Probabilistic Inference and Monadic Second Order Logic

Hans L. Bodlaender

P.O. Box 80.089, Department of Computing Sciences, Utrecht University, The Netherlands, h.l.bodlaender@uu.nl

Abstract. This paper combines two classic results from two different fields: the result by Lauritzen and Spiegelhalter [21] that the probabilistic inference problem on probabilistic networks can be solved in linear time on networks with a moralization of bounded treewidth, and the result by Courcelle [10] that problems that can be formulated in counting monadic second order logic can be solved in linear time on graphs of bounded treewidth. It is shown that, given a probabilistic network whose moralization has bounded treewidth and a property P of the network and the values of the variables that can be formulated in counting monadic second order logic, one can determine in linear time the probability that P holds.

1 Introduction

This paper combines two classic results from two different fields from computer science: the result by Lauritzen and Spiegelhalter [21] that the probabilistic inference problem can be solved in linear time for probabilistic network whose moral graph has treewidth bounded by some constant, and the result by Courcelle [10] that problems that can be formulated in counting monadic second order logic can be solved in linear time on graphs of bounded treewidth.

Probabilistic networks (also called *Bayesian networks* or *belief networks*) are the underlying technology of several modern decision support systems. See for more background, e.g., [16, 22]. A probabilistic network consists of a directed acyclic graph, and for each vertex in the graph a table of conditional probabilities. Each vertex in the graph represents a statistical variable, which can assume one of a fixed number of values; the table for a vertex gives the probability distribution for the values for that vertex, conditional on the values of the parents of the vertex.

Probabilistic networks are a topic of intense study. A central problem for probabilistic networks is the *inference problem*. One of the most commonly used algorithm to solve this problem is the *clique tree propagation* algorithm of Lauritzen and Spiegelhalter [21]. It consists of the following steps: first the *moral graph* or *moralization* is formed from the probabilistic network. (See Section 2 for the definition of moral graph.) To this moral graph, edges are added such that a triangulated (or chordal) graph is formed. To a triangulated graph G, one

can associate a *clique tree*: a tree with every tree node associated to a unique maximal clique in G, with the additional property that for each node of the graph, the cliques to which it belong form a connected subtree of the tree. The clique tree can be used to solve the inference problem, with the time exponential in the sizes of the cliques, but linear when these sizes are bounded by a constant. An alternative way of describing the algorithm of [21] is by means of *tree decompositions*. Doing so, we have an algorithm for the probabilistic inference problem that is linear in the number of variables, but exponential in the *width* of the tree decomposition, in other words: the problem is linear time solvable when the moral graph has its *treewidth* bounded by a constant.

There are many other problems with the property that they are intractable (e.g., NP-hard) for general graphs, but become linear time solvable on graphs with bounded treewidth. A very powerful characterization of a class of such problems is by [10] by the notion of *Monadic Second Order Logic*, or extensions of this notion. In this paper, we use the notion of *Counting Monadic Second Order Logic*and the notion of *regularity*. The result of Courcelle has been extended a number of times (e.g., [1, 8, 12], see also [9].)

In this paper, we show that the results by Lauritzen and Spiegelhalter and by Courcelle can be combined. Suppose we have a probabilistic network whose moral graph has bounded treewidth. We can state some property of the network and of the values of the variables. Our result shows that if the property can be formulated in a language called (Counting) Monadic Second Order Logic, then we can compute the probability that it holds in linear time; i.e., to compute such a probability is *fixed parameter tractable* when parameterized by treewidth. (The result can also be seen as a variant of results shown by Courcelle and Mosbah [12].) Examples of such CMSOL properties are: do the variables that are true form a connected subgraph of the network, does no directed path in the network have all its variables alternatingly true and false along the path, are there an odd number of variables true, is the subgraph of the network induced by the false variables 3-colorable? It includes many properties that are NP-hard to check for arbitrary networks.

In Section 2, we give some preliminary results and definitions, including a hypergraph model that can be used instead of moral graphs. In Section 3, we sketch the proof of the main result. Some variants, extensions, and applications are discussed in Section 4. Some conclusions are given in Section 5.

2 Preliminaries

2.1 Probabilistic networks and the inference problem

To ease descriptions, we assume all variables in the probabilistic networks that are dealt with to be binary (boolean). Notations in the paper sometimes follow conventions from algorithmic graph theory, and sometimes follow conventions from probabilistic networks theory.

