \mathbf{Z})) = \begin{cases} 0 & \text{if } p(X, Y \mid \mathbf{Z}) = 0 \\ \alpha + \frac{1-\alpha}{2+|\mathbf{Z}|} & \text{if } p(X, Y \mid \mathbf{Z}) = 1 \\ 1 - \alpha & \text{if } p(X, Y \mid \mathbf{Z}) = 1 - \alpha. \end{cases}$$

The first constraint (for $p(X, Y \mid \mathbf{Z}) = 0$) is justified by the intuition that when the p-value of the test is close to 0, the test statistic is very far from its value under the model that assumes independence, and thus we would give more preference to the "dependence" decision. The situation for the second case ($p(X, Y \mid \mathbf{Z}) = 1$) is reversed —the statistic is very close from the expected one under independence, and therefore independence is preferred.

The value of the second case is tempered by the number of variables in the conditioning set. This reflects the practical consideration that, as the number $2 + |\mathbf{Z}|$ of variables involved in the test increases, given a fixed data set, the reliability of the test diminishes, going to 0 as $|\mathbf{Z}| \to \infty$. The third case is related to fairness: In the absence of non-propositional arguments (i.e., in the absence of inference rules in the knowledge-base), the independence decisions of the argumentation framework should match those of the purely statistical tests. Otherwise, changes in the outcome of tests may be due to simply bias in the independence decision that favors dependence or independence i.e., it is equivalent to an arbitrary change to the threshold of the statistical test, and the comparison of the two tests would not be a fair one.

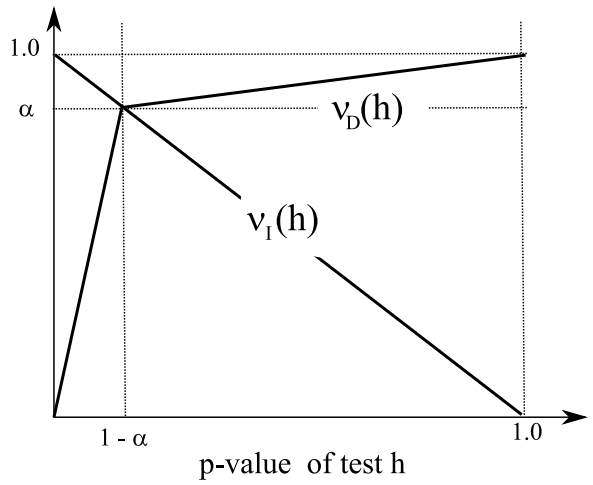


Figure 4.1 The probability of correct independence $\nu_I(h) = 1 - \beta(p(h))$ and the probability of correct dependence $\nu_D(h) = 1 - p(h)$ as a function of the p-value $p(h)$ of test h.

The remaining values of β are approximated by linear interpolation among the above points. The result is summarized in Fig. (4.1), which shows the probabilities of dependence ν_D (i.e., $1 - p$) and ν_I (i.e., $1 - \beta$) versus p.

We now use the following example to illustrate how preference-based argumentation can be used to resolve the inconsistencies of Example 4.1.

Example 4.3. *Example 4.1 considered an IKB with the following propositions*

$$(0 \perp\!\!\!\perp 1 \mid \{2,3\}) \tag{4.17}$$

$$(0 \perp\!\!\!\perp 4 \mid \{2,3\}) \tag{4.18}$$

$$(0 \not\perp\!\!\!\perp \{1,4\} \mid \{2,3\}) \tag{4.19}$$

$$(0 \perp\!\!\!\perp 1 \mid \{2,3\}) \wedge (0 \perp\!\!\!\perp 4 \mid \{2,3\}) \implies (0 \perp\!\!\!\perp \{1,4\} \mid \{2,3\}) \tag{4.20}$$

Let us extend this IKB with the following preference values for its propositions and rules.

$$\begin{aligned} \mathit{Pref}\left[(0 \perp\!\!\!\perp 1 \mid \{2,3\})\right] &= 0.8 \\ \mathit{Pref}\left[(0 \perp\!\!\!\perp 4 \mid \{2,3\})\right] &= 0.7 \\ \mathit{Pref}\left[(0 \not\perp\!\!\!\perp \{1,4\} \mid \{2,3\})\right] &= 0.5 \end{aligned}$$

Following the IKB construction procedure described in the previous section, the above propositions correspond to the following arguments, respectively:

$$\left(\left\{(0 \perp\!\!\!\perp 1 \mid \{2,3\})\right\}, (0 \perp\!\!\!\perp 1 \mid \{2,3\})\right) \tag{4.21}$$

$$\left(\left\{(0 \perp\!\!\!\perp 4 \mid \{2,3\})\right\}, (0 \perp\!\!\!\perp 4 \mid \{2,3\})\right) \tag{4.22}$$

$$\left(\left\{(0 \not\perp\!\!\!\perp \{1,4\} \mid \{2,3\})\right\}, (0 \not\perp\!\!\!\perp \{1,4\} \mid \{2,3\})\right) \tag{4.23}$$

and rule (4.20) corresponds to the following argument

$$\left(\left\{(0 \perp\!\!\!\perp 1 \mid \{2,3\}), (0 \perp\!\!\!\perp 4 \mid \{2,3\})\right\}, (0 \perp\!\!\!\perp \{1,4\} \mid \{2,3\})\right). \tag{4.24}$$

The preference of each argument $(\{\sigma\}, \sigma)$ is equal to the preference value of $\{\sigma\}$, according to Definition 4.13, which, as it contains only a single proposition, is equal to the preference of σ. Therefore,

$$\begin{aligned} \mathit{Pref}\left[\left(\left\{(0 \perp\!\!\!\perp 1 \mid \{2,3\})\right\}, (0 \perp\!\!\!\perp 1 \mid \{2,3\})\right)\right] &= 0.8 \\ \mathit{Pref}\left[\left(\left\{(0 \perp\!\!\!\perp 4 \mid \{2,3\})\right\}, (0 \perp\!\!\!\perp 4 \mid \{2,3\})\right)\right] &= 0.7 \\ \mathit{Pref}\left[\left(\left\{(0 \not\perp\!\!\!\perp \{1,4\} \mid \{2,3\})\right\}, (0 \not\perp\!\!\!\perp \{1,4\} \mid \{2,3\})\right)\right] &= 0.5. \end{aligned}$$

The preference of argument (4.24) equals the preference of the set of its antecedents, which, according to Eq. (4.14), is equal to the product of their individual preferences i.e.,

$$\mathit{Pref}\left[\left(\left\{(0\perp\!\!\!\perp 1 \mid \{2,3\}), (0\perp\!\!\!\perp 4 \mid \{2,3\})\right\}, (0\perp\!\!\!\perp 1 \mid \{2,3\})\right)\right] \;=\; 0.8 \times 0.7 = 0.56.$$

Now, even though proposition (4.19) and rule (4.20) contradict each other logically, i.e., their corresponding arguments (4.23) and (4.24) defeat each other, argument (4.24) defends itself because its preference is 0.56 *which is larger than* 0.5*, the preference of its defeater argument (4.23). Also, since no other argument defeats (4.24), it is acceptable, and (4.23), being attacked by an acceptable argument, must be rejected. We therefore see that using preferences the inconsistency of Example 4.1 has been resolved in favor of rule 4.20.*

We now extend the above example to illustrate the defend relation, i.e., how an argument can be defended by some other argument. The example also illustrates an alternative resolution for the inconsistency of Example 4.1, this time in favor of the dependence proposition 4.19.

Example 4.4. *Let us extend the IKB of Example 4.3 with two additional independence propositions and an additional rule.*

The new propositions and their corresponding preferences are:

$$\begin{aligned}\mathit{Pref}\left[(0\perp\!\!\!\perp 5 \mid \{2,3\})\right] &= 0.8\\ \mathit{Pref}\left[(0\not\perp\!\!\!\perp \{1,5\} \mid \{2,3\})\right] &= 0.9.\end{aligned}$$

and the new rule is:

$$(0\perp\!\!\!\perp 5 \mid \{2,3\}) \wedge (0\not\perp\!\!\!\perp \{1,5\} \mid \{2,3\}) \implies (0\not\perp\!\!\!\perp 1 \mid \{2,3\}).$$

This rule is an instance of the Composition axiom in contrapositive form.

The corresponding arguments are therefore:

$$\begin{aligned}\mathit{Pref}\left[\left(\left\{(0\perp\!\!\!\perp 5 \mid \{2,3\})\right\}, (0\perp\!\!\!\perp 5 \mid \{2,3\})\right)\right] &= 0.8\\ \mathit{Pref}\left[\left(\left\{(0\not\perp\!\!\!\perp \{1,5\} \mid \{2,3\})\right\}, (0\not\perp\!\!\!\perp \{1,5\} \mid \{2,3\})\right)\right] &= 0.9\end{aligned}$$

corresponding to the two propositions, and

$$\mathit{Pref}\left[\left(\left\{(0\perp\!\!\!\perp 5 \mid \{2,3\}), (0\not\perp\!\!\!\perp \{1,5\} \mid \{2,3\})\right\}, (0\not\perp\!\!\!\perp 1 \mid \{2,3\})\right)\right] = 0.8 \times 0.9 = 0.72 \quad (4.25)$$

corresponding to the rule.

As in Example 4.3, argument (4.23) is attacked by argument (4.24). Let us represent this graphically using an arrow from argument a to argument b to denote that a attacks b, i.e.,

$$\textit{Argument } 4.24 \longrightarrow \textit{Argument } 4.23$$

If the IKB was as in Example 4.3, (4.24) would had been accepted and (4.23) would have been rejected. However, the additional argument (4.25) defeats (undercuts) (4.24), by logically contradicting its antecedent $(\{0\} \perp\!\!\!\perp \{1\} \mid \{2,3\})$*. Since (4.25) also attacks (4.24) i.e., its preference* 0.72 *is larger than* 0.56*, the preference of (4.24), (4.25) defends all arguments that are attacked by argument (4.24), in particular (4.23). Graphically,*

$$\textit{Argument } (4.25) \longrightarrow \textit{Argument } (4.24) \longrightarrow \textit{Argument } (4.23).$$

Note this is not sufficient for accepting (4.23) as it has not been proved that its defender (4.25) is itself acceptable. We leave the proof of this as an exercise for the reader.

4.4 Argumentative Independence Tests (AITs)

The independence-based preference argumentation framework described in the previous section provides a semantics for the acceptance of arguments consisting of independence propositions. However, what we need is a procedure for a test of independence that, given as input a triplet $\sigma = (X, Y \mid \mathbf{Z})$ responds whether X is independent or dependent of Y given $\mathbf{Z}$. In other words, we need a semantics for the acceptance of propositions, not arguments. Let us consider the two propositions related to the input triplet $\sigma = (X, Y \mid \mathbf{Z})$, proposition $(\sigma = \texttt{true})$, abbreviated $\sigma_{\texttt{t}}$, and proposition $(\sigma = \texttt{false})$, abbreviated $\sigma_{\texttt{f}}$, that correspond to independence $(X \perp\!\!\!\perp Y \mid \mathbf{Z})$ and dependence $(X \not\perp\!\!\!\perp Y \mid \mathbf{Z})$ of σ, respectively. The basic idea for deciding on the independence or dependence of input triplet σ is to define a semantics for the acceptance or rejection of propositions $\sigma_{\texttt{t}}$ and $\sigma_{\texttt{f}}$ based on the acceptance or rejection of their respective propositional arguments $(\{\sigma_{\texttt{t}}\}, \sigma_{\texttt{t}})$ and $(\{\sigma_{\texttt{f}}\}, \sigma_{\texttt{f}})$. Formally,

$$\begin{aligned} (X \not\perp\!\!\!\perp Y \mid \mathbf{Z}) \text{ is accepted} \quad & \text{iff } (\{(X \not\perp\!\!\!\perp Y \mid \mathbf{Z})\}, (X \not\perp\!\!\!\perp Y \mid \mathbf{Z})) \text{ is accepted, and} \\ (X \perp\!\!\!\perp Y \mid \mathbf{Z}) \text{ is accepted} \quad & \text{iff } (\{(X \perp\!\!\!\perp Y \mid \mathbf{Z})\}, (X \perp\!\!\!\perp Y \mid \mathbf{Z})) \text{ is accepted.} \end{aligned} \tag{4.26}$$

Based on this semantics over propositions, we decide on the dependence or independence of triplet σ as follows:

$$\begin{aligned} \sigma_{\mathtt{t}} = (X \perp\!\!\!\perp Y \mid \mathbf{Z}) \text{ is accepted } &\implies (X \perp\!\!\!\perp Y \mid \mathbf{Z}) \\ \sigma_{\mathtt{f}} = (X \not\perp\!\!\!\perp Y \mid \mathbf{Z}) \text{ is accepted } &\implies (X \not\perp\!\!\!\perp Y \mid \mathbf{Z}). \end{aligned} \tag{4.27}$$

For the above semantics to be well-defined, a triplet σ must be either independent or dependent, not both, or neither. For that, exactly one of the antecedents of the above implications must be true. Formally,

Theorem 4.17. *For any input triplet $\sigma = (X, Y \mid \mathbf{Z})$, the argumentative independence test (**AIT**) defined by Eqs. (4.26) and (4.27) produces a non-ambiguous decision, i.e. it decides σ evaluates to either independence or dependence, but nor both or neither.*

For that to happen, one and only one of its corresponding propositions $\sigma_{\mathtt{t}}$ or $\sigma_{\mathtt{f}}$ must be accepted. A necessary condition for this is given by the following theorem.

Theorem 4.18. *Given a PAF $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ with a strict preference relation π, every propositional argument $(\{\sigma_t\}, \sigma_t) \in \mathcal{A}$ and its negation $(\{\sigma_f\}, \sigma_f)$ satisfy*

$$(\{\sigma_t\}, \sigma_t) \textit{ is accepted iff } (\{\sigma_f\}, \sigma_f) \textit{ is rejected.}$$

The above theorem is not sufficient because the propositions may still be in abeyance, but this possibility is ruled out for strict preference relations by Theorem 4.19, presented later.

The formal proofs of these theorems are presented in Appendix C. We now illustrate the use of AIT with an example.

Example 4.5. *We consider an extension of Example 4.3 to illustrate the use of the AIT to decide on the independence or dependence of input triplet $(\{0\}, \{1, 4\} \mid \{2, 3\})$. According to Eq. (4.26) the decision depends on the status of the two propositional arguments:*

$$(\{(0 \perp\!\!\!\perp \{1, 4\} \mid \{2, 3\})\}, (0 \perp\!\!\!\perp \{1, 4\} \mid \{2, 3\})), \textit{ and} \tag{4.28}$$

$$(\{(0 \not\perp\!\!\!\perp \{1, 4\} \mid \{2, 3\})\}, (0 \not\perp\!\!\!\perp \{1, 4\} \mid \{2, 3\})). \tag{4.29}$$

Argument (4.29) is equivalent to argument (4.23) of Example 4.3, that was proved to be rejected. According to Theorem 4.18, its negated propositional argument Eq. (4.28) must be accepted, and we can conclude triplet $(\{0\}, \{1,4\} \mid \{2,3\})$ *is independent, i.e.,* $(\{0\} \perp\!\!\!\perp \{1,4\} \mid \{2,3\})$*.*

4.5 Top-down AIT algorithm

We now discuss the top-down algorithm introducted in Section 4.3 in more detail. We start by simplifying the recursion of Eq. (4.9) that determines the state (accepted, rejected, or in abeyance) of an argument a. We then explain the algorithm and we conclude by analyzing its computability (i.e., the property that the recursion is finite), its time complexity, and present some approximations to reduce its running time to polynomial.

To simplify the recursion Eq. (4.9) we consider the following theorem (proved in Appendix C).

Theorem 4.19. *Let* $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ *be a PAF with a strict preference relation* π*. Then no argument* $a \in \mathcal{A}$ *is in abeyance.*

This theorem reduces the number of states of each argument, and its corresponding node in the tree, to only two, i.e., an argument can be either accepted or not accepted (rejected). We will use the name of the argument a to denote the predicate "a is accepted" and its negation $\neg a$ to denote the predicate "a is rejected." With this notation, the above theorem, and the fact that we have extended the semantics of acceptability from the defeat to the attack relation, recursion Eq. (4.9) can be expressed as follows,

$$\begin{aligned} a &\iff \forall b \in attackers(a),\ \neg b \\ \neg a &\iff \exists b \in attackers(a),\ b \end{aligned}$$

or alternatively,

$$\begin{aligned} a &\iff \bigwedge_{b \in attackers(a)} \neg b \\ \neg a &\iff \bigvee_{b \in attackers(b)} b. \end{aligned} \tag{4.30}$$

Finally, we notice that the second formula is logically equivalent to the first (simply negate both sides of the double implication to recover the first). Therefore, the boolean value of the dialog tree for a can be computed by the simple expression

$$a \iff \bigwedge_{b \in attackers(a)} \neg b.$$

To illustrate, consider an attacker b of a. If b is rejected, i.e., $\neg b$, the conjunction on the right cannot be determined. Only when all attackers of a are known to be rejected, can the value of a be determined to be a, i.e., accepted. Instead, if b is accepted, i.e., b, the state of $\neg b$ is `false` and the conjunction can be immediately evaluated to `false`, i.e., a is rejected.

An iterative version of the top-down algorithm is shown in Algorithm 13. We assume that the algorithm can access a global PAF $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$, with arguments in $\mathcal{A}$ defined over a knowledge base $\mathcal{K} = \langle \Sigma, \Psi \rangle$.

Given as input a triplet $t = (X, Y \mid \mathbf{Z})$, if the algorithm returns `true` (`false`) then we conclude that t is independent (dependent). It starts by creating a root node u for the propositional argument U of proposition $t =$ `true` (lines 1-6). According to Eqs. (4.26) and (4.27), the algorithm then decides `true` if U is accepted. Otherwise, the algorithm returns `false` (lines 22 and 23). This is because in that case, according to Theorem 4.18, the negation of propositional argument U must be accepted.

The algorithm is an iterative version of a tree traversal algorithm. It maintains a queue of the nodes that have not been expanded yet. A node is expanded when its children are added to the tree. In the algorithm, this is done in the loop of lines 17 to 21, which access subroutine *getAttackers* of Algorithm 15 to obtain all attackers of an argument. This subroutine finds all attackers of the input argument a in a backward-chaining fashion, i.e., given an argument $a = (H, h)$, it searches for all rules in the knowledge base $\mathcal{K}$ whose consequent matches the negation of some proposition in the support H, or the negation of its head h. Every node maintains a three state variable $STATE \in \{\texttt{nil}, \texttt{accepted}, \texttt{rejected}\}$. The `nil` state denotes that the value of the node is not yet known, and a node is initialized to this state when it is added to the tree.

Algorithm 13 *independent*(*triplet* t).

1: $f_{\texttt{true}} \leftarrow$ proposition $(t = \texttt{true})$
2: $U_{\texttt{true}} \leftarrow (\{f_{\texttt{true}}\}, f_{\texttt{true}})$
3: $u_{\texttt{true}} \leftarrow$ node for argument $U_{\texttt{true}}$
4: $u_{\texttt{true}}.parent \leftarrow$ `nil`
5: $u.STATE \leftarrow$ `nil`
6: $fringe \leftarrow [u]$ *// Initialize with u (root).*
7: */* Create global rejected node. */*
8: $\perp \leftarrow$ node with no argument and state `rejected` *//$\perp$ represents a global rejected node.*
9: **while** $fringe \neq \varnothing$ **do**
10: $u \leftarrow pop(fringe)$
11: $attackers \leftarrow getAttackers(u.argument)$
12: **if** $(attackers = \varnothing)$ **then**
13: $u.STATE \leftarrow$ `accepted`
14: **if** $sendMsg(\perp, u) = terminate$ **then** break
15: $attackers \leftarrow$ sort attackers in decreasing order of preference.
16: */* Enqueue attackers after decomposing them. */*
17: **for each** $A \in attackers$ **do**
18: $a \leftarrow$ node for argument A
19: $a.parent \leftarrow u$
20: $a.STATE \leftarrow$ `nil`
21: *enqueue* a in $fringe$
22: **if** $(u.STATE =$ `accepted`$)$ **then return** `true`
23: **if** $(u.STATE =$ `rejected`$)$ **then return** `false`

The algorithm proceeds until a node is found that has no attackers (line 12). Such a node is accepted in line 13 (because it defends itself) and its $STATE$ is propagated to the parent using subroutine $sendMsg$ (Algorithm 14). Every time a node receives a message from a child, if the message is `accepted`, the node is rejected (line 3 of Algorithm 14), otherwise the node is accepted if all its children has been evaluated to `rejected` (line 4 of Algorithm 14). The subroutine $sendMsg$ then proceeds recursively by forwarding a message to the parent whenever a node has been evaluated (line 7). If the root is reached and evaluated, the message is sent to its parent, which is `nil`. In this case, the subroutine returns the special keyword *terminate* back to the caller, indicating that the root has been evaluated and thus the main algorithm (Algorithm 13) can terminate. The caller can be either the subroutine $sendMsg$, in which case

Algorithm 14 $sendMsg(Node\ c, Node\ p)$.

```
1: /* Try to evaluate parent p given new info c.VALUE */
2: if p ≠ nil then
3:     if c.STATE = accepted then p.STATE ← rejected
4:     else if (∀ children q of p, q.STATE ≠ nil) then p.STATE ← accepted
5:     /* If p was successfully evaluated, try to evaluate its parent by sending message upward.
       */
6:     if p.STATE ≠ nil then
7:         return sendMsg(p, p.parent)
8:     else
9:         return continue
10: else
11:     return terminate /* Root node has been evaluated. */
```

```
1: attackers ← ∅
2: /* Get all undercutters or rebutters of a. */
3: for all propositions φ ∈ H ∪ {h} do
4:     /* Get all defeaters of proposition φ. */
5:     for all rules (Φ1 ∧ Φ2 ... ∧ Φn ⟹ ¬φ) ∈ Ψ do
6:         /* Find all propositionalizations of the rule whose consequent matches ¬φ. */
7:         for all subsets {φ1, φ2 ..., φn} of Σ s.t. Φ1 ≡ φ1, Φ2 ≡ φ2 ... Φn ≡ φn do
8:             d ← ({φ1 ∧ φ2 ... φn}, ¬φ) /* Create defeater. */
9:             /* Is the defeater an attacker? */
10:            if ¬(a ≫π d) then
11:                attackers ← attackers ∪ {d}
12: return attackers
```

Algorithm 15 Finds all attackers of input argument a in knowledge base $\mathcal{K} = \langle \Sigma, \Psi \rangle$: $getAttackers(a = (H, h))$

it pushes the returned message up the method-calling stack, or the top-down algorithm in line 14, in which case the "while" loop is terminated.

