Towards Preventing Unnecessary Groundings in the Lifted Dynamic Junction Tree Algorithm*

Marcel Gehrke, Tanya Braun, and Ralf Möller

Institute of Information Systems, University of Lübeck, Germany
{gehrke, braun, moeller}@ifis.uni-luebeck.de

Abstract. The lifted dynamic junction tree algorithm (LDJT) answers filtering and prediction queries efficiently for probabilistic relational temporal models by building and then reusing a first-order cluster representation of a knowledge base for multiple queries and time steps. Unfortunately, a non-ideal elimination order can lead to unnecessary groundings.

1 Introduction

Areas like healthcare, logistics or even scientific publishing deal with probabilistic data with relational and temporal aspects and need efficient exact inference algorithms. These areas involve many objects in relation to each other with changes over time and uncertainties about object existence, attribute value assignments, or relations between objects. More specifically, publishing involves publications (relational) for many authors (objects), streams of papers over time (temporal), and uncertainties for example due to missing information. For query answering, our approach performs deductive reasoning by computing marginal distributions at discrete time steps. In this paper, we study the problem of exact inference and investigate unnecessary groundings can occur in temporal probabilistic models.

We propose parameterised probabilistic dynamic models (PDMs) to represent probabilistic relational temporal behaviour and introduce the lifted dynamic junction tree algorithm (LDJT) to exactly answer multiple filtering and prediction queries for multiple time steps efficiently [5]. LDJT combines the advantages of the interface algorithm [10] and the lifted junction tree algorithm (LJT) [2]. Poole [12] introduces parametric factor graphs as relational models and proposes lifted variable elimination (LVE) as an exact inference algorithm on relational models. Further, de Salvo Braz [14], Milch et al. [8], and Taghipour et al. [15] extend LVE to its current form. Lauritzen and Spiegelhalter [7] introduce the junction tree algorithm. To benefit from the ideas of the junction tree algorithm and LVE, Braun and Möller [2] present LJT, which efficiently performs exact first-order probabilistic inference on relational models given a set of queries. Specifically, this paper shows that a non-ideal elimination order can lead to groundings even though a lifted run is possible for a model. LDJT reuses

* This research originated from the Big Data project being part of Joint Lab 1, funded by Cisco Systems Germany, at the centre COPICOH, University of Lübeck

an first-order junction tree (FO jtree) structure to answer multiple queries and reuses the structure to answer queries for all time steps $t > 0$. Unfortunately, due to a non-ideal elimination order unnecessary groundings can occur.

Most inference approaches for relational temporal models are approximative. Additional to being approximative, these approaches involve unnecessary groundings or are only designed to handle single queries efficiently. Ahmadi et al. [1] propose lifted (loopy) belief propagation. From a factor graph, they build a compressed factor graph and apply lifted belief propagation with the idea of the factored frontier algorithm [9], which is an approximate counterpart to the interface algorithm. Thon et al. [16] introduce CPT-L, a probabilistic model for sequences of relational state descriptions with a partially lifted inference algorithm. Geier and Biundo [6] present an online interface algorithm for dynamic Markov logic networks (DMLNs), similar to the work of Papai et al. [11]. Both approaches slice DMLNs to run well-studied static MLN [13] inference algorithms on each slice individually. Vlasselaer et al. [18,17] introduce an exact approach, which involves computing probabilities of each possible interface assignment.

The remainder of this paper has the following structure: We introduce PDMs as a representation for relational temporal probabilistic models and present LDJT, an efficient reasoning algorithm for PDMs. Afterwards, we show how unnecessary groundings can occur and conclude by looking at extensions.

2 Parameterised Probabilistic Dynamic Models

Parameterised probabilistic models (PMs) combine first-order logic, using logical variables (logvars) as parameters, with probabilistic models [4].

