The Complexity of Plate Probabilistic Models

Fabio G. Cozman and Denis D. Mauá

Universidade de São Paulo, São Paulo, Brazil
{fgcozman,denis.maua}@usp.br

Abstract. Plate-based probabilistic models combine a few relational constructs with Bayesian networks, so as to allow one to specify large and repetitive probabilistic networks in a compact and intuitive manner. In this paper we investigate the combined, data and domain complexity of plate models, showing that they range from polynomial to #P-complete to #EXP-complete.

1 Introduction

The desire to tackle complex decision scenarios, where many variables interact and vast quantities of data are collected, has produced various modeling languages based on graphs. For instance, Bayesian and Markov networks offer visually pleasant tools by which one can represent interacting variables [8, 14, 18]. In practice, several scenarios display repetitive patterns that can be best encoded using relations, domains, and individuals. To address this reality, formalisms have been proposed that combine features of Bayesian and Markov networks with relational languages; for instance, plates [16], Markov logic networks [7], relational Bayesian networks [6].

Plate-based probabilistic models are possibly the simplest and most successfull of these "probabilistic relational models". By capturing symmetries in the model, plates make communication and modelling much more efficient. Plates are simple to draw, easy to understand, and quite powerful in what they can represent. Plate models have been extensively used in statistical practice [15] since they were introduced with the BUGS project [13, 16]. In machine learning, they have been used (often informally) to convey several models since their first appearance [10]. One example is the smoothed Latent Dirichlet Allocation (sLDA) model [9], usually represented with plates as in Figure 1 (explained later).

In this paper we present results on the inferential complexity of plate models, a topic that seems not to have received due attention. There are (at least) three kinds of complexity results that are of interest in this context [11]: first, the *combined* complexity of inferences, where model, evidence and domain are given as input; second, the *data* complexity, where query, evidence and domain are given as input (and we fix a model); finally, the *domain* complexity, where only the domain is given as input (and model, query and evidence are fixed).

We start with the original plate models (Section 2), where nodes in a plate can only have children inside the same plate, and we investigate their inferential complexity (Section 3). We first look into the combined complexity, which we

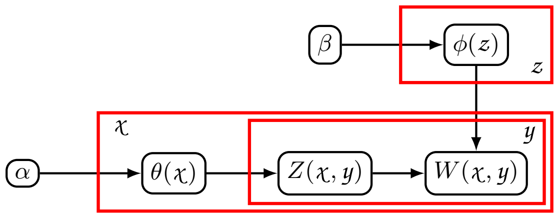


Fig. 1. Smoothed Latent Dirichlet Allocation, a plate model.

show to be #P-complete; this is a surprising result given that plates can generate grounded models that are exponentially large on their description size (hence a naive approach to inference would take exponential time). We show data complexity also to be #P-complete, and domain complexity to be constant. We then move to more general plate models where a node can have children in any plate (Section 4). Here it is necessary to allow "aggregation functions" that specify probability values. We focus on the simplest combination functions, and show that combined complexity leads to #EXP-complete inference, while data complexity leads to #P-complete inference.

2 Plates

In this section we define plate models and some related concepts; because the literature does not have a standard formalization, we start from somewhat basic notions. We only deal with finite spaces, so every variable has finitely many values, leaving for the future a study of continuous variables [5].

A Bayesian network consists of a directed acyclic graph where each one of its n nodes is a random variable X_i, and where the following Markov condition holds: any X_i is independent of its nondescendants given its parents. Additionally, a Bayesian network contains a set of conditional probability distributions: for each X_i, we have $\mathbb{P}(X_i = x_i | \text{pa}(X_i) = \pi_i)$ for all values x_i and π_i ($\text{pa}(X_i)$ denotes the parents of X_i in the graph). The Markov condition implies the factorization $\mathbb{P}(X_1 = x_1, \dots, X_n = x_n) = \prod_{i=1}^{n} \mathbb{P}(X_i = x_i | \text{pa}(X_i) = \pi_i)$, where π_i is the projection of $x_1, \dots, x_n$ on $\text{pa}(X_i)$ [18]. As a simple example [14], assume we want to infer the performance of John (a student), as given by his grade; the performance is affected by John's intelligence and by the course's difficulty. This is graphically represented as $\boxed{\mathsf{Difficulty}} \rightarrow \boxed{\mathsf{Grade}} \leftarrow \boxed{\mathsf{Intelligence}}$. Now we may be interested in a set of students; we use $\mathsf{Grade}(c_i, s_j)$ to denote the grade of student s_j in course c_i; similarly, $\mathsf{Difficulty}(c_i)$ denotes the difficult of the ith course and $\mathsf{Intelligence}(s_j)$ the intelligence of the jth student. A Bayesian network for two students and two courses is shown in Figure 2. The very same model can be described using plates as in Figure 3 [13].

To define plate models more formally, we adopt the following concepts, mixing definitions by Koller and Friedman [14, Chap. 6] and terminology by Poole [19]. We first take any *logical variable (logvar)* to be typed, ranging over a finite

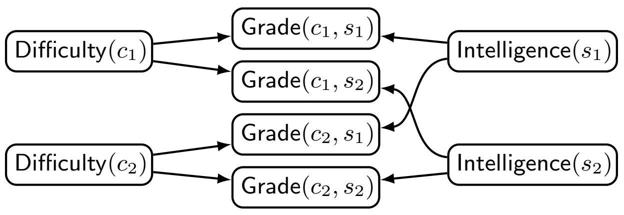


Fig. 2. Bayesian network with repetitive structure.

