International Journal on Artificial Intelligence Tools

LEARNING MARKOV NETWORK STRUCTURES CONSTRAINED BY CONTEXT-SPECIFIC INDEPENDENCES

ALEJANDRO EDERA
Departamento de Sistemas de Información, Universidad Tecnológica Nacional, Rodriguez 273
Mendoza, M5502, Argentina
aedera@frm.utn.edu.ar

FEDERICO SCHLÜTER
Departamento de Sistemas de Información, Universidad Tecnológica Nacional, Rodriguez 273
Mendoza, M5502, Argentina
federico.schluter@frm.utn.edu.ar

FACUNDO BROMBERG
Departamento de Sistemas de Información, Universidad Tecnológica Nacional, Rodriguez 273
Mendoza, M5502, Argentina
fbromberg@frm.utn.edu.ar

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

This work focuses on learning the structure of Markov networks. Markov networks are parametric models for compactly representing complex probability distributions. These models are composed by: a structure and numerical weights. The structure describes independences that hold in the distribution. Depending on the goal of learning intended by the user, structure learning algorithms can be divided into: density estimation algorithms, focusing on learning structures for answering inference queries; and knowledge discovery algorithms, focusing on learning structures for describing independences qualitatively. The latter algorithms present an important limitation for describing independences as they use a single graph, a coarse grain representation of the structure. However, many practical distributions present a flexible type of independences called context-specific independences, which cannot be described by a single graph. This work presents an approach for overcoming this limitation by proposing an alternative representation of the structure that named *canonical model*; and a novel knowledge discovery algorithm called CSPC for learning canonical models by using as constraints context-specific independences present in data. On an extensive empirical evaluation, CSPC learns more accurate structures than state-of-the-art density estimation and knowledge discovery algorithms. Moreover, for answering inference queries, our approach obtains competitive results against density estimation algorithms, significantly outperforming knowledge discovery algorithms.

Keywords: Markov networks; structure learning; context-specific independences; CSI models; knowledge discovery.

1. Introduction

Markov networks are parametric models for compactly representing complex probability distributions of a wide variety of domains. These models are composed by two elements: a *structure* and a set of *numerical weights*. The structure describes a set of independences that holds in the domain; playing an important role, because it makes assumptions about the functional form of the distribution. One of the most important theoretical results in Markov networks show that given a structure whose independences hold in a distribution, then such distribution can be factorized into a set of simpler lower-dimensional functions, parameterized by the weights [1]. The structure of a Markov network can be constructed either by hand from human experts, or by learning algorithms from data sampled from the underlying distribution. In many practical domains, the former is not always enough to design an accurate structure, because human experts do not have sufficient understanding of the domain. Thus, learning algorithms are the most suitable option as they can automatically discovery knowledge unknown by human experts, that is, they can reveal intricate patterns of statistical association among variables. In practice, as insights about patterns in a domain permit us to predict observations, since the structure of a distribution is considered such an important source of knowledge discovery. For instance, the structures learned in Ref. 2 are used to study the co-occurrence between diseases; another example are the structures learned in Ref. 3, called cellular signaling networks, which encode causal influences in cell signaling. For all of these reasons, in the last years, the problem of learning a structure from data has received increasing attention in machine learning [4,5,6,7,8,9]. However, the Markov network structure learning from data is still challenging. One of the most important problems is that it requires weight learning that cannot be solved in closed-form, requirying to perform a convex optimization with inference as a subrutine. Unfortunately, inference in Markov networks is #P-complete [10].

As a result, in practice, structure learning algorithms can only learn structures that approximately captures the underlying structure of a distribution. The quality of the approximation is evaluated differently for the different *goals of learning* (Chapter 16 Ref. 8). In generative learning, we can find two goals of learning: *density estimation*, where a structure is "best" when the resulting Markov network is *accurate* for answering inference queries; and *knowledge discovery*, where a structure is "best" when this structure is *accurate* for describing qualitatively the independences that hold in the underlying distribution. In this work, we use the previous fact to categorize structure learning algorithms according to the goal of learning: *density estimation algorithms* [11,12,13,14,15,16]; and *knowledge discovery algorithms* [17,18,19].

In general, for any learning algorithm, the success of its goal of learning is strongly intertwined with the representation of the structure chosen. The reason for this is because learning algorithms search the best structure from a *space of candidate structures*, where the definition of this space depends on the particular repre-

sentation chosen. A flexible representation can describe a wider space of candidate structures. In contrast, some representations are more restrictive, describing limited spaces. However, restrictive representations are more interpretable, since it is simpler to retrieve independences from them. Moreover, according to the *bias-variance trade-off*, wider spaces of candidate structures tend to introduce variance errors in learning, in contrast to limited spaces which tend to introduce bias errors. In general, as the flexibility of a representation increases, its interpretability decreases [20]. In practice, density estimation and knowledge discovery algorithms use different representations for the structure. Commonly, density estimation algorithms use a set of features, and knowledge discovery algorithms use an undirected graph. The space defined by features is much broader than that defined by graphs. Therefore, density estimation algorithms suffer problems due to variance errors. In contrast, knowledge discovery algorithms suffer problems due to bias errors. For instance, density estimation algorithms tend to overfit the learned structures, requiring to add regularization terms in learning which are tuned by expensive cross-validations [21]. On the other hand, knowledge discovery algorithms can be inhibited to find accurate structures because the bias can exclude such structures from the space of candidates, introducing incorrect structural assumptions. In such situations, for the goal of knowledge discovery, the chosen representation can introduce incorrect assumptions, leading to obscure knowledge discovery and resulting in a contradiction with the goal of learning.

In practice, for knowledge discovery algorithms, the bias errors arises when the structure of the underlying distribution presents a type of independences known as *context-specific independences* [22,23,24,25,26,8]. These independences cannot be captured by a single graph, because they only hold for specific values of the variables of a distribution. In contrast, a set of features is sufficiently flexible to capture them. Therefore, in distributions that present such independences, density estimation algorithms can only learn structures that capture context-specific independences, but knowledge discovery algorithms cannot learn accurate structures. On the other hand, knowledge discovery algorithms learn graphs that cannot capture context-specific independences, resulting in excessively dense graphs that obscure such independences [27]. This shows a limitation of knowledge discovery algorithms which arises from the contradiction between its goal of learning and the representation chosen for encoding the structure.

The thesis of this work consists in relaxing this contradiction between the chosen representation and the goal of knowledge discovery by proposing: i) the use of an alternative representation and ii) a new knowledge discovery algorithm that uses such representation. This alternative representation is called *canonical models*, which are a particular class of *Context Specific Interaction models* (CSI models) [26]. Canonical models offer a balance between flexibility and interpretability, that is, this representation can capture context-specific independences guaranteeing high interpretability. These aspects permit us to adapt well-known ideas used by knowledge discovery algorithms for learning canonical models in a straightforward way.

In this work, we present the CSPC algorithm, a knowledge discovery algorithm that learns canonical models. We note that the proposed algorithm is unlikely to be practical in high-dimensional domains due to its time complexity. However, this algorithm has been designed for providing insight to open new addresses towards the use of alternative representations focused on the goal of knowledge discovery. In an extensive empirical experimentation, our approach is evaluated against density estimation and knowledge discovery state-of-the-art algorithms on synthetic and real-world datasets. The results show that the structures learned by CSPC are more accurate for both goals of learning. First, for the goal of knowledge discovery, the structures learned by CSPC are more accurate for describing context-specific independences than those learned by knowledge discovery algorithms. Second, for the goal of density estimation, the strutures learned by CSPC are significantly more accurate for answering inference queries than those learned by knowledge discovery algorithms; achieving competitive accuracies compared to density estimation algorithms.

The rest of this work is organized as follows: Section 2 introduces an overview of background concepts. Section 3 presents CSI and canonical models as alternative representations for capturing context-specific independences. Section 4 presents CSPC, a simple knowledge discovery algorithm that learns canonical models from data. Next, Section 5 presents our empirical evaluation. This work concludes with a discussion about several open problems and directions for future work. Finally, the Appendices review: basic assumptions about independences (Appendix A); a detailed explanation on how we constructed synthetic datasets used in the evaluation section (Appendix B); and a brief description of AD-Tree, a cache used by CSPC for speeding up its execution (Appendix C).

2. Background

This section reviews the basics, describing probabilistic independences (Section 2.1) and Markov networks by remarking its three most important components: representation (Section 2.2), inference (Section 2.3), weight learning (Section 2.4), and structure learning (Section 2.5). We take special attention in describing the concept of independences and the components of representation and structure learning, because our contribution mainly lies in them. In contrast, for the remaining components (Section 2.3 and 2.4), we offer a brief description in order to facilitate the understanding of our empirical evaluation.

Before starting, we introduce our general notation. Hereon, we use the symbol V to denote a finite set of indexes. Lowercase subscripts denote particular indexes, for instance $a, b \in V$; in contrast, uppercase subscripts denote subsets of indexes, for instance $W \subseteq V$. Let X_V be a set of random variables of a domain, where single variables are denoted by single indexes in V, for instance $X_a, X_b \in X_V$ where $a, b \in V$. We simply use X instead of X_V when V is clear from the context. We focus on the case where X takes discrete values $x \in \text{Val}(V)$, that is, the values for

any $X_a \in X$ are discrete: $\text{Val}(a) = \{x_a^0, x_a^1, \ldots\}$. For instance, for boolean-valued variables, that is $|\,\text{Val}(a)| = 2$, the symbols x_a^0 and x_a^1 denote the assignments $X_a = 0$ and $X_a = 1$, respectively. Moreover, we overload the symbol V to also denote the set of nodes of a graph G. Finally, we use $\mathcal{X} \subseteq \text{Val}(V)$ for denoting an arbitrary set of complete or *canonical assignments*. A *canonical assignment* is an assignment where all the variables in a domain take a fixed value. For instance, $x_V^i \equiv x^i \in \text{Val}(V)$.

2.1. *Conditional and context-specific independences*

Independences are patterns between random variables that define the functional form of a probability distribution, exploiting the form may drastically reduce the space complexity. Context-specific independences only hold when some conditioning variables are assigned to a particular value, called the context. In contrast, when conditioning variables are not assigned, the independence holds for all the values. In this case, we are in the presence of a conditional independence. Formally, context-specific independences are defined as follows:

Definition 2.1. Let $A, B, U, W \subseteq V$ be disjoint subsets of indexes, and let x_W be some assignment in $\text{Val}(W)$. Let $p(X)$ be a probability distribution. We say that variables X_A and X_B are *contextually independent* given X_U and the context $X_W = x_W$, denoted by $I(X_A, X_B \mid X_U, x_W)$, iff it holds in the distribution $p(X)$ that:

$$p(x_A|x_B, x_U, x_W) = p(x_A|x_U, x_W),$$

for all assignments x_A, x_B, and x_U; whenever $p(x_B, x_U, x_W) > 0$.

From the previous definition, it is worth noting two important facts. First, if $I(X_A, X_B \mid X_U, x_W)$ holds in a distribution, then $I(x_A, x_B \mid x_U, x_W)$ also holds in the distribution for any value x_A, x_B, x_U. Second, if $I(X_A, X_B \mid X_U, x_W)$ holds in a distribution for all $x_W \in \text{Val}(W)$, then we say that the variables are conditionally independent. Formally,

Definition 2.2. Let $A, B, U, W \subseteq V$ be disjoint subsets of indexes, and let $p(X)$ be a probability distribution. We say that variables X_A and X_B are *conditionally independent* given X_U and X_W, denoted by $I(X_a, X_b \mid X_U, X_W)$, iff it holds in the distribution $p(X)$ that:

$$p(x_A|x_B, x_U, x_W) = p(x_A|x_U, x_W),$$

for all assignments x_A, x_B, x_U, and x_W; whenever $p(x_B, x_U, x_W) > 0$.

Thus, a conditional independence $I(X_A, X_B \mid X_U, X_W)$ that holds in $p(X)$ can be seen as a conjunction of context-specific independences of the form $I(X_A, X_B \mid X_U, x_W)$ that holds in $p(X)$, for all $x_W \in \text{Val}(W)$ [43]. Moreover, a context-specific

independence $I(X_A, X_B \mid X_U, x_W)$ that holds in $p(X)$ can be seen as a conditional independence $I(X_A, X_B \mid X_U)$ that holds in the conditional distribution $p(X_{V \setminus W}|x_W)$[43]. As is usual, we assume that the independence relation $I(\cdot, \cdot \mid \cdot)$ follows the properties shown in Appendix A [5], and also that the probability distributions are positive[a]. Under this last assumption, the independence relation satisfies the propositions in Appendix A.5, then independence relations are *monotone*. For instance, if $I(X_a, X_b \mid X_U)$ is true, then $I(X_a, X_b \mid X_W)$ is true for all $W \supseteq U$, $W \subseteq V$.

Example 2.1. Consider a probability distribution $p(X)$ with boolean-valued variables that holds independence assertions of the form: $I(X_i, X_j \mid x_w^1)$, for all pairs $i, j \in V \setminus \{w\}$ where $V = \{a, b, c, d, w\}$. For factorizing $p(X)$ by using these independences, we use the chain rule to express the joint probability as a product of conditional distributions, as follows:

$$p(X) = p(X_w)p(X_{V \setminus \{w\}}|X_w).$$

Using Definition 2.1, we can factorize the conditional distribution $p(X_{V \setminus \{w\}} \mid X_w)$ based on the value of the conditioning variable X_w. For instance, for the value x_w^1, the conditional probability factorizes as follows:

$$p(X_{V \setminus \{w\}} \mid x_w^1) = \prod_{i:V \setminus \{w\}} p(X_i \mid x_w^1).$$

However, the previous factorization is not valid for the value x_w^0, formally:

$$p(X_{V \setminus \{w\}} \mid x_w^0) \neq \prod_{i:V \setminus \{w\}} p(X_i \mid x_w^0).$$

A set of independences is commonly called the *structure* of a distribution because these independences determine its factorization, or functional form, of a distribution. In this sense, we say that a structure is more general than another structure when the former makes less independence assumptions than the other. So, the latter is more specific than the former. Therefore, the most general structure is the one that does not make assumptions, namely, all the variables are mutually dependent. In contrast, the most specific structure is one that makes all the possible assumptions, namely, all the variables are mutually independent. When a structure assumes a true independence as a false dependence, it is called a *type I error* and a true dependence as false independence, it is called a *type II error*. In this work, we are interested in structures that only makes correct assumptions over the functional form of a distribution $p(X)$. These structures are called an *I-map* for $p(X)$. Formally, a structure is an I-map for $p(X)$ if every independence described by the structure holds in $p(X)$. In other words, a structure is an I-map if it has no type II errors.

[a] A distribution $p(X)$ is positive if $p(x) > 0$, for all $x \in \text{Val}(V)$.