An important part of the algorithm is yet underspecified, namely the order in which the attackers of a node are explored in the tree (i.e., the manner in which nodes are enqueued in line 21). Possible orderings are depth-first, breadth-first, iterative deepening, as well as informed searches such as best-first when a heuristic is available.

In our experiments we used iterative deepening because it has the benefits of both depth-

first and breadth-first search, i.e., small memory requirements in the same order as depth-first search (i.e., in the order of the maximum number of children a node can have) but also the advantage of finding the shallowest solution like breadth-first search. We also used a heuristic for enqueuing the children of a node. According to iterative deepening, the position in the queue of the children of a node is specified relative to other nodes, but not relative to each other. We therefore specified the relative order of the children according the value of preference function. Children with higher preference are enqueued first (line 15 of the top-down algorithm), and thus, according to iterative deepening, would be dequeued first.

4.5.1 Computability of the Top-Down Algorithm

We now prove that, under certain general conditions, the acceptability of an argument a can always be determined by the top-down algorithm, i.e., the algorithm always terminates. This is shown in the following theorem.

Theorem 4.20. *Given an arbitrary triplet $t = (X, Y \mid \mathbf{Z})$, and a PAF $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ with a strict preference relation π, Algorithm 13 with input t over $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ terminates.*

The proof consists on showing that the path from the root a to any leaf is always finite. For that, let us first define the concept of an attack sequence.

Definition 4.21. *An* attack sequence *is a sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of n arguments such that for every $i \in [2, n]$, a_i attacks a_{i-1}.*

It is clear by the way we construct the tree, that any path from the root to a leaf is an attack sequence. It suffices to show then that any such sequence is finite. This is done by the following theorem.

Theorem 4.22. *Let $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ be a PAF with a strict preference relation π and a finite set $\mathcal{A}$ of arguments. Then any attack sequence is finite.*

Intuitively, if the preference relation is strict then an element can attack its predecessor in the sequence but not vice versa. Since the set of arguments $\mathcal{A}$ is finite, the only way for

an attack sequence to be infinite is to contain a cycle. In that case, an argument would be attacking its predecessor, which cannot happen in a PAF with a strict preference relation. We present a formal proof in Appendix C.

We thus arrived to the important conclusion that, under the conditions of strict preference function and finite argument set, the state of any argument is computable. In Section 4.3.3 we showed that the preference function for independence knowledge bases is strict, and thus the computability of the top-down algorithm is guaranteed.

4.5.2 Time Complexity of the Top-Down Algorithm

Since Algorithm 13 is essentially a tree traversal algorithm, its time complexity can be obtained by standard techniques contained in Algorithms texts, e.g. Cormen et al. (2001), and depends on the exploration procedure we choose. In our case we used iterative deepening. The time complexity of an iterative deepening search algorithm is $O(b^d)$, where b is the *branching factor* and d is the smallest depth at which the algorithm terminates. Therefore, the time is exponential in d. Unfortunately, for the case of independence tests b may itself be exponential. This is because the inference rules of Eqs. (4.5), (4.7), and (4.6) are universally quantified and therefore their propositionalization (lines 7–11 of Algorithm 15), may result in an exponential number of rules with the same consequent.

4.6 Approximate Top-Down Algorithm

In order to obtain a practical algorithm we propose an approximation of the top-down algorithm. To address the exponential behavior of the search we set a *cutoff* depth d for iterative deepening, and to address the exponential behavior of the branching factor b we consider an alternative to Algorithm 15 that produces a polynomial number of defeaters (which bounds the number of attackers) during the propositionalization (lines 7–11 of Algorithm 15). This propositionalization produces, for some given proposition φ, all rules $\{\varphi_1 \wedge \varphi_2 \ldots \wedge \varphi_n \implies \neg\varphi\}$ of Σ. Let $\Sigma_\varphi \subseteq \Sigma$ denote such a set of rules. If we let $\varphi = (\mathbf{X}, \mathbf{Y} \mid \mathbf{Z})$ and $\varphi_i = (\mathbf{X}_i, \mathbf{Y}_i \mid \mathbf{Z}_i)$, $i \in [1, n]$, the approximation considers a subset $\widehat{\Sigma}_\varphi$ of Σ_φ s.t. for all

$\{\varphi_1 \wedge \varphi_2 \ldots \wedge \varphi_n \implies \varphi\} \in \widehat{\Sigma}_\varphi$, and for all $i \in [1, n]$,

$$
\begin{aligned}
|\mathbf{X}| - c < \quad & |\mathbf{X}_i| \quad < |\mathbf{X}| + c \\
|\mathbf{Y}| - c < \quad & |\mathbf{Y}_i| \quad < |\mathbf{Y}| + c \\
|\mathbf{Z}| - c < \quad & |\mathbf{Z}_i| \quad < |\mathbf{Z}| + c
\end{aligned}
$$

where $|\cdot|$ denotes set cardinality, and c is some user-specified integer parameter that defines the approximation. We call this algorithm the **approximate top-down algorithm**. In the experiments shown in the next section we used $c = 1$ and $d = 3$.

4.7 Experimental Results

The main focus of the present chapter is to demonstrate that the argumentation approach does indeed improve the accuracy of independence tests evaluated on small data sets. For that, we conducted experiments on sampled and real-world data sets and compared the accuracy (defined precisely later in the section) of the bottom-up, exact top-down, and approximate top-down versions of the argumentative independence tests (denoted $\mathbf{AIT}_\mathrm{b}$, $\mathbf{AIT}_\mathrm{t}$, and $\widehat{\mathbf{AIT}}$, respectively), versus their statistical counterpart (denoted **SIT**), for varying reliability conditions, obtained by conducting experiments on varying data set sizes. The accuracy was estimated by performing a number of conditional independence tests on data, and comparing the result of each of these (`true` or `false`) with the true value of the corresponding independence, computed by querying the underlying model for the conditional independence value of the same test. This approach is similar to estimating accuracy in a classification task over unseen instances but with inputs here being triplets $(\mathbf{X}, \mathbf{Y} \mid \mathbf{Z})$ and the class attribute being the value of the corresponding conditional independence test. Results for accuracy of the bottom-up algorithm are presented in the following section, followed by the results for the top-down algorithm in Section 4.7.3.

We also compared the running time of the three variations of the argumentative tests. We conducted experiments to compare the running times of the bottom-up vs. the exact top-down algorithms. The results of these experiments are discussed in detail in Section 4.7.2

and demonstrate the great improvements in running time of the top-down algorithm compared against the bottom-up approach. We also conducted experiments to compare the running time of exact vs. approximate top-down algorithms. The results of these experiments are discussed in Section 4.7.4 and also demonstrate improvements in running time of the approximate top-down algorithm of almost an order of magnitude compared against the bottom-up approach.

4.7.1 Bottom-Up Algorithm Experiments

In this section we demonstrate that the argumentation approach, implemented using the bottom-up algorithm of Algorithm 11, improves the accuracy of independence tests on small data sets. For that, we generated the set of all propositional arguments possible i.e., arguments of the form $(\{\sigma\}, \sigma)$, by iterating over all possible triplets $(\mathbf{X}, \mathbf{Y} \mid \mathbf{Z})$, and inserted them in the knowledge base together with their preference, as described in Section 4.3.1. Similarly, for the set of axioms that we used in each case i.e., either the general (Eqs. (4.5)) or the specific ones (Eqs. (4.6)), we iterated over all possible matches of each rule, inserting the corresponding instantiated rule in the knowledge base again together with its preference. The reason for including all propositional and rule-based arguments in our IKB is to allow the argumentation framework to consider all possible arguments in favor of or against an independence query. The time complexity of the bottom-up algorithm is linear with the size $|\mathcal{A}|$ of the space of arguments, but $|\mathcal{A}|$ itself grows super-exponentially with the domain size n. This prevented us from exploring domain sizes larger than $n = 8$. While clearly the bottom-up algorithm is impractical, its (improved) accuracy results demonstrates the utility of our approach. We later present results for larger values of n obtained by using the top-down algorithm, which is much more efficient.

In the next section we present results for data sampled from Bayesian networks, where the underlying model is known and can be queried for conditional independence using d-separation. Following this, we present results of real-wold data experiments where the underlying model is unknown and thus the true values of the independences must be approximated; this is explained in detail below.

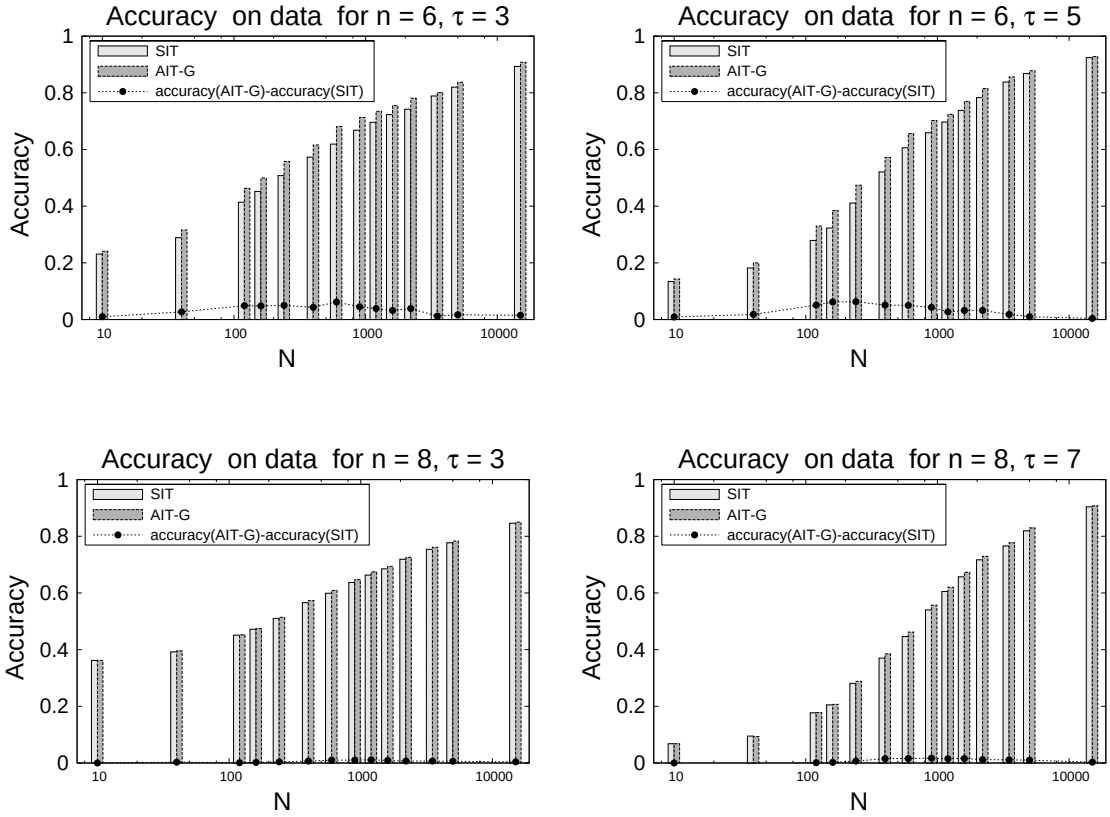


Figure 4.2 Comparison of statistical tests (SIT) vs. argumentative tests on the general axioms (AIT_b-G) for domain size $n = 6$ and $\tau = 3, 5$ and for $n = 8$ and $\tau = 3, 7$. The histograms show the absolute value of the accuracy while the line curves shows their difference i.e., a positive value corresponds to an improvement in the accuracy of AIT_b-G over the accuracy of SIT.

4.7.1.1 Sampled Data Experiments

In this set of experiments we compare the accuracy of argumentative tests versus purely statistical tests on several data sets sampled from a number of randomly generated Bayesian networks. Sampled data experiments have the advantage of a more precise estimation of the accuracy since the underlying model is known. We present experiments for two versions of the argumentative test, one that uses Pearl's general axioms of Eqs. (4.5), denoted $\mathbf{AIT}_\text{b}$**-G**, and another that uses Pearl's directed axioms of Eqs. (4.6), denoted $\mathbf{AIT}_\text{b}$**-D**.

The data was sampled from randomly generated Bayesian networks of different number of

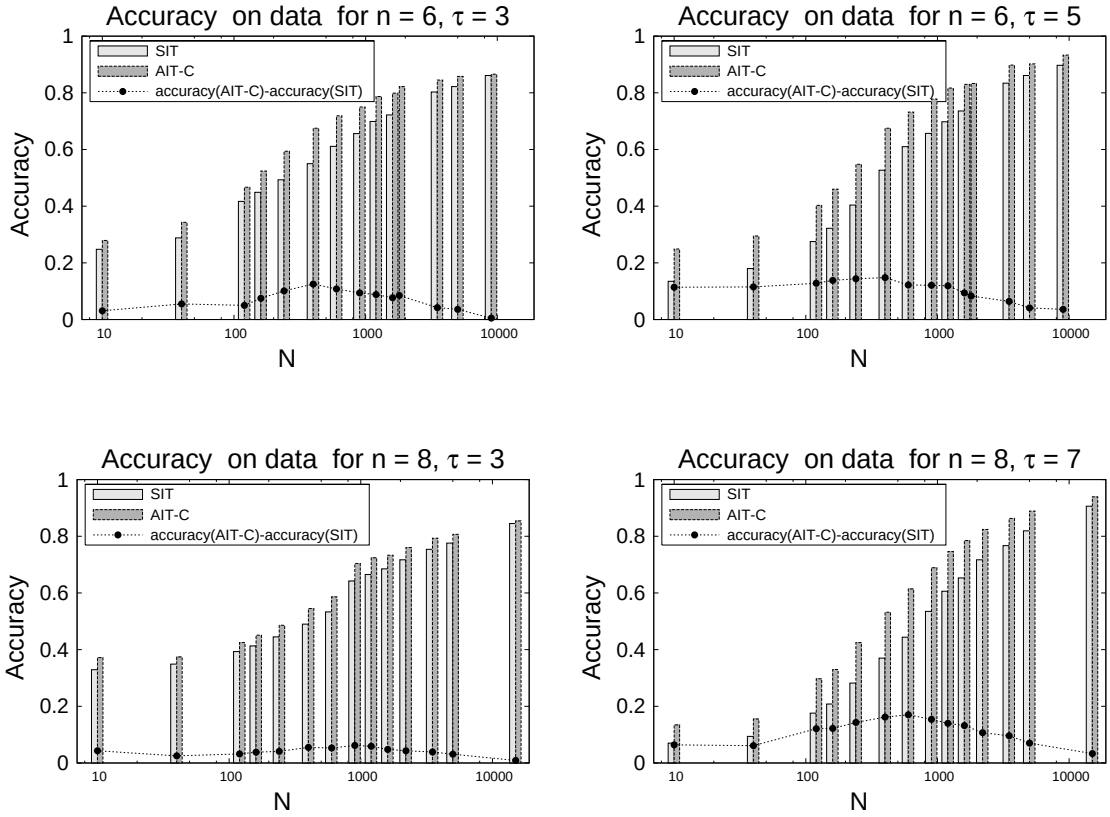


Figure 4.3 Comparison of statistical tests (SIT) vs. argumentative tests on the causal axioms (AIT_b-D) for domain size $n = 6$, maximum degree $\tau = 3, 5$ and domain size $n = 8$, maximum degree $\tau = 3, 7$. The histogram shows the absolute value of the accuracies and the line curve shows their difference i.e., a positive value correspond to an improvement in the accuracy of AIT_b-D over the accuracy of SIT.

nodes n and different maximum degrees per node τ (corresponding to different arc densities in the resulting graphs) using *BNGenerator* (Ide and Cozman, 2002), a publicly available Java package. For $n = 6$ we generated ten networks with $\tau = 3$ and ten networks with $\tau = 5$. For $n = 8$ we generated ten networks for $\tau = 3$ and another ten for $\tau = 7$. For each data set D in these four groups, we conducted experiments on subsets of D containing an increasing number of data points. This was done in order to assess the performance of the independence tests on varying conditions of reliability, as the reliability of a test typically decreases with decreasing data set size.

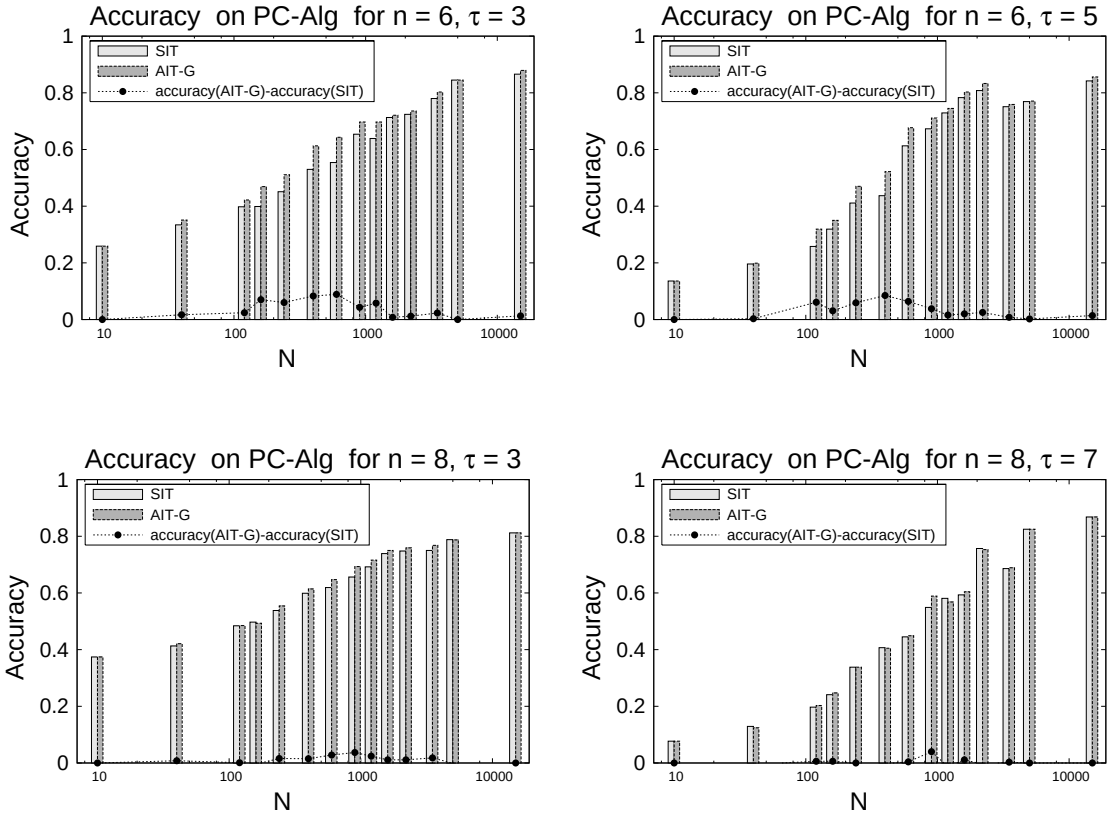


Figure 4.4 Comparison of the output of PC-Algorithm for SIT and AIT_b-G for domain size $n = 6$, maximum degree $\tau = 3, 5$ and domain size $n = 8$, maximum degree $\tau = 3, 7$. The histogram shows the absolute value of the accuracies and the line curve shows their difference i.e., a positive value correspond to an improvement in the accuracy of AIT_b-G over the accuracy of SIT.