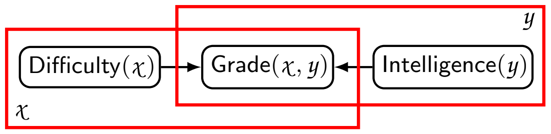


Fig. 3. Plate model for the network in Figure 2.

set, called its *domain*. Note: in the language of plates each logical variable is uniquely attached to a symbol, from which we can deduce its domain. A *parameterized variable (parvariable)* X is a function that yields a random variable $X(\chi_1, \ldots, \chi_k)$ for each instantiation of the logvars $\chi_1, \ldots, \chi_k$ (all typed). All the random variables in the image of a parvariable take values in the same space (hence we can unambiguously talk about the space of values of parvariables). Denote by $\mathsf{logvar}(X)$ the tuple of logvars associated with parvariable X. A *plate graph* consists of a directed acyclic graph where each node X_i is a parvariable, and such that for any X_i and $Y \in \mathrm{pa}(X_i)$, $\mathsf{logvar}(Y) \subseteq \mathsf{logvar}(X_i)$.[1] If two nodes share a logvar they are said to belong to the same plate. Each plate is usually indicated by a rectangle, containing the parvariables that belong to the plate, and information about the logvars in the plate. A *plate model* consists of a plate graph and, additionally, a *template conditional probability distribution* for each parvariable X_i, that yields $\mathbb{P}(X_i = x_i | \mathrm{pa}(X_i) = \pi_i)$ for all possible values x_i and π_i. Template distributions are often called *parfactors* [19]; the latter word is also used to refer to arbitrary functions over parvariables.

To specify the semantics of plate models, we need an additional piece of notation. Suppose we have an ordered set of logvars $\overrightarrow{\chi} = (\chi_1, \ldots, \chi_k)$ and one of its possible instantiations, $\overrightarrow{a} = (a_1, \ldots, a_k)$; then, given another ordered set of logvars, $\overrightarrow{\chi}'$, denote by $\overrightarrow{\chi}'[\overrightarrow{\chi}/\overrightarrow{a}]$ the ordered set where, for each possible i, χ_i is replaced by a_i.

Concerning semantics, a plate model represents a (possibly large) Bayesian network, constructed as follows. First, for each parvariable X, generate all instantiations of $\mathsf{logvar}(X)$ (as they range over their domains), and for each instantiation $\overrightarrow{a}$ create a node $X(\overrightarrow{a})$. Second, for each parvariable X, generate again all instantiations of its logvars; for each instantiation $\overrightarrow{a} = (a_1, \ldots, a_k)$ and for

[1] Note that the definition of plate model in Ref. [14] does not require acyclicity, but this seems to be a necessary requirement in all the relevant literature.

each Y in $\mathrm{pa}(X)$, add an edge from $Y(\mathsf{logvar}(Y)[\mathsf{logvar}(X)/\overrightarrow{a}])$ to $X(\overrightarrow{a})$. Third, associate each grounded parvariable with the corresponding template conditional distribution.

The graph in Figure 2 is the "grounding" of the plates model in Figure 3: $\mathcal{X}$ runs over a set containing courses c_1 and c_2, and y runs over a set containing students s_1 and s_2. To complete specification of the grounded Bayesian network, we must have template probabilities. Suppose, for instance, that all variables are binary, and $\mathbb{P}(\mathsf{Difficulty}(\mathcal{X}) = 0) = 1/3$. Then we have both $\mathbb{P}(\mathsf{Difficulty}(c_1) = 0) = 1/3$ and $\mathbb{P}(\mathsf{Difficulty}(c_2) = 0) = 1/3$ in the grounded Bayesian network.

The plates we have defined so far are the "classic" plates that first appeared with the BUGS system [16]. One of their limitations is that a node only has children inside the same plate (we assume there is a "base plate" containing all nodes). In practice plate models go beyond this, by letting a node to have children in other plates. See for instance the sLDA model in Figure 1: here $\phi(z)$ has a child $W(\mathcal{X}, y)$. The semantics of these enhanced plates is discussed in Section 4.

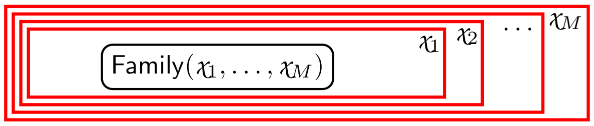
3 The complexity of "classic" plate models

We now examine the complexity of inference with classic plate models. In this context, an inference typically refers to the calculation of a conditional probability $\mathbb{P}(\mathbf{Q}|\mathbf{E})$, where $\mathbf{Q}$ and $\mathbf{E}$ are sets of assignments $\{X(\overrightarrow{a}) = x\}$ (understood as conjunctions). We assume that $\mathbf{Q}$ and $\mathbf{E}$ are not contradictory (i.e., that they do not contain different assignments to the same variable) and that $\mathbb{P}(\mathbf{E}) > 0$ so that the inference is well-defined. Note that checking whether this last assumption holds is again an inference problem. As an example of inference, consider computing $\mathbb{P}(\mathsf{Difficulty}(c_1)=1|\mathsf{Intelligence}(s_2)=1, \mathsf{Grade}(c_1, s_2)=0, \mathsf{Grade}(c_2, s_2)=1)$. The set $\mathbf{Q}$ is the *query* and the set $\mathbf{E}$ is the *evidence*.

To discuss the complexity of inferences, we need a few concepts. The complexity class $\#\mathsf{P}$ is the class of integer-valued functions computed by counting Turing machines in polynomial time; a counting Turing machine is a standard nondeterministic Turing machine that prints in binary notation, on a separate tape, the number of accepting computations induced by the input [21]. Thus, $\#\mathsf{P}$ contains the counting versions of NP-complete problems. If a problem is $\#\mathsf{P}$-hard and can be solved with one call to a $\#\mathsf{P}$ oracle and with polynomial-time computations (i.e., if it belongs to $\mathsf{FP}^{\#P[1]}$), it is said to be $\#\mathsf{P}[1]$-equivalent [12]. Roth [20] showed that inference in Bayesian networks is $\#\mathsf{P}$-hard. Because inference can be reduced to counting solutions of NP-complete problems followed by a normalization step [8], the problem is $\#\mathsf{P}[1]$-equivalent [12]. Note that one cannot assert $\#\mathsf{P}$-completeness of such inferences, as $\#\mathsf{P}$ produces integers and inference produces rationals. Later we will need the class $\#\mathsf{EXP}$; that is, the class of functions computed by counting Turing machines in exponential time (the number of accepting paths may have exponential size) [17]. A problem is $\#\mathsf{EXP}[1]$-equivalent if it is $\#\mathsf{EXP}$-hard and can be solved with one call to a $\#\mathsf{EXP}$ oracle whose result x is processed through a function $h(x)$ that requires polynomial time with respect to the size of x.