For a set S, a *configuration on* S is a function $S \to \{\text{true}, \text{false}\}$. We denote the set of all configurations on S as $\mathcal{C}(S)$. A configuration on a set S is often

denoted as x_S; when S is a single vertex v, we write x_v for $x_{\{v\}}$. $x_\emptyset$ denotes the unique configuration on the empty domain.

For sets S and S' and configurations $x_S \in \mathcal{C}(S)$ and $x_{S'} \in \mathcal{C}(S')$, we say that x_S *agrees with* $x_{S'}$, notated $x_S \sim x_{S'}$, if for all $v \in S \cap S' : x_S(v) = x_{S'}(v)$. Given a configuration $x_S \in \mathcal{C}(S)$ and a subset $S' \subseteq S$, we write $x_S(S')$ for the configuration on S' that is a restriction of x_S to the smaller domain: $x_S(S') \in \mathcal{C}(S')$ with $x_S(S') \sim x_S$.

Given a directed graph $G = (V, A)$ with a vertex $v \in V$, $par(v)$ denotes the set of vertices that have an arc to v: $par(v) = \{w \in V \mid \exists (w, v) \in A\}$. $par(v)$ is called the set of *parents* of v.

Let $G = (V, A)$ be a directed graph. The *moral graph*of G is the undirected graph, obtained by adding an edge between each pair of different vertices that are parents of the same vertex, and then dropping all directions of arcs, i.e., the graph $(V, \{\{v, w\} \mid (v, w) \in A \vee (w, v) \in A \vee \exists x \in V : v, w \in par(x)\}$. (The process of obtaining the moral graph of a directed graph is called *moralization*. The term, frequently used in probabilistic network research, comes from the notion to 'marry the parents'.)

A probabilistic network is a pair (G, κ), with $G = (V, A)$ a directed acyclic graph, and κ a set of functions, as follows. For each $v \in V$, we have a function $\kappa_v : \mathcal{C}(\{v\}) \times \mathcal{C}(par(v)) \to [0, 1]$.

κ_v is meant to describe the conditional probability distribution for v, here conditional on the values of the parents of v, i.e., for configurations x_v, $x_{par(v)}$, $\kappa_v(x_v, x_{par(v)})$ should give the value $Pr(x_v|x_{par(v)})$.

G and κ together define a joint probability distribution on the set of probabilistic variables V in the following way.

A *full configuration* of a probabilistic network (G, κ), $G = (V, A)$ is a configuration $x_V \in \mathcal{C}_V$ on the set V of all vertices in G.

For each full configuration, define the probability of the configuration as the product of the conditional probabilities over all vertices, as follows. Assume $x_V \in \mathcal{C}(V)$.

$$\Pr(x_V) = \prod_{v \in V} \kappa_v(x_V(\{v\}), x_V(par(v))) \tag{1}$$

Throughout this paper, we write x_V as the stochastic variable, that selects one full configuration with the distribution as given by (1) for the considered probabilistic network (G, κ).

For each configuration, its probability now is the sum of the probabilities of all full configurations that agree with it. Assume $x_W \in \mathcal{C}(W)$, $W \subseteq V$.

$$\Pr(x_W) = \sum_{x_V \in \mathcal{C}(V),\ x_V \sim x_W} \Pr(x_V) \tag{2}$$

In applications of probabilistic networks, one generally desires to know the probability that a variable has a certain value, conditional to given values for some other variables. For instance, in a network, modeling a medical application, one may want to know if a patient has a certain disease, given some symptoms. The given values of variables are called the *observations*. The set of *observed*

variables is denoted by $\mathcal{O}$, and the observed configuration is denoted $x_{\mathcal{O}}$. The PROBABILISTIC INFERENCE problem is to compute the conditional probability $\Pr(x_v \mid x_{\mathcal{O}})$, for a variable $v \in V$ and $x_v \in \mathcal{C}(v)$, or, more generally, the probability distribution for each variable, conditional to $x_{\mathcal{O}}$. To compute $\Pr(x_v \mid x_{\mathcal{O}})$, we can use that $\Pr(x_v \mid x_{\mathcal{O}}) = \Pr(x_v \wedge x_{\mathcal{O}})/\Pr(x_{\mathcal{O}})$.