Definition 1. *Let* $\mathbf{L}$ *be a set of logvar names,* Φ *a set of factor names, and* $\mathbf{R}$ *a set of random variable (randvar) names. A parameterised randvar (PRV)* $A = P(X^1, ..., X^n)$ *represents a set of randvars behaving identically by combining a randvar* $P \in \mathbf{R}$ *with* $X^1, ..., X^n \in \mathbf{L}$*. If* $n = 0$*, the PRV is parameterless. The domain of a logvar* L *is denoted by* $\mathcal{D}(L)$*. The term* $range(A)$ *provides possible values of a PRV* A*. Constraint* $(\mathbf{X}, C_{\mathbf{X}})$ *allows to restrict logvars to certain domain values and is a tuple with a sequence of logvars* $\mathbf{X} = (X^1, ..., X^n)$ *and a set* $C_{\mathbf{X}} \subseteq \times_{i=1}^{n} \mathcal{D}(X^i)$*.* $\top$ *denotes that no restrictions apply and may be omitted. The term* $lv(Y)$ *refers to the logvars in some element* Y*. The term* $gr(Y)$ *denotes the set of instances of* Y *with all logvars in* Y *grounded w.r.t. constraints.*

Let us set up a PM for publications on some topic. We model that the topic may be hot, conferences are attractive, people do research, and publish in publications. From $\mathbf{R} = \{Hot, DoR\}$ and $\mathbf{L} = \{A, P, X\}$ with $\mathcal{D}(A) = \{a_1, a_2\}$, $\mathcal{D}(P) = \{p_1, p_2\}$, and $\mathcal{D}(X) = \{x_1, x_2, x_3\}$, we build the boolean PRVs Hot and $DoR(X)$. With $C = (X, \{x_1, x_2\})$, $gr(DoR(X)|C) = \{DoR(x_1), DoR(x_2)\}$.

Definition 2. *We denote a parametric factor (parfactor)* g *with* $\forall \mathbf{X} : \phi(\mathcal{A}) \,|C$*.* $\mathbf{X} \subseteq \mathbf{L}$ *being a set of logvars over which the factor generalises and* $\mathcal{A} = (A^1, ..., A^n)$ *a sequence of PRVs. We omit* $(\forall \mathbf{X} :)$ *if* $\mathbf{X} = lv(\mathcal{A})$*. A function*

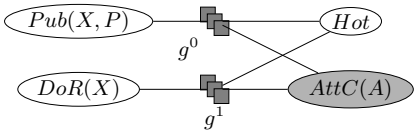

Fig. 1. Parfactor graph for G^{ex}

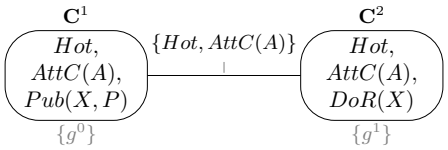

Fig. 2. FO jtree for G^{ex} (local models in grey)

$\phi : \times_{i=1}^{n} range(A^i) \mapsto \mathbb{R}^+$ with name $\phi \in \Phi$ is defined identically for all grounded instances of $\mathcal{A}$. A list of all input-output values is the complete specification for ϕ. C is a constraint on $\mathbf{X}$. A PM $G := \{g^i\}_{i=0}^{n-1}$ is a set of parfactors and semantically represents the full joint probability distribution $P_G = \frac{1}{Z} \prod_{f \in gr(G)} f$ where Z is a normalisation constant.

Adding boolean PRVs $Pub(X, P)$ and $AttC(A)$, $G_{ex} = \{g^i\}_{i=0}^{1}$, $g^0 = \phi^0(Pub(X, P), AttC(A), Hot) \mid \top$, $g^1 = \phi^1(DoR(X), AttC(A), Hot) \mid \top$ forms a model. All parfactors have eight input-output pairs (omitted). Figure 1 depicts G^{ex} with four variable nodes for the PRVs and two factor nodes for g^0 and g^1 with edges to the PRVs involved. Additionally, we can observe the attractiveness of conferences. The remaining PRVs are latent.

The semantics of a model is given by grounding and building a full joint distribution. In general, queries ask for a probability distribution of a randvar using a model's full joint distribution and fixed events as evidence.