A plate model can specify any Bayesian network: just define all parvariables without logvars. Hence the calculation of $\mathbb{P}(\mathbf{Q}|\mathbf{E})$ for plate models is #P-hard. Now consider a generic plate model, with as many logvars (as many plates) as needed. For each plate, a domain must be specified; suppose the domain is given as a list of elements. This assumption means, in essence, that we are not contemplating compact ways to specify the domain (for instance, one might just give a number N in binary notation, with the understanding that the domain is $\{1, 2, \ldots, N\}$; we do not deal with this case here).

Even an explicit specification for domains can lead to exponentially large grounded Bayesian networks. By taking M nested plates, each with a domain consisting of N elements, a single parvariable can specify N^M random variables. Here is a simple example. Suppose we are interested in groups of individuals; for instance, we wish to model the parvariable Family that indicates whether or not M individuals are related. We draw:



Assuming we have N individuals, the grounding of this plate model has N^M grounded nodes. In this example inference is trivial as all random variables are independent. If we instead had several parvariables connected in complex ways, we might face a very dense, exponentially large Bayesian network. This might suggest that calculation of probability values would take us to #EXP in the worst case. The nice fact about plate models is that inference has the same complexity of Bayesian networks, despite the possibly exponential size of grounded Bayesian networks:

Theorem 1 *Inference in plate models is* #P[1]*-equivalent.*

Proof. Clearly, the problem is #P-hard, so it suffices to show it can be computed with one call to #P plus polynomial-time post-processing. We achieve this by showing that, in any given inference, only a polynomially-large fragment of the (possibly exponentially-large) grounded Bayesian network is necessary, and inference in this fragment is known to be #P[1]-equivalent.

We are to compute $\mathbb{P}(\mathbf{Q}|\mathbf{E})$. Suppose we produce the complete grounded Bayesian network. All nodes that appear in $\mathbf{Q}$ and in $\mathbf{E}$ appear in this network. The fragment consisting of these grounded nodes and their (grounded) ancestors is the sub-network that contains all information needed for inference; other grounded nodes can be discarded [18]. This fragment may contain disconnected sub-networks; the only sub-network that matters for the computation of $\mathbb{P}(\mathbf{Q}|\mathbf{E})$ is the sub-network that contains the grounded parvariables in $\mathbf{Q}$. Refer to this fragment of the original grounded as the *requisite* network.

Suppose there are m grounded nodes in $\mathbf{Q} \cup \mathbf{E}$. For any grounded node W in this set, the number of ancestors of W is less than the number of (parvariable) nodes in the plate model. For instance, in our last example, the ancestors of a grounding of Grade contain exactly one grounding of Difficulty and one grounding of Intelligence. Now if there are n parvariables in the plate model, there are at most mn grounded nodes in the requisite network. By running inference in this requisite network, we obtain the desired result. □

This result focuses on the *combined complexity* of plate models; that is, the complexity of inference when plate model, query, evidence, and domains are given as input. However, in practice we may be more interested in the complexity of inferences when the model is fixed. This is justified when we expect the model to be small but the data to be abundant. For instance, we may be interested in modeling relations in a social network; we may have a few relations (friendship, marriage, etc), but an enormous amount of data. The complexity of inference when query, evidence and domains are inputs (and the model is fixed) is called the *data* complexity of inference; similarly, we may be interested in a fixed model and fixed query/evidence, with only the domains as the input; in this case we have *domain* complexity [11]. Data and domain complexities are directly related to the concepts of lqe-liftability and domain-liftability that are often employed in the literature on lifted inference of probabilistic relational models [4]. Lqe-liftability means that data complexity is polynomial, and domain-liftability means that domain complexity is polynomial.

Concerning the data complexity of plates, we have:

Theorem 2 *Inference in plate models when the model is fixed is* $\#\mathsf{P}[1]$*-equivalent.*

Proof. Given Theorem 1, we only need to show hardness. Consider a monotone 2-CNF formula on propositional variables $a_1, \ldots, a_n$ with m clauses. We call a_i and a_j left and right variables, respectively, of the clause $a_i \vee a_k$. We assume an ordering of the clauses, so that we can refer to the left (right) variable of ith clause. Counting the number of assignments to the variables that make the sentence true is a #P-complete problem [21]

Build the plate model in Figure 4. Both the logvars χ and y index the propositions in the CNF formula; their domains are $\{1, \ldots, n\}$. A grounded variable $\mathsf{Left}(i)$ represents the proposition a_i when it appears as the left variable in a clause. Similarly, $\mathsf{Right}(j)$ represents a_j when it appears as the right variable. Impose $\mathbb{P}(\mathsf{Left}(\chi)) = 1/2$, $\mathbb{P}\big(\mathsf{Right}(y)\big) = 1/2$. The $\mathsf{Equivalence}(\chi, y)$ parvariable enforces that $\mathsf{Left}(i)$ and $\mathsf{Right}(j)$ must take on the same value whenever they represent the same proposition; this is achieved by imposing

$$\mathbb{P}\big(\mathsf{Equivalence}(\chi, y) = 1|\mathsf{Left}(\chi), \mathsf{Right}(y)\big) = \begin{cases} 1, & \text{if } \mathsf{Left}(\chi) = \mathsf{Right}(y), \\ 0, & \text{if } \mathsf{Left}(\chi) \neq \mathsf{Right}(y). \end{cases}$$

Finally, $\mathsf{Disjunction}(\chi, y)$ encodes a clause with propositions $\mathsf{Left}(\chi)$ and $\mathsf{Right}(y)$: $\mathbb{P}\big(\mathsf{Disjunction}(\chi, y) = 1|\mathsf{Left}(\chi), \mathsf{Right}(y)\big) = 0$ if $\mathsf{Left}(\chi) = \mathsf{Right}(y) = 0$ and $\mathbb{P}\big(\mathsf{Disjunction}(\chi, y) = 1|\mathsf{Left}(\chi), \mathsf{Right}(y)\big) = 1$ otherwise.