Computational model In the paper, we assume all computations with values of probabilities can be done in constant time. In a practical setting, one could work with standard rounded representations of real numbers. To be precise, we should assume each conditional probability in κ is given as a fraction of two integers. Analysis of the algorithms show that each value remains a fraction of two integers, both expressible with a polynomial number of bits; the algorithms we give involve a linear number of operations with such integers, while each such operation involves a polynomial number of bit operations. The details are not given here.

2.2 A mixed hypergraph model

Instead of using the more standard method of using a moralization of a probabilistic network, we instead associate with each probabilistic network a mixed hypergraph.

A mixed hypergraph is a pair $H = (V, E, A)$, with V a finite set of *vertices*, and E a finite set of *hyperedges*, and A a set of arcs; each hyperedge $e \subseteq V$ is a non-empty subset of the vertices; an arc is an ordered pair of distinct vertices.

To a directed acyclic graph $G = (V, A)$, we associate a mixed hypergraph $H = (V, E, A)$ in the following way. H has the same vertices and arcs as G. For each vertex $v \in V$, we take a hyperedge consisting of v and its parents: $E = \{\{v\} \cup par(v) \mid v \in V\}$. H is the *mixed hypergraph associated with* G. The following lemma shows there is a one-to-one correspondence between the vertices and the hyperedges.

Lemma 1. *Let $G = (V, A)$ be a directed acyclic graph. If $v, w \in V$, $v \neq w$, then $\{v\} \cup par(v) \neq \{w\} \cup par(w)$.*

Besides the hypergraph model of the directed acyclic graph, we also define a set of functions $\kappa_e : \mathcal{C}(e) \to [0, 1]$, one for each hyperedge $e \in E$; these reflect the conditional probability functions κ_v.

Consider an edge $e = \{v\} \cup par(v)$. Let for each configuration on e, $x_e \in \mathcal{C}(e)$:

$$\kappa_e(x_e) = \kappa_v(x_e(\{v\}), x_e(par(v))) \tag{3}$$

We introduce one more notation. Let $x_S \in \mathcal{C}(S)$ be a configuration. Let e be a hyperedge with $e \subseteq S$. Then we write $\kappa(x_S, e) = \kappa(x_e)$ for the configuration x_e, obtained by restricting the function x_S to domain e.

We now can rephrase Equation 1 in terms of the κ_e functions. The next (well known) proposition follows directly from the definitions. See also [25].

Proposition 1. *Let $(G = (V, A), \kappa)$ be a probabilistic network. For each full configuration $x_V \in \mathcal{C}(V)$:*

$$\Pr(x_V) = \prod_{e \in A} \kappa_e(x_V)$$

2.3 Monadic Second Order Logic

The Monadic Second Order Logic language (MSOL) allows us to express properties of (mixed hyper)graphs. The language has the following constructs:

- Quantification over vertices, (hyper)edges, arcs: $\exists v \in V$, $\exists e \in E$, $\forall v \in V$, $\forall e \in E$, $\forall a \in A$,
- Quantification over sets of vertices, sets of (hyper)edges, sets of arcs: $\exists W \subseteq V$, $\exists F \subseteq E$, $\forall W \subseteq V$, $\forall F \subseteq E$, ...
- Membership tests: $v \in W$, $e \in F$.
- Identity tests: $v = w$, $e = f$.
- Adjacency tests: $v \in e$, $\{v, w\} \in E$, $\{v, w\} \in F$, v is tail (head) of a, ...
- Logic operations: $\vee$, $\wedge$, $\Rightarrow$, $\neg$, ...

MSOL is a powerful language. Many graph properties can be expressed in it. E.g, Borie et al. [8] show how many graph properties can be expressed in MSOL. For example, the following property expresses that directed graph $G = (V, A)$ is acyclic:

$$\forall W \subseteq V : \exists v \in W : \neg \exists w \in W : (v, w) \in A$$

Extensions of MSOL can include the following language constructs:

- For fixed constants c_1, c_2: $|W| \bmod c_1 = c_2$, $|F| \bmod c_1 = c_2$. MSOL with these constructs, for all fixed c_1 and c_2 is called Counting MSOL, or CMSOL.
- If the vertices, edges, or arcs of G are labeled with a bounded number of different labels, L the labeling function, we have label tests: $L(v) = c$, $L(e) = e$.

To these, we add one more construct, for the case that G is a probabilistic network.

- Value tests for vertices: $x_v = \text{true}$, $x_v = \text{false}$. (Or: x_v, $\neg x_v$.)