2.2. *Markov networks*

A Markov network is a parametric model for representing probability distributions in a compact way. This model is defined by a structure and a set of potential functions $\{\phi_k(X_{D_k})\}_k$, where $X_{D_k} \subseteq X$ is the *scope* of the potential function. Formally, a potential function $\phi(\cdot)$ is a mapping from $\mathrm{Val}(D)$ to $\mathbb{R}^+$. A usual representation of a structure is an undirected graph G, whose nodes V represent random variables, and the edges encode conditional independences between the variables. For discrete domains, a usual representation of the potential functions is a table-based function. Markov networks represent an important class of probability distributions called *Gibbs distributions*. These distributions have the following general functional form:

Definition 2.3. A *Gibbs distribution* is a distribution parameterized by a set of potential functions as follows:

$$p(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{D_k}),$$

where Z is a global constant that guarantees the normalization of the product. In the general case, the constant Z couples all the potential functions, preventing to perform weight learning in a decomposable way.

For representing a distribution $p(X)$, a Markov network factorizes a Gibbs distribution according to the independences encoded in the graph G. A Gibbs distribution *factorizes* over the graph G, if any scope X_{D_k} corresponds to a complete subgraph (a.k.a. *clique*) of the graph G. Without loss of generality, the Gibbs distribution is often factorized by using the *maximum cliques* of the graph G. For positive distributions, one important theoretical result states the converse [1], that is, $p(X)$ can be represented as a Gibbs distribution that factorizes over G if G is an I-map of $p(X)$. As a result, it can be shown that every influence on any variable X_a can be blocked by conditioning on its adjacencies in the graph G, denoted by $\mathrm{adj}(a) \subseteq V$, a set also known as the *Markov blanket*. Formally, let $p(X)$ be a distribution over X, then for any variable $X_a \in X$ it can be shown that: $p(X_a|X_{V\setminus\{a\}}) = p(X_a|X_{\mathrm{MB}(a)})$, where $\mathrm{MB}(a) \equiv \mathrm{adj}(a)$ stands for the Markov blanket of a. Moreover, for positive distributions, it can be shown that $p(X_a, X_b|X_{V\setminus\{a,b\}}) = p(X_a, X_b|X_{\mathrm{MB}(a)})$, for any pair $a, b \in V$ (Section 3.2.1 Ref. 5).

For factorizing a distribution over a graph G, we must retrieve the independences encoded by G. This is performed by using the graph-theoretic notion of *reachability* [7]. Let A, B, U be arbitrary disjoint subsets of V, variables X_A and X_B are *conditionally independent* given X_U, if there is no path between nodes A and B when nodes U are removed from G. Thus, from a graph G, we can only retrieve/encode conditional independences. Therefore, when a distribution holds context-specific independences, encoding them by using a single graph leads to excessively dense graphs [22,28,27].

Example 2.2. Let us conclude with an illustrating example how to encode the structure for the distribution of Example 2.1. Figure 1 shows a graph constructed by using reachability. Given that any pair of variables are only independent given the value x_w^1, a graph cannot encode independences over specific values, leading to an excessively dense graph which obscures such independences.

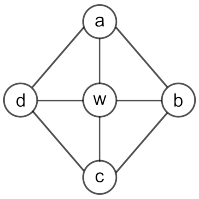


Fig. 1. Graph representation that encodes the independence assertions described in Example 2.1.

An alternative representation of a Markov network that can capture context-specific independences is the *log-linear model*. A log-linear model can be constructed from a Gibbs distribution as follows: for the ith row of the table-based potential function ϕ_k is defined an indicator function $f_k^i(\cdot)$ associated to a weight $w_{i,k} = \log \phi_k(x_{D_k}^i)$. From the Gibbs distribution shown in Eq. 2.3, a log-linear model is formally defined as:

$$p(x) = \frac{1}{Z} \exp \left\{ \sum_k \sum_i w_{i,k} f_k^i(x_{D_k}) \right\},$$

where each feature $f_k^i(\cdot)$ is commonly represented as an indicator function, that is, a conjunction of single assignment tests. For instance, given an arbitrary assignment x and a feature f_k^i, $f_k^i(x_{D_k})$ returns 1 if $x_a = x_a^i$ for all $a \in D_k$; and 0 otherwise.

In a log-linear model, the structure is represented by the set of *features*, denoted by $\mathcal{F}$. Moreover, from a set $\mathcal{F}$ of features, we can induce a graph by adding an edge between every pair of nodes whose variables appear together in some feature $f \in \mathcal{F}$ [29]. Therefore, a conditional independence $I(X_a, X_b \mid X_W)$ is encoded in $\mathcal{F}$, when for any feature $f \in \mathcal{F}$ the variables X_a and X_b do not appear in its scope, that is, either $a \notin D_k$ or $b \notin D_k$ when $W \subset D_k$.

Example 2.3. Let us now illustrate how a set of features $\mathcal{F}$ can capture the context-specific independences shown in Example 2.1, resulting in a more flexible representation than the graph shown in Example 2.2. A simple way to define the set $\mathcal{F}$ in order to capture such independences is by generating an initial set of features using all the $2^{|V|}$ complete assignments of the variables. Then, for encoding each independence of the form $I(X_i, X_j \mid x_w^1)$, the subset of features whose scopes contain the assignment x_w^1 must be modified as follows: removing all the variables from the

scopes except the variables X_i and X_w. The complete list of resulting features is the following:

(1) $(x_a^0 \wedge x_b^0 \wedge x_c^0 \wedge x_d^0 \wedge x_w^0)$
(2) $(x_a^1 \wedge x_b^0 \wedge x_c^0 \wedge x_d^0 \wedge x_w^0)$
(3) $(x_a^0 \wedge x_b^1 \wedge x_c^0 \wedge x_d^0 \wedge x_w^0)$
(4) $(x_a^1 \wedge x_b^1 \wedge x_c^0 \wedge x_d^0 \wedge x_w^0)$
(5) $(x_a^0 \wedge x_b^0 \wedge x_c^1 \wedge x_d^0 \wedge x_w^0)$
(6) $(x_a^1 \wedge x_b^0 \wedge x_c^1 \wedge x_d^0 \wedge x_w^0)$
(7) $(x_a^0 \wedge x_b^1 \wedge x_c^1 \wedge x_d^0 \wedge x_w^0)$
(8) $(x_a^1 \wedge x_b^1 \wedge x_c^1 \wedge x_d^0 \wedge x_w^0)$
(9) $(x_a^0 \wedge x_b^0 \wedge x_c^0 \wedge x_d^1 \wedge x_w^0)$
(10) $(x_a^1 \wedge x_b^0 \wedge x_c^0 \wedge x_d^1 \wedge x_w^0)$
(11) $(x_a^0 \wedge x_b^1 \wedge x_c^0 \wedge x_d^1 \wedge x_w^0)$
(12) $(x_a^1 \wedge x_b^1 \wedge x_c^0 \wedge x_d^1 \wedge x_w^0)$
(13) $(x_a^0 \wedge x_b^0 \wedge x_c^1 \wedge x_d^1 \wedge x_w^0)$
(14) $(x_a^1 \wedge x_b^0 \wedge x_c^1 \wedge x_d^1 \wedge x_w^0)$
(15) $(x_a^0 \wedge x_b^1 \wedge x_c^1 \wedge x_d^1 \wedge x_w^0)$
(16) $(x_a^1 \wedge x_b^1 \wedge x_c^1 \wedge x_d^1 \wedge x_w^0)$
(17) $(x_a^0 \wedge x_w^1)$
(18) $(x_a^1 \wedge x_w^1)$
(19) $(x_b^0 \wedge x_w^1)$
(20) $(x_b^1 \wedge x_w^1)$
(21) $(x_c^0 \wedge x_w^1)$
(22) $(x_c^1 \wedge x_w^1)$
(23) $(x_d^0 \wedge x_w^1)$
(24) $(x_d^1 \wedge x_w^1)$

The two previous representations of a structure, a single graph and a set of features, highlight the trade-off between interpretability and flexibility. On the one hand, a single undirected graph is more interpretable than a set of features because a graph make conditional independences clearly visible, that is, these independencies can be retrieved by rechability. For instance, any conditional independence can be efficiently read off from a graph by using reachability, in contrast from a set of features, this requires to check if such independence is hold in every feature in the set. On the other hand, a set of features is more flexible than a single graph because it describes the structure as combinations between variables and their values. Instead, graphs only describe combinations between variables. This flexibility permits a set of features to capture a wider range of structures. For instance, a set of features can be much more convenient when a distribution holds context-specific independences; or when the table-based potential functions shown in Eq. 2.3 have repeated values [30].

2.3. *Inference*

Markov networks are mainly used to answer a broad range of inference questions that are of interest. For this task, the structure of a Markov network is critial to perform inference effectively, because the independences assertions can be exploted for reducing the computational complexity (Part II Ref. 8). The most common query is the conditional probability query, that is, the conditional probability of some query variables X_Q given the values of the evidence variables X_E. This query is computed by summing out all the remaining variables X_W, $W = V \setminus Q \cup E$, that is:

$$p(X_Q|X_E) = \sum_{x_W \in \mathrm{Val}(W)} p(X_Q|X_E, x_W).$$

In the general case, inference in Markov networks is #P-complete [10]; requiring approximate inference techniques. These techniques exploit the structure of a Markov network in order to improve substantially the efficiency of inference. For instance, inference in Sum-product networks, a class of Markov networks that assumes a very rigid structure, results in tractable inferences [31]. Moreover, inference techniques can exploit context-specific independences in order to reduce expensive inferences [22,28,32]. One widely used approximate technique for inference is *Gibbs sampling*, a special case of Markov-chain Monte Carlo (MCMC) [33]. Gibbs sampling answers inference queries taking a sample from a Markov network by sampling each variable X_a in turn given its Markov blanket $\mathrm{MB}(a)$ (Chapter 12.3 Ref. 8).

2.4. *Weight learning*

Given the structure of a distribution, the task of constructing a Markov network consists in learning its weights. The weight learning automatically estimates the weights from data by optimizing an objective function of the weights. When data is fully observed, the log-likelihood is an often used convex objective function. Unfortunately, the log-likelihood cannot be solved in closed-form to obtain the optimal weights, thus requiring to perform convex optimization. At each iteration of this convex optimization, it is needed to compute inference queries to estimate the partition function Z. In the general case, these queries must be computed by using approximate inference techniques, producing inaccurate results [34]. In practice, an efficient alternative objective function is the pseudo-likelihood [35]. The pseudo-likelihood is defined so as to avoid the computation of the partition function. Formally, let $\mathcal{D} = \{x^0, x^1, \ldots\}$ be a discrete set of examples in a domain, the pseudo-loglikelihood is defined as follows:

$$\log p(x) = \sum_{a \in V} \sum_{x^i \in \mathcal{D}} \log p(x_a | x^i_{\mathrm{MB}(a)}),$$

which uses the independence assumptions encoded by the structure, namely, the independence of each variable conditioned by its Markov blanket. However, inference in Markov networks trained by using pseudo-likelihood tend to be only accurate on queries with large amounts of evidence [36].

Finally, both objective functions are prone to overfit data. This can be reduced by adding regularization terms. In practice, the most common regularation terms are L_1 or L_2 norms, which assume a Laplacian or Gaussian prior on the weights, respectively. These priors are parameterized by additional weights known as *hyper-parameters*. L_1 prior tends to select models that are much sparser than L_2, thus L_1 is increasingly used by feature selection algorithms to reduce variance errors. For instance, L_1 has been shown to be used for selecting relevant features of a log-linear model [29,14,37,2,16]. However, L_1 is very sensitive to the choice of its hyperparameters, requiring expensive cross-validations to find them [21].

2.5. *Structure learning*

The aim of structure learning is to elicit the structure of a Markov network from data. This subsection is not meant to be a compendium of every structure learning algorithm, but rather a comprehensive and representative selection in order to describe general approaches for searching a structure through a space of candidate structures. We review two categories of approaches: *general approaches* that search the structure regardless of the representation of the structure chosen; and *specific approaches* that are intertwined with the chosen representation, taking advantage of the topology of the specific search space. In practice, structure learning algorithms can be seen as an instantiation that combines both approaches.

In practice, the most common general approaches are: *top-down* and *bottom-up* approaches. The top-down approach consists in starting from a very general structure, i.e. the fully connected structure, which is specified through independence assumptions, resulting at the end in a more specific structure [17,15]. Alternatively, the bottom-up approach consists in starting from a very specific structure, i.e. the empty structure, which is generalized by removing independence assumptions, resulting at the end in a more general structure [11,13,37,18,19]. Roughly speaking, the top-down approach is more prone to make type I errors than the bottom-up approach; in contrast, the bottom-up approach is more prone to make type II errors than the top-down approach. Therefore, some structure learning algorithms may use both approaches. For instance, the GSMN algorithm starts the search from a very specific structure which is generalized and specialized in two phases respectively: the grow phase that follows the bottom-up approach, and the shrink phase that follows the top-down approach [18].

The previous approaches can use two strategies for searching the structure. The global strategy consists in directly searching a structure defined over all the variables [11,12,17,19]. On the other hand, the local strategy consists in searching a local structure for each variable in turn, and then combining them into a unique structure [18,13,37,16].

In practice, several specific approaches can be found, one for each different learning algorithm, but they can be analized in a general way according to the goal of learning. As previously discussed, the goal of learning is interwined by the representation of the structure. Density estimation algorithms use a set of features for its flexibility, in contrast, knowledge discovery use a single graph for its interpretability. For the search, a set of features defines a more large space of candidate structure than a single graph. For this reason, density estimation algorithms search the structure in two phases: a feature generation phase, and a feature selection phase. The generation phase consists in a cheap way to generate many candidate features that have high support in data; leading to many irrelevant features as well as inconsistent independence assumptions. The selection phase consists in removing irrelevant

features by using a bias. Two common bias applied are: positive features[b] and regularization terms. In the search, the former reduces the space of candidate structures because there are fewer possible features, as discussed in (Section 5 Ref.13), and (Section IV.C Ref. 38). In the weight learning, the latter removes features that have low support in data, in other words, these features have weights equal to zero. To avoid discarding of relevant features by using regularization terms, their hyperparameters must be tuned by cross-validation. In contrast to density estimation algorithms, knowledge discovery algorithms search the structure as constraint-based search, that is, they limit the space of candidate structure by using independence assumptions as constraints. So, under this approach, any candidate structure must satisfy the constraints. Moreover, independence properties, as those in Appendix A, can be used to infer new independence assumptions that further reduce the search space. However, if the independence assumptions are incorrect, the inference of new assumptions produces cascading errors, leading to incorrect structures [39,19]. For determining the truth value of an independence assumption, it is not required to perform weight learning. A common method for determining the truth value is by using statistical independence tests. However, statistical independence tests can be incorrect due to sample size problems [40]. This problem is exhacerbated by the fact that common statistical independence tests, such as χ^2, requires a sample size that increases exponentially with the number of variables [41].