For each experiment we report the estimated accuracy, calculated by comparing the result of a number of conditional independence tests (SITs or AITs) on data with the true value of independence, computed by querying the underlying model for the conditional independence of the same test using d-separation. Since the number of possible tests is exponential, we estimated the independence accuracy by sampling 2,000 triplets $(X, Y, \mathbf{Z})$ randomly, evenly distributed among all possible conditioning set sizes $m \in \{0, \ldots, n-2\}$ (i.e., $2000/(n-1)$ tests for each m). Denoting by $\mathcal{T}$ this set of 2,000 triplets, by $t \in \mathcal{T}$ a triplet, by $I_{\text{true}}(t)$ the result of a test performed on the underlying model, and by $I_{\text{data-}\mathcal{Y}}(t)$ the results of performing a test

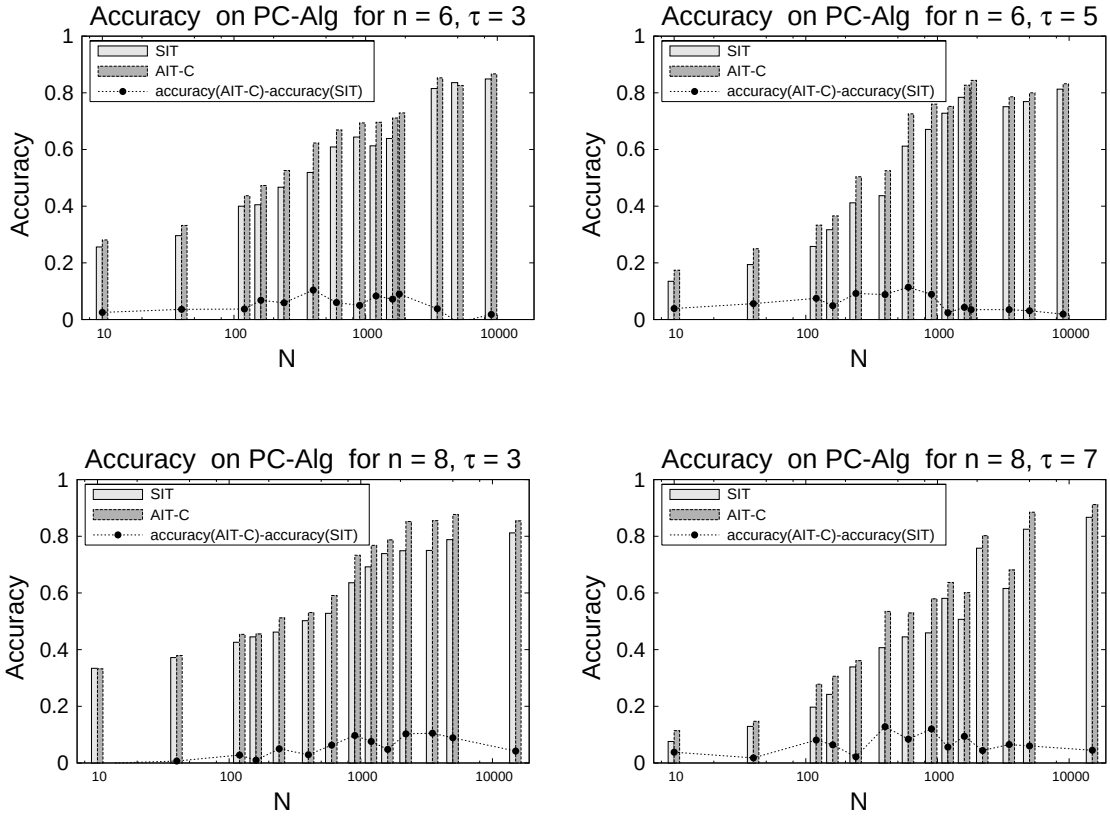


Figure 4.5 Comparison of the output of PC-Algorithm for SIT and $\mathrm{AIT_b}$-D for domain size $n = 6$, maximum degree $\tau = 3, 5$ and domain size $n = 8$, maximum degree $\tau = 3, 7$. The histogram shows the absolute value of the accuracies and the line curve shows their difference i.e., a positive value correspond to an improvement in the accuracy of $\mathrm{AIT_b}$-D over the accuracy of SIT.

of type $\mathcal{Y}$ on data, for $\mathcal{Y}$ equal to SIT, $\mathrm{AIT_b}$-G or $\mathrm{AIT_b}$-D, the estimated accuracy of test type $\mathcal{Y}$ is defined as

$$\widehat{accuracy}_{\mathcal{Y}}^{\text{data}} = \frac{1}{|\mathcal{T}|}\left|\left\{t \in \mathcal{T} \mid I_{\text{data-}\mathcal{Y}}(t) = I_{\text{true}}(t)\right\}\right|. \tag{4.31}$$

Figure 4.2 shows a comparison of the argumentative test $\mathrm{AIT_b}$-G using the general axioms with the corresponding SIT. The figure shows four plots for different values of n and τ of the mean values (over runs for ten different networks) of $\widehat{accuracy}_{\text{SIT}}^{\text{data}}$ and $\widehat{accuracy}_{\mathrm{AIT_b-G}}^{\text{data}}$ (histograms), and the difference ($\widehat{accuracy}_{\mathrm{AIT_b-G}}^{\text{data}} - \widehat{accuracy}_{\text{SIT}}^{\text{data}}$) (line graph) for different data set sizes N. A positive value of the difference corresponds to an improvement of $\mathrm{AIT_b}$-G

over SIT. We can observe modest improvements over the entire range of data set sizes in all four cases of up to 6% for $n = 6$, $\tau = 5$ and $N = 240$.

In certain situations it may be the case that the experimenter knows that the underlying distribution belongs to the class of Bayesian networks. In these situations it is appropriate to use the specific axioms of Eqs. (4.6) instead of the general axioms of Eqs. (4.5). Figure 4.3 compares the argumentative test AIT_b-D that uses the directed axioms vs. statistical tests. The plots follow the same format as Figure 4.2, with histograms plotting the mean values of $\widehat{accuracy}^{\text{data}}_{\text{SIT}}$ and $\widehat{accuracy}^{\text{data}}_{\text{AIT}_\text{b}-\text{D}}$ and the line graphs showing the difference $(\widehat{accuracy}^{\text{data}}_{\text{AIT}_\text{b}-\text{D}} - \widehat{accuracy}^{\text{data}}_{\text{SIT}})$. As in the case for the AIT using the general axioms, we can observe improvements over the entire range of data set sizes in all four cases. In this case however, the improvement is considerably larger, with sustained increases in the accuracy in the order of 5% and above, and improvements of up to 17% for $n = 8$, $\tau = 7$ and $N = 600$. We also notice in both AIT_b-G and AIT_b-D that larger improvements tend to appear in more connected domains i.e., for larger values of τ.

We also studied the effect that the improvement in the accuracy of argumentative tests has on the learning of the structure of Bayesian networks. In the following experiments we used the PC algorithm (Spirtes et al., 2000), an independence-based algorithm, to learn the structure. We compared the true structure of the underlying model to the resulting structure of the PC algorithm when it uses SITs as independence tests, denoted PC-SIT, and its output when it uses argumentative independence tests, denoted PC-AIT_b-G and PC-AIT_b-D when using general and directed axioms respectively. We evaluated the resulting networks by their ability to accurately represent the true independences in the domain, calculated by comparing the results (`true` or `false`) of a number of conditional tests conducted using d-separation, with the results on the output networks (PC-SIT, PC-AIT_b-G or PC-AIT_b-D). Denoting by $\mathcal{T}$ this set of 2,000 triplets, by $t \in \mathcal{T}$ a triplet, by $I_{\text{true}}(t)$ the result of a test performed on the underlying model, and by $I_{\text{PC-}\mathcal{Y}}(t)$ the result of performing a d-separation test on the output network PC-$\mathcal{Y}$ with $\mathcal{Y}$ equal to SIT, AIT_b-G or AIT_b-D, the estimated accuracy of network

PC-$\mathcal{Y}$ is defined as

$$\widehat{accuracy}_{\mathcal{Y}}^{\text{PC}} = \frac{1}{|\mathcal{T}|}\left|\left\{t \in \mathcal{T} \mid I_{\text{PC-}\mathcal{Y}}(t) = I_{\text{true}}(t)\right\}\right|. \tag{4.32}$$

The comparison of the accuracy of the PC algorithm using SITs vs. using argumentative tests AIT_b-G or AIT_b-D is shown in Figures 4.4 and 4.5, respectively. The figures contain four plots each for the different values of n and τ, and have the same format as in previous figures. Once again, all four plots show improvements of the argumentation approach over the entire range of data set sizes, with improvements of up to 8% for the general axioms (for $n = 6$, $\tau = 5$ and $N = 400$), and up to 17% for the causal axioms (for $n = 8$, $\tau = 7$ and $N = 400$ and 900).

4.7.1.2 Real-world Data Experiments

While the sampled data set studies of the previous section have the advantage of a more controlled and systematic study of the performance of the algorithms, experiments on real-world data are necessary for a more realistic assessment.

We conducted experiments on a number of real-world data sets obtained from the UCI machine learning repository (D.J. Newman and Merz, 1998a) and the Knowledge Discovery Data repository (D.J. Newman and Merz, 1998b). For each data set D, we conducted experiments on subsets d of D containing an increasing number of data points N. In this way we could assess the performance of the independence tests (SITs or AITs) on varying conditions of reliability, as the reliability of a test varies (typically increases) with the amount of data available. To reduce variance, each experiment was repeated for ten data subsets d of equal size, obtained by permuting the data points of D randomly and using the first N as the subset d.

Because for real-world data sets the underlying model is unknown, we can only be sure the general axioms of Eqs. (4.5) apply. Therefore in the following experiments we only report the accuracy of AIT_b-G, the bottom-up argumentative independence test defined over the general axioms. The accuracy for this set of experiments is now defined as

$$\widehat{accuracy}_{\mathcal{Y}}^{\text{data}} = \frac{1}{|\mathcal{T}|}\left|\left\{t \in \mathcal{T} \mid I_{\text{data-}\mathcal{Y}}(t) = I_{\text{true}}(t)\right\}\right| \tag{4.33}$$

where $\mathcal{Y}$ is equal to either SIT or AIT_b-G. Unfortunately, since the underlying model is not known, it is also impossible to know the true value I_{true} of any independence t. We therefore approximate it by a statistical test on the entire data set, and limit the size N of the data set subsets that we use up to a third of the size of the entire data set. This corresponds to the hypothetical scenario that a much smaller data set is available to the researcher, allowing us to evaluate the improvement of argumentation under these more challenging situations.

As mentioned in the beginning of the section, because of the exponential nature of the bottom-up algorithm we had to limit the size of our domain. For real-world data sets we limited our bottom-up algorithm experiments to 6 variables by selecting multiple random subsets of 6 variables from each data set D, resulting in a number of projections of D of size 6. We later report results over the complete data sets using the top-down argumentation algorithm, which can be run over larger domains. As noted in the sampled data experiments of the previous section, the amount of improvement in accuracy is greater for more connected models, as measured by the maximum degree parameter τ used to create the underlying model. For this reason we investigated analogous situations for real-world data sets as well. As for the latter the underlying model is unknown, no connectivity parameter τ is available; instead we used as measure of dependence the ratio of the triplets that are dependent (obtained using a statistical independence test) in a collection of tests, and generated and evaluated a number of data set projections of various different ratios.

Table 4.1 shows the results of our comparison between the argumentative test AIT_b-G and statistical test SIT for real-world data sets. The best-performing method (SIT or AIT_b-G, the latter shown abbreviated as AIT in the table) is shown in bold. As we can verify, the argumentative test improves accuracy for most data set sizes with the exception of very small data sets i.e., 10 data points. The same numbers are plotted in Figure 4.6. The figure contains 8 plots, one per data set D, each containing 5 curves for each of the variable projections of D. The plots depict the average of the difference between the accuracy of AIT_b-G and that of SIT, where as usual a positive value denotes an improvement of AIT_b-G over SIT. The figure demonstrates a clear advantage of the argumentative approach, with all data sets reaching

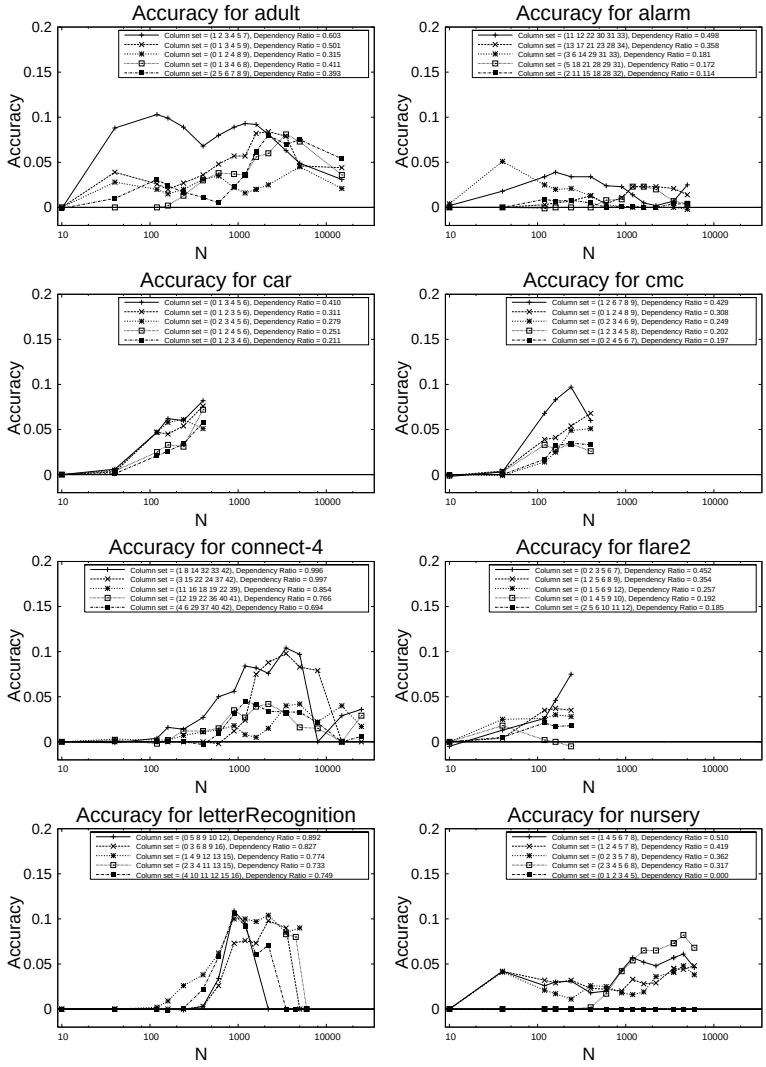


Figure 4.6 Accuracy improvements of $\mathrm{AIT_b}$ over SIT for a number of 6-column projections of several real-world data sets.

positive improvements in accuracy of up to 10% (with the exception of the alarm data set).

4.7.2 Top-down vs. Bottom-up Experiments

In Theorem 4.10 (Section 4.3) we proved that for every input argument, the bottom-up and top-down algorithms are guaranteed to produce the same output. However, the top-down algorithm is expected to be considerably more efficient. To prove this we conducted experiments that compare the running time of the argumentative test that uses the bottom-up algorithm $\mathrm{AIT_b}$ and the running time of the argumentative test that uses the top-down algorithm $\mathrm{AIT_t}$

Table 4.1 Accuracies (in percentage) of SIT and AIT_b-G (denoted AIT in the table) for several 6-variable projections of real-world data sets. For each data set projection, the table shows the ratio of dependencies in the data set and the accuracy for number of data points $N = 40, 240, 600, 1200, 3500$. The best performance between SIT and AIT_b-G is indicated in bold. Blank entries correspond to cases where the original data set was smaller than the value of N in that column.

Data set				N=40		N=240		N=600		N=1200		N=3500	
Name	total N	Dep-Ratio	Variable set	SIT	AIT	SIT	AIT	SIT	AIT	SIT	AIT	SIT	AIT
alarm	20000	0.498	(11 12 22 30 31 33)	59.4	**61.2**	73.3	**76.7**	81.1	**83.5**	85.9	**87.3**	89.2	**89.9**
		0.358	(13 17 21 23 28 34)	64.7	64.7	74.0	**74.7**	77.7	**78.1**	81.5	**83.8**	91.5	**93.6**
		0.181	(3 6 14 29 31 33)	90.9	**96.0**	96.8	**98.9**	98.4	**98.7**	98.7	98.7	98.6	98.6
		0.172	(5 18 21 28 29 31)	86.4	86.4	90.2	90.2	90.4	**91.2**	92.1	**94.4**	96.2	**96.9**
		0.114	(2 11 15 18 28 32)	88.9	88.9	93.8	**94.6**	95.0	95.0	95.0	**95.1**	95.2	**95.6**
adult	32560	0.603	(1 2 3 4 5 7)	42.8	**51.6**	54.6	**63.5**	63.1	**71.1**	69.3	**78.6**	80.4	**86.7**
		0.501	(0 1 3 4 5 9)	51.4	**55.3**	56.9	**59.6**	60.9	**65.7**	66.8	**72.5**	74.5	**82.4**
		0.315	(0 1 2 4 8 9)	71.1	**73.9**	75.0	**77.0**	77.8	**81.3**	82.2	**83.8**		
		0.411	(0 1 3 4 6 8)	58.9	58.9	59.7	**61.0**	61.7	**65.5**	64.7	**68.3**	71.4	**79.5**
		0.393	(2 5 6 7 8 9)	62.1	**63.1**	65.4	**67.1**	67.2	**67.7**	69.3	**72.9**	75.3	**82.3**
nursery	12959	0.510	(1 4 5 6 7 8)	50.8	**55.0**	58.1	**61.2**	63.8	**65.8**	68.7	**74.4**	82.7	82.7
		0.419	(1 2 4 5 7 8)	60.6	**64.8**	66.6	**69.8**	70.6	**72.8**	74.0	**77.3**	84.0	84.0
		0.362	(0 2 3 5 7 8)	66.4	**70.5**	71.0	**72.1**	73.7	**76.2**	76.3	**77.9**	85.6	85.6
		0.317	(2 3 4 5 6 8)	68.3	68.3	68.3	68.3	69.3	**71.0**	72.4	**77.8**	83.5	83.5
		0.000	(0 1 2 3 4 5)	100.0	100.0	99.9	99.9	100.0	100.0	100.0	100.0	100.0	100.0
connect-4	65534	0.996	(1 8 14 32 33 42)	**0.9**	0.8	8.0	**9.4**	13.1	**18.1**	24.6	**33.0**	52.0	**62.4**
		0.997	(3 15 22 24 37 42)	0.7	0.7	0.4	0.4	**1.2**	1.0	6.3	**8.7**	31.3	**41.1**
		0.854	(11 16 18 19 22 39)	19.5	**19.8**	23.1	**23.8**	25.6	**26.9**	32.6	**33.4**	43.1	**47.1**
		0.766	(12 19 22 36 40 41)	23.9	**24.1**	25.9	**27.1**	32.7	**34.2**	39.2	**41.9**	58.0	**61.2**
		0.694	(4 6 29 37 40 42)	31.2	31.2	31.2	31.2	33.8	**34.7**	39.6	**44.1**	55.3	**58.6**
letter-rec	19999	0.892	(0 5 8 9 10 12)	10.8	10.8	**11.5**	11.4	13.8	**17.2**	21.9	**31.4**		
		0.827	(0 3 6 8 9 16)	17.3	17.3	17.3	17.3	19.2	**21.8**	26.1	**33.7**	52.1	**61.1**
		0.774	(1 4 9 12 13 15)	22.6	22.6	23.7	**26.3**	26.9	**33.1**	32.2	**42.2**	54.1	**62.6**
		0.734	(2 3 4 11 13 15)	26.6	26.6	28.6	28.6	31.6	31.6	37.8	37.8	57.6	57.6
		0.749	(4 10 11 12 15 16)	25.1	25.1	25.0	25.0	26.8	26.8	33.3	33.3	100.0	100.0
car	1727	0.410	(0 1 3 4 5 6)	60.1	**60.7**	69.7	**75.7**						
		0.311	(0 1 2 3 5 6)	69.8	**70.1**	77.9	**83.3**						
		0.279	(0 2 3 4 5 6)	73.3	**73.8**	80.7	**86.8**						
		0.251	(0 1 2 4 5 6)	75.2	**75.6**	80.4	**83.5**						
		0.211	(0 1 2 3 4 6)	79.1	**79.2**	83.4	**86.8**						
cmc	1472	0.429	(1 2 6 7 8 9)	58.3	**58.6**	68.9	**78.6**						
		0.308	(0 1 2 4 8 9)	70.1	**70.5**	76.6	**82.0**						
		0.249	(0 2 3 4 6 9)	**75.6**	75.5	78.0	**82.9**						
		0.202	(1 2 3 4 5 8)	81.1	**81.4**	89.0	**92.4**						
		0.197	(0 2 4 5 6 7)	79.9	79.9	84.1	**87.6**						
flare2	1065	0.452	(0 2 3 5 6 7)	62.8	**64.1**	81.5	**89.0**						
		0.354	(1 2 5 6 8 9)	67.6	**68.0**	82.9	**86.4**						
		0.257	(0 1 5 6 9 12)	79.4	**81.9**	89.2	**92.0**						
		0.192	(0 1 4 5 9 10)	83.4	**85.2**	**89.1**	88.6						
		0.185	(2 5 6 10 11 12)	82.4	**82.9**	86.6	**88.4**						

when they are used to discover the structure of Bayesian and Markov networks. We consider two versions of each test, one that uses the general axioms of Eq.(4.5), denoted AIT_i-G ($i \in \{t, b\}$), and one that uses the specific axioms of Eqs.(4.7) (applicable to Markov networks) and (4.6) (applicable to Bayesian networks), denoted AIT_i-U and AIT_i-D respectively ($i \in \{t, b\}$).

For Bayesian networks, we compare the running time taken by the PC algorithm to learn the structure when it uses AIT_b-G (or AIT_b-D) vs. the time taken by the PC algorithm when it uses AIT_t-G (or AIT_t-D). Similarly, for Markov networks we compare the running time taken by the GSMN algorithm of Chapter 2 to learn the structure when it uses AIT_b-G (or AIT_b-U) vs. the time taken by GSMN when it uses AIT_t-G (or AIT_t-U).

We conducted experiments on data sampled from randomly generated Bayesian networks containing $n = 8$ variables, and connectivity parameters $\tau = 3, 5, 7$. (These data sets are the same used for the experiments of the previous section).

We also conducted experiments on data sampled from Markov networks of different number of nodes n generated randomly by selecting the first $m = \mu\frac{n}{2}$ pairs in a random permutation of all possible edges, with μ being a *connectivity* parameter (the factor $1/2$ is necessary because each edge contributes to the degree of two nodes). In the experiments we used $\mu = 1, 2, 4$. For each data set D in these six groups, we conducted experiments on subsets of D containing an increasing number of data points. This was done in order to assess the running time of AIT_b and AIT_t on the same conditions of reliability considered in the experiments that measures the accuracy.

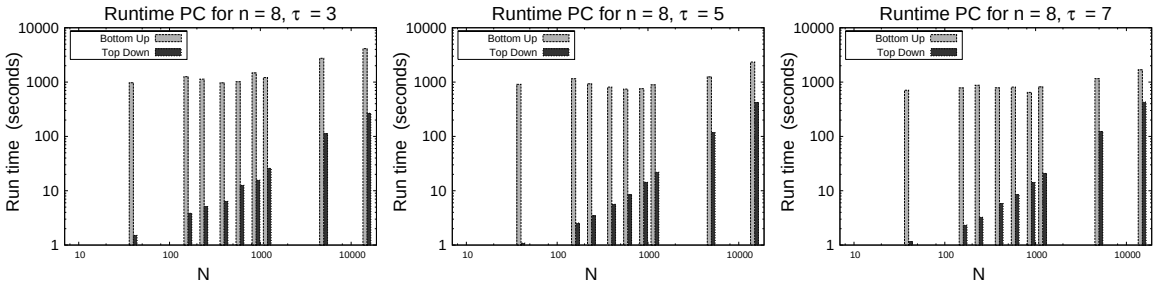


Figure 4.7 Comparison of the running times of the PC algorithm when it uses the bottom-up and top-down argumentative tests on the general axioms (AIT_b-G and AIT_t-G respectively) to learn the structure of a Bayesian network from data sampled from Bayesian models with domain size $n = 8$ and maximum degrees $\tau = 1, 3, 7$. The bars show the absolute value of the running time using a logarithmic scale.