Now create the evidence $\mathbf{E}$ that contains for each $i = 1, \ldots, N$ (that is, for each proposition), the assignment $\{\mathsf{Equivalence}(i, i) = 1\}$. Likewise, create the query $\mathbf{Q}$ containing for each clause the assignment $\{\mathsf{Disjunction}(i, j) = 1\}$, where

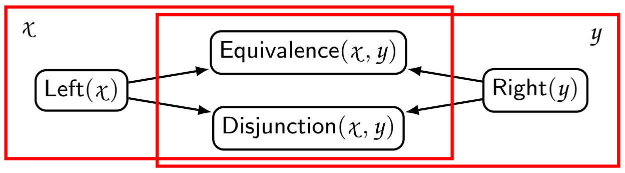


Fig. 4. A plate model that counts satisfying assignments of monotone 2-CNF formulas.

a_i and a_j are, respectively, the left and right variables of the ith clause. Building this plate model takes polynomial time in the size of the CNF formula. One can check that $\mathbb{P}(\mathbf{Q}|\mathbf{E})$ equals the number of satisfying assignments of the formula up to a (polynomial-time computable) constant. □

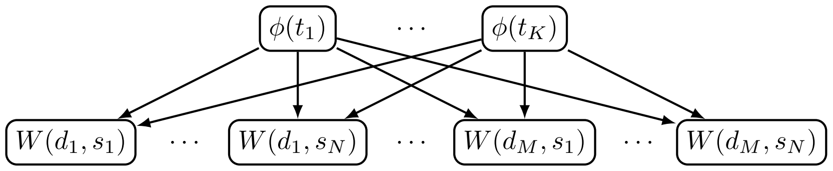
The proof of Theorem 1 shows that the size of domain is irrelevant once the model, query and evidence are fixed. Hence, domain complexity is constant:

Theorem 3 *Inference in plate models, when the model, query and evidence are fixed, takes constant time.*

4 Enhanced plate models

We now consider plate models where a node in a plate can have children in other plates; we refer to these as *enhanced plate models*. As before, to guarantee that such a definition works for all cases, we assume that there is a "base plate" encompassing all nodes, so that a node outside of all drawn plates is already in the base plate, and it can have children in other plates. We do not draw the base plate in our plate models.

A popular model that employs enhanced plate models is sLDA, depicted in Figure 1. Here the logvar z runs over a set of topics, while χ runs over a set of documents, and y runs over a designated set of strings. The node $W(\chi, y)$ is the child of $\phi(z)$; grounding produces:



Now consider a particular grounded variable $W(d, s)$; the number of parents of this variable in the grounded graph depends on the number of topics (that is, on the size of domain of z). Hence the template probability $\mathbb{P}\big(W(\chi, y)|\phi(z)\big)$ must specify which procedure is to be used to produce probability values given the domains. The typical solution is to allow *aggregation functions* to be given [14], where an aggregation function takes a set of groundings, and assignments for them, and produces a probability value out of them. Another strategy is used in relational Bayesian networks, where *combination functions* are adopted to compute probability values [6]. Now we need to be careful when selecting a family of aggregation functions lest the complexity of inference may be dictated by the complexity of a single aggregation function. For example, consider allowing a function that counts the number of solutions of an EXP-complete problem. Then clearly the inference problem is $\#\mathsf{EXP}$-hard. On the other extreme, consider aggregation functions that return constant values; these functions act as if disconnecting the parents from the child, and do not add any expressivity over classic plate models.

We choose to investigate the simplest possible language, where all parvariables are binary (have two values), and aggregation functions are specified using existential quantifiers. Such quantifiers can describe many common phenomena; for instance, they can be used to specify Noisy-Or models [18]. Moreover, existential quantifiers are easily computed, and concisely specified. Say we have a parvariable Y with parent $X(\mathcal{x})$, and $\mathcal{x}$ takes values in $\{a_1, \dots, a_N\}$. The grounded Bayesian network contains the variable Y with parents $X(a_1), \dots, X(a_N)$. Then the corresponding conditional distribution is $\mathbb{P}(Y = 1|X(a_1), \dots, X(a_N)) = 0$ if $X(a_1) = \cdots = X(a_N) = 0$ and $\mathbb{P}(Y = 1|X(a_1), \dots, X(a_N)) = 1$ otherwise. This can be stated more concisely as $Y = \exists \mathcal{x} X(\mathcal{x})$.

Now suppose the model has another variable Z with $\mathbb{P}(Z|Y) = 1$ if $Z \neq Y$ and $\mathbb{P}(Z|Y) = 0$ if $Z = Y$. That is, $Z = \neg Y$. Then $\mathbb{P}(Z|X(\mathcal{x})) = \neg\exists\mathcal{x} X(\mathcal{x}) = \forall\mathcal{x}\neg X(\mathcal{x})$. Thus we assume, as a syntactic sugar, that we can specify aggregation functions containing arbitrary logical formulas with existential and universal quantifiers, and that we specify the aggregation function as a first-order logical expression: for example, $Y = \big(\exists\mathcal{x}\forall\mathcal{y} Z(\mathcal{x}, \mathcal{y}) \wedge \neg X(\mathcal{x}) \rightarrow W(\mathcal{y})\big)$.