3. Towards a more flexible representation

As discussed before, the aim of this work is to propose a new knowledge discovery algorithm that learns structures that can capture context-specific independences of the form $I(X_A, X_B \mid X_U, x_W)$, that is, conditional independences $I(X_A, X_B \mid X_U)$ that only hold in the context x_W. However, current knowledge discovery algorithms use a single graph as the representation of the structure which cannot capture such independences. For overcoming this limitation, we present an alternative representation of the structure based on *CSI models*, first introduced in two works [25,26], which can capture such independences. Roughly speaking, a CSI model is a collection $\mathcal{G}$ of pairs defined by a graph G and an assignment x^i_W, denoted by (G, x^i_W). Given a pair $(G, x^i_W) \in \mathcal{G}$, the graph G captures conditional independences in the context x^i_W allowing the representation of context-specific independences. Unfortunately, the use of CSI models as representation presents some issues for learning them because CSI models are too flexible. The flexibility of CSI models arises from the fact that each of its graphs can be associated to any possible context, resulting in an enormous space of candidate structures (Section 9 Ref. 26). In other words, the space of candidate CSI models can be seen as a combination of two exponential spaces: the space of candidate graphs and the space of candidate contexts. In this manner, for

[b]Features whose scopes are not for false values, formally, a feature f_j is positive if $x^j_a \neq 0$ for all $a \in D_j$.

each possible context x^i_W in the space of candidate contexts, the context x^i_W can be associated to any graph G in the space of candidate graphs. For overcoming this problem, we propose a more rigid class of CSI models whose graphs can be only associated to very rigid contexts. These models, that we name as *canonical models*, lead to a substantial simplification of the space of candidate structures, but keeping the flexibility to encode context-specific independences qualitatively. Moreover, we show that canonical models permit us to design new learning algorithms as simple adaptations of current knowledge discovery algorithms.

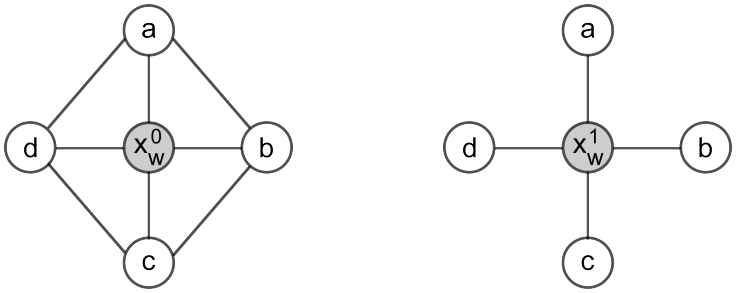
Following the previous discussion, the aim of this section is to describe the CSI models and, particularly, the canonical models, highlighting how CSI and canonical models can capture context-specific independences. This section concludes showing how canonical models permit us to learn them by using simple adaptations of current knowledge discovery algorithms. Next, Section 4 presents a learning algorithm that learns canonical models from data. The rest of this section is divided into two parts: Section 3.1, where the CSI models are presented; and Section 3.2, where the canonical models are presented.

3.1. *CSI models*

As shown in [26], a CSI model is a hierarchical log-affine model. These models present an interesting property: they can be represented by using graphs [5]. A graphical representation of a CSI model $\mathcal{G}$ is a collection of *instantiated graphs* $G(x^i_W)$, one for each pair (G, x^i_W). A graph G is instantiated by a context x^i_W when its nodes W are associated to an *assignment* x^i_W, and its remaining nodes $V \setminus W$ are associated to the *variables* $X_{V \setminus W}$. Then, the edges of the instantiated graph $G(x^i_W)$ can capture statistical associations among x^i_W and $X_{V \setminus W}$. Therefore, by a straightforward adaptation of the concept of reachability, context-specific independences that hold in the context x^i_W can be retrieved from an instantiated graph $G(x^i_W)$. As a result, a CSI model $\mathcal{G}$ can capture the context-specific independences present in a joint distribution $p(X)$ by using a set of instantiated graphs $G(x^i_W) \in \mathcal{G}$, each one capturing independences present in a conditional distribution of the form $p(X_{V \setminus W} | x^i_W)$. In contrast, a single graph G can only capture conditional independences that hold in the joint distribution $p(X)$.

Example 3.1. Figure 2 shows a CSI model $\mathcal{G}$ that represents the structure for the distribution of Example 2.1. This model is composed by two instantiated graphs $G(x^0_w)$ and $G(x^1_w)$, shown in Figure 2(a) and Figure 2(b) respectively. In these graphs, gray nodes indicate assigned variables. The context-specific independences can be read off from each instantiated graph by using reachability. The graph $G(x^0_w)$ encodes no independences, but the graph $G(x^1_w)$ encodes that each pair of variables $X_a, X_b \in X_{V \setminus \{w\}}$ are mutually independent given the context x^1_w.

The use of CSI models as the representation of the structure offers flexibility and interpretability, that is, they are sufficiently flexible for capturing context-specific



(a) The graph $G(x^0_w)$ is instantiated by the context x^0_w.

(b) The graph $G(x^1_w)$ is instantiated by the context x^1_w.

Fig. 2. a CSI model $\mathcal{G}$ composed of two instantiated graphs: the graph $G(x^0_w)$ shown in figure 2(a), and the graph $G(x^1_w)$ shown in Figure 2(b).

independences which can read off from the model by using reachability. However, this model is too flexible due to the fact that a graph can be associated to any possible context, resulting in an enormous space of candidate CSI models. For instance, the definition of a CSI model consists in defining a set of instantiated graphs, where the definition of each instantiated graph requires to decide which nodes will be instantiated, and which edges will be present. For limiting the space of candidates, we propose the canonical models, a more inflexible class of CSI models that avoids the need of deciding which nodes to assign.

3.2. *Canonical models*

We introduce in this section a subclass of CSI models that we named canonical models. A canonical model, denoted by $\bar{\mathcal{G}}$, is a CSI model that is only composed by *canonical graphs*. A canonical graph is an instantiated graph where all their nodes are instantiated by using a canonical assignment. Formally, an instantiated graph $G(x^i)$ is canonical if the assignment x^i is canonical, that is, $x^i \in \text{Val}(V)$. As a result, for defining a canonical graph $G(x^i)$, it is not necessary to decide which nodes would be assigned, because all the nodes V are instantiated. Therefore, the definition of canonical models avoids the need of deciding which nodes to assign, leading to a substantial simplification of the space of candidate models. In other words, the space of canonical models is a proper subset of the space of candidate CSI models. From this fact, the definition of a canonical model $\bar{\mathcal{G}}$ can be realized in a simple way: defining a set of canonical assignments $\mathcal{X}$ and then associating a canonical graph to each assignment $x^i \in \mathcal{X}$, formally $\bar{\mathcal{G}} = \{G(x^i) \colon x^i \in \mathcal{X}\}$. Given a particular set of canonical assignments $\mathcal{X}$, the size of the space of candidate canonical models depends on the number $|\mathcal{X}|$ of contexts. A possible definition for the set $\mathcal{X}$ is to use all the possible canonical assignments of the domain variables, that is, $\mathcal{X} = \text{Val}(V)$.

Despite of the simplification of the space of candidates, canonical models still capture context-specific independences. In contrast to general CSI models, a canon-

ical model requires several canonical graphs for capturing a context-specific independence. For instance, let us suppose that we want to encode a context-specific independence $I(X_a, X_b \mid x_w)$ in a canonical model $\bar{\mathcal{G}}$. From Definition 2.1, this independence implies a set of independences of the form $I(x_a, x_b \mid x_w)$, for all the assignments x_a and x_b in $\mathrm{Val}(a)$ and $\mathrm{Val}(b)$, respectively. Then, each independence $I(x_a, x_b \mid x_w)$ is captured by a particular $G(x^i) \in \bar{\mathcal{G}}$, one whose context x^i satisfies: $x^i_a = x_a$, $x^i_b = x_b$, and $x^i_w = x_w$. In this manner, the independence $I(X_a, X_b \mid x_w)$ is captured by using several canonical graphs.

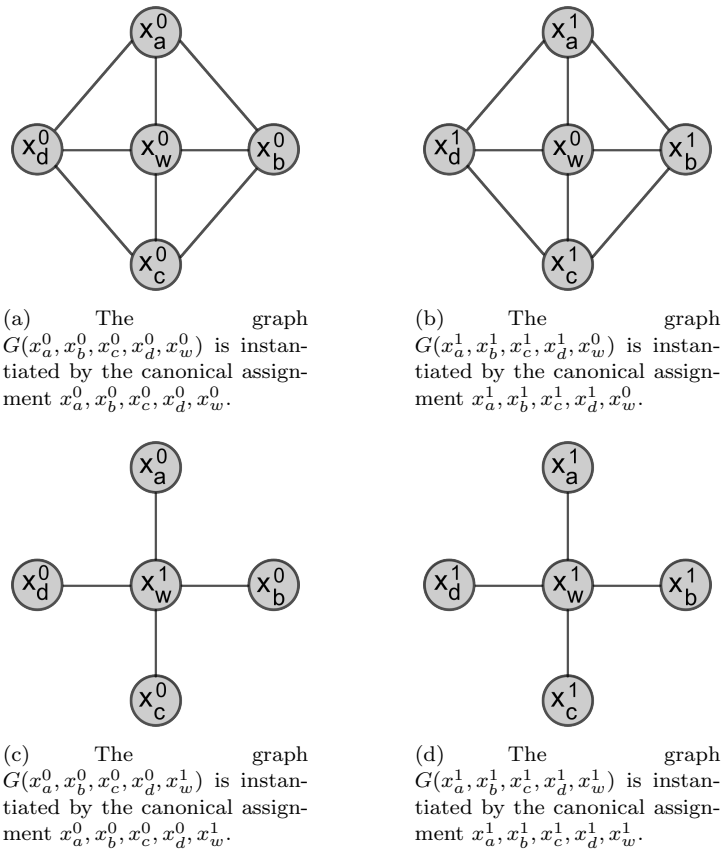
Example 3.2. Let us illustrate how the structure of the distribution of Example 2.1 is captured by a canonical model $\bar{\mathcal{G}}$. A trivial way to define $\bar{\mathcal{G}}$ is using the set of all the canonical assignments $\mathcal{X} = \mathrm{Val}(V)$, defining $2^{|V|} = 32$ canonical graphs, that is, a graph $G(x^i)$ per canonical assignment $x^i \in \mathrm{Val}(V)$. According to the value of X_w in the canonical assignments, the canonical graphs have two different structures: a fully connected structure when $X_w = x^0_w$, and the star structure when $X_w = x^1_w$. For this reason, Figure 3 only shows four canonical graphs in $\bar{\mathcal{G}}$. The figures 3(a), 3(b), 3(b), 3(d) show, in order, the following canonical graphs: $G(x^0_a, x^0_b, x^0_c, x^0_d, x^0_w)$, $G(x^1_a, x^1_b, x^1_c, x^1_d, x^0_w)$, $G(x^0_a, x^0_b, x^0_c, x^0_d, x^1_w)$, and $G(x^1_a, x^1_b, x^1_c, x^1_d, x^1_w)$. Similary to Example 3.1, the context-specific independences can be read off from each canonical graph by using reachability. For instance, the graph $G(x^0_a, x^0_b, x^0_c, x^0_d, x^1_w)$ encodes that any pair of values between $x^0_a, x^0_b, x^0_c, x^0_d$ are mutually independent given the context x^1_w; in contrast, the graph $G(x^1_a, x^1_b, x^1_c, x^1_d, x^0_w)$ encodes no independences between any pair of values $x^1_a, x^1_b, x^1_c, x^0_d$.

An interesting property of using canonical models for representing the structure is that a canonical model is defined from the set $\mathcal{X}$, that is, a set of mutually independent canonical assignment $x^i \in \mathrm{Val}(V)$. This follows from the fact that each canonical assignment can be seen as a row in the table-based representation of the joint distribution. Thus, the learning of canonical models from data can be decomposed into a set of independent subproblems, that is, the learning of each single canonical graph $G(x^i)$, $x^i \in \mathcal{X}$. In a general perpective, knowledge discovery algorithms are designed to learn a single graph. Hence, each canonical graph can be learned by using basic ideas of knowledge discovery algorithms. The next section presents a simple learning algorithm that learns canonical models by learning each canonical graph in turn by using basic concepts of knowledge discovery algorithms.

4. Learning canonical models

This section presents the CSPC[c] algorithm for learning structures represented as canonical models. In a general perspective, CSPC uses a top-down approach for learning the structure, that is, CSPC defines an initial general canonical model and

[c]CSPC is the acronym of Context-Specific Parent and Children algorithm, because its design was inspired by the well-known Parent and Children algorithm [17].



(a) The graph $G(x_a^0, x_b^0, x_c^0, x_d^0, x_w^0)$ is instantiated by the canonical assignment $x_a^0, x_b^0, x_c^0, x_d^0, x_w^0$.

(b) The graph $G(x_a^1, x_b^1, x_c^1, x_d^1, x_w^0)$ is instantiated by the canonical assignment $x_a^1, x_b^1, x_c^1, x_d^1, x_w^0$.

(c) The graph $G(x_a^0, x_b^0, x_c^0, x_d^0, x_w^1)$ is instantiated by the canonical assignment $x_a^0, x_b^0, x_c^0, x_d^0, x_w^1$.

(d) The graph $G(x_a^1, x_b^1, x_c^1, x_d^1, x_w^1)$ is instantiated by the canonical assignment $x_a^1, x_b^1, x_c^1, x_d^1, x_w^1$.

Fig. 3. Several canonical graphs that compose a canonical model $\bar{\mathcal{G}}$.

then this model is specified by using context-specific independences elicited from data. The specification of the initial canonical model $\bar{\mathcal{G}}$ consists in specifying each canonical graph $G(x^i) \in \bar{\mathcal{G}}$, where $x^i \in \mathcal{X}$. A graph $G(x^i)$ is specified by removing edges according to a well-known criterion often used by knowledge discovery algorithms [17,18,42,19]. For determining whether the criterion is satisfied, it is required to elicit context-specific independences from data. Finally, once the canonical model has been specified, CSPC uses the canonical graphs for generating a set of features. Instead of a canonical model, a set of features is a more common representation which permits us to use standard software packages available for weight learning and inference. For instance, the set of features generated can be taken together in a log-linear model, whose weights can be obtained by performing any standard weight learning package.

The key elements of CSPC are: *i)* the definition of the initial canonical model; *ii)* the specialization of each canonical graph; *iii)* the elicitation of context-specific independeces from data; and *iv)* the feature generation from the learned canonical

model. Algorithm 1 shows the pseudo-code of CSPC, offering an overview of how these key elements are related. Moreover, the rest of this section is organized according to these key elements. Section 4.1 describes lines 1 and 2, which define the initial canonical model. Section 4.2 describes lines 3 and 4, which specify the initial canonical graphs. Section 4.3 describes how to elicit context-specific independences, which are used to specify a canonical graph. Section 4.4 describes line 5, which generates a set of features from the specified canonical model. Finally, Section 4.5 describes several tips for speeding up the execution of CSPC.