Figures 4.7 and 4.8 shows a comparison of the argumentative tests AIT_b-G vs. AIT_t-G and AIT_b-D vs. AIT_t-D on the PC algorithm, respectively. Each figure shows three plots for

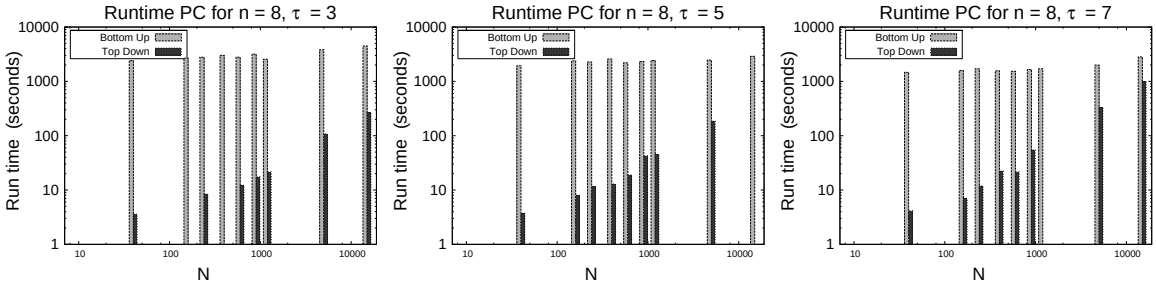


Figure 4.8 Comparison of the running times of the PC algorithm when it uses the bottom-up and top-down argumentative tests on the specific axioms ($\mathrm{AIT_b}$-D and $\mathrm{AIT_t}$-D respectively) to learn the structure of a Bayesian network from data sampled from Bayesian models with domain size $n = 8$ and maximum degrees $\tau = 1, 3, 7$. The bars show the absolute value of the running time using a logarithmic scale.

different values of τ of the mean values (over runs for ten different networks) of the running times of $\mathrm{AIT_b}$-G and $\mathrm{AIT_t}$-G shown as histogram bars in the first figure, and the running times of $\mathrm{AIT_b}$-D and $\mathrm{AIT_t}$-D shown as histogram bars in the second figure. Note that both the x and y-axes are plotted in log-scale. We can observe improvements in the running time of the top-down algorithm of several orders of magnitude over the entire range of data set sizes in all three plots of each figure. For instance, for the general axioms and $\tau = 3$ (Fig. 4.7, leftmost plot), the running time for $N = 40$ is 2 seconds for $\mathrm{AIT_t}$ against 1000 seconds for $\mathrm{AIT_b}$, while for $N = 15000$ is 500 seconds for $\mathrm{AIT_t}$ against 5000 seconds for $\mathrm{AIT_b}$.

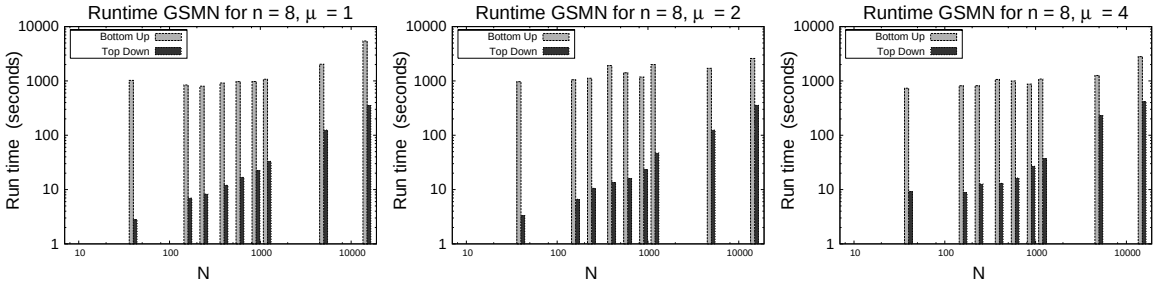


Figure 4.9 Comparison of the running times of the GSMN algorithm when it uses the bottom-up and top-down argumentative tests on the general axioms ($\mathrm{AIT_b}$-G and $\mathrm{AIT_t}$-G respectively) to learn the structure of a Markov network from data sampled from Markov models with domain size $n = 8$ and connectivities $\mu = 1, 2, 4$. The bars show the absolute value of the running time using a logarithmic scale.

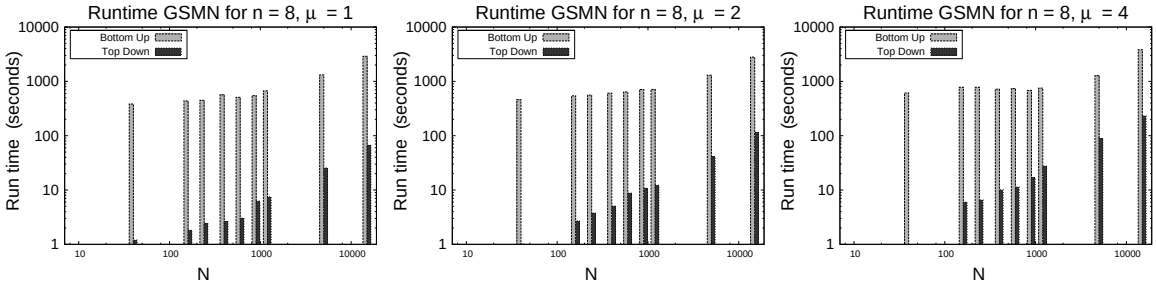


Figure 4.10 Comparison of the running times of the GSMN algorithm when it uses the bottom-up and top-down argumentative tests on the specific axioms ($\mathrm{AIT_b}$-U and $\mathrm{AIT_t}$-U respectively) to learn the structure of a Markov network from data sampled from Markov models with domain size $n = 8$ and connectivities $\mu = 1, 2, 4$. The bars show the absolute value of the running time using a logarithmic scale.

Figures 4.9 and 4.10 shows a comparison of the argumentative tests $\mathrm{AIT_b}$-G vs. $\mathrm{AIT_t}$-G and $\mathrm{AIT_b}$-U vs. $\mathrm{AIT_t}$-U using the GSMN algorithm. Each figure shows three plots for different values of μ of the mean values (over runs for ten different networks) of the running times shown as histogram bars. Again, both the x and y-axes are plotted in log-scale. We can again observe improvements in running time of the top-down algorithm of several orders of magnitude over the entire range of data set sizes in all plots. For instance, for the general axioms and $\mu = 3$ (Fig. 4.9, leftmost), the running time for $N = 40$ is approximately 4 seconds for $\mathrm{AIT_t}$ against 1000 seconds for $\mathrm{AIT_b}$, and for $N = 15000$ is 500 seconds for $\mathrm{AIT_t}$ against 8000 seconds for $\mathrm{AIT_b}$.

4.7.3 Top-down Approximate Algorithm Experiments

The goal of the experiments presented in this section is to demonstrate that the top-down approximate algorithm of Section 4.6 improves the accuracy of independence tests on small data sets while requiring a considerably smaller run time. We conducted experiments on sampled and real data sets. For sampled data, we conducted a similar sequence of experiments as the ones described in Section 4.7.1 on data sets sampled from Bayesian networks. We also conducted experiments on data sampled from Markov networks. For each case we compared the performance of the SITs against the performance of an argumentative independence test

that uses the approximate algorithm of Section 4.6 to determine acceptance or rejection of the corresponding propositional arguments. As in the bottom-up case we consider the general and specific argumentative tests. Since this time we also consider experiments on Markov networks, the corresponding tests are $\widehat{\text{AIT}}$-G for the general axioms, and $\widehat{\text{AIT}}$-D and $\widehat{\text{AIT}}$-U for the specific axioms.

4.7.3.1 Sampled Data Experiments

For the case of Bayesian networks, we considered data sampled from randomly generated Bayesian networks of sizes $n = 8$, and maximum degrees $\tau = 1, 3, 7$ (same as the data sets used in the experiments of Section 4.7.1). For each data set D in these three groups, we conducted experiments on subsets of D containing an increasing number of data points. We report the estimated accuracy calculated using Eq. 4.31, with $\mathcal{Y} =$ SIT, $\widehat{\text{AIT}}$-G, and $\widehat{\text{AIT}}$-D.

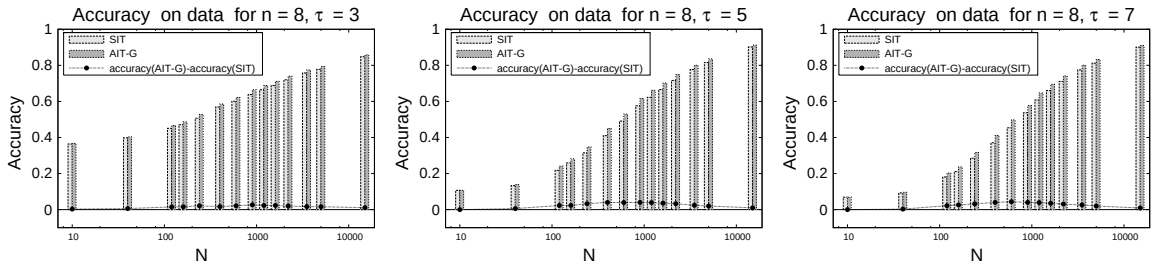


Figure 4.11 Comparison of statistical tests (SIT) vs. approximate argumentative tests on the general axioms ($\widehat{\text{AIT}}$-G) for data sets sampled from Bayesian models for domain size $n = 8$ and maximum degrees $\tau = 1, 3, 7$. The bars show the absolute value of the accuracy, and the line curves show the difference between their accuracy with a positive value corresponding to an improvement in the accuracy of $\widehat{\text{AIT}}$-G over the accuracy of SIT.

Figure 4.11 shows a comparison of the argumentative tests $\widehat{\text{AIT}}$-G using the general axioms with the corresponding SIT. The figure shows three plots for different values of τ of the mean values (over runs for ten different networks) of $\widehat{accuracy}^{\text{data}}_{\text{SIT}}$ and $\widehat{accuracy}^{\text{data}}_{\widehat{\text{AIT}}-\text{G}}$ (histograms), and the difference $(\widehat{accuracy}^{\text{data}}_{\widehat{\text{AIT}}-\text{G}} - \widehat{accuracy}^{\text{data}}_{\text{SIT}})$ (line graph) for different data set sizes N. A positive value of the difference corresponds to an improvement of $\widehat{\text{AIT}}$-G over SIT. We can

observe modest improvements over the entire range of data set sizes in all three cases of up to 5% for $n = 8$, $\tau = 5$ and $N = 600$.

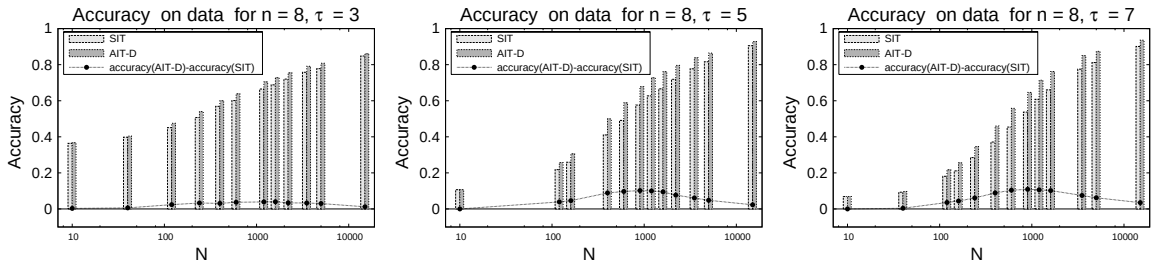


Figure 4.12 Comparison of statistical tests (SIT) vs. approximate argumentative tests on the directed axioms ($\widehat{\text{AIT}}$-D) for data sets sampled from Bayesian models for domain size $n = 8$ and maximum degrees $\tau = 1, 3, 7$. The bars show the absolute value of the accuracy, and the line curves show the difference between their accuracy with a positive value corresponding to an improvement in the accuracy of $\widehat{\text{AIT}}$-D over the accuracy of SIT.

In those situations where the experimenter knows that the underlying distribution belongs to the class of Bayesian networks it is appropriate to use the directed axioms of Eqs. (4.6) instead of the general axioms of Eqs. (4.5). Figure 4.12 compares the argumentative test $\widehat{\text{AIT}}$-D that uses the directed axioms vs. statistical tests. The plots follow the same format as Figure 4.11, with histograms plotting the mean values of $\widehat{accuracy}^{\text{data}}_{\text{SIT}}$ and $\widehat{accuracy}^{\text{data}}_{\widehat{\text{AIT}}-\text{D}}$ and the line graphs showing the difference $(\widehat{accuracy}^{\text{data}}_{\widehat{\text{AIT}}-\text{D}} - \widehat{accuracy}^{\text{data}}_{\text{SIT}})$. As in the case for the AIT using the general axioms, we can observe improvements over the entire range of data set sizes in all three cases. In this case however, the improvements are larger, reaching values of up to 11% for $n = 8$, $\tau = 7$ and $N = 900$. We also notice in both $\widehat{\text{AIT}}$-G and $\widehat{\text{AIT}}$-D that larger improvements tend to appear in more connected domains i.e., for larger values of τ.

We also studied the effect that the improvement in the accuracy of the approximate argumentative test has on the discovery of the structure of Bayesian networks. We use again the PC algorithm to learn the structure from data sampled from randomly generated Bayesian networks of sizes $n = 8, 16, 24$, and maximum degrees $\tau = 1, 3, 7$. We compared the true structure of the underlying model to the resulting structure of the PC algorithm when it uses

SITs as independence tests, denoted PC-SIT, and its output when it uses the approximate argumentative independence tests, denoted PC-$\widehat{\text{AIT}}$-G and PC-$\widehat{\text{AIT}}$-D when using the general and directed axioms respectively. We evaluated the resulting networks by their ability to accurately represent the true independences in the domain, estimated using the estimated accuracy of Eq. (4.32), with $\mathcal{Y} = \text{SIT}$, $\widehat{\text{AIT}}$-G and $\widehat{\text{AIT}}$-D.

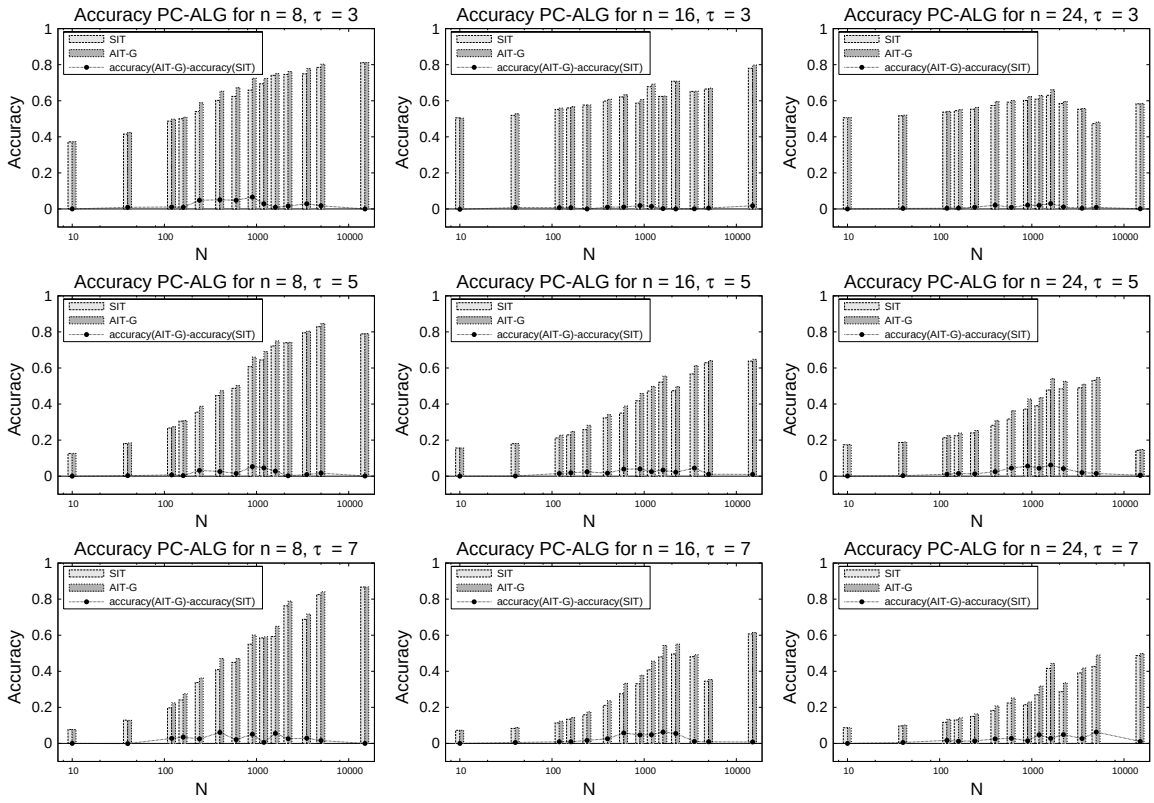


Figure 4.13 PC-Algorithm for SIT and $\widehat{\text{AIT}}$-G for domain sizes $n = 8, 16, 24$ (columns), and maximum degree $\tau = 3, 5, 7$ (rows). The bars show the absolute value of the accuracies and the line curves shows the difference between their accuracy with a positive value correspond to an improvement in the accuracy of $\widehat{\text{AIT}}$-G over the accuracy of SIT.

The comparison of the accuracy of the PC algorithm using SITs vs. using argumentative tests $\widehat{\text{AIT}}$-G and $\widehat{\text{AIT}}$-D is shown in Figures 4.13 and 4.14, respectively. The figures contain nine plots each for the different values of n and τ, and have the same format as in previous figures. Once again, all nine plots show improvements of the argumentation approach over the

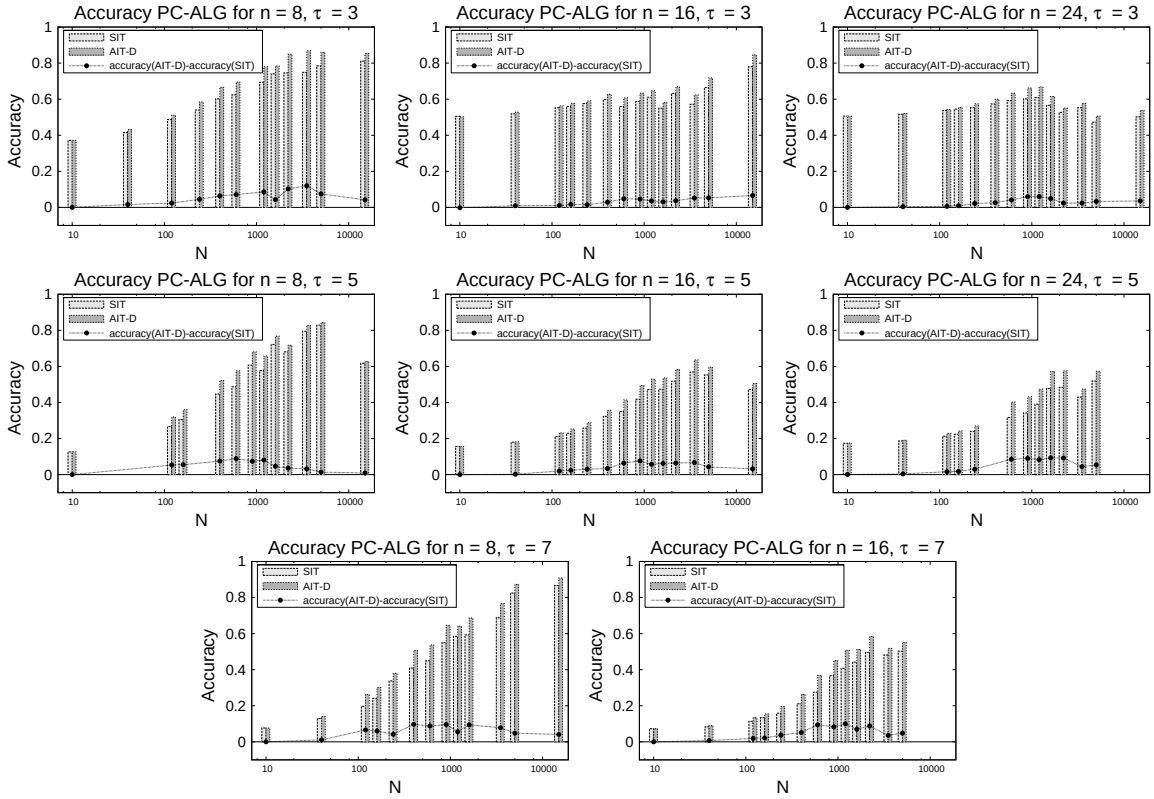


Figure 4.14 PC-Algorithm for SIT and $\widehat{\text{AIT}}$-D for domain sizes $n = 8, 16, 24$ (columns), and maximum degree $\tau = 3, 5, 7$ (rows). The bars show the absolute value of the accuracies and the line curves shows the difference between their accuracy with a positive value correspond to an improvement in the accuracy of $\widehat{\text{AIT}}$-D over the accuracy of SIT.

entire range of data set sizes, with improvements of up to 7% for the general axioms (e.g., for $n = 16$, $\tau = 5$ and $N = 600$), and up to 13% for the specific axioms (e.g., for $n = 16$, $\tau = 7$ and $N = 1200$ or $n = 24$, $\tau = 7$, and $N = 900, 1200, 1600$ and 2200). We also note that the improvements are sustained with increasing domain sizes n.

A similar sequence of experiments for assessing the performance of $\widehat{\text{AIT}}$ was performed for the case of Markov networks, with data sampled from randomly generated Markov networks of sizes $n = 8$, and connectivities $\mu = 1, 2, 4$. Once again, for each data set D in each of these three groups, we conducted experiments on subsets of D containing an increasing number of data points. We report the estimated accuracy calculated using Eq. (4.31), for each of the

three tests $\mathcal{Y} =$ SIT, $\widehat{\text{AIT}}$-G, and $\widehat{\text{AIT}}$-U.