Call the resulting language *CMSOL with value tests*. For example, one can write the property that an even number of variables is true as follows:

$$\exists W : (\forall v : v \in W \leftrightarrow x_v) \wedge (|W| \bmod 2) = 0$$

Given a property in CMSOL with value tests, we are interested in the probability that this property holds for a given probabilistic network.

2.4 Treewidth, terminal hypergraphs, and parse trees

The notion of treewidth was introduced by Robertson and Seymour in [23]. There are several equivalent notions known, e.g., the treewidth of a graph is exactly one larger than the minimum over all chordal supergraphs of the maximum clique size; and graphs of treewidth at most k are also known as partial k-trees. See [3] for an overview. The definition here is given in terms of mixed hypergraphs.

Definition 1. *A* tree decomposition *of a mixed hypergraph $H = (V, E, A)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a collection of subsets of vertices (called* bags*), and T a tree, such that*

- $\bigcup_{i \in I} X_i = V$.
- *For each hyperedge $e \in E$ and each arc $(v, w) \in A$: there exists an $i \in I$ with $e \subseteq X_i$, or $\{v, w\} \subseteq X_i$, respectively.*
- *For all vertices $v \in V$: the set of nodes $\{i \in I \mid v \in X_i\}$ is connected in T.*

The width *of tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph G is the minimum width over all possible tree decompositions of G.*

Instead of using tree decompositions, we use an equivalent framework, where we use an 'algebra' on terminal mixed hypergraphs. A *k-terminal mixed hypergraph* is a 4-tuple $G = (V, E, A, (x_1, \ldots, x_k))$ with V a finite set of vertices, $E \subseteq \mathcal{P}(V)$ a set of hyper edges, $A \subseteq V \times V$ a set of arcs, and $(x_1, \ldots, x_k) \in V^k$ an ordered set of k vertices from V, called the *terminals*. A *terminal mixed hypergraph* is a k-terminal mixed hypergraph for some $k \geq 0$.

We define the following operations on terminal mixed hypergraphs.

- CreateHyperedge$_k()$. Has no arguments. Yields a k-terminal hypergraph with k vertices and one hyperedge, containing all vertices: $(\{v_1, \ldots, v_k\}, \{\{v_1, \ldots, v_k\}\}, \emptyset, (v_1, \ldots, v_k))$.
- CreateArc(). As CreateHyperedge, but creates a directed arc: $(\{v_1, v_2\}, \emptyset, \{(v_1, v_2)\}, (v_1, v_2))$.
- DropTerminal$_{k,\ell}(G)$. Has a k-terminal mixed hypergraph as argument, and turns the ℓth terminal into a non-terminal: $(V, E, A, (x_1, \ldots, x_k))$ maps to $(V, E, A, (x_1, \ldots, x_{\ell-1}, x_{\ell+1}, \ldots, x_k))$.
- AddTerminal$_k(G)$. Has a $(k-1)$-terminal mixed hypergraph as argument, and adds a new terminal vertex. Hyperedges and arcs are not affected. So, $(V, E, A, (x_1, \ldots, x_{k-1}))$ maps to $(V \cup \{x_k\}, E, A, (x_1, \ldots, x_k))$.
- Join$_k(G, H)$. Has two k-terminal mixed hypergraphs as argument. Yields the k-terminal mixed hypergraph obtained by taking the union of the arguments and then identifying the corresponding terminals. So, Join$_k((V, E, A, (x_1, \ldots, x_k)), (V', A', E', (x_1, \ldots, x_k))) = (V \cup V', E \cup E', A \cup A', (x_1, \ldots, x_k))$, with $V \cap V' = \{x_1, \ldots, x_k\}$.

Let $\mathcal{O}_k$ be the set of operations, containing for all k', ℓ, $1 \leq \ell \leq k' \leq k$, the operations CreateHyperedge$_{k'}$, CreateArc$_{k'}$, DropTerminal$_{k',\ell}$, AddTerminal$_{k'}$, Join$_{k'}$.

Lemma 2. *Let $(G = (V, A), \kappa)$ be a probabilistic network, let H be the moral graph of G, and let $H' = (V, E', A)$ be the associated mixed hypergraph of G. The following statements are equivalent.*

1. *H has treewidth at most $k - 1$.*
2. *H is subgraph of a triangulated graph with maximum clique size k.*
3. *H' has treewidth at most $k - 1$.*
4. *The 0-terminal mixed hypergraph $(V, E',\ A, \emptyset)$ can be constructed using the operations from $\mathcal{O}_k$.*

This can be shown using proofs and techniques from [26], see also [3]. Note that the notion is a minor twist on the well known notion of *nice tree decompositions*, see e.g., [3, 19].