Algorithm 1: Overview of CSPC

Input: domain V, dataset $\mathcal{D}$

1 $\mathcal{X} \leftarrow$ Define the set of canonical assignments
2 $\bar{\mathcal{G}} \leftarrow$ Define a set of initial graphs $\{G(x^i) : x^i \in \mathcal{X}\}$
3 **foreach** $G(x^i) \in \bar{\mathcal{G}}$ **do**
4 $\quad G(x^i) \leftarrow$ Specialize($G(x^i)$, V, $\mathcal{D}$)
5 $\mathcal{F} \leftarrow$ Feature generation from each $G(x^i) \in \bar{\mathcal{G}}$

4.1. *The initial canonical model*

CSPC starts the top-down approach from an initial canonical model $\bar{\mathcal{G}}$. For defining the initial $\bar{\mathcal{G}}$, CSPC define an initial set of canonical assignments $\mathcal{X}$, where each canonical assignment $x^i \in \mathcal{X}$ is used to instantiate an initial graph. A naïve definition of the set of canonical assignments is $\mathcal{X} = \text{Val}(V)$. However, in this case, the size of $\mathcal{X}$ is exponential with respect to the number of variables. A more practical alternative is to use a *data-driven approach* for defining it, that is, adding a context x^i per unique training example in $\mathcal{D}$. This alternative is often used as a starting point in the search for density estimation algorithms [13,15]. This initial model guarantees that any feature obtained at the end matches at least one training example. Once defined the set $\mathcal{X}$, CSPC constructs the initial canonical model $\bar{\mathcal{G}}$ as a collection of fully connected graphs, that is, a fully connected graph $G(x^i)$ for each $x^i \in \mathcal{X}$. We note that this approximation may be drastic for dense distributions and large datasets, but the approximation permits us to drastically reduce the time complexity of the algorithm. For evaluating the impact of this approximation in the quality of the learned structures, we performed an offline empirical evaluation over synthetic data, confirming that both ways of defining $\mathcal{X}$ result in equivalent learned structures.

4.2. *Specialization of a canonical graph*

For encoding the context-specific independences present in data, CSPC specifies each canonical graph by using a combination of two criteria: the *local Markov property* and the *pairwise Markov property* (Chapter 3.2.1 Ref. 5). This criterion says

that each variable X_a is conditionally independent of the variable $X_b \in X_{V \setminus \mathrm{MB}(a)}$ given its Markov blanket variables $X_{\mathrm{MB}(a)}$. Therefore, given an initial canonical graph $G(x^i)$, any edge (a, b) can be removed if $I(X_a, X_b \mid x^i_{\mathrm{MB}(a)})$ is present in data. This criterion allows to use a local strategy for specifying a graph: specifying the adjacencies of each node $a \in V$ in turn. This local strategy is the same used by the PC algorithm [17]. Algorithm 2 shows how this local strategy is performed. Roughly speaking, the algorithm iterates over each node $a \in V$ in line 4. Then, each node $b \in \mathrm{adj}(a)$ is iterated in line 5 in order to remove the edge (a, b) if this satisfies the criterion. For that, line 6 searches the Markov blanket $\mathrm{MB}(a)$, denoted by W, from the remaining nodes $\mathrm{adj}(a) \setminus \{b\}$. For determining whether an edge satisfies the criterion, line 7 elicits the independence $I(X_a, X_b \mid x^i_W)$ from data. If it is satisfied, then the edge (a, b) is removed from $G(x^i)$, as shown in line 8. In such case, the node b is removed from the set $\mathrm{adj}(a)$, specifying in this way the graph. When all the edges from the node a have been checked, the search proceeds by taking some next node, repeating the previous process.

Algorithm 2: SPECIALIZATION OF $G(x^i)$

Input: graph $G(x^i)$, domain V, dataset $\mathcal{D}$

```
1  k ← 0
2  repeat
3  |  foreach node a ∈ V do
4  |  |  adj(a) ← adjacencies of a obtained from G(x^i)
5  |  |  foreach b ∈ adj(a) do
6  |  |  |  foreach W ⊆ adj(a) \ {b} s.t. |W| = k do
7  |  |  |  |  if I(X_a, X_b | x^i_W) is true in D then
8  |  |  |  |  |  G(x^i) ← remove the edge (a, b) in G(x^i)
9  |  |  |  |  |  break // go to line 5
10 |  k ← k + 1
11 until |adj(a)| < k, for all a ∈ V;
12 return G(x^i)
```

Then, before removing an edge (a, b), it is required to find the Markov blanket $\mathrm{MB}(a)$, that is, the conditioning nodes W. This nodes are found by using an incremental search in line 6. This incremental search uses an integer-valued threshold k which is initialized with zero in line 1. This threshold fixes the maximum number of conditioning nodes in W, that is, any independence assertion $I(X_a, X_b \mid x^i_W)$ has up to $k = |W|$ conditioning variables. When all the adjacencies have been specialized, this threshold is increased in line 10. Therefore, the termination condition of the local search is reached when $|\mathrm{adj}(a)| < k$ for all $a \in V$, in other words, when the size of each adjacency is smaller than the threshold.

4.3. *Elicitation of context-specific independences*

For eliciting the independences present in data, several statistical methods can be used (Section 3.A Ref. 38), (Section 4.3 Ref. 13), (Section 4.1 Ref. 51), and (Section 3 Ref.14). However, CSPC uses a very common method used by knowledge discovery algorithms called statistical independence tests [17,52,42,19]. A statistical independence test determines the truth value of an independence assertion $I(X_a, X_b \mid X_W)$ by using statistics computed from a contingency table constructed from data. This can be seen as a typical problem of hypothesis testing. In our case, for determining the truth value of a context-specific independence $I(X_a, X_b \mid x_W)$, CSPC uses a straightforward adaptation of a statistical independence test. As discussed in Section 2.1, this adaptation is based on the following fact: any arbitrary context-specific independence $I(X_a, X_b \mid x_W)$ that holds in $p(X)$ is a conditional independence, $I(X_a, X_b \mid \emptyset)$, in the conditional distribution $p(X \setminus \{X_W\} \mid x_W)$ [43]. Moreover, given that $\mathcal{D}$ is a representative sample of $p(X)$, a partition $\mathcal{D}[x^i_W] = \{x \in \mathcal{D} \colon x_W = x^i_W\}$ is a representative sample of $p(X_{V \setminus W} | x^i_W)$. Using $\mathcal{D}[x^i_W]$, CSPC obtains the truth value for any assertion $I(x^i_a, x^i_b \mid x^i_W)$ by performing a statistical test for the assertion $I(X_a, X_b \mid \emptyset)$. If $I(X_a, X_b \mid \emptyset)$ is true in $\mathcal{D}[x^i_W]$, then the X_a and X_b are conditionally independent. This fact implies that x^i_a and x^i_b are conditionally independent in $p(X_{V \setminus W} | x^i_W)$, and they are contextually independent given the context x^i_W in $p(X)$.

4.4. *Feature generation from a canonical model*

When the specialization of each canonical graph is finished, the resulting structure is a canonical model $\bar{\mathcal{G}}$. This structure is converted into an equivalent representation in the form of a set of features $\mathcal{F}$. Algorithm 3 shows how the set of features is generated from a canonical model $\bar{\mathcal{G}}$. For each canonical graph $G(x^i) \in \bar{\mathcal{G}}$, a feature x^i_C is generated for each maximum cliques $C \in G(x^i)$. In this way, the resulting set of features guarantees that the context-specific independences captured by $\bar{\mathcal{G}}$ are correctly encoded. In our implementation, we use the Bron-Kerbosch algorithm for obtaining the maximum cliques of a graph [44]. For ensuring the positiveness of the distribution, a simple way is to generate a unitary feature for each $x^i \in \mathrm{Val}(a)$ and for all $a \in V$.

Example 4.1. Let us now illustrate how a set of features $\mathcal{F}$ is generated from the canonical graphs shown in Example 3.2. The set of features is generated in two steps. First, we search the maximum cliques in each canonical graph. Second, for each maximum clique, the context of the canonical graph is used for constructing a feature. For instance, the canonical graphs shown in Figure 3(a) and 3(b) have the same maximum clique: $\{a, b, c, d, w\}$. Using this clique and the two canonical assignments, the following features can be constructed:

Algorithm 3: Feature Generation

Input: Canonical model $\bar{\mathcal{G}}$

```
1 F ← ∅
2 foreach G(x^i) ∈ Ḡ do
3 |   foreach maximum clique C ∈ G(x^i) do
4 |   |   F ← F ∪ {x^i_C}
5 F ← Add unit feature x^i_a for each variable X_a ∈ X
6 return F
```

(1) $(x_a^0 \wedge x_b^0 \wedge x_c^0 \wedge x_d^0 \wedge x_w^0)$ (2) $(x_a^1 \wedge x_b^1 \wedge x_c^1 \wedge x_d^1 \wedge x_w^0)$

On the other hand, the canonical graphs shown in Figure 3(c) and 3(d) have the same maximum cliques: $\{a, w\}, \{b, w\}, \{c, w\}, \{d, w\}$. Using these cliques and the two canonical assignments, the following features can be constructed:

(1) $(x_a^0 \wedge x_w^1)$
(2) $(x_a^1 \wedge x_w^1)$
(3) $(x_b^0 \wedge x_w^1)$
(4) $(x_b^1 \wedge x_w^1)$
(5) $(x_c^0 \wedge x_w^1)$
(6) $(x_c^1 \wedge x_w^1)$
(7) $(x_d^0 \wedge x_w^1)$
(8) $(x_d^1 \wedge x_w^1)$

4.5. *Implementation details for speeding up computation*

CSPC naively implemented is computationally infeasible, because it must perform $|\mathcal{X}|$ mutual independent structure learning steps for each canonical graph $G(x^i)$; as shown in line 4 of Algorithm 1. For obtaining a graph $G(x^i)$, it is required to decide the presence of $\binom{|V|}{2}$ edges by performing independence tests. This is discussed in Section 4.2, for determining the presence of an edge (a, b), it is required to perform several independence tests, due to the incremental strategy used for finding the Markov blanket (the conditioning nodes W). To perform an independence test implies the counting of sufficient statistics for constructing contingency tables from the dataset $\mathcal{D}$. Therefore, this subsection describes some tips for speeding up the execution of CSPC in order to reduce the execution time: i) the use of AD-Tree [45], a cache for sufficient statistics to fast constructing contingency tables (a brief description in Appendix C); ii) an approximate strategy for finding Markov blankets, reducing the amount of independence tests performed; and iii) a naïve way to parallelize the learning of a canonical model for reducing the execution time.

CSPC uses AD-Tree for fast construction of contingency tables. However, in high-dimensional domains, AD-Tree can incurre in intractable memory requirements. Thus, CSPC takes advantage of Leaf-Lists for not incurring in intractable memory, as follows. Given a graph $G(x^i)$ in Algorithm 2, the presence of any edge (a, b) requires to perform incremental independence assertions with the form $I(X_a, X_b \mid x_W^i)$, that is, for incremental contexts x_W^i where $W \subseteq V$ and $|W| = k$.

Therefore, Algorithm 2 only requires sufficient statistics from specific subtress of the AD-Tree, that is, those that satisfy the contexts $x^i_W \subseteq x^i$. For this reason, Algorithm 2 starts with a very sparse AD-Tree by using a very large pruning threshold. Then, for each increment of k, the AD-Tree is expanded on demand, that is, the depths of the subtrees that satisfy x^i_W increase, reusing all the computations performed in the previous iterations due to the Leaf-Lists.

The use of AD-Tree reduces the time complexity of constructing contingency tables. However, CSPC must still perform a large amount of statistical tests, because $|\mathcal{X}|$ canonical graphs must be specified. This is exacerbated by the incremental way used for obtaining the conditioning set W. When the underlying structure is very dense, the amount of tests for obtaining $G(x^i)$ grows exponentially because all the subsets W must be checked before finishing (line 6 in Algorithm 4), that is, the power set of each set of adjacencies. One simple strategy to overcome this is to use a maximum size K_{max} for conditioning sets. This threshold changes the termination condition of line 2 in Algorithm 2 as follows: ($|\operatorname{adj}(a)| < k$ or $|\operatorname{adj}(a)| = K_{max}$), for all $a \in V$. Given a fixed *maximum degree*[d] of the underlying structure, three situations are present for using K_{max}: (i) if the maximum degree is greater than K_{max}, the resulting structure is more dense than the underlying structure; (ii) if the maximum degree equals K_{max}, then the resulting structure is correct; and (iii) if the maximum degree is smaller than K_{max}, then the termination condition $|\operatorname{adj}(a)| < k$ is reached first, and thus the correct structure is found. Therefore, for any case, the resulting structure is an I-map. However, for the situation (iii), the resulting structures are approximations because these structures do not encode all the independences.

Finally, it is worth noting that Algorithm 1 can be performed in a parallel way due to the definition of canonical model. A canonical model is a collection of mutually independent canonical graphs, in other words, the definition of each canonical graph does not dependent of the remaining canonical graphs. As a result, the set of canonical assigments $\mathcal{X}$ can be splitted into any set of partitions, that is, $\mathcal{X} = \bigcup_j \mathcal{X}_j$ where $\mathcal{X}_j$ is a partition.

5. Empirical evaluation

This section shows the improvements that can be reached in the accuracy of the learned structures when context-specific independences are explicitly learned; highlighting the direct correlation between the correctness of the structure and the accuracy of the distribution. These improvements cannot be obtained by existing knowledge discovery and density estimation algorithms. Knowledge discovery algorithms focus on learning accurate structures but they use an inflexible representation that cannot encode context-specific independences. On the other hand, density estimation algorithms do not focus on learning accurate structures but they

[d]The maximum degree of a graph structure is the maximum number of edges incident to a node.

use a flexible representation that can encode context-specific independences. This trade-off is mitigated by our approach allowing it to reach more accurate structures. Our approach is evaluated on two empirical scenarios: *synthetic datasets*, and *real-world datasets*. The former offers well-understood conditions which permit us to obtain precise measures to show the potential improvements that can be reached in a single application domain. On the other hand, the latter shows the improvements on multiple application domains. The evaluation consists in comparing the structures learned by CSPC and several knowledge discovery and density estimation state-of-the-art algorithms. This evaluation is conducted under the following performance measures: accuracy for encoding qualitatively independences that hold in the underlying structures; accuracy for answering inference queries; and general statistics over the learned structures and the underlying structures. The synthetic data used in our experiment, together with an open source implementation of CSPC algorithm, and the learned models are publicly available[e].