Given that we may have a polynomial number of nested plates, and this produces an exponential number of groundings, one might suspect that inference with enhanced plates requires exponential effort. However, it is not obvious how to prove this, because the language of plate models does not allow us to directly build standard complete problems for exponential classes. We have the restriction that each logvar is tied to a plate/domain; hence we cannot write a logical expression such as $X(\mathcal{x}) \rightarrow \neg X(\mathcal{y})$, where the same parvariable X appears with distinct logvars. However, our main result in this section shows that exponential behavior is actually realized:

Theorem 4 *Inference in enhanced plate models is* $\#\mathsf{EXP}[1]$*-equivalent.*

Proof. To prove pertinence, note that an enhanced plate model can be grounded into an exponentially larger Bayesian network, and inference can be carried out in that network (which implies it can be solved with one call to a $\#\mathsf{EXP}$ machine and some exponential-time post-processing).

To prove hardness, we resort to bounded domino problems; indeed, we will build a plate model (Figure 5) that encodes a domino problem. A *domino system* consists of a finite set D of *tiles* and a pair of *compatibility relations* H and V, both on $D \times D$, respectively expressing horizontal and vertical constraints between tiles. The idea is that tiles are to be placed in points of a *torus* $\mathbb{N}_s \times \mathbb{N}_t$, where $\mathbb{N}_s$ denotes the integers modulo s, and that adjacent tiles need to satisfy the constraints H and V. Such a torus is denoted compactly by $U(s,t)$.

Some tiles, the *initial conditions*, are assigned to the first n points in the bottom row of torus $U(s,t)$. We denote by d_i^0 the ith initial condition; that is, the initial condition for point $(i,0)$. The torus has a *tiling* if it is possible to cover the whole torus while respecting the compatibility relations and the initial conditions. That is, there must be a mapping $\tau : U(s,t) \to D$ such that for all $(x,y) \in U(s,t)$, (i) $(\tau(x,y), \tau(x+1 \mod s, y)) \in H$; (ii) $(\tau(x,y), \tau(x, y+1 \mod t)) \in V$; (iii) $\tau(i,0) = d_i^0$ for $0 \leq i < n$.

Börger et al. showed that given a (time/space) bounded Turing machine one can construct a bounded domino system that reproduces its behavior [2, Thm. 6.1.2]. Unfortunately in their construction the number of accepting paths in the Turing machine and the number of tilings in the domino system may differ, and this is inappropriate for a counting class such as #EXP. We need to produce a *parsimonious* reduction [3, Sec. 18.1]; that is, a reduction that preserves the number of accepting paths in the Turing machine. To do it, we must recapitulate the construction by Börger et al. They start by assuming that we have a *simple* nondeterministic Turing machine M over alphabet Σ containing a *blank* character. That is, the Turing machine works on a single semi-infinite tape where cells are numbered from 0 on; the machine never tries to move to the left of the first cell, and at every stage of the computation there is some integer n such that cells 0 to n contain non-blank characters and all other cells contain blanks; finally, the machine has a unique accepting state q_a, in which the tape contains only blanks and the head is in the first cell. Given any Turing machine, we can enlarge it polynomially so that it satisfies these restrictions, as described by the following result (the proof is omitted due to space constraints, but it can be produced by an explicit construction):

Lemma 1 *Let M be a simple nondeterministic Turing machine with alphabet Σ, input alphabet Σ', and set of states Q. An input x is a sequence $\sigma'_0\sigma'_1\ldots\sigma'_{n-1}$. Then there exists a domino system and a linear-time reduction that takes any input x to a sequence d^0 of n tiles such that:*
(i) if M accepts x in time t_0 and space s_0 then for any accepting computation there is a single tiling for torus $U(s,t)$ with initial condition d^0 where s and t are polynomials on s_0, t_0, and M;
(ii) if M does not accept x then the torus $U(s,t)$ is not tiled with initial condition d^0 for all $s, t \geq 2$.

Hence, counting the number of tilings is a #EXP-complete problem. From now on we assume that we have a domino system with m tiles ($|D| = m$) specifying a torus $U(2^n, 2^n)$ and initial conditions $d_i^0 \ldots d_{n-1}^0$. Our goal is to reduce the problem of counting tilings to an inference in an enhanced plate model. Our reduction is inspired by a similar result by Tobies [1].

First we need to represent the positions of the torus; we do so by creating $2n$ logical variables $\chi_{0,0}, \ldots, \chi_{0,n-1}$ and $\chi_{1,0}, \ldots, \chi_{1,n-1}$. All these variables have the same binary domain $\{0,1\}$. The idea is that these variables represent the