If we have a method to construct a terminal mixed hypergraph with operations from $\mathcal{O}_k$, we can express this in a *parse tree*. Each node of such a tree is labeled with one operation from $\mathcal{O}_k$. To each tree node i, we associate a terminal mixed hypergraph $G_i = (V_i, E_i, A_i, (x_1^i, \ldots, x_{k'}^i)$; the terminal mixed hypergraph associated to the node is the graph obtained by applying its operation to the terminal mixed hypergraphs associated with its children. We assume that the root r of the parse tree is a 0-terminal mixed hypergraph, otherwise, use some DropTerminal operations at the top of the tree. If $G_r = (V_r, E_r, A_r, \emptyset)$, (V_r, E_r, A_r) is the mixed terminal hypergraph represented by the parse tree.

It is well known (see e.g., [6]) that if W is a clique in G, then a tree decomposition of G has a bag that contains all vertices in W. .Note that for each vertex v, $v \cup par(v)$ forms a clique in the moral graph and thus, for each tree decomposition of the moral graph, there is a bag that contains $v \cup par(v)$.

2.5 Regular properties and finite state tree automata

Many of the linear time algorithms on graphs of bounded treewidth can be expressed as a *finite state tree automaton*, a generalization of the classic finite state automaton. Such an automaton can be written as a 4-tuple $(S, S_A, t, \Box)$, with S a finite set of *states*, $S_A \subseteq S$ a set of *accepting states*, $\Box$ a special symbol, and t a *state transition function*: $t : (S \cup \{\Box\}) \times (S \cup \{\Box\}) \times \mathcal{O}_k \to S$. The automaton works on a binary tree with nodes labeled with elements from $\mathcal{O}_k$ (and in particular: on a parse tree), in the following way. Each element i of the tree is associated a state $s(i)$ with $s(i) = t(s_L, s_R, o(i))$, with s_L (s_R) the state $s(j)$ of the left (right) child j of i, $s(i) = \Box$ if i has no left (right) child, and $o(i)$ the operation from $\mathcal{O}_k$ with which i is labeled. Note that these states can be computed in bottom-up order (e.g., in postorder) in the tree. We say an automaton *accepts* the labeled tree, if the state of the root belongs to the set S_A.

A property P of graphs (or, of mixed hypergraphs) is *regular*, if for each k, there is a finite state tree automaton M_k, such that for each G: $P(G)$ holds, if and only if M_k accepts all parse trees of G with operations from $\mathcal{O}_k$, if and only if M_k accepts at least one such parse tree. Courcelle's theorem can be stated as follows.

Theorem 1. *[10] A graph property in CMSOL is regular.*

Courcelle conjectured that all regular properties belong to CMSOL; for some special cases, the conjecture has been shown by Kabanets [17] and Kaller [18]. Regularity implies that for each k, there is a linear time algorithm that, given a (mixed hyper)graph G of treewidth at most k, decides if the property holds for G or not: the parse tree with operations in $\mathcal{O}_k$ can be constructed in linear time, and then the automaton M is run in linear time on the parse tree; decide 'yes', if M ends in an accepting state.

We need to look at a minor variant of Theorem 1 for graph expressions with one free vertex set variable, i.e., properties of the form $P(G, W)$, $G = (V, E, A)$ a mixed hyper graph, $W \subseteq V$ a set of vertices. Our finite state tree automaton gets as input a labeled tree, *and* for each node in the tree that gives a new vertex (i.e., one that is not used by any node that is a descendant in the tree) whether this vertex belongs to the vertex set we denote by W. The state transition function gets a fourth argument, that is empty, except for the AddTerminal, CreateArc, and CreateHyperedge$_{k'}$ operations, where it gets a series of 1, 2, or k' booleans telling whether the new vertices belong to set W or not.

Theorem 2. *[10] Let P be an expression in CMSOL with free set variable W. Then P is regular.*

We write as shorthand notation: $\{x_V\} = \{v \in V \mid x_V(v) = \text{true}\}$, i.e., the set of variables with value true in configuration x_V. For a probabilistic network (G, κ) and (regular) property P with free vertex set variable W, we can ask for the probability that $P(G, x_V\})$ holds. Examples of such properties are given in the introduction. We can write:

$$\Pr(P(G, \{x_V\}) = \sum_{W \subseteq V,\ P(G,W)} \Pr(x_V^W) \tag{4}$$

with x_V^W the full configuration with $\{x_V\} = W$. This generalizes (2).