Definition 3. *Given a PM G, a ground PRV Q and grounded PRVs with fixed range values $\mathbf{E} = \{E^i = e^i\}_i$, the expression $P(Q|\mathbf{E})$ denotes a query w.r.t. P_G.*

To define PDMs, we use PMs and the idea of how Bayesian networks give rise to dynamic Bayesian networks [5]. We define PDMs based on the first-order Markov assumption. Further, the underlying process is stationary.

Definition 4. *A PDM is a pair of PMs $(G_0, G_\rightarrow)$ where G_0 is a PM representing the first time step and $G_\rightarrow$ is a two-slice temporal parameterised model representing $\mathbf{A}_{t-1}$ and $\mathbf{A}_t$ where $\mathbf{A}_\pi$ is a set of PRVs from time slice π.*

Figure 3 shows how the model G^{ex} behaves over time. $G^{ex}_{\rightarrow}$ consists of G^{ex} for time step $t-1$ and for time step t with inter-slice parfactor for the behaviour over time. In this example, the parfactor g^H is the inter-slice parfactors.

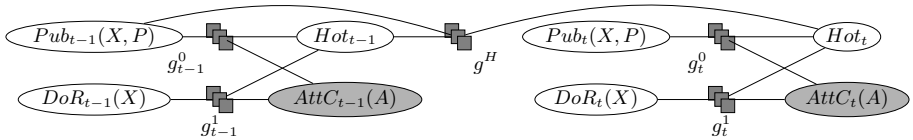

Fig. 3. $G^{ex}_{\rightarrow}$ the two-slice temporal parfactor graph for model G^{ex}

Definition 5. *Given a PDM G, a ground PRV Q_t and grounded PRVs with fixed range values $\mathbf{E}_{0:t} = \{E_t^i = e_t^i\}_{i,t}$, $P(Q_t|\mathbf{E}_{0:t})$ denotes a query w.r.t. P_G.*

The problem of answering a marginal distribution query $P(A_\pi^i|\mathbf{E}_{0:t})$ w.r.t. the model is called *prediction* for $\pi > t$ and *filtering* for $\pi = t$.

3 Lifted Dynamic Junction Tree Algorithm

To provide means to answer queries for PMs, we introduce LJT, mainly based on [3]. Afterwards, we present LDJT [5] consisting of FO jtree constructions for a PDM and a *filtering* and *prediction* algorithm.

3.1 Lifted Junction Tree Algorithm

LJT provides efficient means to answer queries $P(\mathbf{Q}|\mathbf{E})$, with a set of query terms, given a PM G and evidence $\mathbf{E}$, by performing the following steps: (i) Construct an FO jtree J for G. (ii) Enter $\mathbf{E}$ in J. (iii) Pass messages. (iv) Compute answer for each query $Q^i \in \mathbf{Q}$. We first define an FO jtree and then go through each step. To define an FO jtree, we need to define parameterised clusters (parclusters), the nodes of an FO jtree.

Definition 6. *A parcluster $\mathbf{C}$ is defined by $\forall \mathbf{L} : \mathbf{A}|C$. $\mathbf{L}$ is a set of logvars, $\mathbf{A}$ is a set of PRVs with $lv(\mathbf{A}) \subseteq \mathbf{L}$, and C a constraint on $\mathbf{L}$. We omit $(\forall \mathbf{L} :)$ if $\mathbf{L} = lv(\mathbf{A})$. A parcluster $\mathbf{C}^i$ can have parfactors $\phi(\mathcal{A}^\phi)|C^\phi$ assigned given that (i) $\mathcal{A}^\phi \subseteq \mathbf{A}$, (ii) $lv(\mathcal{A}^\phi) \subseteq \mathbf{L}$, and (iii) $C^\phi \subseteq C$ holds. We call the set of assigned parfactors a local model G^i.*
An FO jtree for a model G is $J = (\mathbf{V}, \mathbf{E})$ where J is a cycle-free graph, the nodes $\mathbf{V}$ denote a set of parcluster, and the set $\mathbf{E}$ edges between parclusters. An FO jtree must satisfy the following properties: (i) A parcluster $\mathbf{C}^i$ is a set of PRVs from G. (ii) For each parfactor $\phi(\mathcal{A})|C$ in G, $\mathcal{A}$ must appear in some parcluster $\mathbf{C}^i$. (iii) If a PRV from G appears in two parclusters $\mathbf{C}^i$ and $\mathbf{C}^j$, it must also appear in every parcluster $\mathbf{C}^k$ on the path connecting nodes i and j in J. The separator $\mathbf{S}^{ij}$ of edge $i - j$ is given by $\mathbf{C}^i \cap \mathbf{C}^j$ containing shared PRVs.