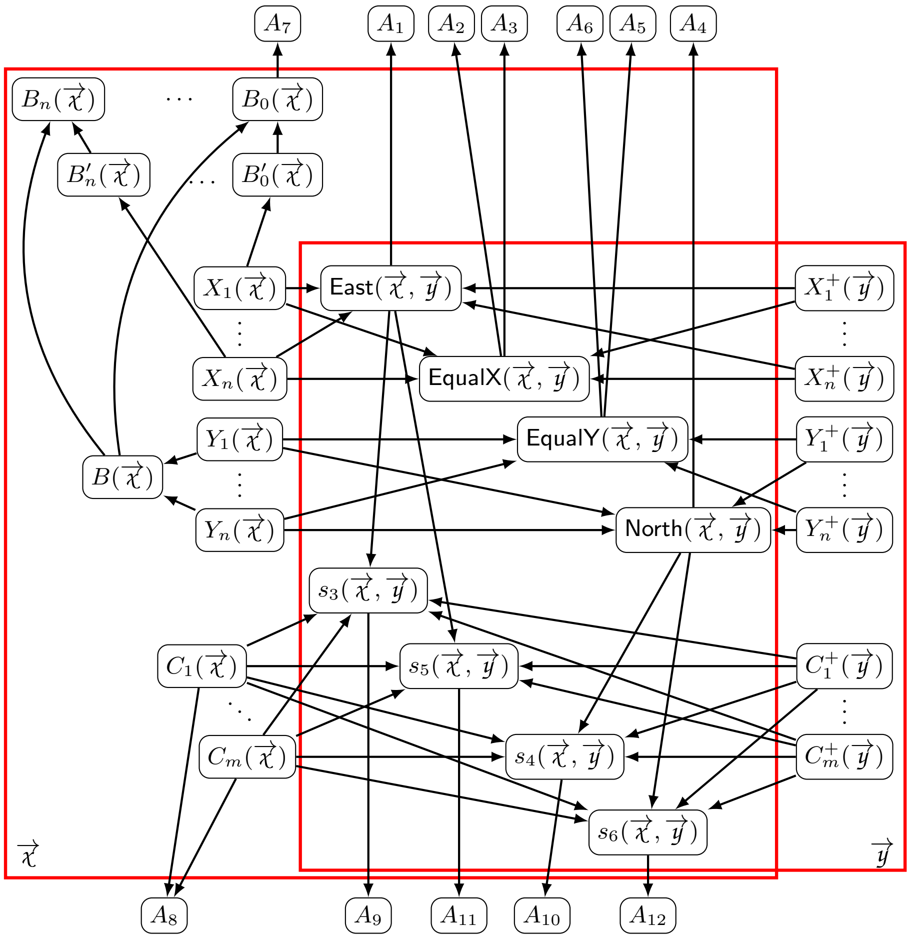


Fig. 5. Counting tilings of a torus with a plate model. Each plate is actually a set of n nested plates, one for each logvar in the indicated vector of logvars. Nodes s_3, s_4, s_5 and s_5 encode auxiliary logical expressions indicated in the text.

coordinates of a position (x, y) in the torus in the following way: an assignment $\vec{a}$ to $\vec{x}_0 = (x_{0,0}, \ldots, x_{0,n-1})$ represents the value of x (the column) in binary notation, while an assignment $\vec{b}$ to $\vec{x}_1 = (x_{1,0}, \ldots, x_{1,n-1})$ represents the value of y (the row) in binary notation. To make the presentation more clear and succinct, we treat all these logical variables as a single variable $\vec{x}$ whose domain are the natural numbers between zero and 2^{2n}. One should bear in mind that this is simply syntactic sugar (so the reduction is polynomial in the input).

The proof builds two torus, which we force to be identical. The positions of the second torus are represented by the logical variables $\vec{y}$ whose combined domain are the natural numbers between zero and $2^{2n} - 1$. Here again, this is simply syntactic sugar to avoid writing $2n$ logical variables with binary domains.

We create parvariables $X_0(\overrightarrow{x}), \ldots, X_{n-1}(\overrightarrow{x})$ and $Y_0(\overrightarrow{x}), \ldots, Y_{n-1}(\overrightarrow{x})$ to represent the x- and y-coordinates of the positions in binary notation. Thus a position (x, y) is encoded as $x = \sum_{i=0}^{n-1} X_i(x, y) \cdot 2^i$ and $y = \sum_{i=0}^{n-1} Y_i(x, y) \cdot 2^i$. The row and column of a position in the second torus are represented by parvariables $X_0^+(\overrightarrow{y}), \ldots, X_{n-1}^+(\overrightarrow{y})$ and $Y_0^+(\overrightarrow{y}), \ldots, Y_{n-1}^+(\overrightarrow{y})$. We impose $\mathbb{P}(X_i = 1) = \mathbb{P}\big(X_i^+ = 1\big) = \mathbb{P}(Y_i = 1) = \mathbb{P}\big(Y_i^+ = 1\big) = 1/2$ (here and elsewhere we omit logvars to save space). We need to specify the concept of adjacent positions; to this end we introduce parvariables $\mathsf{East}(\overrightarrow{x}, \overrightarrow{y})$ and $\mathsf{North}(\overrightarrow{x}, \overrightarrow{y})$, and specify:

$$\begin{aligned}
\mathsf{East}(\overrightarrow{x}, \overrightarrow{y}) = \bigwedge_{k=0}^{n-1} & (\wedge_{j=0}^{k-1} X_j(\overrightarrow{x})) \rightarrow (X_k(\overrightarrow{x}) \leftrightarrow \neg X_k^+(\overrightarrow{y})) \wedge \\
& \bigwedge_{k=0}^{n-1} (\vee_{j=0}^{k-1} \neg X_j(\overrightarrow{x})) \rightarrow (X_k(\overrightarrow{x}) \leftrightarrow X_k^+(\overrightarrow{y})), \\
& \qquad \wedge \bigwedge_{k=0}^{n-1} (Y_k(\overrightarrow{x}) \rightarrow Y_k^+(\overrightarrow{y})) \wedge (\neg Y_k(\overrightarrow{x}) \rightarrow \neg Y_k^+(\overrightarrow{y}))), \\
\mathsf{North}(\overrightarrow{x}, \overrightarrow{y}) = \bigwedge_{k=0}^{n-1} & (\wedge_{j=0}^{k-1} Y_j(\overrightarrow{x})) \rightarrow (Y_k(\overrightarrow{x}) \leftrightarrow \neg Y_k^+(\overrightarrow{y})) \wedge \\
& \bigwedge_{k=0}^{n-1} (\vee_{j=0}^{k-1} \neg Y_j(\overrightarrow{x})) \rightarrow (Y_k(\overrightarrow{x}) \leftrightarrow Y_k^+(\overrightarrow{y}) \\
& \qquad \wedge \bigwedge_{k=0}^{n-1} ((X_k(\overrightarrow{x}) \rightarrow X_k^+(\overrightarrow{y})) \wedge (\neg X_k(\overrightarrow{x}) \rightarrow \neg X_k^+(\overrightarrow{y}))).
\end{aligned}$$