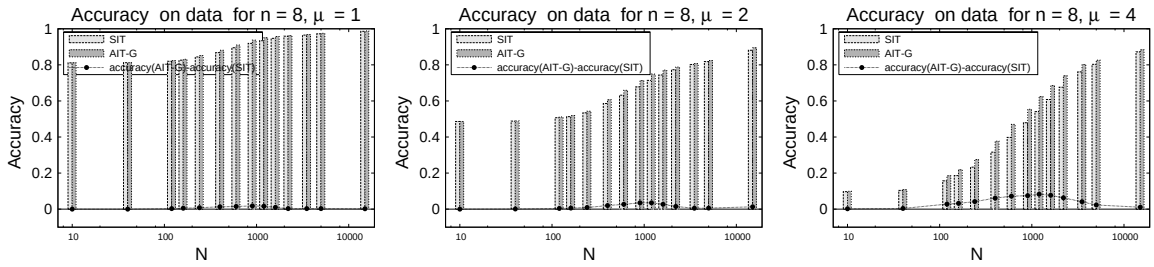


Figure 4.15 Comparison of statistical tests (SIT) vs. approximate argumentative tests on the general axioms ($\widehat{\text{AIT}}$-G) for data sets sampled from Markov networks with $n = 8$ variables and connectivities $\mu = 1, 2, 4$. The bars show the absolute value of the accuracy, and the line curves show the difference between their accuracy with a positive value corresponding to an improvement in the accuracy of $\widehat{\text{AIT}}$-G over the accuracy of SIT.

Figure 4.15 shows a comparison of the argumentative test $\widehat{\text{AIT}}$-G using the general axioms with the corresponding SIT. The figure shows three plots for different values of μ of the mean values (over runs for ten different networks) of $\widehat{accuracy}^{\text{data}}_{\text{SIT}}$ and $\widehat{accuracy}^{\text{data}}_{\widehat{\text{AIT}}-\text{G}}$ (histograms), and the difference ($\widehat{accuracy}^{\text{data}}_{\widehat{\text{AIT}}-\text{G}} - \widehat{accuracy}^{\text{data}}_{\text{SIT}}$) (line graph) for different data set sizes N. As usual, a positive value of the difference corresponds to an improvement of $\widehat{\text{AIT}}$-G over SIT. We can observe modest improvements over the entire range of data set sizes in all three cases of up to 7% (e.g., for $n = 8$, $\mu = 4$ and $N = 1200$).

When the experimenter knows that the underlying distribution belongs to the class of Markov networks it is appropriate to use the undirected axioms of Eq. (4.7) instead of the general axioms of Eq. (4.5). Figure 4.16 compares the argumentative test $\widehat{\text{AIT}}$-U that uses the undirected axioms vs. statistical tests. The plots follow the same format as Figure 4.16, with histograms plotting the mean values of $\widehat{accuracy}^{\text{data}}_{\text{SIT}}$ and $\widehat{accuracy}^{\text{data}}_{\widehat{\text{AIT}}-\text{U}}$ and the line graphs showing the difference ($\widehat{accuracy}^{\text{data}}_{\widehat{\text{AIT}}-\text{U}} - \widehat{accuracy}^{\text{data}}_{\text{SIT}}$). Once again, we can observe improvements over the entire range of data set sizes in all four cases. In this case however, the improvement is considerably larger, with increases in the accuracy of up to 20% for $n = 8$, $\mu = 7$ and $N = 1600$. As in all previous cases, we notice in both $\widehat{\text{AIT}}$-G and $\widehat{\text{AIT}}$-U that larger improvements tend to appear in more connected domains i.e., for larger values of μ.

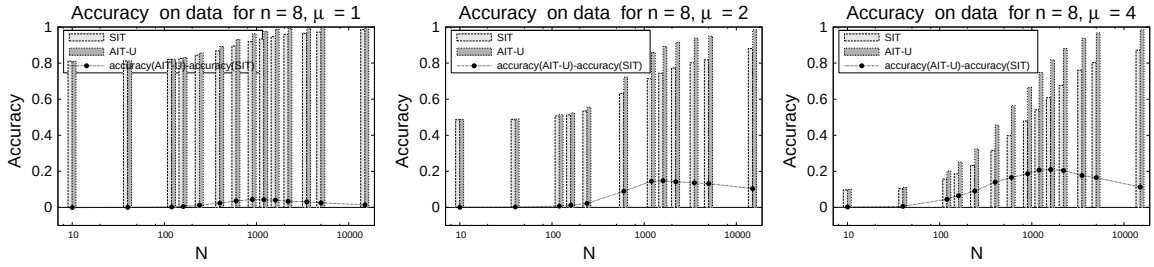


Figure 4.16 Comparison of statistical tests (SIT) vs. approximate argumentative tests on the undirected axioms ($\widehat{\text{AIT}}$-U) for data sets sampled from Markov networks with $n = 8$ variables and connectivities $\mu = 1, 2, 4$. The bars show the absolute value of the accuracy, and the line curves show the difference between their accuracy with a positive value corresponding to an improvement in the accuracy of $\widehat{\text{AIT}}$-U over the accuracy of SIT.

As in the case of Bayesian networks, we also studied the effect that the improvement in the accuracy of the approximate argumentative test has on the discovery of the structure of Markov networks. In the following experiments we used the GSMN algorithm of Chapter 2 to learn the structure from data sampled from randomly generated Markov networks of sizes $n = 8, 16, 24$, and connectivities $\mu = 1, 2, 4, 8$. We compared the true structure of the underlying model to the resulting structure of the GSMN algorithm when it uses SITs as independence tests, denoted GSMN-SIT, and its output when it uses the approximate argumentative independence tests, denoted GSMN-$\widehat{\text{AIT}}$-G and GSMN-$\widehat{\text{AIT}}$-U when using general and undirected axioms respectively. We evaluated the resulting networks using the following estimated accuracy, similar to Eq. (4.32),

$$\widehat{accuracy}_{\mathcal{Y}}^{\text{GSMN}} = \frac{1}{|\mathcal{T}|}\left|\left\{t \in \mathcal{T} \mid I_{\text{GSMN-}\mathcal{Y}}(t) = I_{\text{true}}(t)\right\}\right|. \tag{4.34}$$

with $\mathcal{Y} =$ SIT, $\widehat{\text{AIT}}$-G and $\widehat{\text{AIT}}$-U.

The comparison of the accuracy of the GSMN algorithm using SITs vs. using argumentative tests $\widehat{\text{AIT}}$-G or $\widehat{\text{AIT}}$-U is shown in Figures 4.17 and 4.18, respectively. The figures contain twelve plots each for the different values of n and μ, and have the same format as in previous figures. In these figures, plots with the same values of n are shown in the same column, and plots with the same value of μ are shown in the same row. All plots show improvements of

the argumentation approach over the entire range of data set sizes, with improvements of up to 17% for the general axioms (e.g., for $n = 16$, $\mu = 8$ and $N = 600$), and also up to 17% for the undirected axioms (e.g., for $n = 16$, $\mu = 8$ and $N = 600$). A few cases in the undirected axioms (e.g., $n = 8$, $\tau = 2$, and $N = 600$) show a minor decrease in accuracy in the order of at most 1%.

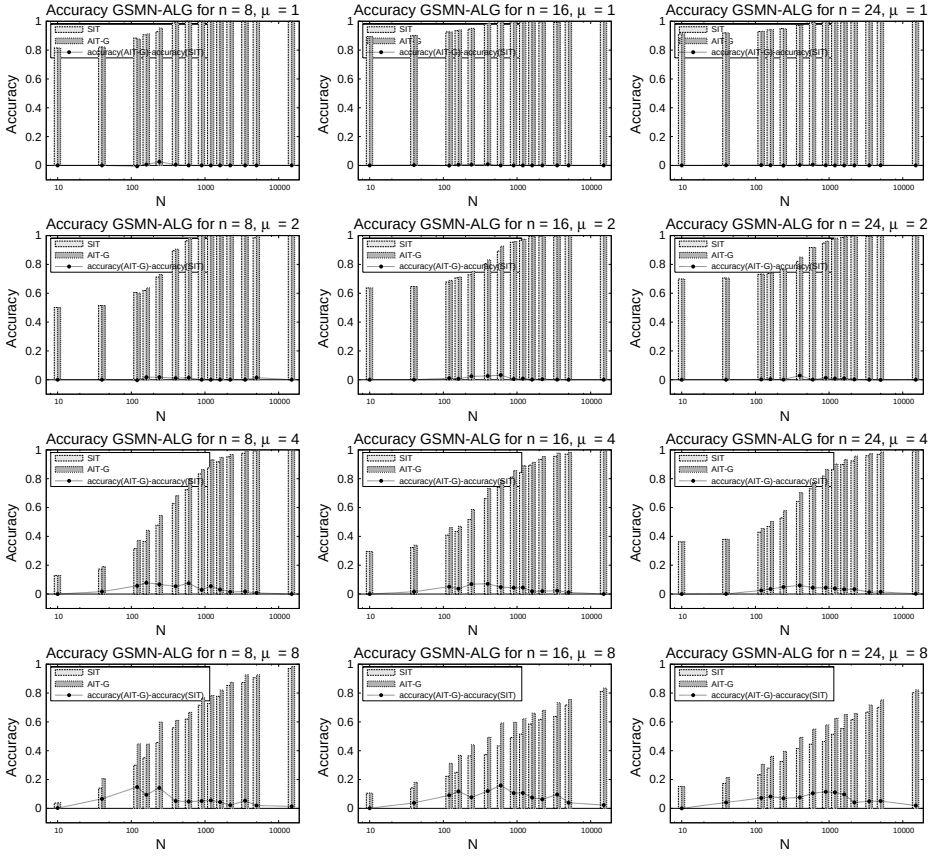


Figure 4.17 GSMN-Algorithm for SIT and $\widehat{\text{AIT}}$-G for domain sizes $n = 8, 16, 24$ (columns), and connectivities $\mu = 1, 2, 4, 8$ (rows). The bars show the absolute value of the accuracies and the line curves shows the difference between their accuracy with a positive value correspond to an improvement in the accuracy of $\widehat{\text{AIT}}$-G over the accuracy of SIT.

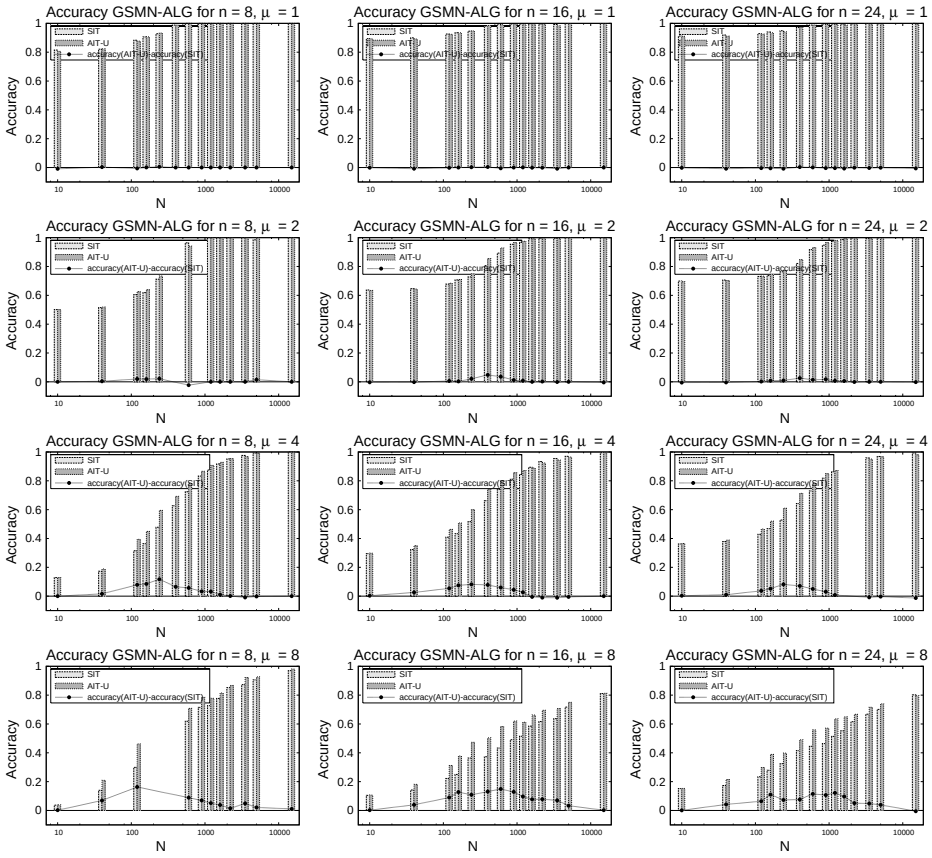


Figure 4.18 GSMN-Algorithm for SIT and $\widehat{\text{AIT}}$-U for domain sizes $n = 8, 16, 24$ (columns), and connectivities $\mu = 1, 2, 4, 8$ (rows). The bars show the absolute value of the accuracies and the line curves shows the difference between their accuracy with a positive value correspond to an improvement in the accuracy of $\widehat{\text{AIT}}$-U over the accuracy of SIT.

4.7.3.2 Real-world Data Experiments

We also tested the top-down approximate algorithm under more challenging conditions using real-world data sets obtained from the UCI machine learning repository (D.J. Newman and Merz, 1998a) and the Knowledge Discovery Data repository (D.J. Newman and Merz, 1998b). For each data set D, we conducted experiments on subsets d of D containing an increasing number of data points N. Since the reliability of a test typically increases with the amount of data available, this allowed us to assess the performance of the independence tests (statistical or argumentative) on conditions of increasing reliability. To reduce variance, each

experiment was repeated for ten subsets d of equal size, obtained by permuting the data points of D randomly and using the first N as the subset d.

For real-world data sets the underlying model is unknown. Therefore, we can only be sure the general axioms of Eq. (4.5) hold, and thus in the following experiments we only report the accuracy of $\widehat{\text{AIT}}$-G, the approximate argumentative independence test defined over the general axioms. The accuracy for this set of experiments is defined by Eq. (4.33) introduced for the real-world experiments of the bottom-up argumentative algorithm. Here instead, $\mathcal{Y}$ is equal to either SIT or $\widehat{\text{AIT}}$-G. Also, since the underlying model is not known, it is impossible to know the true value I_{true} of any independence t. We therefore approximate it as we did for the bottom-up case, namely, by conducting a statistical test on the entire data set, and limit the size N of the data set subsets that we use to a third of the size of the entire data set.

Figure 4.19 shows the results of our comparison between approximate argumentative tests $\widehat{\text{AIT}}$-G and statistical tests SIT for real-world data sets. The figure contains 6 plots, one per data set D, that depict the difference between the accuracy of $\widehat{\text{AIT}}$-G and that of SIT, where as usual a positive value denotes an improvement of $\widehat{\text{AIT}}$-G over SIT. The figure demonstrates the advantage of the argumentative approach, with all data sets reaching positive improvements in accuracy of up to 4%.

4.7.4 Approximate vs. Exact Top-down Algorithm Experiments

In this section we conduct experiments to assess the performance of the approximate top-down algorithm when compared against its exact counterpart. We conduct experiments to compare the running time of the two algorithms in order to prove that the approximation does improve efficiency. We also conduct experiments to compare the improvements in accuracy (against a statistical test) of the approximate and exact versions to demonstrate that the approximation does not produce a substantial loss in accuracy improvements.

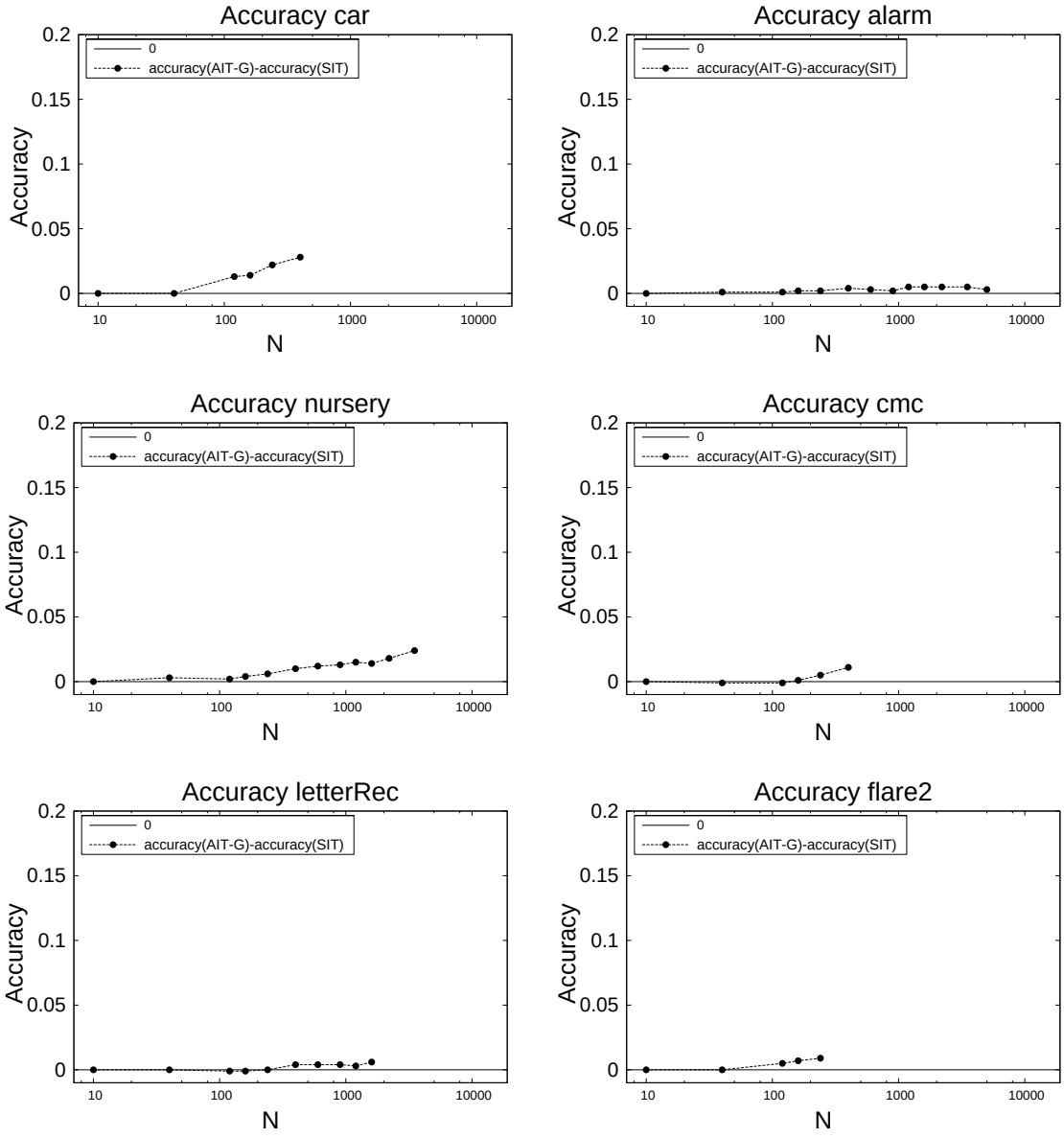


Figure 4.19 Accuracy improvements of $\widehat{\text{AIT}}$-G over SIT for a number of real-world data sets.

4.7.4.1 Running Time

The first set of experiments compare the running time of the exact top-down argumentative test AIT_t and the running time of the approximate top-down argumentative test $\widehat{\text{AIT}}$, when they are used to discover the structure of Bayesian and Markov networks. As usual, we consider three versions of each test, one that uses the general axioms (AIT_t-G and $\widehat{\text{AIT}}$-G,

respectively), one that uses the specific axioms (AIT_t-U and $\widehat{\text{AIT}}$-U, respectively), and one that uses the directed axioms (AIT_t-D and $\widehat{\text{AIT}}$-D, respectively). For Bayesian networks, we compare the running time taken by the PC algorithm to learn the structure when it uses AIT_t-G (or AIT_t-D) vs. the time taken by the PC algorithm when it uses $\widehat{\text{AIT}}$-G (or $\widehat{\text{AIT}}$-D). Similarly, for Markov networks we compare the running time taken by the GSMN algorithm to learn the structure when it uses AIT_t-G (or $\widehat{\text{AIT}}$-G) vs. the time taken by GSMN when it uses AIT_t-U (or $\widehat{\text{AIT}}$-U).

We conducted experiments on data sampled from randomly generated Bayesian and Markov networks containing $n = 8$ variables, and connectivity parameters $\tau = 3, 5, 7$, for the case of Bayesian networks, and $\mu = 1, 2, 4$, for the case of Markov networks. (These data sets are the same used for the experiments of the previous sections). For each data set D in these six groups, we conducted experiments on subsets of D containing an increasing number of data points. This was done in order to assess the running time of AIT_t and $\widehat{\text{AIT}}$ on the same conditions of reliability considered in the experiments that measures the accuracy.

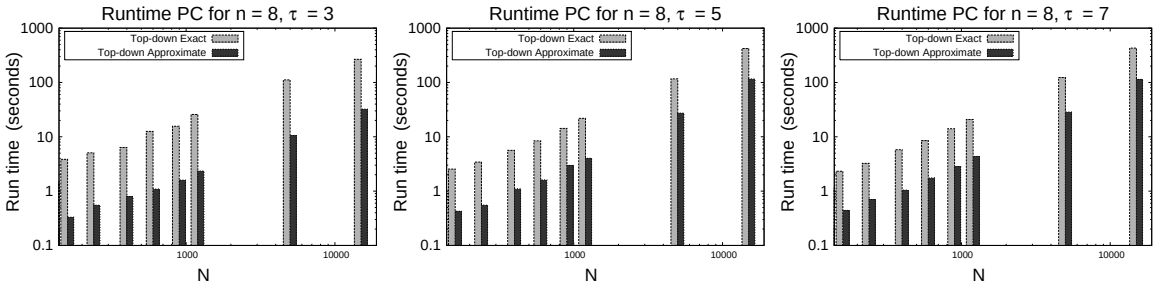


Figure 4.20 Comparison of the running times of the PC algorithm when it uses the exact and approximate top-down argumentative tests on the general axioms (AIT_t-G and $\widehat{\text{AIT}}$-G respectively) to learn the structure of a Bayesian network from data sampled from Bayesian models with domain size $n = 8$ and maximum degrees $\tau = 1, 3, 7$. The bars show the absolute value of the running time.