LJT constructs an FO jtree using a first-order decomposition tree (FO dtree), enters evidence in the FO jtree, and passes messages through an *inbound* and an *outbound* pass, to distribute local information of the nodes through the FO jtree. To compute a message, LJT eliminates all non-seperator PRVs from the parcluster's local model and received messages. After message passing, LJT answers queries. For each query, LJT finds a parcluster containing the query term and sums out all non-query terms in its local model and received messages.

Figure 2 shows an FO jtree of G^{ex} with the local models of the parclusters and the separators as labels of edges. During the *inbound* phase of message passing, LJT sends messages from $\mathbf{C}^1$ to $\mathbf{C}^2$ and for the *outbound* phase a message from $\mathbf{C}^2$ to $\mathbf{C}^1$. If we want to know whether Hot holds, we query for $P(Hot)$ for which LJT can use either parcluster $\mathbf{C}^1$ or $\mathbf{C}^2$. Thus, LJT can sum out $AttC(A)$ and $DoR(X)$ from $\mathbf{C}^2$'s local model G^2, $\{g^1\}$, combined with the received messages.

3.2 LDJT: Overview

LDJT efficiently answers queries $P(\mathbf{Q}_t|\mathbf{E}_{0:t})$, with a set of query terms $\{\mathbf{Q}_t\}_{t=0}^{T}$, given a PDM G and evidence $\{\mathbf{E}_t\}_{t=0}^{T}$, by performing the following steps: (i) Construct offline two FO jtrees J_0 and J_t with *in-* and *out-clusters* from G. (ii) For $t = 0$, using J_0 to enter $\mathbf{E}_0$, pass messages, answer each query term $Q_\pi^i \in \mathbf{Q}_0$, and preserve the state. (iii) For $t > 0$, instantiate J_t for the current time step t, recover the previous state, enter $\mathbf{E}_t$ in J_t, pass messages, answer each query term $Q_\pi^i \in \mathbf{Q}_t$, and preserve the state.

Next, we show how LDJT constructs the FO jtrees J_0 and J_t with *in-* and *out-clusters*, which contain a minimal set of PRVs to m-separate the FO jtrees. M-separation means that information about these PRVs make FO jtrees independent from each other. Afterwards, we present how LDJT connects the FO jtrees for reasoning to solve the *filtering* and *prediction* problems efficiently.

3.3 LDJT: FO Jtree Construction for PDMs

LDJT constructs FO jtrees for G_0 and $G_\rightarrow$, both with an incoming and outgoing interface. To be able to construct the interfaces in the FO jtrees, LDJT uses the PDM G to identify the interface PRVs $\mathbf{I}_t$ for a time slice t.