3 Main results

The main result of this paper is given now. It shows that we can combine the results of Courcelle (Theorem 2) and [21]. Note that the treewidth of the moral graph of G is at least the treewidth of G.

Theorem 3. *Let P be a property of (mixed hyper)graphs with one free vertex set variable. Suppose P is regular. For each constant k, there is a linear time algorithm that, when given a probabilistic network (G, κ) with the treewidth of the moral graph of G at most k, computes the probability of $P(G, \{x_V\})$.*

Proof. Let k be fixed. Assume we have the finite state tree automaton $M = (S, S_A, t, \Box)$ for P and $\mathcal{O}_{k+1}$. We first build in linear time a parse tree T with operations in $\mathcal{O}_{k+1}$ for the hypergraph associated with (G, κ). Then, we compute

for each node in T a table called Q_i. These tables can be computed in bottom up order in the tree. Given the table of the root node, the requested value can be computed.

For each node i in T, we denote its associated terminal graph $G_i = (V_i, E_i, A_i, (x_1^i, \ldots, x_{k_i}^i))$, and write $X_i = \{x_1^i, \ldots, x_{k_i}^i\}$. Q_i maps each pair (x_{X_i}, s) to a real value in the interval $[0, 1]$, with x_{X_i} a configuration on X_i, and $s \in S$ a state, such that

$$Q_i(C_{X_i}, s) = \sum \prod_{e \in E_i} \kappa(x_{V_i,e})$$

where we sum over all configurations x_{V_i} on V_i that agree with x_{X_i} and that have the property that when we run machine M on the subtree with root i with $W = \{x_{V_i}\}$ then M gives state s in i. Tables have size $2^{|X_i|} \cdot |S| \leq 2^{k+1} \cdot |S| = O(1)$.

Claim. Let i be a node in the parse tree. Given tables Q_j for all children j of i, we can compute the table Q_i in $O(1)$ time.

Proof. i is labeled with some operation from $\mathcal{O}_{k+1}$. For each type of operation, we must show the claim separately. We consider two more interesting cases, and omit the others here.

Join Suppose i is labeled $\textsc{Join}_{k'}$ and has children j_1 and j_2. $X_i = X_{j_1} = X_{j_2} = V_{j_1} \cap V_{j_2}$. For each configuration x_{X_i} and state s:

$$Q_i(x_{X_i}, s) = \sum Q_{j_1}(x_{X_i}, s_1) \cdot Q_{j_2}(x_{X_i}, s_2) \tag{5}$$

where the sum is taken over all pairs (s_1, s_2) with $t(s_1, s_2, \textsc{Join}_{k'}) = s$. There are $O(1)$ such pairs, and thus each of the $O(1)$ values $Q_i(x_{X_i}, s)$ can be computed in $O(1)$ time.

AddTerminal Suppose j is the child of node i, i is labeled $\textsc{AddTerminal}'_k$, with z the added new vertex. Consider a configuration x_{X_i} and a state s. We look at the case that z has value *true* in x_{X_i}, the other case is similar. Now, one can show

$$Q_i(x_{X_i}, s) = \sum Q_j(x_{X_i - \{z\}}, s') \tag{6}$$

where $x_{X_i - \{z\}}$ is the restriction of x_{X_i} to domain $X_i - \{z\}$, and the sum is taken over all $s' \in S$ with $t(s, \square, \textsc{AddTerminal}'_k) = s$. This implies that Q_i can be computed in $O(1)$ time, given Q_j.

Claim. Let r be the root of the parse tree of G.

$$\Pr(P(G, \{v \mid v = \text{true}\})) = \sum_{s \in S_A} Q_r(x_\emptyset, s)$$

Proof. G_r has 0 terminals, so there is a unique configuration $x_{X_r} = x_\emptyset$.
$\Pr(P(G, \{x_V\}))$ equals

$$\sum \Pr(x_V) = \sum \prod_{e \in E} \kappa(x_{V_i}, e)$$

where the sum is taken over all configurations x_V on $V = V_r$ where $P(G, \{x_V\})$ holds. Each such configuration trivially agrees with $x_\emptyset$. As M is the finite state tree automaton for P, these configurations are exactly those where M accepts when $W = \{x_V\}$. So, the sum equals

$$\sum_{s \in S_A} \sum \prod_{e \in E} \kappa(x_{V_i}, e)$$

where the second sum is taken over the configurations x_V on V that have M end in state s when W is as above. This equals $\sum_{s \in S_A} Q_r(x_\emptyset, s)$.