Parvariable $\mathsf{East}(\overrightarrow{x}, \overrightarrow{y})$ indicates whether $\overrightarrow{y}$ is the position immediately to the right of $\overrightarrow{x}$; similarly, $\mathsf{North}(\overrightarrow{x}, \overrightarrow{y})$ indicates whether $\overrightarrow{y}$ is the position immediately above $\overrightarrow{x}$. We need to enforce that the positions of $\overrightarrow{x}$ and $\overrightarrow{y}$ with $\overrightarrow{x} = \overrightarrow{y}$ have the same encoding: $\mathsf{EqualX}(\overrightarrow{x}, \overrightarrow{y}) = \bigwedge_{k=0}^{n-1} X_k(\overrightarrow{x}) \leftrightarrow X_k^+(\overrightarrow{y})$, $\mathsf{EqualY}(\overrightarrow{x}, \overrightarrow{y}) = \bigwedge_{k=0}^{n-1} Y_k(\overrightarrow{x}) \leftrightarrow Y_k^+(\overrightarrow{y})$. We can now create variables to define the adjancency of every position: $A_1 = \forall \overrightarrow{x} : \exists \overrightarrow{y} : \mathsf{East}(\overrightarrow{x}, \overrightarrow{y})$, $A_2 = \forall \overrightarrow{x} : \exists \overrightarrow{y} : \mathsf{EqualX}(\overrightarrow{x}, \overrightarrow{y})$, $A_3 = \forall \overrightarrow{y} : \exists \overrightarrow{x} : \mathsf{EqualX}(\overrightarrow{x}, \overrightarrow{y})$, $A_4 = \forall \overrightarrow{x} : \exists \overrightarrow{y} : \mathsf{North}(\overrightarrow{x}, \overrightarrow{y})$, $A_5 = \forall \overrightarrow{x} : \exists \overrightarrow{y} : \mathsf{EqualY}(\overrightarrow{x}, \overrightarrow{y})$, $A_6 = \forall \overrightarrow{y} : \exists \overrightarrow{x} : \mathsf{EqualY}(\overrightarrow{x}, \overrightarrow{y})$.

We then need to represent the base row (so that we can establish an origin and initial conditions for the torus). We create a parvariable $B_i'(\overrightarrow{x})$, for each $i = 0, \ldots, n-1$, such that $B_i'(\overrightarrow{x})$ reflects the binary encoding of i, as follows: $B_0'(\overrightarrow{x}) = \bigwedge_{k=0}^{n-1} \neg X_k(\overrightarrow{x})$, $B_1'(\overrightarrow{x}) = X_0(\overrightarrow{x}) \wedge \bigwedge_{k=1}^{n-1} \neg X_k(\overrightarrow{x})$, $B_2'(\overrightarrow{x}) = \neg X_0(\overrightarrow{x}) \wedge X_1(\overrightarrow{x}) \wedge \bigwedge_{k=2}^{n-1} \neg X_k(\overrightarrow{x})$, and so on. Now specify $B(\overrightarrow{x}) = \bigwedge_{k=0}^{n-1} \neg Y_k(\overrightarrow{x})$ and, for each $i \in \{0, \ldots, n-1\}$, $B_i(\overrightarrow{x}) = B_i'(\overrightarrow{x}) \wedge B(\overrightarrow{x})$. The parvariable $B(\overrightarrow{x})$ indicates that the position is in the base row, and the parvariable $B_i(\overrightarrow{x})$ indicates that the position is in the ith column. Together, they specify the relevant part of the base row that we need to specify the initial tiles. We must enforce an origin for the torus, so we introduce: $A_7 = \exists \overrightarrow{x} : B_0(\overrightarrow{x})$.

At this point, by fixing $\bigwedge_{i=1}^{7} A_i$ we build a torus of size $2^n \times 2^n$. It remains to represent the horizontal, vertical and initial constraints.

We introduce a pair $C_j(\overrightarrow{\chi})$ and $C_j^+(\overrightarrow{y})$ for each possible tile. For each tile $j = 1, \dots, m$, we impose $\mathbb{P}\big(C_j(\overrightarrow{\chi})\big) = \mathbb{P}\big(C_j^+(\overrightarrow{y})\big) = 1/2$. Each position of the torus must have one and only one tile:

$$A_8 = \forall \overrightarrow{\chi} : \left(\bigvee_{j \in \mathcal{C}} C_j(\overrightarrow{\chi}) \right) \wedge \left(\bigwedge_{j \in \mathcal{C}, k \in \mathcal{C}, j \neq k} \neg (C_j(\overrightarrow{\chi}) \wedge C_k(\overrightarrow{\chi})) \right).$$

Moreover, the tiles must satisfy the horizontal and vertical restrictions:

$$A_9 = \forall \overrightarrow{\chi} : \bigwedge_{j \in \mathcal{C}} C_j(\overrightarrow{\chi}) \rightarrow (\forall \overrightarrow{y} : \mathsf{East}(\overrightarrow{\chi}, \overrightarrow{y}) \rightarrow \vee_{k:(j,k) \in H} C_k^+(\overrightarrow{y})),$$

$$A_{10} = \forall \overrightarrow{\chi} : \bigwedge_{j \in \mathcal{C}} C_j(\overrightarrow{\chi}) \rightarrow (\forall \overrightarrow{y} : \mathsf{North}(\overrightarrow{\chi}, \overrightarrow{y}) \rightarrow \vee_{k:(j,k) \in V} C_k^+(\overrightarrow{y})).$$

Now make both sets of parvariables related to tiles behave the same:

$$A_{11} = \bigwedge_{j \in \mathcal{C}} \forall \overrightarrow{\chi} : \forall \overrightarrow{y} : C_j^+(\overrightarrow{y}) \wedge \mathsf{East}(\overrightarrow{\chi}, \overrightarrow{y}) \rightarrow C_j(\overrightarrow{\chi}),$$

$$A_{12} = \bigwedge_{j \in \mathcal{C}} \forall \overrightarrow{\chi} : \forall \overrightarrow{y} : C_j^+(\overrightarrow{y}) \wedge \mathsf{North}(\overrightarrow{\chi}, \overrightarrow{y}) \rightarrow C_j(\overrightarrow{\chi}).$$

Finally, we impose the initial conditions: $A_{13} = \forall \chi : \bigwedge_{i=0}^{n-1} B_i(\overrightarrow{\chi}) \rightarrow C_i^0(\overrightarrow{\chi})$, where C_i^0 represents the ith tile as given by initial condition.