5.1. *Datasets*

The synthetic datasets were generated through Gibbs sampling on synthetic binary Markov networks (more details in Appendix B). The underlying structure of these models is similar to those shown in Example 3.1. This structure encodes independence assertions of the form $I(X_a, X_b \mid x_w^1)$ for all pairs $a, b \in V \setminus \{w\}$, but these assertions are not valid when $X_w = x_w^0$. In this way, the underlying structure can be seen as two instantiated graphs depending on the context: a fully connected graph $G(x_w^0)$, and a star graph $G(x_w^1)$ where the central node is x_w^1 and the remaining nodes $V \setminus \{w\}$ are leaves. Despite of the simplicity of this structure, this cannot be correctly captured by using a single graph, while it can be captured by a set of features or CSI models; as is shown in Example 2.2, Example 2.3 and Example 3.1. The number of variables in the synthetic models ranges from 6 to 9. The maximum degree of these underlying structures is equal to the number of variables n, resulting in a challenging problem for structure learning algorithms, which are usually tested on synthetic models with maximum degree at most 6 to 8 [14,29,19]. The generated datasets are partitioned into: a *training set* (70%) and a *validation set* (30%). The reason of this partition is due to the fact that density estimation algorithms use the validation set to tune their parameters for learning the models: they learn several models from the training set by using different parameters, and then they pick the best one using the validation set. On the other hand, CSPC and knowledge discovery algorithms do not use tunning parameters, they therefore use the whole dataset, i.e. the union of both sets, to learn the models.

The real-world datasets used here have been used by several recent works on structure learning [36,30,31,2]. Their characteristics are described in Table 1. These datasets are samples coming from multiple application domains, for instance: click

[e]http://dharma.frm.utn.edu.ar/papers/ijait14

logs, nucleic acid sequences, collaborative filtering, and etcetera. The number of variables ranges from 16 to 1556, and the amount of datapoints varies from 2k to 291k. For a more detailed description of those datasets, see [13,15,16]. These datasets are partitioned in three parts: a *training set*, a *validation set*, and a *test set*; where the latter is denoted by $\mathcal{D}_{test}$. The validation set is used to tune parameters hyperparameters as well as input parameters of the density estimation algorithms. In contrast to synthetic datasets, we use the test set in real-world datasets for evaluation.

Table 1. Characteristics of all real-world datasets used in our experiments. The first column shows the number of variables. The three following columns show the sizes of each partitions: train set, validation set and test set. The last column $|\mathcal{X}|$ shows the number of canonical contexts obtained from the unique training examples in the training set.

Dataset	$\|X\|$	Size of train set	Size of tune set	Size of test set	$\|\mathcal{X}\|$
NLTCS	16	16181	2157	3236	2671
MSNBC	17	291326	38843	58265	9228
Abalone	31	3134	417	626	648
Wine	48	4874	650	975	3898
KDDCup 2000	65	180092	19907	34955	6864
Plants	69	17412	2321	3482	8258
Covertype	84	30000	4000	6000	22679
Audio	100	15000	2000	3000	14969
Netflix	100	15000	2000	3000	14998
Accidents	111	12758	1700	2551	12756
Adult	125	36631	4884	7327	25060
Retail	135	22041	2938	4408	10510
Pumsb Star	163	12262	1635	2452	12037
DNA	180	1600	400	1186	1545
MSWeb	294	29441	3270	5000	10294
WebKB	839	2803	558	838	2754
BBC	1058	1670	225	330	1585
Ad	1556	2461	327	491	1573

5.2. *Methodology*

In this subsection we explain the methodology used for evaluating our approach against several structure learning algorithms. First, we describe the two methods used for measuring the accuracies of the structures learned by the algorithms; and, second, we explain which structure learning algorithms are used in our experimentation and their configuration settings.

We use different evaluation methods for each empirical scenary. In synthetic datasets, the underlying distributions are known, in contrast, in the real-world datasets they are not. Therefore, for synthetic datasets, we use the *Kullback-Leibler divergence* (KL), a method that evaluates how similar are the learned distributions against the underlying distributions [46,47]. On the other hand, for real-world datasets, we use the *conditional marginal log-likelihood* (CMLL), a method that

evaluates the accuracy of the learned distributions for answering inference queries [29,13,16,15,31].

The KL is a "distance measure" widely used to compare two distributions, $p(X)$ and $q(X)$, by measuring the information lost between them, as follows:

$$KL(p \mid\mid q) = \sum_{x \in \text{Val}(V)} p(x) \log \frac{p(x)}{q(x)},$$

where KL is zero if $p(X) = q(X)$, and positive otherwise. KL is not symmetric, but it satisfies the basic property of an error measure [47]. Therefore, let $p(x)$ be the underlying distribution, and let $q_1(x)$ and $q_2(x)$ be two learned distributions; then the learned distribution with the smallest KL divergence with respect to $p(X)$ is considered to be the most accurate. On the other hand, given a learned distribution $q(X)$ and a test set $\mathcal{D}_{test}$, the CMLL is computed by summing the conditional marginals of each variable in a query set Q, conditioned on the remaining variables corresponding to an evidence set $E = V \setminus Q$, as follows:

$$CMLL(q, \mathcal{D}_{test}) = \sum_{x \in \mathcal{D}_{test}} \sum_{a \in Q} \log q(x_a | x_E).$$

The query and evidence variables are defined by dividing the variables into four disjoint subsets, taking one subset as query variables, and the remaining as evidence variables. Then, this procedure was repeated such that each subset is at least once in the query variables. The conditional marginals can be computed by performing a Gibbs sampling on the learned model. In our experiment, Gibbs sampling was run with 100 burn-in and 1000 sampling iterations, as commonly used in other works.

CSPC is compared against 8 state-of-the-art structure learning algorithms, four for knowledge discovery, and four for density estimation. The knowledge discovery algorithms are two for Markov networks: GSMN [18], and IBMAP-HC [19]; and two for Bayesian networks: HHC [42], and PC [17]. The Bayesian networks structure learning algorithms were selected for the following reasons: HHC has shown competitive results against IBMAP-HC [19]; and against PC [17], because it is similar to the specialization of a canonical graph in CSPC, shown in Algorithm 2. PC offers a baseline to highlight the improvements resulting from learning context-specific independences. The previous two algorithms were modified for learning the structure of a Markov network by omitting the step of edges orientation [48,19]. For a fair comparison, we use the Pearson's χ^2 as the statistical independent test with a significance level of 0.05 for all algorithms, except IBMAP-HC that only works with the Bayesian statistical test of independence [49], using a threshold of 0.5. The knowledge discovery algorithms learn the structure as a graph, thus a set of features was generated from its maximum cliques in a similar fashion than Algorithm 3. On the other hand, the density estimation algorithms are GSSL [15], L1 [37], DTSL [16], and BLM [13]. For a fair comparison, we replicate the recommended tunning parameters for all the density estimation algorithms. For GSSL, we generated 0.5, 1, 2, and 5 million features; with pruning thresholds of 1, 5, and 10; Gaussian standard deviations of 0.1, 0.5,

and 1, combined with L_1 priors of 1, 5, and 10; giving a total of 108 configurations. For L1, we employed the same procedure used in [30], with L_1 priors of 0.1, 0.5, 1, 2, 5 ,10 15, and 20, giving a total of 8 configurations. For DTSL, we used the κ values of 1.0, 0.1, 0.01, 0.001, 0.0001 to obtain an initial model, and then 4 feature generation methods for positive features (NONZERO, PRUNE-NONZERO, PRUNE-5-NONZERO, PRUNE-10-NONZERO); and Gaussian standard deviations combined to L_1 priors similar to GSSL; giving a total of 50 configurations. For BLM, we use 1, 2, 5, 10, 20, 25, 50, 100, 200 nearest examples; frequency of accepted features of 1, 5, 10, and 25; and Gaussian standard deviation of 0.1, 0.5, and 1; given a total of 108 configurations. The objective function used by knowledge discovery and density estimation algorithms is pseudo-likelihood[f]. For all the density estimation algorithms, the best learned model was that with the maximum pseudo-likelihood on the validation set. Finally, for all baseline algorithms, we used publicly available code [g].

5.3. *Results for synthetic datasets*

The evaluation on synthetic datasets consists in varying the available data from 20 to 10k datapoints, comparing at each case the KL divergences of the Markov networks learned by each algorithm. The Markov networks are obtained by performing weight learning from the learned structures. As density estimation algorithms depend on the regularization, weight learning is performed with regularization. In contrast, for knowledge discovery algorithms, weight learning is performed without regularization in order to avoid unnecessary changes in the learned structures. Before evaluating the KL divergences obtained by the algorithms, we show the KL divergences obtained by using hand-coded structures in order to show insights on the behaviour of KL divergence. These structures are representative results of learning: structures that are an I-map for the underlying distribution, and structures that are not an I-map. The I-map structures show us the impact in KL divergence of encoding correct independences. In contrast, the non-I-map structures show us the impact of encoding incorrect independences. For each type, we propose two particular structures: the fully connected and underlying structures for the I-map case, and the empty and star structures for the non-I-map case. The empty structure encodes the maximum number of incorrect independences. The star structure encodes context-specific independences as conditional independences. This kind of structures are usually learned when data is scarce. The fully connected structure encodes incorrect dependences which obscures the context-specific independences. It is worth noting that the fully connected structure would be learned by knowledge discovery algorithms under the assumption that the statistical tests are correct, for instance,

[f]Weight learning was performed by using the version 0.5.0 of the Libra toolkit (http://libra.cs.uoregon.edu/)
[g]https://dharma.frm.utn.edu.ar/papers/ijait14

for large datasets. The underlying structure encodes the correct independences, that is, models with this structure should have the best KL divergences.

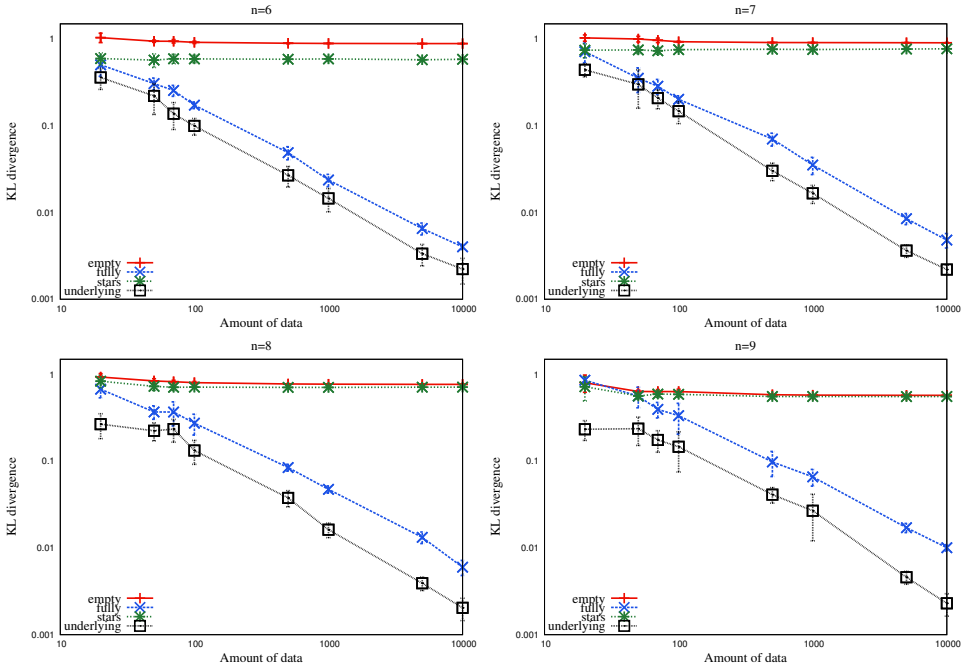


Fig. 4. KL divergences over increasing amounts of data for four different structures: the empty structure, the star structure, the fully structure, and the underlying structure. Every point represents the average and standard deviation over ten datasets with a fixed size. Each figure shows the KL divergences for different domain sizes: 6 (top left), 7 (top right), 8 (bottom left), and 9 (bottom right) variables. Lower values of KL divergence are better.

Figure 4 shows the KL divergence for the four proposed structures, each point showing the average and standard deviation for 10 random datasets obtained from the synthetic Markov networks. In these results, we can see that the KL divergences of each structure tend to zero while the amount of data increases. This is because the weight learning is more accurate for large data (Theorem 20.3 Ref. 8), improving the divergences for a fixed structure. However, the quality of these improvements depends on the structure used. As shown in Figure 4, the I-map structures (fully and underlying curves) reduce the KL divergence more quickly than the non-I-map structures (empty and star curves). This is due to the fact that the weight learning cannot learn accurate weights when the structure is not an I-map, resulting in imprecise distributions. Moreover, the quality of the KL divergences of the I-map structures are different. As shown in Figure 4, the underlying structures obtain better divergences than the fully structures. This fact highlights that the weight learning is more accurate when the structure used captures the context-specific independences.

Moreover, star structures obtain better divergences than empty structures, showing that encoding more incorrect independences (empty structures) produces worst divergences. This very simple experiment shows the strong correlation between the KL divergence measure and structural correctness.

Table 2. Average feature length (AFL) in the structures learned by structure learning algorithms. Every cell represents the average over ten datasets with a fixed size.

Datasets				Knowledge discovery algorithms				Density estimation algorithms			
n	$\|D\|$	Underlying	CSPC	GSMN	IBMAP	PC	HHC	GSSL	DTSL	BLM	L1
6	20	2.80±0.4	1.363	1.890	2.342	1.457	1.888	2.973	1.000	1.750	0.516
	50		1.625	2.057	2.309	1.796	2.028	3.000	1.301	1.221	0.311
	70		1.659	2.210	2.595	1.930	2.120	3.023	1.389	1.564	1.095
	100		1.722	2.579	2.610	2.148	2.270	3.033	1.529	1.883	1.050
	500		2.048	3.733	3.222	2.829	3.640	3.023	1.964	2.097	1.317
	1k		2.242	4.333	3.933	3.501	4.366	3.033	2.342	2.350	1.321
	5k		3.027	6.000	6.000	6.000	6.000	3.023	2.817	2.350	1.321
	10k		3.486	6.000	1.000	6.000	6.000	3.014	2.884	2.443	1.321
7	20	2.83±0.37	1.310	1.804	2.525	1.417	1.804	2.677	1.000	1.191	0.524
	50		1.558	2.114	2.606	1.650	2.046	3.365	1.170	1.582	0.832
	70		1.644	2.209	2.475	1.986	2.157	3.464	1.337	1.724	1.032
	100		1.724	2.349	2.730	1.905	2.130	3.475	1.393	1.678	1.064
	500		1.946	3.611	3.402	2.708	3.141	3.516	1.961	2.142	1.317
	1k		2.215	4.243	3.676	3.303	3.460	3.516	2.244	2.047	1.322
	5k		2.811	5.666	5.633	5.848	5.600	3.513	2.952	2.366	1.324
	10k		3.195	5.933	1.000	5.840	5.933	3.513	3.258	2.426	1.324
8	20	2.85±0.34	1.392	1.815	2.617	1.341	1.821	3.440	1.000	1.011	0.399
	50		1.606	1.994	2.673	1.672	2.022	3.564	1.000	1.342	1.031
	70		1.588	1.991	2.854	1.689	1.989	3.688	1.091	1.442	1.030
	100		1.667	2.003	2.802	1.725	1.983	3.872	1.224	1.722	1.000
	500		1.894	3.133	3.288	2.167	2.807	4.009	1.891	2.091	1.277
	1k		2.092	3.528	3.665	2.495	3.356	4.004	2.051	2.199	1.321
	5k		2.557	5.276	4.998	4.662	4.424	4.004	2.826	2.313	1.323
	10k		2.865	6.640	1.000	6.907	6.371	4.007	3.071	2.248	1.323
9	20	2.87±0.37	1.206	1.447	2.597	1.200	1.448	2.845	0.900	1.080	0.238
	50		1.314	1.608	2.944	1.322	1.632	3.556	1.000	1.170	1.091
	70		1.553	1.914	2.974	1.490	1.921	3.936	1.000	1.338	1.032
	100		1.563	1.911	2.992	1.645	1.987	4.151	1.040	1.697	1.064
	500		1.816	2.534	2.943	1.961	2.460	4.449	1.702	1.997	1.000
	1k		1.911	3.175	3.032	2.234	2.863	4.450	1.939	2.090	1.305
	5k		2.368	3.639	3.579	3.039	3.766	4.498	2.458	2.454	1.324
	10k		2.596	4.835	4.514	3.724	4.822	4.501	2.851	2.407	1.325