So, working bottom-up in the parse tree, we can compute in linear time all tables Q_i, and then compute the requested answer in $O(1)$ time using the table of the root of the parse tree.

4 Extensions

We briefly mention a few extensions of the result.

4.1 Non-binary variables

While most results were described in terms of binary variables, the same results hold when variables are non-binary but can attain a number of values bounded by some constant.

4.2 Conditional probabilities

Suppose we want to compute the probability that property P holds, conditional to the observations $x_\mathcal{O}$. The value can be computed with the same method. We use that $\Pr(P(G, x_V)|x_\mathcal{O}) = \Pr(P(G, x_V) \wedge x_\mathcal{O}) / \Pr(x_\mathcal{O})$ and that regularity and CMSOL are closed under conjunction, and compute $\Pr(P(G, x_V) \wedge x_\mathcal{O})$ and $\Pr(x_\mathcal{O})$ separately.

4.3 Different types of edges

To the mixed hypergraph model, we can add additional edges and arcs that are not part of the probabilistic network, but can be used in CMSOL queries on the network. Assuming that we work with bounded width tree decompositions that also fulfill the property that for each of the different types of edges there are bags that contain the endpoints, we can also answer compute the probability of CMSOL queries on properties about the graph formed by these additional edges in linear time.

4.4 Different models

Possibly, the results can also be applied to different probabilistic models, whose structure can be modeled by hypergraphs. See e.g. [13].

5 Conclusions

Examples The results in this paper can be applied to a large number of problems, as many graph properties can be formulated in CMSOL. See e.g., [8] for many examples of CMSOL graph properties. Some examples are:

- 'Suggested causality': for two variables x and y: what is the probability that x and y hold *and* there is a path from x to y of variables that all are true?
- Independence: what is the probability that no true variables are adjacent?
- In graphical games: some vertices represent an agent. Agents have states, which are modeled by a probabilistic network. For a collection of pairs of agents F, we ask: what is the probability that no two pair in F of agents have the same state? Or: can we partition the agents into three (or any other constant) number of groups, such that adjacent agents with the same state belong to a different group?

Time and space considerations The time and space used by the algorithm of Theorem 3 is approximately the product of the time, respectively the space, of the Lauritzen Spiegelhalter algorithm and the number of states of the tree automaton. In some cases this number of states is large, in particular, when the property whose probability is to be computed is NP-hard when the treewidth is not bounded. In some other cases, the number of states can be reasonably small. For instance, when P is the property that an even number of variables has the value true, then a machine with only two states suffices. So, while in some cases, the algorithm implied by Theorem 3 is not practical, in other cases, we get algorithms that are sufficiently efficient.

It is also often possible to use some optimizations that decrease the running time. For instance, many machines will have a state s_r that will never lead to an accepting state later. Here, we can see that we do not need to compute values of the form $Q_i(c, s_r)$ for any node i and any configuration c. (For notation, see the proof of Theorem 3.) More Myhill-Nerode type optimizations (compare [15]) are also possible. For instance, computing the probability that there is at most one true variable needs tables with $\ell + 3$ entries per node in the parse tree, ℓ the treewidth.

Finally, in practical cases, one can try to design more efficient algorithms by hand. The situation can resemble the experiences with the use of Courcelle's theorem: Consider some practical problem P, which can be formulated in monadic second order logic. Courcelle's theorem directly tells us that the problem can be solved in linear time for graphs of bounded treewidth. However, a direct use of the machinery behind the proof of Courcelle's theorem would most probably give an algorithm that is too slow in practice, due to large hidden constants in the

O-notation. However, in most cases, algorithms that are tailor made for problem P will have much better constant factors, and may be practical for small values of treewidth (see e.g. [20].)

A direct application of Courcelle's theorem may also yield an automaton that is too large to effectively build.A recent technique by Courcelle and Durand [11] helps to (partially) overcome such problems. It is interesting to investigate if the techniques from [11] can give practical improvements for implementations of Theorem 3.