Figures 4.20 and 4.21 shows a comparison of the argumentative tests AIT_t-G vs. $\widehat{\text{AIT}}$-G and AIT_t-D vs. $\widehat{\text{AIT}}$-D on the PC algorithm. Each figure shows three plots for different values of τ of the mean values (over runs for ten different networks) of the running times of PC-AIT_t-G and PC-$\widehat{\text{AIT}}$-G shown as histogram bars in the first figure, and the running times of

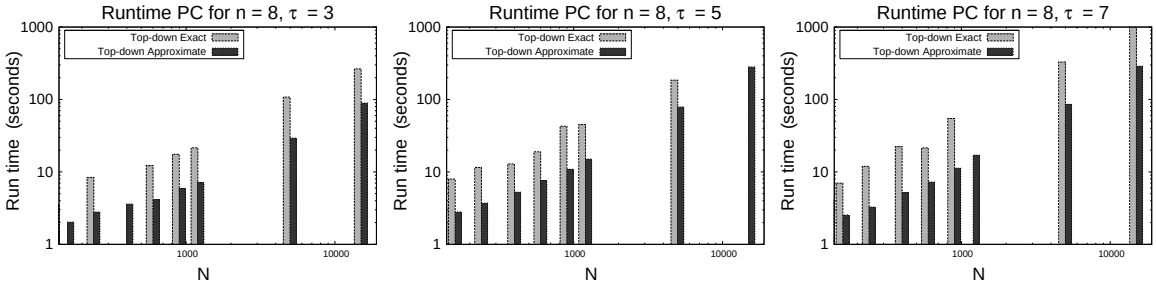


Figure 4.21 Comparison of the running times of the PC algorithm when it uses the exact and approximate top-down argumentative tests on the specific axioms ($\mathrm{AIT_t}$-D and $\widehat{\mathrm{AIT}}$-D respectively) to learn the structure of a Bayesian network from data sampled from Bayesian models with domain size $n = 8$ and maximum degrees $\tau = 1, 3, 7$. The bars show the absolute value of the running time.

PC-$\mathrm{AIT_t}$-D and PC-$\widehat{\mathrm{AIT}}$-D shown as histogram bars in the second figure. Note that both the x and y-axes are plotted in log-scale. We can observe improvements in the running time of the approximate algorithm of almost one order of magnitude in the entire range of data set sizes in all three plots of each figure. For instance, for the general axioms and $\tau = 3$ (Fig. 4.20 left), the running time for $N = 40$ is 0.2 seconds for PC-$\widehat{\mathrm{AIT}}$-G against 3 seconds for PC-$\mathrm{AIT_t}$-G, and for $N = 15000$ is 30 seconds for PC-$\widehat{\mathrm{AIT}}$-G against 250 seconds for $\mathrm{AIT_t}$-G.

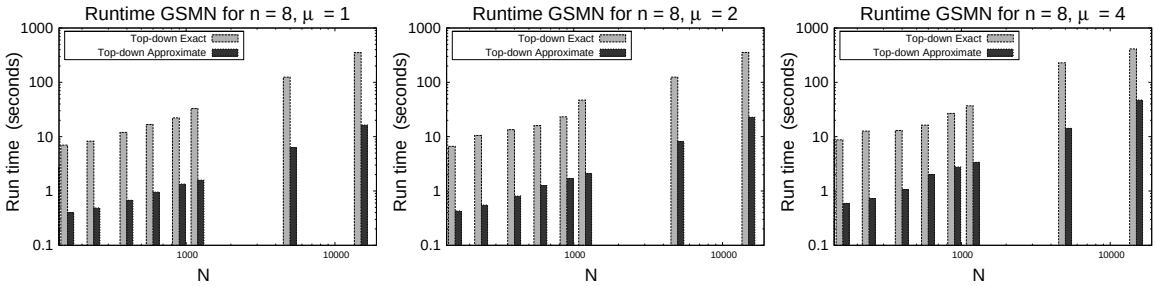


Figure 4.22 Comparison of the running times of the GSMN algorithm when it uses the exact and approximate top-down argumentative tests on the general axioms ($\mathrm{AIT_t}$-G and $\widehat{\mathrm{AIT}}$-G respectively) to learn the structure of a Markov network from data sampled from Markov models with domain size $n = 8$ and connectivities $\mu = 1, 2, 4$. The bars show the absolute value of the running time.

Figures 4.22 and 4.23 shows a comparison of the argumentative tests $\mathrm{AIT_t}$-G vs. $\widehat{\mathrm{AIT}}$-G and

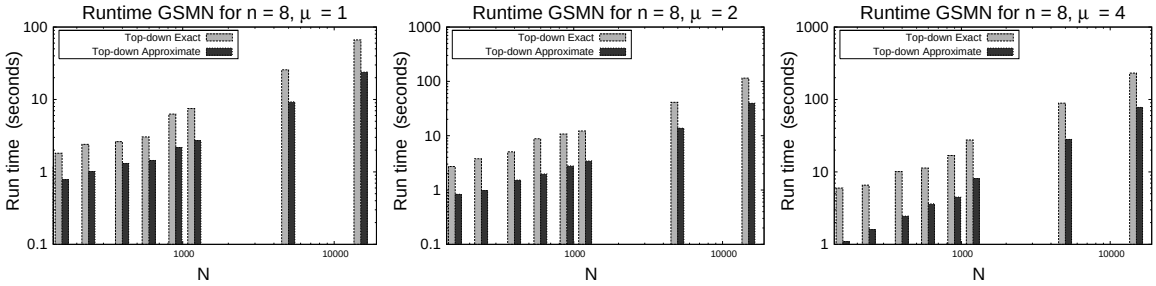


Figure 4.23 Comparison of the running times of the GSMN algorithm when it uses the exact and approximate top-down argumentative tests on the specific axioms ($\mathrm{AIT_t}$-U and $\widehat{\mathrm{AIT}}$-U respectively) to learn the structure of a Markov network from data sampled from Markov models with domain size $n = 8$ and connectivities $\mu = 1, 2, 4$. The bars show the absolute value of the running time.

$\mathrm{AIT_t}$-U vs. $\widehat{\mathrm{AIT}}$-U on the GSMN algorithm. Each figure shows three plots for different values of τ of the mean values (over runs for ten different networks) of the running times of GSMN-$\mathrm{AIT_t}$-G and GSMN-$\widehat{\mathrm{AIT}}$-G shown as histogram bars in the first figure, and the running times of GSMN-$\mathrm{AIT_t}$-U and GSMN-$\widehat{\mathrm{AIT}}$-U shown as histogram bars in the second figure. Again, both the x and y-axes are plotted in log-scale. We can again observe improvements in running time of the approximate algorithm of almost one order of magnitude in the entire range of data set sizes in all three plots of each figure. For instance, for the general axioms and $\mu = 7$ (Fig. 4.20, rightmost plot), the running time for $N = 40$ is 1.5 seconds for GSMN-$\widehat{\mathrm{AIT}}$-U against 8 seconds for GSMN-$\mathrm{AIT_t}$-U, and for $N = 15000$ is 200 seconds for GSMN-$\widehat{\mathrm{AIT}}$-U against 1000 seconds for GSMN-$\mathrm{AIT_t}$-U.

4.7.4.2 Accuracy

In this second set of experiments we compare the accuracy of the exact argumentative test and the approximate argumentative test.

For Bayesian networks, we report the accuracy of the network output by the PC algorithm computed using Eq. (4.32), with $\mathcal{Y} = \mathrm{AIT_t}$-G, $\mathrm{AIT_t}$-D, $\widehat{\mathrm{AIT}}$-G, and $\widehat{\mathrm{AIT}}$-D, whereas for Markov networks we use Eq. (4.34) for $\mathcal{Y} = \mathrm{AIT_t}$-G, $\mathrm{AIT_t}$-U, $\widehat{\mathrm{AIT}}$-G, and $\widehat{\mathrm{AIT}}$-U.

We conducted experiments on the same data sets used in the run time experiments of the

previous section, i.e., sampled from randomly generated Bayesian and Markov networks with $n = 8$ variables, and connectivity parameters $\tau = 3, 5, 7$, for the case of Bayesian networks, and $\mu = 1, 2, 4$, for the case of Markov networks. And, as usual, for each data set D in these six groups we conducted experiments on subsets of D containing an increasing number of data points.

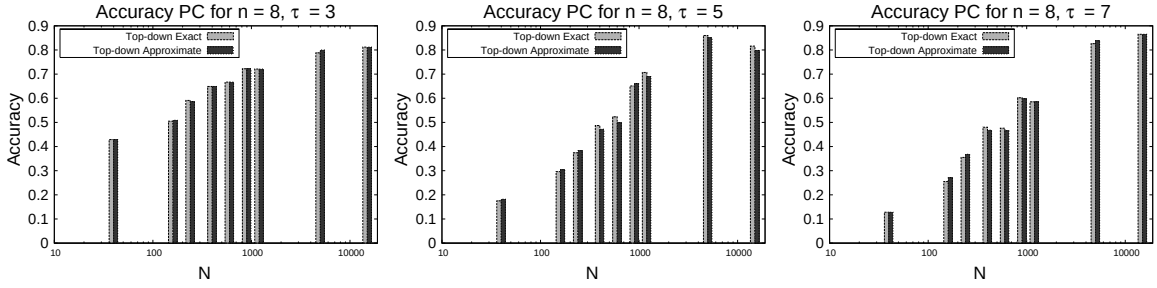


Figure 4.24 Comparison of the accuracy of the network output by the PC algorithm when it uses the exact and approximate top–down argumentative tests on the general axioms (AIT_t-G and $\widehat{\text{AIT}}$-G respectively). We show plots for data sampled from Bayesian networks with $n = 8$ variables and maximum degrees $\tau = 1, 3, 7$. The bars show the absolute value of the accuracy.

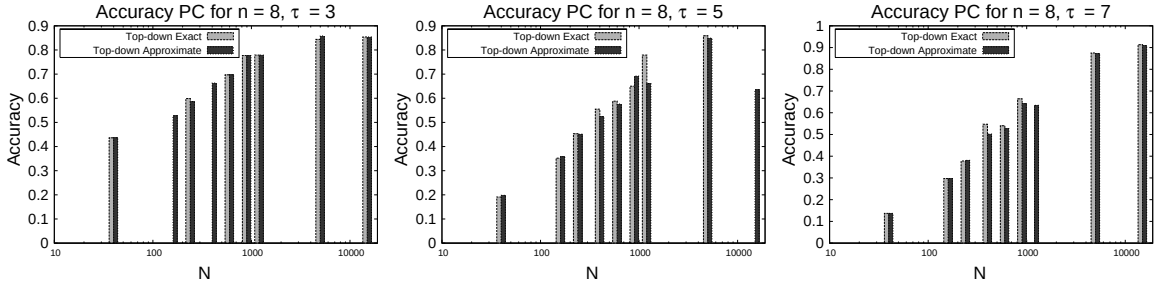


Figure 4.25 Comparison of the accuracy of the network output by the PC algorithm when it uses the exact and approximate top–down argumentative tests on the directed axioms (AIT_t-D and $\widehat{\text{AIT}}$-D respectively). We show plots for data sampled from Bayesian networks with $n = 8$ variables and maximum degrees $\tau = 1, 3, 7$. The bars show the absolute value of the accuracy.

Figures 4.24 and 4.25 show a comparison of the argumentative tests AIT_t-G vs. $\widehat{\text{AIT}}$-G and AIT_t-D vs. $\widehat{\text{AIT}}$-D on the PC algorithm. Each figure shows three plots for different values of τ of the mean values (over runs for ten different networks) of the accuracy of AIT_t-G and $\widehat{\text{AIT}}$-G shown as histogram bars in the first figure, and the accuracy of AIT_t-D and $\widehat{\text{AIT}}$-D shown as

histogram bars in the second figure. We can observe the accuracy of both algorithms matches in all but few cases. We found only a single case in which the accuracy of the approximate case is considerably smaller than its exact counterpart. This case is for the directed axioms figure, $\tau = 5$, $N = 1200$, where the accuracy of the approximate case is almost 10% lower than the accuracy of the exact algorithm.

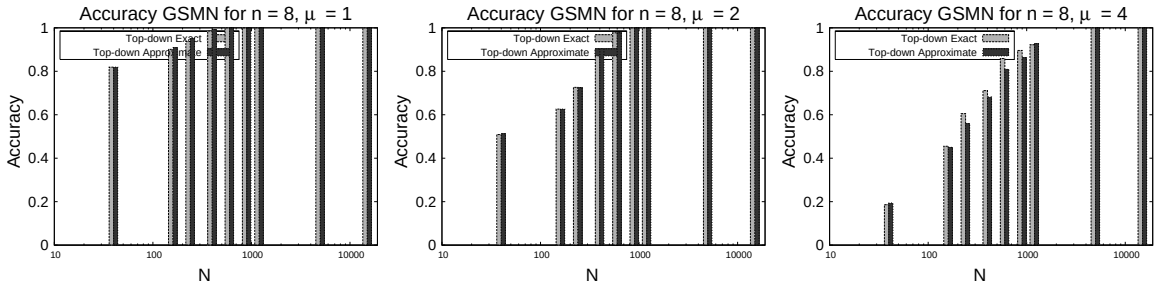


Figure 4.26 Comparison of the accuracy of the network output by the GSMN algorithm when it uses the exact and approximate top–down argumentative tests on the general axioms ($\mathrm{AIT_t}$-G and $\widehat{\mathrm{AIT}}$-G respectively). We show plots for data sampled from Markov networks with $n = 8$ variables and maximum degrees $\mu = 1, 2, 4$. The bars show the absolute value of the accuracy.

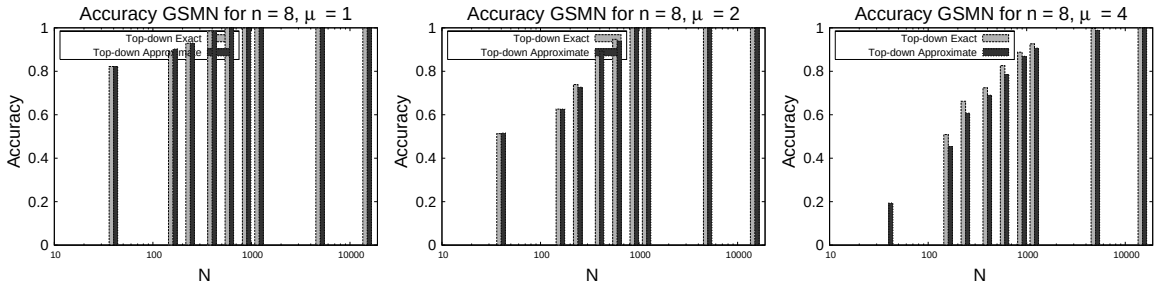


Figure 4.27 Comparison of the accuracy of the network output by the GSMN algorithm when it uses the exact and approximate top–down argumentative tests on the undirected axioms ($\mathrm{AIT_t}$-U and $\widehat{\mathrm{AIT}}$-U respectively). We show plots for data sampled from Markov networks with $n = 8$ variables and maximum degrees $\mu = 1, 2, 4$. The bars show the absolute value of the accuracy.

Figures 4.26 and 4.27 show a similar comparison of the argumentative tests $\mathrm{AIT_t}$-G vs. $\widehat{\mathrm{AIT}}$-G and $\mathrm{AIT_t}$-U vs. $\widehat{\mathrm{AIT}}$-U on the GSMN algorithm. Each figure shows three plots for different values of τ of the mean values (over runs for ten different networks) of the accuracy of $\mathrm{AIT_t}$-G and $\widehat{\mathrm{AIT}}$-G shown as histogram bars in the first figure, and the accuracy of $\mathrm{AIT_t}$-U

and $\widehat{\text{AIT}}$-U shown as histogram bars in the second figure. Again, both the x and y-axes are plotted in log-scale. We can again observe that the accuracy of both algorithms matches in all but few cases (e.g., $\mu = 4$, $N = 200$–900 of the general case, and $\mu = 4$, $N = 200$–900 of the specific case) with the largest difference being less than 4%.

4.8 Summary

We presented a framework for addressing one of the most important problems of independence-based structure discovery algorithms, namely the problem of unreliability of statistical independence tests. Our main idea was to recognize that there exist constraints in the outcome of conditional independence tests—in the form of Pearl's axiomatic characterization of the conditional independence relation—that can be exploited to correct unreliable statistical tests. We modeled this setting as a knowledge base containing conditional independences that are potentially inconsistent, and used the preference-based argumentation framework to reason with and possibly resolve these inconsistencies. We presented in detail how to apply the argumentation framework to independence knowledge bases and how to compute the preference among the independence propositions. We also presented an approximate algorithm with polynomial running time and showed experimentally its outcome is comparable to the exact case. Our experimental results on sampled and real-world data sets show improvements in the number of correct tests (as measured by accuracy on independences) for an overwhelming majority of situations considered, with maximum improvements of up to 20% in certain cases.

CHAPTER 5. Conclusions

In this thesis we addresses the problem of efficiently learning graphical probabilistic models from data using the independence-based approach. We presented, to the best of our knowledge, the first algorithms for learning the structure of Markov networks of a domain from data using the independence-based approach. We also proposed and evaluated a framework based on argumentation that addresses one of the most important problems of the independence-based approach, namely the problem of the unreliability of statistical independence tests.

As pointed out in the introduction, in the past the problem of structure learning of undirected graphical models has proved to be extremely challenging. Existing approaches require the computation of an expensive score measure to guide the search of the optimal model. The computation of this score requires the complete model, i.e., structure and parameters, which for undirected models involves a computationally expensive normalization step. The independence-based approach in general, and our algorithms in particular, do not require the computation of the model parameters during structure discovery, and thus result in more efficient algorithms.

We presented three structure learning algorithms: the GSMN and GSIMN algorithms in Chapter 2, and the PFMN algorithm in 3. To the best of the author's knowledge, GSMN is the first independence-based algorithm for learning Markov networks that has appeared in the literature. Although GSMN is interesting and useful in itself, it can be further extended and made more efficient in regards of number of tests required and accuracy; we achieve that goal in the GSIMN and PFMN algorithms. GSIMN is an extension of GSMN that avoids executing unnecessary independence tests by exploiting Pearl's axioms, a set of well-known and very general constraints that are satisfied by the conditional independence relation of a domain

whose probabilistic distribution is faithful to a Markov network. During the structure discovery process, these axioms were used as inference rules to infer the value of unseen independencies from those seen so far. The result is a very efficient algorithm that scales up polynomially in the weighted number of tests required. Moreover, both the GSMN and GSIMN algorithms are provably correct under assumptions—their proof of correctness is presented in Appendices A and B.

One disadvantage of the GSMN and GSIMN algorithms is that they order the tests to be performed in a relatively rigid way, resulting in potential inefficiencies in the computational cost that they require to learn the structure. The PFMN algorithm instead greedily selects the optimal sequence of tests according to information gain, resulting in significant efficiency improvements. Together with the PFMN algorithm, we presented in Chapter 3 an analysis of the domain of structures and independencies using an explicit generative model. The generative model proposed was also used to derived an expression for the information gain, but it is also an interesting and informative contribution in itself.

The experiments demonstrate that both the GSIMN and PFMN algorithms satisfy our main design goal of improving efficiency in terms of the weighted number of tests without adversely affecting the quality of the output network. When compared against GSMN, GSIMN showed a decrease in the weighted number of tests of up to 38% for difficult (large) artificial domains and up to 75% for real-world data, and output network accuracy comparable to that of GSMN, with some cases showing improvement. Compared against GSIMN, PFMN showed a decrease in the weighted number of tests of up to 85% for both sampled and real-world data, and output network accuracy comparable to that of GSIMN.

In addition to these algorithms, we addressed in Chapter 4 one of the main criticisms of independence-based structure discovery algorithms, namely the unreliability of statistical independence tests for small data sets. The chapter introduced the argumentative independence tests (AIT), whose idea is to exploit the constraints in the outcome of conditional independence tests (formalized by Pearl's axioms) to correct unreliable statistical tests. We modeled the set of the outcomes of all possible tests in a domain as a knowledge base containing potentially

inconsistent conditional independencies, and used the sound theoretical framework argumentation framework to reason with and possibly resolve some of these inconsistencies. Under this framework, we presented an exact and also a more practical approximate top-down algorithm that runs in polynomial time and presents improvements in the accuracy with respect to the accuracy achieved by statistical tests of up to 20%. Experimental results show that the approximate algorithm's result are similar to the outcomes of its exact counterpart.

In summary, our contributions include a set of practical algorithms that efficiently and accurately discover the conditional independence structure of Markov networks from data.

APPENDIX A. Correctness of GSMN

For the proof of correctness we make the following assumptions.

- The axioms of Eqs. (1.7) hold.
- The probability distribution of the domain is strictly positive.
- Tests are conducted by querying an oracle, which returns the true value of each test in the underlying model.

For each variable $X \in \mathbf{V}$ visited during the main loop of the GSMN algorithm (lines 8–24), the set $\mathbf{B}^X$ of variable $X \in \mathbf{V}$ is constructed by growing and shrinking a set $\mathbf{S}$, starting from the empty set. We denote by $\mathbf{MB}(X)$ the true Markov blanket of X, and prove here that the set $\mathbf{B}^X$ (returned by GSMN) is in fact equal to $\mathbf{MB}(X)$.

The algorithm examines every variable $Y \in \lambda^X$ for inclusion to $\mathbf{S}$ (and thus to $\mathbf{B}^X$) during the grow phase (lines 15 to 21) and, if Y was added to $\mathbf{S}$ during the grow phase, it considers it for removal during the shrinking phase (line 22). Each variable Y that has been visited already (and thus its blanket $\mathbf{B}^Y$ has been computed) is excluded from the grow and shrink phases of X by removing it from queue λ^X (line 14). At the end of the iteration for X, a subset of these variables (the set $\mathbf{T}$) is added to set $\mathbf{S}$ to form the blanket of X returned (line 24).