We show now the KL divergences obtained for the structures learned by the competitor algorithms. The learning was performed by using the same datasets previously used in order to contrast the results. Since the KL divergence clearly depends on the structure, first we analyze qualitatively the learned structures, and later their divergences. Given that the learned structures are more complex than the undirected graph, at least those learned by CSPC and density estimation algorithms, we do not use standard metrics such as the Hamming distance and F-measure for measuring the correctness of learned structures. Instead, we use a simple but

effective measure used by several recent works on density estimation algorithms: the *average of feature lengths* and the *number of features* [38,15,30]. These measures are statistics that permit us to determine, in an approximated way, qualitative aspects of the learned structures. For instance, for $n = 6$ variables, if the learned structure is the empty structure, the average feature length should be equal to 1 and the number of features should be equal to $2 \times 6 = 12$, namely, one feature per each single assigment. In contrast, if the learned structure is the fully structure, the average feature length should be equal to 6 and the number of features should be equal to $2^6 = 64$, namely, one feature per each complete assigment. Table 2 and 3 show the average feature lengths, and the number of features for the learned structures, respectively. In Table 2, the column named Underlying shows the average feature lengths of the underlying structure, which are: 2.80 ± 0.4 for 6 variables, 2.83 ± 0.37 for 7 variables, 2.85 ± 0.34 for 8 variables, and 2.87 ± 0.37 for 9 variables. Using the previous averages, we can approximately determine whether a learned structure is accurate, that is, when its average feature lengths is close to the average feature length of the underlying structure. In this manner, according to Table 2, we can see that the average feature lengths of CSPC and density estimation algorithms are more accurate than those obtained by knowledge discovery algorithms for large data ($|\mathcal{D}| \geq 500$), showing that a more flexible representation can capture better structures against a single graph. The best feature lengths for density estimation algorithms is obtained by GSSL and DTSL. The structures found by GSSL are more accurate than those obtained by CSPC when data is scarce ($|\mathcal{D}| < 500$). However, the performance of GSSL is degraded while the number of variables increases. This is due to the randomized approach used by GSSL which tends to increase the number of selected features in large domains, as shown in Table 3. For instance, the average number of features learned by GSSL ranges from 59.900 to 62.300 for 6 variables, increasing up to 508.900 for 9 variables, resulting in the biggest number of features. On the other hand, DTSL presents more accurate feature lengths than CSPC, especially for $|\mathcal{D}| \geq 500$, but for the remaining cases, $|\mathcal{D}| < 500$, CSPC is more accurate. Finally, L1 has similar average feature lengths across all the cases, this is because L1 only learns pairwise features.

As shown in Table 2, one important trend followed by CSPC and knowledge discovery algorithms is the difference between them in average feature lengths when the amount of data increases, starting from $|\mathcal{D}| \geq 500$. In these situations, CSPC obtains average feature lengths that are more similar to the underlying ones. In contrast, knowledge discovery algorithms only tend to obtain larger features which correspond to fully connected structures. For instance, in $n = 6$ and $|\mathcal{D}| \geq 5k$, GSMN learns fully connected structures because the feature lengths are equal to 6 and the numbers of features are $2^6 = 64$. This trend is because the independence tests are more reliable when data is sufficient, reducing statistical errors. However, this reliability in tests results in very different structural qualities due to the limitation of undirected graphs to encode context-specific independences, resulting in highly connected structures.

Table 3. Number of features in the structures learned by structure learning algorithms. Every cell represents the average over ten datasets with a fixed size.

Datasets			Knowledge discovery algorithms				Density estimation algorithms			
n	$\|\mathcal{D}\|$	CSPC	GSMN	IBMAP	PC	HHC	GSSL	DTSL	BLM	L1
6	20	19.800	17.400	22.400	12.200	17.800	59.500	5.900	6.600	31.400
	50	31.800	21.800	25.600	14.400	19.800	61.100	9.000	9.000	2.200
	70	35.100	23.200	28.000	16.000	20.000	62.500	10.100	9.100	52.600
	100	42.100	26.400	30.800	17.400	24.000	62.700	11.900	11.800	13.400
	500	70.900	46.000	44.800	29.200	44.800	62.500	20.900	19.800	147.000
	1000	90.200	51.200	59.200	39.800	52.800	62.700	32.500	24.100	186.000
	5000	111.500	64.000	64.000	64.000	64.000	62.500	51.300	29.000	192.000
	10000	78.200	64.000	12.000	64.000	64.000	62.300	54.700	32.600	192.000
7	20	22.000	19.000	28.200	14.200	19.400	62.300	6.100	7.500	44.000
	50	32.500	24.000	33.800	16.600	23.200	109.300	8.900	9.600	23.800
	70	40.600	29.200	37.200	18.200	25.600	121.700	10.900	10.500	31.600
	100	48.600	28.800	38.800	20.600	24.800	123.800	11.600	11.300	57.200
	500	78.400	55.200	64.400	31.200	44.800	126.600	23.300	21.800	152.000
	1000	115.200	76.000	62.000	41.000	53.600	126.600	35.600	27.100	219.200
	5000	203.000	96.000	121.600	66.000	96.000	126.500	70.300	34.800	264.800
	10000	229.200	121.600	14.000	66.200	121.600	126.500	89.100	36.900	266.000
8	20	27.400	23.200	35.000	16.200	24.800	120.667	6.700	8.100	40.200
	50	41.200	27.400	44.400	16.600	27.000	170.200	7.400	9.300	22.000
	70	39.300	27.800	44.000	17.800	25.800	178.800	8.700	10.000	23.800
	100	48.000	30.000	46.800	19.800	29.200	222.600	10.600	11.500	16.000
	500	82.500	51.200	75.200	31.800	45.200	254.600	24.900	21.700	90.400
	1000	111.700	77.600	103.200	38.800	63.600	254.200	31.200	28.400	243.400
	5000	228.600	140.800	164.800	105.200	132.000	254.300	78.500	39.900	309.400
	10000	336.600	150.400	16.000	130.000	145.600	254.500	99.100	42.300	311.800
9	20	23.900	22.000	41.600	18.000	22.400	112.333	4.400	9.200	28.800
	50	28.800	23.800	50.600	18.400	24.600	236.167	6.600	9.300	47.600
	70	40.400	26.600	52.000	18.600	28.600	250.400	8.000	10.000	35.400
	100	45.500	28.800	55.200	21.400	28.000	347.778	9.000	11.300	53.400
	500	77.300	47.400	75.200	31.200	47.800	478.900	21.800	19.200	18.000
	1000	93.600	61.800	91.200	35.600	58.600	482.300	28.700	25.000	173.400
	5000	211.700	134.000	150.000	67.600	127.200	507.200	63.700	46.500	358.800
	10000	310.300	184.800	294.400	110.800	188.000	508.900	101.700	47.500	396.000

It is worth noting that IBMAP-HC is the only algorithm that obtains the best feature lengths when data is scarce. In this situation, independence tests are not reliable, producing statistical errors. When many statistical tests are performed in an independent way, it is produced cascading errors. IBMAP-HC was designed for avoiding cascading errors, combining the outcomes of all the performed tests. Therefore, IBMAP-HC can obtain highly accurate structures when data is scarce. However, for $|\mathcal{D}| = 10k$ in all the domains except with $n = 9$, IBMAP-HC obtains strange results that does no follow the expected tendency of the remaining knowledge discovery algorithms. As shown in Table 2 and 3, IBMAP-HC obtains empty structures because the features lengths are equal to 1 and the numbers of features correspond to the number of single values of the variables. This is due to arithmetic underflow errors when large amounts of data are used in the current implementation of the algorithm, that were not detected by the authors, when large amounts of

data are used. For this reason, IBMAP-HC is not included in the experimentation of the next subsection, because the most of the datasets used are very large.

Table 4. Average lengths for the features that satisfy either the context x_w^0 or x_w^1 in the structures learned by all the algorithms. Only the features obtained by CSPC can satisfy both contexts.

Datasets		Underlying		CSPC		Knowledge discovery algorithms				Density estimation algorithms			
n	$\|\mathcal{D}\|$	x_w^0	x_w^1	x_w^0	x_w^1	GSMN	IBMAP	PC	HHC	GSSL	DTSL	BLM	L1
6	20	6	2	1.383	1.375	1.800	2.133	1.600	1.800	3.225	0.900	1.450	0.539
	50			1.629	1.751	1.900	2.266	1.700	1.900	3.425	1.300	2.216	0.300
	70			1.839	1.856	2.000	2.646	1.500	2.000	3.459	1.366	1.500	1.144
	100			1.708	1.880	2.100	2.566	1.300	2.200	3.475	1.500	2.483	1.000
	500			2.108	1.966	3.766	3.133	2.166	3.500	3.459	2.233	2.745	1.462
	1k			2.396	1.909	4.000	4.000	2.333	4.100	3.475	2.669	2.924	1.484
	5k			3.158	1.945	6.000	6.000	6.000	6.000	3.459	3.245	2.866	1.486
	10k			3.866	1.909	6.000	1.000	6.000	6.000	3.443	3.345	2.991	1.486
7	20	7	2	1.158	1.240	1.300	2.680	1.300	1.300	1.890	0.300	1.300	0.447
	50			1.553	1.557	1.800	2.856	1.500	1.900	3.340	0.950	1.300	0.848
	70			1.582	1.742	2.000	2.656	1.600	2.066	3.876	1.200	1.500	1.048
	100			1.521	1.863	2.000	2.863	1.100	2.000	3.911	1.150	1.500	1.096
	500			1.800	1.914	2.700	3.266	1.100	2.766	3.980	1.903	3.150	1.000
	1k			1.955	2.009	3.100	3.200	1.200	2.766	3.980	2.525	2.741	1.471
	5k			2.450	2.015	5.000	5.500	1.000	5.000	3.976	3.536	3.424	1.489
	10k			2.905	1.953	5.800	1.000	1.100	5.800	3.976	3.900	3.474	1.489
8	20	8	2	1.087	1.083	1.100	2.743	1.100	1.100	1.277	0.100	1.000	0.300
	50			1.373	1.418	1.500	2.886	1.400	1.500	2.686	0.500	1.000	1.047
	70			1.251	1.494	1.600	3.121	1.400	1.600	2.947	0.600	1.000	1.000
	100			1.283	1.684	1.800	2.968	1.100	1.800	3.739	0.850	1.000	1.000
	500			1.224	1.893	2.000	3.183	1.000	2.100	4.488	1.550	2.000	1.000
	1k			1.255	1.914	2.700	3.516	1.000	2.400	4.481	1.850	3.566	1.000
	5k			1.746	2.096	3.800	4.400	1.000	3.400	4.480	3.151	3.626	1.481
	10k			2.009	2.066	4.400	1.000	1.000	3.900	4.486	3.541	3.264	1.482
9	20	9	2	1.000	1.000	1.000	2.483	1.000	1.000	1.000	0.000	1.000	0.200
	50			1.000	1.088	1.100	3.076	1.100	1.100	1.664	0.100	1.000	1.000
	70			1.090	1.088	1.100	3.269	1.100	1.100	1.395	0.100	1.000	1.000
	100			1.317	1.350	1.400	3.220	1.200	1.500	2.720	0.400	1.000	1.048
	500			1.100	1.805	1.900	3.163	1.000	1.900	4.595	1.150	1.000	1.000
	1k			1.166	1.806	1.900	3.155	1.000	1.900	4.590	1.250	1.000	1.048
	50k			1.050	1.928	2.900	3.487	1.000	2.700	4.987	2.050	5.800	1.096
	10k			1.000	1.942	3.000	4.280	1.000	2.700	4.988	2.350	5.780	1.482

We present an additional result for measuring the quality of the learned structures. We measure the average feature length for two important subset of features: the features that satisfy the context x_w^0, and the features that satisfy the context x_w^1. These averages permit us to evalute if the context-specific independencies have been correctly encoded. Ideally, both averages would be different because the context-specific independences are only present on the context x_w^1. In this manner, the feature length for x_w^1 should be equal to 2; in contrast, the features length for x_w^0 should be equal to the number of variables. For instance, this trend can be seen in the set of features shown in Example 2.3. Table 4 shows the average feature lengths for all the algorithms. This table only shows the two average feature lengths

for CSPC because, for the remaining algorithms, both averages are the same. On the one hand, for knowledge discovery algorithms, the features generated from the learned graphs results in feature lengths that are the same for both contexts. On the other hand, for density estimations algorithms, the learned features are positive, resulting in an unique feature length for x_w^1. As a result, the structures learned by knowledge discovery and density estimation algorithms cannot capture the context-specific independences. On the other hand, as shown in Table 4, the average feature lengths for CSPC relatively follow the expected trend, obtaining feature lengths closed in 2 for x_w^1, and feature lengths greater than 2 for x_w^0. Moreover, for the most reliable situation, namely $n = 6$ for $|\mathcal{D}| = 10k$, CSPC obtains very accurate averages, that is average feature length equals 3.866 for x_w^0 and 1.909 for x_w^1. On the other hand, this trend cannot be seen in the remaining algorithms, resulting in structures that do not encode correctly the context-specific independences. Additionally, it is important to remark the trend that presents CSPC for the feature lengths for x_w^0, which tend to decrease for increasing number of variables, resulting in less accurate structures. This trend shows the problems caused by the sample size when using independence tests for determining the independences. As discussed in Section 4.3, this problem is exacerbated because CSPC performs independence tests by using partitions of data. However, it is worth remarking the improvements that CSPC obtains against PC, for instance, the average feature lengths for CSPC in $n \in \{7, 8\}$ are more accurate than those obtained by PC.