If a tree decomposition of bounded width is not given with the input, it is possible to find one in linear time (assuming the width bound is a constant): run the algorithm of [2] on the undirected graph, obtained by dropping directions of edges and replacing hyperedges by a clique. Unfortunately, this algorithm is not practical, even for small values of the treewidth. However, there are good heuristics that often perform very well, see [4, 5] for overviews.

Final conclusions In this paper, we have shown that two famous results from different fields of computer science can be combined: the algorithm for probabilistic inference by Lauritzen and Spiegelhalter, and the result by Courcelle that problems in CMSOL can be solved in linear time when the treewidth of the graph is bounded. The formalism chosen in this paper to present the results may depart from what is sometimes usual; a description in other formalisms (tree decompositions or clique trees) is also possible, but seems to require more clumsy details.

The result allows us to compute the probability of several properties of the network and the values of the variables in linear time. For some properties, the constant factor hidden in the 'O' may yield it impractical, but for other properties, one can expect that the resulting algorithm is indeed sufficiently efficient.

An interesting question is whether other problems that were studied for probabilistic networks have a similar CMSOL variant. Other interesting theoretical questions include:

- Is Inference on probabilistic networks with moral graphs of bounded treewidth solvable in logspace? (Compare [14].)
- Inference is $\#P$-hard [24]. What complexities have computing the probabilities of CMSOL properties on general probabilistic networks (i.e., without bounds on treewidth)?
- Are there other graph problems for which a variant of Theorem 3 holds? One possible direction may be to look at optimization problems, e.g., with the finite integer index property [7]. Also, when we allow polynomial running time instead of linear, it is to be expected that a larger class of problems can be handled.

Acknowledgement

I thank the referees for several very helpful comments, and colleagues from the Decision Support Systems and the Algorithmic Systems groups at the Depart-

ment of Information and Computing Science of Utrecht University for useful discussions.

References

1. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.
2. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
3. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.
4. H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208:259–275, 2010.
5. H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations II. Lower bounds. *Information and Computation*, 209:1103–1119, 2011.
6. H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 6:181–188, 1993.
7. H. L. Bodlaender and B. van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167:86–119, 2001.
8. R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.
9. R. B. Borie, R. G. Parker, and C. A. Tovey. Solving problems on recursively constructed graphs. *ACM Computing Surveys*, 41(4), 2008.
10. B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.
11. B. Courcelle and I. Durand. Fly-automata, their properties and applications. In B. Bouchou-Markhoff, P. Caron, J.-M. Champarnaud, and D. Maurel, editors, *Proceedings of the 16th International Conference on Implementation and Application of Automata, CIAA 2011*, volume 6807 of *Lecture Notes in Computer Science*, pages 264–272. Springer Verlag, 2011.
12. B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109:49–82, 1993.
13. C. Daskalakis and C. H. Papadimitriou. Computing pure Nash equilibria in graphical games via Markov random fields. In J. Feigenbaum, J. C.-I. Chuang, and D. M. Pennock, editors, *Proceedings 7th ACM Conference on Electronic Commerce EC-2006*, pages 91–99. ACM, 2006.
14. M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. pages 143–152, 2010.
15. M. R. Fellows and M. A. Langston. An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science, FOCS'89*, pages 520–525, 1989.
16. F. V. Jensen and T. D. Nielsen. *Bayesian Networks and Decision Graphs (2nd ed.)*. Information Science and Statistics series. Springer Verlag, 2007.
17. V. Kabanets. Recognizability equals definability for partial k-paths. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proceedings of the 24th International Colloquium on Automata, Languages and Programming, ICALP'97*, pages 805–815. Springer Verlag, Lecture Notes in Computer Science, vol. 1256, 1997.

18. D. Kaller. Definability equals recognizability of partial 3-trees and k-connected partial k-trees. *Algorithmica*, 27:348–381, 2000.
19. T. Kloks. *Treewidth. Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1994.
20. A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40(3):170–180, 2002.
21. S. J. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *The Journal of the Royal Statistical Society. Series B (Methodological)*, 50:157–224, 1988.
22. J. Pearl. *Probablistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Palo Alto, 1988.
23. N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.
24. D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82:273–302, 1996.
25. L. C. van der Gaag. *Probability-Based Models for Plausible Reasoning*. PhD thesis, University of Amsterdam, 1990.
26. T. V. Wimer. *Linear Algorithms on k-Terminal Graphs*. PhD thesis, Dept. of Computer Science, Clemson University, 1987.