To guide the reader, in Figure A.1 we provide a graph showing the dependence relationship of observations, lemmas and theorems appearing in this proof. It also allows one to verify that there are no cycles in the entire proof of correctness. The general idea behind the proof is to show that a variable Y appears in the final blanket of X if and only if $(X \not\perp\!\!\!\perp Y \mid \mathbf{V} - \{X, Y\})$. The proof proceeds by extending the conditioning set of tests executed between X and Y (the

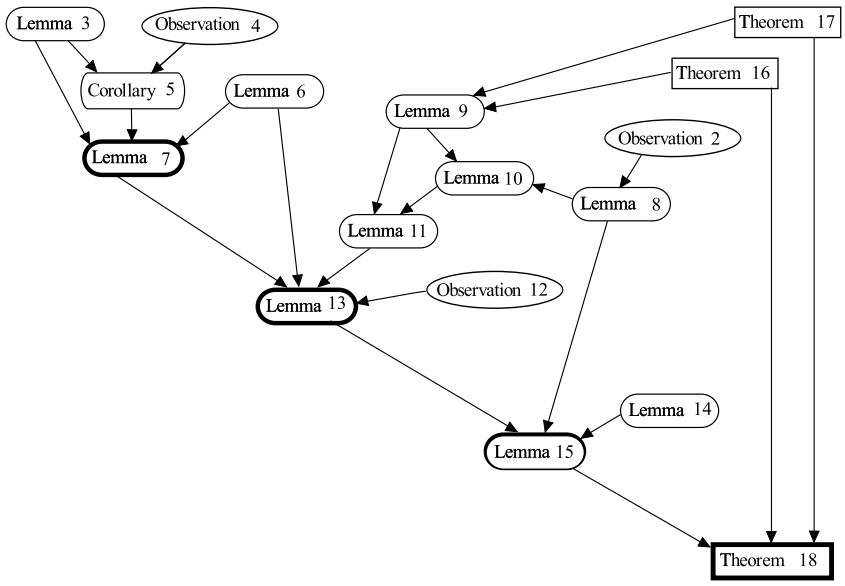


Figure A.1 Proof graph for Appendix A.

grow and shrink tests) to the all remaining variables $\mathbf{V} - \{X, Y\}$. This is done in Lemmas A.6 and A.12.

We now introduce notation and definitions and prove preliminary lemmas. We call the test executed between X and Y during the grow phase the *grow test* of Y on X (line 20). The test executed between X and Y during the shrinking phase is called the *shrink test* of Y on X (line 23). If line 21 is reached, that is, if $p_{XY} < 1 - \alpha$ (indicating X and Y are unconditionally dependent) and the grow test of Y is `false`, we say that Y is *grown* into $\mathbf{B}^X$. Otherwise, if $p_{XY} \geq 1 - \alpha$ (indicating X and Y are unconditionally independent) or the grow test is `true`, we say that Y is *not grown* into $\mathbf{B}^X$. If the shrink test of Y is `true`, indicating independence (`false`, indicating dependence), we say that Y was *shrunk* (*not shrunk*) from $\mathbf{B}^X$. This assumes that X precedes Y in the index permutation π (the "variable visit" order); the roles of X and Y are exchanged if Y precedes X in π.

Note that if X precedes Y in π during the execution of the loop for X, then $Y \notin \mathbf{T}$ and $Y \notin \mathbf{F}$. In particular, we can make the following observation.

Observation A.1. *The grow and (if necessary) the shrink test of Y on X are the only ones*

done between X and and Y during the execution of the algorithm.

We denote by $\mathbf{S}_G$ the set of variables found dependent of X during the grow phase, i.e., the value of $\mathbf{S}$ at the end of the grow phase (line 22), and by $\mathbf{S}_S$ the set of variables found dependent of X during the shrink phase, i.e., the value of $\mathbf{S}$ at the end of the shrink phase (line 24). We also denote by $\mathbf{G}$ the set of variables found independent of X during the grow phase and by $\mathbf{U} = [U_0, \ldots, U_k]$ the sequence of variables shrunk from $\mathbf{B}^X$, i.e., found independent of X during the shrink phase. The sequence $\mathbf{U}$ is assumed ordered as follows: if $i < j$ then variable U_i was found independent from X before U_j during the shrinking phase. A prefix of the first i variables $[U_0, \ldots, U_{i-1}]$ of $\mathbf{U}$ is denoted by $\mathbf{U}_i$. For some test t performed during the algorithm, we define $k(t)$ as the integer such that $\mathbf{U}_{k(t)}$ is the prefix of $\mathbf{U}$ containing the variables that were found independent of X in this loop before t. Furthermore, we abbreviate $\mathbf{U}_{k(t)}$ by $\mathbf{U}_t$.

Lemma A.2. *Let $Y \in \mathbf{S}_S$ and $t = (X, Y \mid \mathbf{Z})$ denote the shrink test of Y. Then $\mathbf{Z} = \mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_t$.*

Proof. According to line 23 of the algorithm, $\mathbf{Z} = \mathbf{S} \cup \mathbf{T} - \{Y\}$. At the beginning of the shrink phase (line 22) $\mathbf{S} = \mathbf{S}_G$, but variables found independent afterward and until t is conducted are removed from $\mathbf{S}$ in line 23. Thus, by the time t is performed, $\mathbf{S} = \mathbf{S}_G - \mathbf{U}_t$ and the conditioning set becomes $\mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_t$. □

From the definition of $\mathbf{U}$ and the fact that in the grow phase the conditioning set increases by dependent variables only, we can immediately make the following observation:

Observation A.3. *For some variable $U_i \in \mathbf{U}$, if t denotes the shrink test performed on U_i then $\mathbf{U}_t = \mathbf{U}_{i-1}$.*

Corollary A.4. $(X \perp\!\!\!\perp U_i \mid \mathbf{S}_G \cup \mathbf{T} - \mathbf{U}_i)$.

Proof. The proof follows immediately from Lemma A.2, Observation A.3, and the fact that $\mathbf{U}_i = \mathbf{U}_{i-1} \cup \{U_i\}$. □

The following lemma shows that if a certain independence holds, the conditioning set of a dependence can be increased by one variable.

Lemma A.5. *Let* $X, Y \in \mathbf{V}$, $\mathbf{Z} \subseteq \mathbf{V} - \{X, Y\}$, *and* $\mathbf{Z}' \subseteq \mathbf{Z}$. *Then* $\forall W \in \mathbf{V}$,

$$(X \not\perp\!\!\!\perp Y \mid \mathbf{Z}) \wedge (X \perp\!\!\!\perp W \mid \mathbf{Z}' \cup \{Y\}) \implies (X \not\perp\!\!\!\perp Y \mid \mathbf{Z} \cup \{W\}).$$

Proof. We prove by contradiction, and make use of the axioms of *Intersection* (I), *Strong Union* (SU), and *Decomposition* (D). Let us assume that $(X \not\perp\!\!\!\perp Y \mid \mathbf{Z})$ and $(X \perp\!\!\!\perp W \mid \mathbf{Z}' \cup \{Y\})$ but $(X \perp\!\!\!\perp Y \mid \mathbf{Z} \cup \{W\})$. Then

$$\begin{aligned}
&(X \perp\!\!\!\perp Y \mid \mathbf{Z} \cup \{W\}) \wedge (X \perp\!\!\!\perp W \mid \mathbf{Z}' \cup \{Y\}) \\
&\overset{\text{SU}}{\implies} \quad (X \perp\!\!\!\perp Y \mid \mathbf{Z} \cup \{W\}) \wedge (X \perp\!\!\!\perp W \mid \mathbf{Z} \cup \{Y\}) \\
&\overset{\text{I}}{\implies} \quad (X \perp\!\!\!\perp \{Y, W\} \mid \mathbf{Z}) \\
&\overset{\text{D}}{\implies} \quad (X \perp\!\!\!\perp Y \mid \mathbf{Z}) \wedge (X \perp\!\!\!\perp W \mid \mathbf{Z}) \\
&\implies \quad (X \perp\!\!\!\perp Y \mid \mathbf{Z}).
\end{aligned}$$

This contradicts the assumption $(X \not\perp\!\!\!\perp Y \mid \mathbf{Z})$. □

The following lemma shows that if a variable Y was not shrunk from $\mathbf{B}^X$ i.e., $Y \in \mathbf{S}_S$, then X and Y are remain dependent given a set of variables that contains all variables in set $\mathbf{U}_t$, where t is the shrink test between X and Y.

Lemma A.6. *Let* $Y \in \mathbf{S}_S$ *and let* t *denote the shrink test of* Y, *then* $\forall i \in \{0, 1, \ldots, k(t)\}$, $(X \not\perp\!\!\!\perp Y \mid \mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_i)$.

Proof. The proof is by induction on decreasing values of i, starting at $i = k(t)$.

- **Base case** $(i = k(t))$: From Lemma A.2, $t = (X, Y \mid \mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_t)$, which is equal to $(X, Y \mid \mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_{k(t)})$ by the definition of $\mathbf{U}_t$. Since $Y \in \mathbf{S}_S$, it must be the case that this test was found dependent, i.e., $(X \not\perp\!\!\!\perp Y \mid \mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_{k(t)})$.

- **Inductive step**: Let us assume that the statement is true for $i = m, 1 \leq m \leq k(t)$:

$$(X \not\perp\!\!\!\perp Y \mid \mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_m). \tag{A.1}$$

We need to prove that this is also true for $i = m - 1$:

$$(X \not\perp\!\!\!\perp Y \mid \mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_{m-1}).$$

By corollary A.4, we have

$$(X \perp\!\!\!\perp U_m \mid \mathbf{S}_G \cup \mathbf{T} - \mathbf{U}_m)$$

and by Strong Union,

$$(X \perp\!\!\!\perp U_m \mid (\mathbf{S}_G \cup \mathbf{T} - \mathbf{U}_m) \cup \{Y\})$$

or

$$(X \perp\!\!\!\perp U_m \mid (\mathbf{S}_G \cup \mathbf{T} - \mathbf{U}_m - \{Y\}) \cup \{Y\}). \tag{A.2}$$

From Eqs. (A.1), (A.2) and Lemma A.5 we get the desired relation:

$$(X \not\perp\!\!\!\perp Y \mid (\mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_m) \cup \{U_m\}) = (X \not\perp\!\!\!\perp Y \mid \mathbf{S}_G \cup \mathbf{T} - \{Y\} - \mathbf{U}_{m-1}).$$

□

Lemma A.7. *If $Y \notin \mathbf{S}_S \cup \mathbf{T}$ then exists a set $\mathbf{A} \subseteq \mathbf{V} - \{X, Y\}$ such that $(X \perp\!\!\!\perp Y \mid \mathbf{A})$.*

Proof. If $Y \notin \mathbf{S}_S$ then: (i) $Y \notin \lambda^X$, or (ii) Y was not grown into $\mathbf{B}^X$, or (iii) Y was grown into $\mathbf{B}^X$ but later shrunk from it. Cases (ii) and (iii) imply X and Y must have been found independent in the corresponding tests, which implies the lemma consequent. Case (i) can occur if either $Y \in \mathbf{T}$ or $Y \in \mathbf{F}$. Since the former is false by assumption, it must be true that $Y \in \mathbf{F}$ i.e., Y was visited before X and $X \notin \mathbf{B}^Y$. Let us assume that there is no set $\mathbf{A} \subseteq \mathbf{V} - \{X, Y\}$ such that $(X \perp\!\!\!\perp Y \mid \mathbf{A})$. Then, while visiting Y, both the grow and shrink tests between Y and X must be dependent, and thus X must be added to $\mathbf{B}^Y$ in line 24. Since these are the only opportunities for X to be in or out of $\mathbf{B}^Y$ (Observation A.1), this is a contradiction. □

Lemma A.8. *There exists a set $\mathbf{A}$ such that $(X \perp\!\!\!\perp Y \mid \mathbf{A})$ if and only if $Y \notin \mathbf{MB}^X$.*

Proof. The proof follows from Theorems A.15 and A.16 (presented near the end of this appendix) and the Strong Union axiom. □

Lemma A.9. $\mathbf{MB}^X \subseteq \mathbf{S}_S \cup \mathbf{T}$.

Proof. We must prove that $Y \in \mathbf{MB}^X \implies Y \in \mathbf{S}_S \cup \mathbf{T}$. Let us assume that $Y \in \mathbf{MB}^X$ but $Y \notin \mathbf{S}_S \cup \mathbf{T}$. From $Y \notin \mathbf{S}_S \cup \mathbf{T}$ and Lemmas A.7 and A.8 we can conclude that $Y \notin \mathbf{MB}^X$, a contradiction. □

Lemma A.10. *For every* $Y \in \mathbf{F}$, $(X \perp\!\!\!\perp Y \mid \mathbf{S}_S \cup \mathbf{T})$.

Proof. By the definition of $\mathbf{F}$ we have that for some $\mathbf{A} \subseteq \mathbf{V} - \{X, Y\}$:

$$(X \perp\!\!\!\perp Y \mid \mathbf{A})$$

By Lemma A.8 this implies

$$X \notin \mathbf{MB}^Y.$$

By the definition of the Markov blanket of Y, this implies

$$(X \perp\!\!\!\perp Y \mid \mathbf{MB}^Y)$$

which, by Lemma A.8 again, this implies

$$Y \notin \mathbf{MB}^X.$$

By the definition of the Markov blanket again this implies

$$(X \perp\!\!\!\perp Y \mid \mathbf{MB}^X)$$

which by Lemma A.9 and the Strong Union axiom implies

$$(X \perp\!\!\!\perp Y \mid \mathbf{S}_S \cup \mathbf{T}).$$

□

Observation A.11. *By definition of* $\mathbf{S}_G$, *we have that for every test* $t = (X, Y \mid \mathbf{Z})$ *performed during the grow phase,* $\mathbf{Z} \subseteq \mathbf{S}_G \subseteq \mathbf{S}_G \cup \mathbf{T}$.

The following lemma shows that if a variable Y was not shrunk from $\mathbf{B}^X$ i.e., $Y \in \mathbf{S}_S$, then X and Y remain dependent given a set of variables that contains all variables in set $\mathbf{G}$, the set of variables found independent of X during the grow phase, and all variables in $\mathbf{F}$, the set of variables already visited that do not contain X in their blankets (as defined in line 13).

Lemma A.12. *Let $Y \in \mathbf{S}_S$, and let $\mathbf{C} = \{C_0, \ldots, C_{|\mathbf{C}|}\} = \mathbf{G} \cup \mathbf{F}$. If we denote by $\mathbf{C}_i$ the first i elements of an arbitrary ordering of set $\mathbf{C}$, then for all $i = 0, \ldots, |\mathbf{C}|$, $(X \not\perp\!\!\!\perp Y \mid \mathbf{S}_G \cup \mathbf{T} \cup \mathbf{C}_i - \{Y\})$.*

Proof. The proof is by induction on increasing values of i.

- **Base Case** ($i = 0$): Follows directly from Lemma A.6 for $i = 0$ since $\mathbf{C}_0 = \mathbf{U}_0 = \emptyset$.
- **Inductive Step**: Let us assume that the statement is true for $i = m, 0 \leq m \leq |\mathbf{C}|$:

$$(X \not\perp\!\!\!\perp Y \mid \mathbf{S}_G \cup \mathbf{T} \cup \mathbf{C}_m - \{Y\}). \tag{A.3}$$

  We need to prove that it is also true for $i = m + 1$:

$$(X \not\perp\!\!\!\perp Y \mid \mathbf{S}_G \cup \mathbf{T} \cup \mathbf{C}_{m+1} - \{Y\}). \tag{A.4}$$

  We consider two cases:

  - **Case 1** ($C_m \in \mathbf{G}$): From Observation A.11 the grow test of C_m results in the independence:

$$(X \perp\!\!\!\perp C_m \mid \mathbf{Z}), \text{ where } \mathbf{Z} \subseteq \mathbf{S}_G \cup \mathbf{T}.$$

  - **Case 2** ($C_m \in \mathbf{F}$): From Lemma A.10 and the fact that $\mathbf{S}_S \subseteq \mathbf{S}_G \implies \mathbf{S}_S \cup \mathbf{T} \subseteq \mathbf{S}_G \cup \mathbf{T}$ we have that:

$$(X \perp\!\!\!\perp C_m \mid \mathbf{Z}), \text{ where } \mathbf{Z} \subseteq \mathbf{S}_G \cup \mathbf{T}.$$

  That is, both cases yields the same independence statement. By the Strong Union axiom this can become:

$$(X \perp\!\!\!\perp C_m \mid \mathbf{Z} \cup \{Y\}) \text{ where } \mathbf{Z} \subseteq \mathbf{S}_G \cup \mathbf{T} \tag{A.5}$$

or equivalently

$$(X \perp\!\!\!\perp C_m \mid (\mathbf{Z} - \{Y\}) \cup \{Y\}) \text{ where } \mathbf{Z} \subseteq \mathbf{S}_G \cup \mathbf{T}. \tag{A.6}$$

Since $\mathbf{Z} \subseteq \mathbf{S}_G \cup \mathbf{T} \subseteq \mathbf{S}_G \cup \mathbf{T} \cup \mathbf{C}_m$, we have that $\mathbf{Z} - \{Y\} \subseteq \mathbf{S}_G \cup \mathbf{T} \cup \mathbf{C}_m$, and so from Eq. (A.3) and Lemma A.5 we get the desired relation:

$$(X \not\perp\!\!\!\perp Y \mid (\mathbf{S}_G \cup \mathbf{T} \cup \mathbf{C}_m - \{Y\}) \cup C_m) = (X \not\perp\!\!\!\perp Y \mid \mathbf{S}_G \cup \mathbf{T} \cup \mathbf{C}_{m+1} - \{Y\}).$$

□

The following lemma, combined with Lemma A.12, implies that for every variable $Y \in \mathbf{S}_S$, X and Y are dependent given the universe $\mathbf{V} - \{X, Y\}$.

Lemma A.13. $\mathbf{S}_G \cup \mathbf{T} \cup \mathbf{G} \cup \mathbf{F} - \{Y\} = \mathbf{V} - \{X, Y\}$.

Proof. In loop 4-7, the queue λ^X is populated with all elements in $\mathbf{V} - \{X\}$, and then, in line 14, all elements in $\mathbf{T}$ and $\mathbf{F}$ are removed from it. Thus, at the beginning of X's visit, $\lambda^X \cup \mathbf{F} \cup \mathbf{T} = \mathbf{V} - \{X\}$. The grow phase then partitions λ^X into variables dependent of X (set $\mathbf{S}_G$) and independent of X (set $\mathbf{G}$). □

Lemma A.14. $\forall\, X, Y \in \mathbf{V},\ Y \in \mathbf{B}^X \iff (X \not\perp\!\!\!\perp Y \mid \mathbf{V} - \{X, Y\})$.

Proof. The set $\mathbf{B}^X$ is set to $\mathbf{S}_S \cup \mathbf{T}$ in line 24, and thus it is equivalent to prove:

$$(Y \in \mathbf{S}_S) \vee (Y \in \mathbf{T}) \iff (X \not\perp\!\!\!\perp Y \mid \mathbf{V} - \{X, Y\}).$$

- (Proof of $\Longleftarrow$): We prove by contradiction. Let us assume that (A) $(X \not\perp\!\!\!\perp Y \mid \mathbf{V} - \{X, Y\})$, (B) $Y \notin \mathbf{S}_S$, and (C) $Y \notin \mathbf{T}$ are true. From (B), (C), and Lemma A.7 there must be a set $\mathbf{A}$ such that $(X \perp\!\!\!\perp Y \mid \mathbf{A})$, which in turn implies $(X \perp\!\!\!\perp Y \mid \mathbf{V} - \{X, Y\})$ by Strong Union, contradicting (A).

- (Proof of $\Longrightarrow$): We consider the cases $Y \in \mathbf{S}_S$ and $Y \in \mathbf{T}$ separately:

  - ($Y \in \mathbf{S}_S$): Here, $(X \not\perp\!\!\!\perp Y \mid \mathbf{V} - \{X, Y\})$ follows from lemma A.12 as a special case for $i = |\mathbf{C}|$, since $\mathbf{S}_G \cup \mathbf{T} \cup \mathbf{G} \cup \mathbf{F} - \{Y\} = \mathbf{V} - \{X, Y\}$ as proved in Lemma A.13.

– ($Y \in \mathbf{T}$): We use $\mathbf{T}^Z$ and $\mathbf{S}_S^Z$ to denote sets $\mathbf{T}$ and $\mathbf{S}_S$ during the visit of variable Z. By definition of $\mathbf{T}^X$ (line 12), if $Y \in \mathbf{T}^X$ then (i) X must be in $\mathbf{B}^Y$ which equals $\mathbf{S}_S^Y \cup \mathbf{T}^Y$, and (ii) X is visited after Y. From (ii) it follows that $X \notin \mathbf{T}^Y$, which together with (i) proves that $X \in \mathbf{S}_S^Y$.

The proof reduces then to proving that $X \in \mathbf{S}_S^Y$ implies $(X \not\perp\!\!\!\perp Y \mid \mathbf{V} - \{X, Y\})$, which follows from Lemma A.12 ($i = |\mathbf{C}|$) and Lemma A.13.

□

We now reproduce a theorem and a corollary from Pearl (1988) (first published in Pearl and Paz (1985)):

Theorem A.15. *(Pearl and Paz, 1985) Every dependence model M satisfying symmetry, decomposition, and intersection (Eqs. (1.7)) has a unique Markov network $G = (\mathbf{V}, \mathbf{E})$ produced by deleting from the complete graph every edge (X, Y) for which $(X \perp\!\!\!\perp Y \mid \mathbf{V} - \{X, Y\})$ holds in M, i.e.,*

$$(X, Y) \notin \mathbf{E} \iff (X \perp\!\!\!\perp Y \mid \mathbf{V} - \{X, Y\}) \text{ in } M.$$

The following results is also proved in Pearl (1988) (as a corollary of a theorem not shown here):

Theorem A.16. *(Pearl and Paz, 1985) The Markov network $G = (\mathbf{V}, \mathbf{E})$ of any strictly positive distribution can be constructed by connecting each variable X to all members of its Markov blanket $\mathbf{MB}(X)$, i.e.*

$$Y \in \mathbf{MB}(\mathbf{X}) \iff (X, Y) \in \mathbf{E}.$$

We can now prove our main theorem that shows that GSMN recovers the correct Markov blanket of each variable in the domain.