So far, we have been discussing some characteristics present in the structures learned by the different algorithms. In an approximate way, these characteristics permit us to analyze qualitatively the learned structures. For instance, using the average feature lengths shown in Table 4, one can approximately determine which learned structures can capture the context-specific independences. As we saw in the beginning of this section, KL divergences depend on the quality of the structures, permitting us to determine if a structure is accurate in a more precise way. In this manner, in the following we present the KL divergences computed from the structures learned by the algorithms. For clarity, the computed divergences are divided into two figures: Figure 5 shows the KL divergences for the knowledge discovery algorithms; and Figure 6 shows the KL divergences for density estimation algorithms. The KL divergences for CSPC are added in both figures to facilitate comparisons. Additionally, the KL divergences of the underlying structures are drawn in both figures for indicating the best KL divergence obtained in Figure 4. In Figure 5 and 6, the structures learned by CSPC reach the lowest divergences in most cases, showing the impact of encoding context-specific independences in the structure. For instance, for $|\mathcal{D}| = 10k$ in $n = 6$, the divergence obtained by CSPC is remarkably similar to the KL divergence of the underlying structure. As shown in Figure 5, knowledge discovery algorithms obtain the better divergences for $|\mathcal{D}| < 500$, but the quality of their learned models start to decline for large datasets just when the learned structures tend to be very connected. Moreover, as shown in Figure 6, density estimation algorithms obtain competitive divergences for large datasets, but these divergences

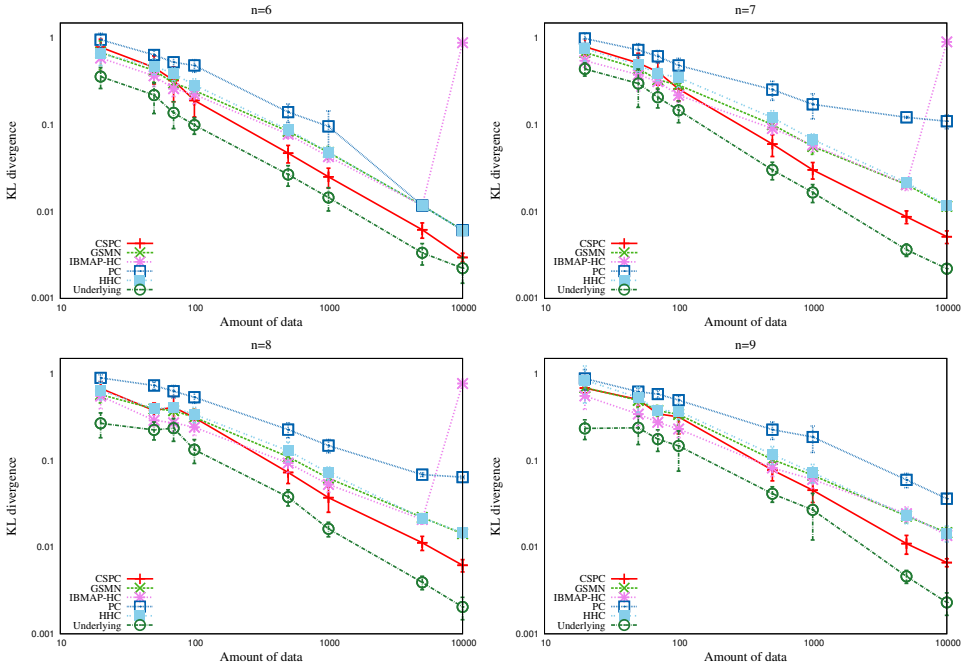


Fig. 5. KL divergences over increasing amounts of data for structures learned by: CSPC, GSMN, IBMAP-HC, PC, and HHC. For comparison reasons, the KL divergence of the underlying structure is shown. Every point represents the average and standard deviation over ten datasets with a fixed size. Each figure shows the KL divergences for different domain sizes: 6 (top left), 7 (top right), 8 (bottom left), and 9 (bottom right) variables. Lower values of KL divergence are better.

are not better than those obtained by CSPC. The improvements obtained by CSPC remark the balance between flexibility and interpretability offered by the structures learned by CSPC. On the one hand, the structures learned by CSPC are more flexible than the undirected graph learned by knowledge discovery, capturing the context-specific independences. On the other hand, the structures learned by CSPC are more interpretable than the set of features learned by density estimation, permitting CSPC to focus on the goal of knowledge discovery. The most competitive knowledge discovery algorithm is IBMAP-HC that obtains the better divergences when data is scarce; but as discussed previously, the divergences of IBMAP-HC are poor, closed in 1, for $|\mathcal{D}| = 10k$ due to the fact that empty structures are learned. The most competitive density estimation algorithm is GSSL that obtains competitive divergences but these are worst while the dimensionality of the domain increases. The worst divergences are obtained by L1 which range from 1.85 to 1.65 for 6 variables; 2.86 to 3.35 for 7 variables; 4.65 to 5.61 for 8 variables; and 4.98 to 7.58 for 9 variables.

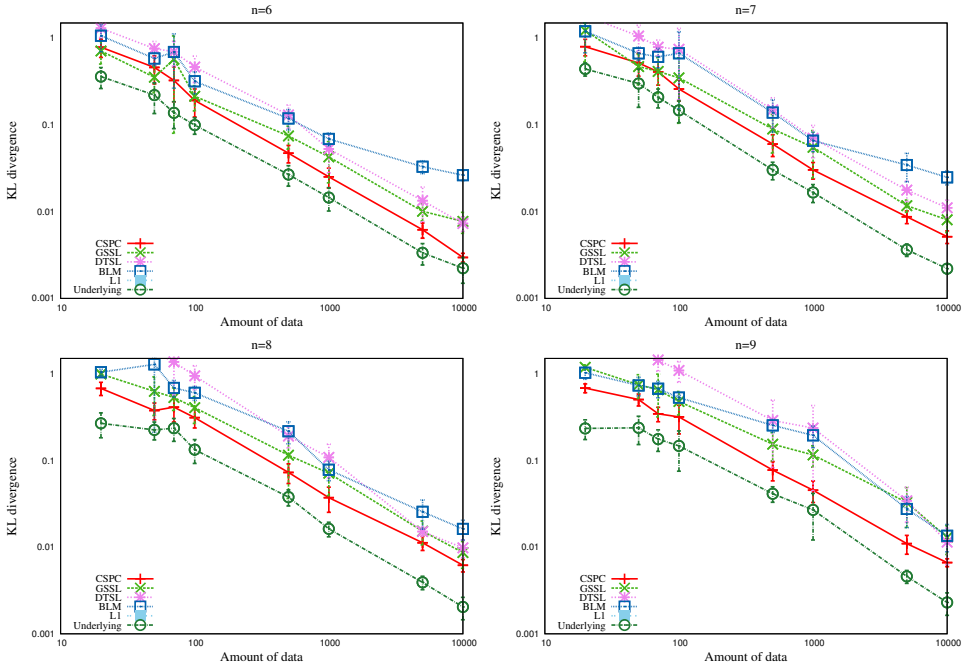


Fig. 6. KL divergences over increasing amounts of data for structures learned by: CSPC, GSSL, DTSL, BLM, and L1. For comparison reasons, the KL divergence of the underlying structure is shown. Every point represents the average and standard deviation over ten datasets with a fixed size. Each figure shows the KL divergences for different domain sizes: 6 (top left), 7 (top right), 8 (bottom left), and 9 (bottom right) variables. Lower values of KL divergence are better.

5.4. *Results for real-world datasets*

The evaluation on 18 real-world datasets consists in computing the CMLL over the test sets, using the learned Markov networks. The Markov networks are obtained by performing a weight learning from the learned structures. All the algorithms use the same regularization: Gaussian standard deviations of 0.1, 0.5, and 1, combined with L_1 priors of 1, 5, and 10. The best of these learned models is selected as the one that maximizes the pseudo-log-likelihood over the validation set. Table 5 shows the CMLL scores for all the algorithms on all 18 datasets. We use the CMLL scores presented in (Table 2 Ref. 15) and (Table 2 Ref. 13) in order to avoid the recomputation of the results of density estimation algorithms (GSSL, L1, DTSL, and BLM); but the CMLL scores of GSMN, HHC, PC and CSPC were computed. The blank cells in Table 5 represent CMLL scores that cannot be computed because the implementation of PC never finished, with waiting times of at least 3 weeks per dataset. As shown in Table 5, for learning the structure for several datasets, CSPC required to use the threshold K_{max} in order to reduce the time complexity; obtaining worst structures as cost. Using the CMLL scores in Table 5, Figure 7 shows how the CMLL scores obtained by the competitor algorithms are compared

in terms of CSPC, that is, the scores of CSPC are subtracted from the scores of each competitor algorithm. In this manner, for each dataset, a positive value represents that a competitor algorithm performs worse than CSPC, and a negative value represents that a competitor algorithm performs better than CSPC.

Table 5. Test set CMLL score summary. Lower scores represent better accuracy. The best CMLL score for each dataset is shown in bold.

Dataset	GSSL	L1	DTSL	BLM	GSMN	HHC	PC	CSPC	K_{max}
NLTCS	-5.050	-5.232	-5.025	-5.248	-5.012	-5.033	-5.028	**-4.975**	-
MSNBC	-5.947	-6.281	**-5.727**	-5.815	-5.777	-5.908		-5.991	-
Abalone	-7.440	-7.780	**-2.603**	-8.075	-5.766	-5.373	-10.245	-2.734	-
Wine	-22.846	-50.615	**-7.626**	-24.559	-14.298	-14.365	-15.819	-7.832	-
KDDCup	-2.071	-2.108	**-2.046**	-2.077	-2.102	-2.107		-2.068	4
Plants	**-9.854**	-10.739	-10.709	-10.445	-9.935	-10.381	-10.257	-9.875	6
Covertype	-18.942	-24.527	**-8.641**	-19.464	-31.692	-30.127	-34.194	-12.164	4
Audio	**-36.803**	-36.878	-37.484	-37.452	-37.872	-38.044		-37.320	3
Netflix	**-52.339**	-52.401	-53.342	-56.521	-53.890	-53.953		-52.703	5
Accidents	-18.180	**-16.543**	-16.957	-37.558	-22.974	-26.073		-22.730	-
Adult	-16.714	-48.982	**-10.315**	-28.673	-22.825	-22.972		-16.573	4
Retail	-10.547	-10.534	-10.447	-10.620	-10.477	-10.513	-10.560	**-10.428**	6
Pumsb Star	-17.245	**-13.905**	-19.508	-133.155	-19.087	-21.439		-14.279	4
DNA	-81.034	**-69.035**	-69.197	-99.560	-69.340	-69.629	-69.928	-69.475	-
MSWeb	**-8.819**	-8.959	-16.201	-8.848	-9.276	-9.318		-9.230	4
WebKB	-144.206	-143.290	-148.234	-164.844	-147.570	-148.528		**-142.470**	3
BBC	-242.424	-239.642	-251.036	-265.486	-247.697	-249.763	-257.497	**-239.319**	2
Ad	-14.848	-15.393	-16.751	-45.638	**-12.908**	-20.865	-40.122	-14.837	4

As shown in Table 5, CSPC achieves the best overall performance on 4 of the 18 datasets, being the only algorithm focused on knowledge discovery that achieves this performance. For instance GSMN only achieves the best overall performance on the Ad domain. Comparing CSPC against all the knowledge discovery algorithms, it beats them on 15/18 datasets. As shown in Figure 7, the CMLL scores obtained by CSPC are substantially better than those obtained by knowledge discovery algorithms on the Wine, Covertype, Adult, and BBC domains. Furthermore, the CMLL values of CSPC outperforms PC on 9 of the 9 datasets succesfully computed. On the other hand, comparing CSPC against all the density estimation algorithms, it cannot beat them; CSPC only beats them on 5/18 datasets. The best density estimation algorithm is DTSL that produces the best overall performance on 6/18 datasets, following by GSSL on 4/18 datasets. Comparing the CMLL scores obtained by CSPC against each density estimation algorithm, CSPC achieve better score on 6/18 datasets against GSSL, on 6/18 datasets against L1, on 8/18 datasets against DTSL, and on 16/18 datasets against BLM.

Using the CMLL values, we compare CSPC against the competitor algorithms by using a Wilcoxon signed-rank test with continuity correction in the normal approximation for the p-value. For computing the Wilcoxon signed-rank test, we defined 18 paired samples by using the CMLL value of CSPC and every algorithm, for each dataset. Using the Wilcoxon signed-rank test, CSPC significantly outperforms all

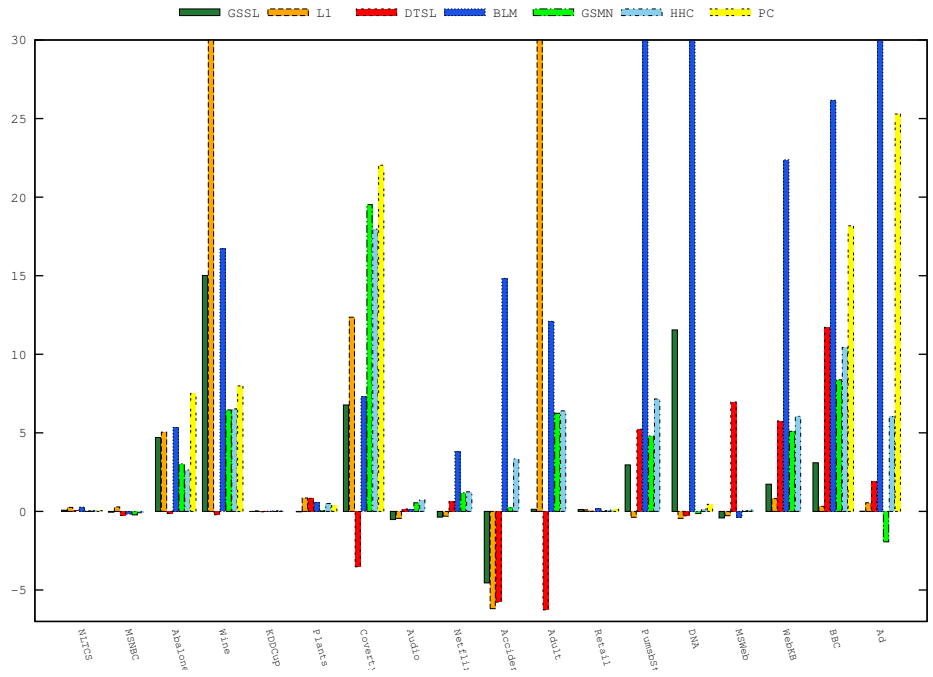


Fig. 7. CMLL scores relative to CSPC. Positive values represent that the CMLL of a competitor algorithm is worse than the CMLL of CSPC. Negative values represent that the CMLL of a competitor algorithm is better than the CMLL of CSPC.

the knowledge discovery algorithms: GSMN at the 0.002 significance level, HHC at the 0.00001 significance level, and PC at the 0.001 significance level. On the other hand, comparing with density estimation algorithms, CSPC only outperforms BLM at the 0.0001 significant level. CSPC obtains equivalently accuracy to GSSL and does not present significant differences with respect to DTSL and L1.