Definition 7. *The forward interface is defined as* $\mathbf{I}_t = \{A_t^i \mid \exists \phi(\mathcal{A}) | C \in G : A_t^i \in \mathcal{A} \wedge \exists A_{t+1}^j \in \mathcal{A}\}$*, i.e., the PRVs which have successors in the next slice.*

For $G_\rightarrow^{ex}$, which is shown in Fig. 3, PRVs Hot_{t-1} and $Pub_{t-1}(X, P)$ have successors in the next time slice, making up $\mathbf{I}_{t-1}$. To ensure interface PRVs $\mathbf{I}$ ending up in a single parcluster, LDJT adds a parfactor g^I over the interface to the model. Thus, LDJT adds a parfactor g_0^I over $\mathbf{I}_0$ to G_0, builds an FO jtree J_0 and labels the parcluster with g_0^I from J_0 as *in-* and *out-cluster*. For $G_\rightarrow$, LDJT removes all non-interface PRVs from time slice $t-1$, adds parfactors g_{t-1}^I and g_t^I, constructs J_t, and labels the parcluster containing g_{t-1}^I as *in-cluster* and the parcluster containing g_t^I as *out-cluster*.

The interface PRVs are a minimal required set to m-separate the FO jtrees. LDJT uses these PRVs as separator to connect the *out-cluster* of J_{t-1} with the *in-cluster* of J_t, allowing to reusing the structure of J_t for all $t > 0$.

3.4 LDJT: Proceeding in Time with the FO Jtree Structures

Since J_0 and J_t are static, LDJT uses LJT as a subroutine by passing on a constructed FO jtree, queries, and evidence for step t to handle evidence entering, message passing, and query answering using the FO jtree. Further, for proceeding to the next time step, LDJT calculates an α_t message over the interface PRVs using the *out-cluster* to preserve the information about the current state. Afterwards, LDJT increases t by one, instantiates J_t, and adds α_{t-1} to the *in-cluster* of J_t. During message passing, α_{t-1} is distributed through J_t.

Figure 4 depicts how LDJT uses the interface message passing between time step three to four. First, LDJT sums out the non-interface PRV $AttC_3(A)$ from

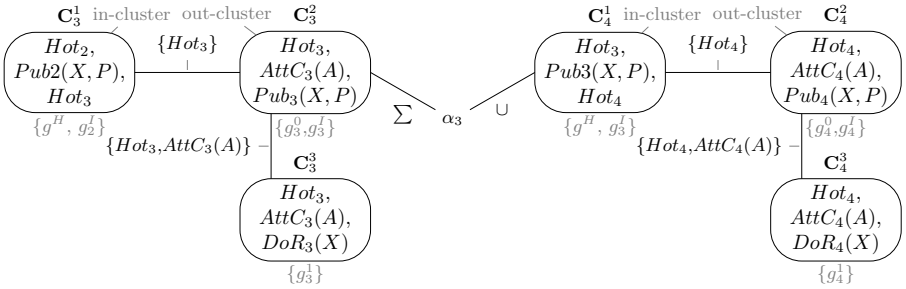

Fig. 4. Forward pass of LDJT (local models and labeling in grey)

$\mathbf{C}_3^2$'s local model and the received messages and saves the result in message α_3. After increasing t by one, LDJT adds α_3 to the *in-cluster* of J_4, $\mathbf{C}_4^1$. α_3 is then distributed by message passing and accounted for during calculating α_4.

4 Unnecessary Groundings in LDJT

Unnecessary groundings have a huge impact on temporal models, as groundings during message passing can propagate through the complete model. LDJT has an intra and inter FO jtree message passing phase. Intra FO jtree message passing takes place inside of an FO jtree for one time step. Inter FO jtree message passing takes place between two FO jtrees. To prevent groundings during intra FO jtree message passing, LJT successfully proposes to fuse parclusters [3]. Unfortunately, having two FO jtrees, LDJT cannot fuse parclusters from different FO jtrees. Hence, LDJT requires a different approach to prevent unnecessary groundings during inter FO jtree message passing.

Let us now have a look at Fig. 4 to understand inter FO jtree message pass can induce unnecessary groundings due to the elimination order. Fig. 4 shows J_t instantiated for time step 3 and 4. To compute α_3, LDJT eliminates $AttC_3(A)$ from $\mathbf{C}_3^2$'s local model. The elimination of $AttC_3(A)$ leads to groundings, as $AttC_3(A)$ does not contain all logvars, X and P are missing. Additionally, $AttC_3(A)$ is not count-convertible. Assuming $AttC_3(A)$ would also be included in the parcluster $\mathbf{C}_4^1$, LDJT would not need to eliminate $AttC_3(A)$ in $\mathbf{C}_3^2$ anymore and therefore calculating α_3 would not lead to groundings. Therefore, the elimination order can lead to unnecessary groundings.