Theorem A.17. *Assuming the axioms of Eqs. 1.7 and domain distribution positivity, the GSMN algorithm recovers the correct Markov blanket $\mathbf{MB}(X)$ of each variable $X \in \mathbf{V}$, that is,*

$$\forall X \in \mathbf{V},\ \mathbf{B}^X = \mathbf{MB}(X).$$

Proof. The axioms in Eqs. 1.7 include the assumptions of Theorem A.15, and thus its consequent holds. Theorem A.16 holds under our assumption of domain distribution positivity. We thus have:

$$\begin{aligned} Y \in \mathbf{B}^X &\overset{Lemma\ A.14}{\iff} (X \not\perp\!\!\!\perp Y \mid \mathbf{V}(G) - \{X, Y\}) \\ &\overset{Thm\ A.15}{\iff} (X, Y) \in \mathbf{E}(G) \\ &\overset{Thm\ A.16}{\iff} Y \in \mathbf{MB}(X). \end{aligned}$$

□

APPENDIX B. Correctness of GSIMN

The GSIMN algorithm differs from GSMN in four places: (A) the initialization phase, lines 4–6, (B) the use of test subroutine I' instead of I, (C) the change of grow order of currently grown variable Y, lines 26–29, and (D) the change in visit order of variables not yet visited, lines 31–34.

The new initialization phase in GSIMN simply initializes the knowledge base entries K_{XY} and K_{YX}, which influence the algorithm only through subroutine I'. The subroutine I' is correct as long as each of the independence tests is correct, which is true by assumption, since it is only used as a repository of independence information gathered though such tests.

The change in grow and visit orders (cases (C) and (D) above) affects only the order of variables not yet visited. The proof of correctness of GSMN presented in the previous section is done for the loop of an arbitrary variable X and made no assumption on any particular grow order (see Lemma A.12). The only place where any ordering in the variables is considered is in Lemma A.6. However, this lemma, although it considers the order in which variables are shrunk, it does not make any assumptions about this ordering, i.e., the proof works for any given ordering. Therefore the proof of correctness of GSMN transfers directly to GSIMN.

APPENDIX C. Computability and Validity of the Argumentative Independence Test

In this appendix we prove the computability and validity of the argumentative independence test under the assumption that the preference-based argumentation framework it uses contains a strict preference relation (c.f. Definition 4.14), which is the case for independence-based PAFs (c.f. Section 4.3.3). We first prove computability in Theorem (4.20) (repeated below for convenience), followed by validity in Theorem (4.17) (also repeated below).

C.1 Computability of the Argumentative Independence Test

In this section we prove the computability of the argumentative independence test. We first introduce some notation. We denote independence propositions (e.g. $(X \perp\!\!\!\perp Y \mid \mathbf{Z})$) by σ and their negation (e.g., $(X \not\perp\!\!\!\perp Y \mid \mathbf{Z})$) by $\neg\sigma$. We abbreviate their corresponding propositional arguments $(\{\sigma\}, \sigma)$ and $(\{\neg\sigma\}, \neg\sigma)$ by a_σ and $a_{\neg\sigma}$, respectively, and we will refer to $a_{\neg\sigma}$ as the *negation* of a_σ (and vice versa). Also, we use the predicates $A(a)$, $R(a)$, $Ab(a)$ to denote the fact the argument a is accepted, rejected, or in abeyance, respectively.

For completeness we repeat here the definition of a strict preference relation.

Definition C.1. *We say preference relation π over arguments is* strict *if the ordering of arguments induced by it is strict and total, i.e., for every pair a and b of arguments,*

$$a \gg_\pi b \iff \neg\big(b \gg_\pi a\big). \tag{C.1}$$

Lemma C.2. *A strict preference relation π satisfies the condition that for every pair of arguments such that a defeats b and b defeats a, it is the case that a attacks b or b attacks a, i.e., at least one of a and b attacks the other.*

Proof. We prove by contradiction: Let us assume that a defeats b and b defeats a but neither a attacks b nor b attacks a. By definition of the attack relation (Definition 4.15),

$$\neg\big(a \; attacks \; b\big) \implies \neg\big(\neg(b \gg_\pi a)\big) \implies b \gg_\pi a$$

and

$$\neg\big(b \; attacks \; a\big) \implies \neg\big(\neg(a \gg_\pi b)\big) \implies a \gg_\pi b.$$

However, this is a contradiction since, by assumption the preference ordering is strict, and therefore it cannot be true that both conclusions $a \gg_\pi b$ and $b \gg_\pi a$ hold true at the same time. □

Lemma C.3. *A strict preference π satisfies the condition that for every pair a and b of arguments, it is not the case that both a attacks b and b attacks a, i.e., there can be no mutual attack.*

Proof. We prove by contradiction. Let us consider two mutually defeating arguments a and b. By the definition of the attack relation, and because π is a total order, we have that

$$a \; attacks \; b \implies \neg(b \gg_\pi a) \implies \big(a \gg_\pi b \vee a \equiv_\pi b\big)$$

and

$$b \; attacks \; b \implies \neg(a \gg_\pi b) \implies \big(b \gg_\pi a \vee b \equiv_\pi a\big)$$

where $a \equiv_\pi b$ means a is equally preferable to b. However, this latter case is not possible in a strict preference relation. Therefore it must be the case that $a \gg_\pi b$ and $b \gg_\pi a$, which is a contradiction Eq.(C.1), again due to strictness. □

We next prove that no argument is in abeyance if the preference relation over arguments is strict. For that, we first prove that an argument in abeyance is always attacked by at least another argument in abeyance.

Lemma 4.9. *For every argument a it holds that*

$$Ab(a) \implies \exists b \in attackers(a), \; Ab(b).$$

Proof. By definition, an argument a is in abeyance if it is neither accepted, nor rejected. Applying the definitions of acceptance and rejection and manipulating the boolean formulae we obtain,

$$\begin{aligned}
Ab(a) &\iff \neg A(a) \wedge \neg R(a) \\
&\iff \neg\big(\forall b \in attackers(a),\ R(b)\big) \wedge \neg\big(\exists b \in attackers(a),\ A(b))\big) \\
&\iff \big(\exists b \in attackers(a),\ \neg R(b)\big) \wedge \big(\forall b \in attackers(a),\ \neg A(b))\big) \\
&\iff \big(\exists b \in attackers(a),\ (A(b) \vee Ab(b))\big) \wedge \big(\forall b \in attackers(a),\ \neg A(b))\big) \\
&\iff \big(\exists b \in attackers(a),\ Ab(b)\big) \wedge \big(\forall b \in attackers(a),\ \neg A(b))\big) \\
&\implies \exists b \in attackers(a),\ Ab(b).
\end{aligned}$$

□

Definition 4.21. *An* attack sequence *is a sequence* $\langle a_1, a_2, \ldots, a_n\rangle$ *of* n *arguments such that for every* $i \in [2, n]$, a_i *attacks* a_{i-1}.

Lemma C.4. *Let* $\langle \mathcal{A}, \mathcal{R}, \pi\rangle$ *be a PAF with a strict preference relation* π. *Then, no argument can appear more than once in any attack sequence, i.e., for every attack sequence* $\langle a_1, a_2, \ldots, a_n\rangle$ *and every pair of integers* $i, j \in [1, n]$, $i \neq j$, $a_i \neq a_j$.

Proof. We first note that by definition of the attack relation, it must be the case that for any two consecutive arguments a_i, a_{i+1}, it is true that $\neg(a_i \gg_\pi a_{i+1})$. Since π is strict, this is equivalent to $a_{i+1} \gg_\pi a_i$ (c.f. Definition C.1). That is,

$$a_n \gg_\pi a_{n-1} \gg_\pi \ldots \gg_\pi a_2 \gg_\pi a_1 \tag{C.2}$$

We now assume, for contradiction, there exists an argument $a^\star$ that appears twice in the attack sequence at indexes $i^\star$ and $j^\star$, i.e.,

$$\exists\ i^\star, j^\star \in [1, n], i^\star \neq j^\star, \text{ such that } a_{i^\star} = a_{j^\star} = a^\star. \tag{C.3}$$

Since no argument defeats itself, it cannot attack itself, and thus the smallest possible attack sequence with a repeated argument must have at least length 3. From this fact and

Eq. (C.2), there must exist an argument $b \neq a^\star$ such that $a^\star \gg_\pi b$ and $b \gg_\pi a^\star$, which contradicts Eq. (C.1). □

A corollary of this lemma is the following theorem.

Theorem 4.22. *Every attack sequence $\langle a_1, a_2, \ldots, a_n \rangle$ in a PAF $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ with strict π and finite $\mathcal{A}$ is finite.*

Proof. Follows directly from Lemma (C.4) and the fact that $\mathcal{A}$ is finite. □

We now prove the main result of this section in the following theorem.

Theorem 4.20. *Given an arbitrary triplet $t = (X, Y \mid \mathbf{Z})$, and a PAF $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ with a strict preference relation π and finite arguments set $\mathcal{A}$, the top-down algorithm of Algorithm 13 run for input t over $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ terminates.*

Proof. In the tree traversed by the top-down algorithm, any path from the root to a leaf is an attack sequence. Since for strict π and finite $\mathcal{A}$ each such sequence is finite, the algorithm always terminates. □

C.2 Validity of the Argumentative Independence Test

In this section we prove the validity of the argumentative independence test.

We start we proving that under the assumption of a strict preference relation no argument is in abeyance.

Theorem 4.19. *Let $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ be a PAF with a strict preference relation π. Then no argument $a \in \mathcal{A}$ is in abeyance.*

Proof. Let us assume, for contradiction, that there is an argument a in abeyance. From Lemma (4.9), not only a has an attacker in abeyance, say argument b, but b also has an attacker in abeyance, and so on. That is, we can construct an attack sequence starting at a that contains only arguments in abeyance. Moreover, this sequence must be infinite, since the lemma assures as we always has at least one attacker in abeyance. This is in direct contradiction with Theorem 4.22. □

Corollary C.5. *For every argument a in a PAF $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ with strict π,*

$$A(a) \iff \neg R(a).$$

We now prove a number of lemmas that hold only for the sub-class of propositional arguments (arguments whose support only contains the head of that argument). We start with a lemma that demonstrates that it cannot be the case that an attacker of a propositional argument a_σ and an attacker of its negation $a_{\neg\sigma}$ do not attack each other. The former must attack the latter or vice versa.

Lemma C.6. *Let $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ be a PAF with a strict preference relation π, $a_\sigma \in \mathcal{A}$ be a propositional argument, and $a_{\neg\sigma}$ its negation. For every pair of arguments b and c that attacks a_σ and $a_{\neg\sigma}$ respectively, it follows that:*

$$(b \textit{ attacks } c) \vee (c \textit{ attacks } b).$$

Proof. Since a_σ and $a_{\neg\sigma}$ are propositional arguments, their support contains the head and only the head, and thus any defeater (i.e., rebutter or undercutter) must have as head $\neg\sigma$ and σ, respectively, i.e., the head of b must be $\neg\sigma$ and the head of c must be σ. Thus, b rebuts (and thus defeats) c and vice versa. The lemma then follows directly from Lemma (C.2). □

Lemma C.7. *Let $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ be a PAF with a strict preference relation π, and a_σ and $a_{\neg\sigma}$ be a propositional argument and its negation. Then,*

$$R(a_\sigma) \implies \neg R(a_{\neg\sigma}).$$

Proof. By assumption, $R(a_\sigma)$. We assume, for contradiction, that $R(a_{\neg\sigma})$. Therefore, by definition of rejection, $\exists b \in \mathit{attackers}(a_\sigma)$ such that $A(b)$, and $\exists c \in \mathit{attackers}(a_{\neg\sigma})$ such that $A(c)$. Also, by Lemma C.6, (i) b *attacks* c, or (ii) c *attacks* b. In either case, an accepted argument is attacking an accepted argument, which contradicts the definition of acceptance. □

Lemma C.8. *Given a PAF $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ with a strict preference relation π, every propositional argument $a_\sigma \in \mathcal{A}$ satisfies*

$$A(a_\sigma) \implies \neg A(a_{\neg\sigma})$$

Proof. We prove by contradiction. Let us assume that both a_σ and $a_{\neg\sigma}$ are accepted. Since a_σ and $a_{\neg\sigma}$ are propositional arguments, they defeat each other. Then, by Lemma C.2 either a_σ attacks $a_{\neg\sigma}$ or vice versa. In both cases an accepted argument has an accepted attacker, which is a contradiction. □

We now prove Theorem 4.18 introduced in Section 4.4.

Theorem 4.18. *Given a PAF $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ with a strict preference relation π, every propositional argument $a_\sigma \in \mathcal{A}$ and its negation $a_{\neg\sigma}$ satisfy*

$$A(a_\sigma) \iff R(a_{\neg\sigma}).$$

Proof. The ($\implies$) direction follows from Theorem 4.19 and Lemma (C.8). The ($\Longleftarrow$) direction follows from Theorem 4.19 and Lemma (C.7). □

In Section 4.4 we defined the following semantics for deciding on the dependence or independence of an input triplet $(X, Y \mid \mathbf{Z})$:

$$(\{(X \not\perp\!\!\!\perp Y \mid \mathbf{Z})\}, (X \not\perp\!\!\!\perp Y \mid \mathbf{Z})) \text{ is accepted} \iff (X \not\perp\!\!\!\perp Y \mid \mathbf{Z}) \text{ is accepted} \implies (X \not\perp\!\!\!\perp Y \mid \mathbf{Z})$$

$$(\{(X \perp\!\!\!\perp Y \mid \mathbf{Z})\}, (X \perp\!\!\!\perp Y \mid \mathbf{Z})) \text{ is accepted} \iff (X \perp\!\!\!\perp Y \mid \mathbf{Z}) \text{ is accepted} \implies (X \perp\!\!\!\perp Y \mid \mathbf{Z}) \quad \text{(C.4)}$$

where acceptance is defined over the independence-based PAF defined in Section 4.3.3.

For this argumentative test of independence to be valid, its decision must be non-ambiguous, i.e., it must decide either independence or dependence, but not both or neither. For that, exactly one of the antecedents of the above implications must be true. Formally:

Theorem 4.17. *For any input triplet $\sigma = (X, Y \mid \mathbf{Z})$, the argumentative independence test defined by Eqs. (C.4) produces a non-ambiguous decision, i.e., it decides that σ evaluates to either independence or dependence, but not both or neither.*

Proof. Let us denote $(X \perp\!\!\!\perp Y \mid \mathbf{Z})$ by $\sigma_{\mathtt{t}}$ and $(X \not\perp\!\!\!\perp Y \mid \mathbf{Z})$ by $\sigma_{\mathtt{f}}$. It suffices to prove that exactly one of the corresponding propositional arguments $a_{\sigma_{\mathtt{t}}}$ and $a_{\sigma_{\mathtt{f}}}$ is accepted in the corresponding independence-based PAF. Since any independence-based PAF is strict (c.f. Lemma 4.16 in Section 4.3.3), Theorems 4.18 and 4.19 hold. From Theorem 4.19 we know that neither of the

propositional arguments is in abeyance. Thus, since $a_{\sigma_{\mathtt{t}}}$ corresponds to the negation of $a_{\sigma_{\mathtt{f}}}$ it follows from Theorem 4.18 that exactly one of them is accepted. □

BIBLIOGRAPHY

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

Agresti, A. (2002). *Categorical Data Analysis*. Wiley, 2nd edition.

Amgoud, L. and Cayrol, C. (2002). A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34:197–215.

Andrieu, C., de Freitas, N., Doucet, A., and Jordan, M. (2003). An introduction to MCMC for machine learning. *Machine Learning*, 50:5–43.

Anguelov, D., Taskar, B., Chatalbashev, V., Koller, D., Gupta, D., Heitz, G., and Ng, A. (2005). Discriminative learning of Markov random fields for segmentation of 3D range data. *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*.

Barahona, F. (1982). On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241–3253.

Besag, J. (1974). Spacial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society, Series B*.

Besag, J., York, J., and Mollie, A. (1991). Bayesian image restoration with two applications in spatial statistics. *Annals of the Institute of Statistical Mathematics*, 43:1–59.

Bromberg, F. and Margaritis, D. (2007). Efficient and robust independence-based Markov network structure discovery. In *Proceedings of The Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*.

Bromberg, F., Margaritis, D., and Honavar, V. (2006). Efficient Markov network structure discovery using independence tests. In *SIAM International Conference on Data Mining*.

Bromberg, F., Margaritis, D., and Honavar, V. (2007). Efficient Markov network structure discovery using independence tests. *Submitted to Journal of Machine Learning Research*.

Buntine, W. L. (1994). Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225.

Butz, C. J. (2000). *The Relational Database Theory of Bayesian Networks*. PhD dissertation, University of Regina, Saskatchewan, Canada.

Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462 – 467.

Cochran, W. G. (1954). Some methods of strengthening the common χ^2 tests. *Biometrics*, 10:417–451.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. Mit Press, 2nd edition.

Dawid, A. P. (1979). Conditional independence in statistical theory. *Journal of the Royal Statistical Society*, 41:1–31.

Della Pietra, S., Della Pietra, V., and Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):390–393.

D.J. Newman, S. Hettich, C. B. and Merz, C. (1998a). UCI repository of machine learning databases. *Irvine, CA: University of California, Department of Information and Computer Science.*

D.J. Newman, S. Hettich, C. B. and Merz, C. (1998b). UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Com puter Science.

Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357.

Friedman, N., Linial, M., Nachman, I., and Pe'er, D. (2000). Using Bayesian networks to analyze expression data. *Computational Biology*, 7:601–620.

Gärdenforst, P. (1992). *Belief Revision*. Cambridge Computer Tracts. Cambridge University Press, Cambridge.

Gärdenforst, P. and Rott, H. (1995). Belief revision. In Gabbay, D. M., Hogger, C. J. and Robinson, J. A., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4. Clarendon Press, Oxford.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian relation of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741.

Heckerman, D. (1995). A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research.

Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*.

Hofmann, R. and Tresp, V. (1998). Nonlinear Markov networks for continuous variables. In *NIPS '98*, volume 10, pages 521–529.

Ide, J. and Cozman, F. G. (2002). Random generation of Bayesian networks. In *Brazilian Symposium on Artificial Intelligence*, volume Brazil, Denver, Colorado.

Isard, M. (2003). Pampas: Real-valued graphical models for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 613–620.

Isham, V. (1981). An introduction to spatial point processes and markov random fields. *Intl. Statist. Review*, 49:21–43.

Jerrum, M. and Sinclair, A. (1993). Polynomoial-time approximation algorithms for the ising model. *SIAM Journal on Computing*, 22:1087–1116.

Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA.

Kohlas, J. (2003). Probabilistic argumentation systems a new way to combine logic with probability. *Journal of Applied Logic*, 1(3-4):225–253.

Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In *International Conference on Machine Learning*, pages 284–292.

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

Lauritzen, S. L. (1982). *Lectures in contingency tables*. University of Aalborg Press, Aalborg, Denmark, 2nd edition.

Loui, R. P. (1987). Defeat among arguments: a system of defeasible inference. *Computational Intelligence*, 2:100–106.

Margaritis, D. (2005). Distribution-free learning of Bayesian network structure in continuous domains. In *National Conference on Artificial Intelligence (AAAI)*. AAAI Press.

Margaritis, D. and Thrun, S. (2000). Bayesian network induction via local neighborhoods. In Solla, S., Leen, T., and Müller, K.-R., editors, *Advances in Neural Information Processing Systems 12*, pages 505–511. MIT Press.

Martins, J. P. (1992). Belief revision. In Shapiro, S. C., editor, *Encyclopedia of Artificial Intelligence*, pages 110–116. John Wiley & Sons, New York, second edition.

McCallum, A. (2003). Efficiently inducing feeatures of conditional random fields. *Proceedings of Uncertainty in Artificial Intelligence (UAI)*.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc.

Pearl, J. (2000). *Causality*. Cambridge University Press.

Pearl, J. and Paz, A. (1985). Graphoids: A graph-based logic for reasoning about releveance relations. Technical Report 850038 (R-53-L), Cognitive Systems Laboratory, University of California.

Pearl, J. and Paz, A. (1987). Graphoids: A graph-based logic for reasoning about relevance relations. *Advances in Artificial Intelligence*, II:357–363.

Pollock, J. L. (1992). How to reason defeasibly. *Artificial Intelligence*, 57:1–42.

Prakken, H. (1997). *Logical Tools for Modelling Legal Argument. A Study of Defeasible Reasoning in Law.* Kluwer Law and Philosophy Library, Dordrecht.

Prakken, H. and Vreeswijk, G. (2002). *Logics for Defeasible Argumentation*, volume 4 of *Handbook of Philosophical Logic*. Kluwer Academic Publishers, Dordrecht, 2 edition.

Rebane, G. and Pearl, J. (1989). The recovery of causal poly-trees from statistical data. In Kanal, L. N., Levitt, T. S., and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence 3*, pages 175–182, Amsterdam. North-Holland.

Saha, S. and Sen, S. (2004). A Bayes net approach to argumentation based negotiation. In *Proc. of the AAMAS-2004 workshop on Argumentation in Multi-Agent Systems (ArgMAS).*

Shapiro, S. C. (1998). Belief revision and truth maintenance systems: An overview and a proposal. Technical Report 98-10, Dept of Computer Science and Engineering, State University of New York at Buffalo.

Shekhar, S., Zhang, P., Huang, Y., and Vatsavai, R. (2004). *Trends in Spatial Data Mining*, chapter 19, pages 357–379. AAAI Press / The MIT Press.

Spirtes, P., Glymour, C., and Scheines, R. (2000). *Causation, Prediction, and Search*. Adaptive Computation and Machine Learning Series. MIT Press, 2nd edition.

Spohn, W. (1980). Sotochastic independence, causal independence, and shieldability. *Journal of Philosophical Logic*, 9:73–99.

Srebro, N. and Karger, D. (2001). Learning Markov networks: Maximum bounded tree-width graphs. In *ACM-SIAM Symposium on Discrete Algorithms*.

Tong, S. and Koller, D. (2001). Active learning for structure in Bayesian networks. In *IJCAI*.

Vreeswijk, G. A. W. (2005). Argumentation in Bayesian belief networks. In *Argumentation in Multi-Agent Systems*, volume 3366 of *Lecture Notes in Computer Science*, pages 111–129. Springer Berlin / Heidelberg.