Computing the probability of $A_{14} = \bigwedge_{i=1}^{13} A_i$ produces the probability that a tiling is built satisfying all horizontal and vertical restrictions and the initial condition. If we can recover the number of tilings of the torus from this probability, we obtain the number of accepting computations of the exponentially-bounded Turing machine we started with. Assume we have $\mathbb{P}(A_{14} = 1)$. Then $\mathbb{P}(A_{14} = 1) \times 2^{\delta}$ is the number of truth assignments that build the torus satisfying horizontal and vertical relations and initial conditions, where $\delta = 2^{2n}(2n + 2^{2n+1} + m)$. However, this number is *not* equal to the number of tilings of the torus. To see this, consider the grounded Bayesian network where each a in the domain is associated with a "slice" containing groundings $X_i(a)$, $Y_i(a)$, $C_j(a)$ and so on. If a particular configuration of these indicator variables corresponds to a tiling, then we can produce the same tiling by permuting all elements of the domain with respect to the slices of the network. Intuitively, we can fix a tiling and imagine that we are labelling each point of the torus with an element of the domain; clearly every permutation of these labels produces the same tiling (this intuition is appropriate because each a corresponds to a different point in the torus). So, in order to produce the number of tilings of the torus, we must compute $\mathbb{P}(A_{14} = 1) \times 2^{\delta}/(2^{2n}!)$, where we divide the number of satisfying truth assignments by the number of repeated tilings. Consequently we obtain the number of accepting computations of the original Turing machine just by processing the inference $\mathbb{P}(A_{14} = 1)$, proving the desired result. $\square$

Concerning the data complexity of enhanced plates, we have:

Theorem 5 *Inference in enhanced plate models when the model is fixed is* $\#\mathsf{P}[1]$*-equivalent.*

Proof. Hardness follows from Theorem 2. To show pertinence, consider that, once the plate model is fixed, the arity of any relation is fixed. And given the domains as input, the combined domain has size that is a polynomial on the domains (where the maximum arity appears in the exponent). So one can then produce a grounded Bayesian network of size polynomial in the input. The result follows as inference in the grounded Bayesian network belongs to $\mathsf{FP}^{\#\mathsf{P}[1]}$. □

Concerning domain complexity, the fact that one can build complex logical expressions using plates (see the proof of Theorem 4) suggests that polynomial behavior cannot be expected [4]. However, we have not been able to provide precise lower and upper bounds on domain complexity, so we leave this as a challenge for future work.

5 Conclusion

Plates allow large Bayesian networks to be concisely described, and are particularly useful when one faces scenarios with many variables and intricate relations. Despite the popularity of plate models, few results on their complexity are available. We have presented here a number of results concerning the complexity of "classic" and enhanced plates; the former display $\#\mathsf{P}[1]$-equivalent combined/data complexity (despite the fact that they may induce exponentially large groundings), while the latter display $\#\mathsf{EXP}[1]$-equivalent combined complexity and $\#\mathsf{P}[1]$-equivalent data complexity. The results on enhanced plates are obtained when all relations are binary and aggregation functions are based on existential quantification. It is not difficult to see that exponential complexity there stems from the nesting of plates; in fact, if the level of nesting is limited, the combined complexity goes down to $\#\mathsf{P}[1]$-equivalent.

There are several avenues open for future work. The domain complexity of enhanced plate models is an open problem. Also, plate models are often augmented with additional resources to allow recursive descriptions and structural uncertainty [14]; the complexity of these more sophisticated languages deserves analysis. Finally, it would be interesting to examine more restricted languages; for instance, languages where evidence can only be "positive", or where aggregation functions can only have some bounded complexity.

Acknowledgements

The first author was partially supported by CNPq and the second author was partially supported by FAPESP.

References

1. Stephan Tobies The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *Journal of Artificial Intelligence Research*, 12:199–217, 2000.
2. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Cambridge University Press, 1997.
3. Papadimitriou, C. H. *Computational Complexity.* Addison-Wesley Publishing, 1994.
4. Manfred Jaeger, and Guy van den Broeck. Liftability of probabilistic inference: Upper and lower bounds. In *Statistical Relational AI Workshop*, 2012.
5. David Sontag and Dan Roy. Complexity of inference in Latent Dirichlet Allocation. *Advances in Neural Information Processing Systems 24*, pages 1008–1016, 2011.
6. Manfred Jaeger. Relational Bayesian networks. *Conference on Uncertainty in Artificial Intelligence*, pages 266–273, 1997.
7. Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1–2):107–136, 2006
8. Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.
9. David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research 3*, 3:993–1022, 2003.
10. W. L. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.
11. Fabio G. Cozman, and Denis D. Mauá. Bayesian networks specified using propositional and relational constructs: Combined, data, and domain complexity. In *AAAI Conference on Artificial Intelligence*, pages 3519–3525, 2015.
12. Cassio P. de Campos, Georgios Stamoulis, and Dennis Weyland. A structured view on weighted counting with relations to quantum computation and applications. Technical Report 133, Electronic Colloquium on Computational Complexity, 2013.
13. W. Gilks, A. Thomas, and D. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43:169–178, 1993.
14. Daphne Koller, and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, 2009.
15. David Lunn, Christopher Jackson, Nicky Best, Andrew Thomas, and David Spiegelhalter. *The BUGS Book: A Practical Introduction to Bayesian Analysis.* CRC Press/Chapman and Hall, 2012.
16. David Lunn, David Spiegelhalter, Andrew Thomas, and Nicky Best. The BUGS project: Evolution, critique and future directions. *Statistics in Medicine*, 28:3049–3067, 2009.
17. Christos H. Papadimitriou. A note on succinct representations of graphs. *Information and Control*, 71:181–185, 1986.
18. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, California, 1988.
19. David Poole. First-order probabilistic inference. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 985–991, 2003.
20. Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.
21. Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8(3):410–421, 1979.