5 Conclusion

We present the need to prevent unnecessary groundings in LDJT by changing the elimination order. We currently work on an approach to prevent unnecessary groundings, as well as extending LDJT to also calculate the most probable explanation. Other interesting future work includes a tailored automatic learning for PDMs, parallelisation of LJT, and improved evidence entering.

References

1. Ahmadi, B., Kersting, K., Mladenov, M., Natarajan, S.: Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. Machine learning 92(1), 91–132 (2013)
2. Braun, T., Möller, R.: Lifted Junction Tree Algorithm. In: Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz). pp. 30–42. Springer (2016)
3. Braun, T., Möller, R.: Preventing Groundings and Handling Evidence in the Lifted Junction Tree Algorithm. In: Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz). pp. 85–98. Springer (2017)
4. Braun, T., Möller, R.: Counting and Conjunctive Queries in the Lifted Junction Tree Algorithm. In: Postproceedings of the 5th International Workshop on Graph Structures for Knowledge Representation and Reasoning. Springer (2018)
5. Gehrke, M., Braun, T., Möller, R.: Lifted Dynamic Junction Tree Algorithm. In: Proceedings of the 23rd International Conference on Conceptual Structures. Springer (2018)
6. Geier, T., Biundo, S.: Approximate Online Inference for Dynamic Markov Logic Networks. In: Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI). pp. 764–768. IEEE (2011)
7. Lauritzen, S.L., Spiegelhalter, D.J.: Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems. Journal of the Royal Statistical Society. Series B (Methodological) pp. 157–224 (1988)
8. Milch, B., Zettlemoyer, L.S., Kersting, K., Haimes, M., Kaelbling, L.P.: Lifted Probabilistic Inference with Counting Formulas. In: Proceedings of AAAI. vol. 8, pp. 1062–1068 (2008)
9. Murphy, K., Weiss, Y.: The Factored Frontier Algorithm for Approximate Inference in DBNs. In: Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence. pp. 378–385. Morgan Kaufmann Publishers Inc. (2001)
10. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. Ph.D. thesis, University of California, Berkeley (2002)
11. Papai, T., Kautz, H., Stefankovic, D.: Slice Normalized Dynamic Markov Logic Networks. In: Proceedings of the Advances in Neural Information Processing Systems. pp. 1907–1915 (2012)
12. Poole, D.: First-order probabilistic inference. In: Proceedings of IJCAI. vol. 3, pp. 985–991 (2003)
13. Richardson, M., Domingos, P.: Markov Logic Networks. Machine learning 62(1), 107–136 (2006)
14. de Salvo Braz, R.: Lifted First-Order Probabilistic Inference. Ph.D. thesis, Ph. D. Dissertation, University of Illinois at Urbana Champaign (2007)
15. Taghipour, N., Fierens, D., Davis, J., Blockeel, H.: Lifted Variable Elimination: Decoupling the Operators from the Constraint Language. Journal of Artificial Intelligence Research 47(1), 393–439 (2013)
16. Thon, I., Landwehr, N., De Raedt, L.: Stochastic relational processes: Efficient inference and applications. Machine Learning 82(2), 239–272 (2011)
17. Vlasselaer, J., Van den Broeck, G., Kimmig, A., Meert, W., De Raedt, L.: TP-Compilation for Inference in Probabilistic Logic Programs. International Journal of Approximate Reasoning 78, 15–32 (2016)
18. Vlasselaer, J., Meert, W., Van den Broeck, G., De Raedt, L.: Efficient Probabilistic Inference for Dynamic Relational Models. In: Proceedings of the 13th AAAI

Conference on Statistical Relational AI. pp. 131–132. AAAIWS'14-13, AAAI Press (2014)