6. Conclusions and future work

In this work, we have been discussing an approach to overcome the limitation of knowledge discovery algorithms for learning Markov network structures that can capture context-specific independences. This limitation mainly arises from the representation of the structure used. Knowledge discovery algorithms use a single graph for representing the structure, which is a rigid representation that cannot capture context-specific independences. Thus, our approach proposes an alternative representation called *canonical models*, a particular class of CSI models. Canonical models are flexible, permiting us to encode context-specific independences; and are also interpretable, permiting us to represent a canonical model as a collection of graphs. These aspects allow the straightforward adaptation of well-known ideas used by knowledge discovery algorithms for learning canonical models. In this manner, we

present the CSPC algorithm, a simple knowledge discovery algorithm that learns structures by using canonical models as the representation. CSPC uses a combination between the pairwise and local Markov properties as a criterion for encoding context-specific independences in each canonical graph of the canonical model. The context-specific independences are elicited from data by using independence statistical tests. We evaluated our approach against several state-of-the-art learning algorithms on synthetic and real-world datasets. On synthetic datasets, CSPC learned the most accurate structures when data is large. On real-world datasets, CSPC learned structures that were competitive for inference tasks against density estimation algorithms, showing high performances against traditional knowledge discovery algorithms. However, CSPC presents two important downsides: first, it has a high computational complexity because it performs a structure learning over a collection of graphs; second, the elicitation of independences has lack of robustness because independent statistical tests are performed on partitions of data.

The directions of future work are focused on improving the downsides of CSPC, mainly reducing the time complexity and improving the quality of the learned structures. For reducing the time complexity, we discuss two stategies that basically consist in reducing the number of test performed. The first strategy is to define an alternative set of canonical assignments. In this work, we suggested a very simple set of canonical assignments based on the data-driven approach. As a future work, we investigate alternative sets of canonical assignments. For instance, a set could be obtained by clustering data into the most likely canonical assigments by using *the generalized Hamming distance* or *the generalized value different metric* (Section 4.3 Ref.13) as a way of measuring the distances between canonical assignment and then using the EM algorithm [50] or K-means to obtain the clusters. The second strategy is to explote common structures between canonical graphs in order to avoid the computation of redundant statistical tests that could be inferred from other canonical graphs previously learned. On the other hand, we investigate alternative methods for eliciting independences from data, in order to evaluate its impacts in the quality of the learned structures. For instance, some interesting methods include: logistic regression [37] and conditional entropy [51]. Finally, an interesting future work includes the exploration of other strategies for learning canonical graphs. In a general perspective, CSPC can be seen as an adaptation of the PC algorithm. In our experimentation, PC was the knowledge discovery algorithm with worst performance. However, CSPC, an adaptation of PC, obtained improvements against all the remaining competitor algorithms. For this fact, we investigate alternatives adaptations and strategies: the GSMN and IBMAP-HC on the side of knowledge discovery algorithms [52,19], and the L1 and GSSL on the side of density estimation algorithms [37,15].

Appendix A. Properties of independences

The independence relation $I(\cdot,\cdot \mid \cdot)$ has the following properties, where h denotes an arbitrary function on the variables X_A (Chapter 3.1 Ref. 5):

Proposition Appendix A.1. *if* $I(X_A, X_B \mid X_U)$*, then* $I(X_B, X_A \mid X_U)$*;*

Proposition Appendix A.2. *if* $I(X_A, Y_B \mid X_U)$ *and* $X_W = h(X_A)$*, then* $I(X_W, X_B \mid X_U)$*;*

Proposition Appendix A.3. *if* $I(X_A, X_B \mid X_U)$ *and* $X_W = h(X_A)$*, then* $I(X_A, X_B \mid X_U, X_W)$*;*

Proposition Appendix A.4. *if* $I(X_A, X_B \mid X_U)$ *and* $I(X_A, X_W \mid X_B, X_U)$ *then* $I(X_A, X_{W,B} \mid X_U)$*.*

The following property only holds for positive distributions:

Proposition Appendix A.5. *if* $I(X_A, X_B \mid X_U)$ *and* $I(X_A, X_U \mid X_B)$*, then* $I(X_A, X_{B,U} \mid \emptyset)$*.*

Graphs are an algebraic structure called *graphoid*. This structure satisfies all the properties where A, B, U and W are disjoint subsets, and $X_W = h(X_U)$ is defined as $W \subseteq U$ [53,5].

Appendix B. Synthetic datasets

This section presents a detailed explaination of how to generate the synthetic datasets used in our empirical evaluation in Section 5.3. The synthetic Markov networks are defined over $n = |X|$ binary-valued random variables. The underlying structure of these models consists in two instantiated graphs, one for each possible value of the "flag" variable X_w: a star graph $G(x_w^1)$ whose center is x_w^1, and a fully connected graph $G(x_w^0)$. This underlying structure encodes the context-specific independences shown in Example 3.1. In this way, the underlying structure encodes $\frac{(n-1)(n-2)}{2}$ context-specific independences of the form $I(X_a, X_b \mid x_w^1)$, for all pairs $a, b \in V \setminus \{w\}$. From both instantiated graphs and all value $x \in \mathrm{Val}(V)$, a set of features is generated as shown in Algorithm 3. This set of features are divided into two subsets: *i)* the set of pairwise features which are obtained from $G(x_w^1)$ whose form is $f(x_a, x_w^1)$ for all $a \in V \setminus \{w\}$; and *ii)* the set of triplet features which are obtained from $G(x_w^0)$ whose form is $f(x_a, x_b, x_w^0)$ for all pair $a, b \in V \setminus \{w\}$. One example of these set of features is shown in Example 2.3. From these features, we generated 10 different Markov networks, varying their weights. The weights were selected to satisfy the log-odds ratio in order to force strong dependencies [56,57,19]. Therefore, the weights of the pairwise features were forced to satisfy the log-odds ratio as follows:

$$\varepsilon = \log\left(\frac{w_{00} f(x_a^0, x_w^0) w_{11} f(x_a^1, x_w^1)}{w_{01} f(x_a^0, x_w^1) w_{10} f(x_a^1, x_w^0)}\right),$$

where $w_{00} = w_{11}$ and $w_{01} = w_{10}$. In our experiments we set $\varepsilon = 1.0$ which guarantees strong dependencies. Since this ratio has 2 unknowns, we choose w_{01} sampled from $\mathcal{N}(0.5; 0.001)$, and solving for w_{00}. On the other hand, the weights of the triplet features were obtained from the weight of the pairwise features as follows: $w_{000} = w_{00} \times w_{00} = w_{110}$, and $w_{010} = w_{00} \times w_{10} = w_{100}$, and $w_{010} = w_{00} \times w_{10}$. The datasets were generated by sampling from the log-linear models using Rao-Blackwellized Gibbs sampler [h] with 10 chains, 100 burn-in and 1000 sampling.

Appendix C. AD-Tree

This section gives a brief description of AD-Tree. We refer the interested reader to [45]. The AD-Tree is an efficient representation for caching sufficient statistics from data. The use of AD-Tree for constructing contingency tables has shown dramatical improvements in time complexity for structure learning and feature induction algorithms [54,55]. Roughly speaking, AD-Tree is a sparse tree structure whose nodes represent assignments of a single variable $x_a^i \in \text{Val}(a)$, and branches represent complete assignments $x^i \in \text{Val}(V)$. A node saves the sufficient statistic for the partial assignment that results of picking all the nodes towards the root. For instance, the root saves $|\mathcal{D}|$, and each one of its children, that is x_a^i, saves $|\mathcal{D}[x_a^i]|$ for all $a \in V$. Therefore, a contingency table can be constructed by using the sufficient statistics of a subtree. For not incurring in an intractable memory demand, [45] presents several *cache reductions*. One of them, called *Leaf-Lists* (Section 5 Ref. 45), prunes any subtree when the number of datapoints used for counting is fewer than a given *pruning threshold*. In these cases, a Leaf-List is used to save the indexes of such datapoints to quickly access them for computing on demand these sufficient statistics at some later time.

References

1. J. M. Hammersley and P. Clifford. Markov fields on finite graphs and lattices. 1971.
2. J Van Haaren, J Davis, GAM Lappenschaar, and AJ Hommersom. Exploring disease interactions using Markov networks. 2013.
3. Karen Sachs, Omar Perez, Dana Pe'er, Douglas A Lauffenburger, and Garry P Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308(5721):523–529, 2005.
4. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1re edition, 1988.
5. Steffen L Lauritzen. *Graphical models*. Oxford University Press, 1996.
6. Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.
7. Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2): 1–305, 2008.

[h]This version of Gibbs sampler is available in the open-source Libra toolkit http://libra.cs.uoregon.edu/

8. D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, 2009.
9. Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012.
10. Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1): 273–302, 1996.
11. S. Della Pietra, V. J. Della Pietra, and J. D. Lafferty. Inducing Features of Random Fields. *IEEE Trans. PAMI.*, 19(4):380–393, 1997.
12. Andrew McCallum. Efficiently inducing features of conditional random fields. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 403–410. Morgan Kaufmann Publishers Inc., 2002.
13. Jesse Davis and Pedro Domingos. Bottom-up learning of Markov network structure. In *Proceedings of the 27th International Conference on Machine Learning*, pages 271–280, 2010.
14. Vibhav Gogate, William Webb, and Pedro Domingos. Learning efficient Markov networks. In *Advances in Neural Information Processing Systems*, pages 748–756, 2010.
15. Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the Twenty-Sixth National Conference on Artificial Intelligence. AAAI Press*, 2012.
16. Daniel Lowd and Jesse Davis. Improving Markov network structure learning using decision trees. *Journal of Machine Learning Research*, 15:501–532, 2014.
17. Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9(1):62–72, 1991.
18. Facundo Bromberg, Dimitris Margaritis, Vasant Honavar, et al. Efficient Markov network structure discovery using independence tests. *Journal of Artificial Intelligence Research*, 35(2):449, 2009b.
19. Federico Schlüter, Facundo Bromberg, and Alejandro Edera. The IBMAP approach for Markov network structure learning. *Annals of Mathematics and Artificial Intelligence*, pages 1–27, 2014. ISSN 1012-2443. doi: 10.1007/s10472-014-9419-5.
20. Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.
21. Yutian Chen and Max Welling. Bayesian structure learning for Markov random fields with a spike and slab prior. 2012.
22. C. Boutilier, N. Friedman, M. Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth international conference on Uncertainty in artificial intelligence*, pages 115–123. Morgan Kaufmann Publishers Inc., 1996.
23. Jozef L Teugels and Johan Van Horebeek. Generalized graphical models for discrete data. *Statistics & probability letters*, 38(1):41–47, 1998.
24. P Svante Eriksen. *Context specific interaction models*. Citeseer, 1999.
25. Søren Højsgaard. Yggdrasil: a statistical package for learning split models. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 274–281. Morgan Kaufmann Publishers Inc., 2000.
26. Søren Højsgaard. Statistical inference in context specific interaction models for contingency tables. *Scandinavian journal of statistics*, 31(1):143–158, 2004.
27. Arthur Fridman. Mixed Markov models. *Proceedings of the National Academy of Sciences*, 100(14):8092–8096, 2003.
28. D. Poole and N. L. Zhang. Exploiting contextual independence in probabilistic inference. *J. Artif. Intell. Res. (JAIR)*, 18:263–313, 2003.
29. S. I. Lee, V. Ganapathi, and D. Koller. Efficient structure learning of Markov networks using L1-regularization. In *Neural Information Processing Systems*. Citeseer, 2006.

30. Daniel Lowd and Amirmohammad Rooshenas. Learning Markov networks with arithmetic circuits. *The Journal of Machine Learning Research*, 31:406–414, 2013.
31. Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *Proceedings of The 30th International Conference on Machine Learning*, volume 28, 2013.
32. D. Fierens. Context-specific independence in directed relational probabilistic models and its influence on the efficiency of Gibbs sampling. In *European Conference on Artificial Intelligence*, pages 243–248, 2010.
33. Walter R Gilks, Sylvia Richardson, and David J Spiegelhalter. Introducing Markov chain monte carlo. In *Markov chain Monte Carlo in practice*, pages 1–19. Springer, 1996.
34. Alex Kulesza and Fernando Pereira. Structured learning with approximate inference. In *Advances in neural information processing systems*, pages 785–792, 2007.
35. Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302, 1986.
36. Daniel Lowd. Closed-form learning of Markov networks from dependency networks. 2012.
37. Pradeep Ravikumar, Martin J. Wainwright, and John D. Lafferty. High-dimensional Ising model selection using L1-regularized logistic regression. *Annals of Statistics*, 38: 1287–1319, 2010. doi: 10.1214/09-AOS691.
38. Daniel Lowd and Jesse Davis. Learning Markov network structure with decision trees. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 334–343. IEEE, 2010.
39. Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of statistics*, pages 1165–1188, 2001.
40. Eliot Brenner and David Sontag. Sparsityboost: A new scoring function for learning Bayesian network structure. 2013.
41. MH DeGroot and MJ Schervish. Probability and statistics-international edition, 2001.
42. Constantin F Aliferis, Alexander Statnikov, Ioannis Tsamardinos, Subramani Mani, and Xenofon D Koutsoukos. Local causal and Markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation. *The Journal of Machine Learning Research*, 11:171–234, 2010.
43. Alejandro Edera, Facundo Bromberg, and Federico Schlüter. Markov random fields factorization with context-specific independences. *arXiv preprint arXiv:1306.2295*, 2013.
44. Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of The ACM*, 16:575–577, 1973. doi: 10.1145/362342.362367.
45. Andrew Moore and Mary Soon Lee. Cached Sufficient Statistics for Efficient Machine Learning with Large Datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998.
46. Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.
47. Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991. ISBN 0-471-06259-6.
48. Markus Kalisch and Peter Bühlmann. Estimating high-dimensional directed acyclic graphs with the PC-algorithm. *The Journal of Machine Learning Research*, 8:613–636, 2007.
49. Dimitris Margaritis and Sebastian Thrun. Bayesian network induction via local neighborhoods. Technical report, DTIC Document, 2000.
50. Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from

incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

51. Pieter Abbeel, Daphne Koller, and Andrew Y Ng. Learning factor graphs in polynomial time and sample complexity. *The Journal of Machine Learning Research*, 7: 1743–1788, 2006.
52. F. Bromberg, D. Margaritis, and Honavar V. Efficient Markov Network Structure Discovery Using Independence Tests. *JAIR*, 35:449–485, July 2009a.
53. J. Pearl and A. Paz. GRAPHOIDS : A graph based logic for reasonning ab relevance relations. Technical Report 850038 (R-53-L), Cognitive Systems Laboratory, University of California, Los Angeles, 1985.
54. Shunkai Fu and Michel C Desmarais. Fast Markov blanket discovery algorithm via local learning within single pass. In *Advances in Artificial Intelligence*, pages 96–107. Springer, 2008.
55. Jo-Anne Ting, Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal. Efficient learning and feature selection in high-dimensional regression. *Neural computation*, 22(4): 831–886, 2010.
56. Parichey Gandhi, Facundo Bromberg, and Dimitris Margaritis. Learning Markov Network Structure using Few Independence Tests. In *SIAM International Conference on Data Mining*, pages 680–691, 2008.
57. D. Margaritis and F. Bromberg. Efficient Markov Network Discovery Using Particle Filter. *Comp. Intel.*, 25(4):367–394, 2009.