Lifted Temporal Most Probable Explanation[⋆]

Marcel Gehrke[0000−0001−9056−7673], Tanya Braun[0000−0003−0282−4284], and Ralf Möller

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

Abstract. The lifted dynamic junction tree algorithm (LDJT) answers filtering and prediction queries efficiently for temporal probabilistic relational models by building and then reusing a first-order cluster representation of a knowledge base for multiple queries and time steps. Another type of query asks for a most probable explanation (MPE) for given events. Specifically, this paper contributes (i) LDJT^{mpe} to efficiently solve the temporal MPE problem for temporal probabilistic relational models and (ii) a combination of LDJT and LDJT^{mpe} to efficiently answer assignment queries for a given number of time steps.

Keywords: Relational Temporal Probabilistic Models, Lifting, MPE, MAP

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 (the relational part) for many authors (the objects), streams of papers over time (the temporal part), and uncertainties for example due to missing or incomplete information. By performing model counting, probabilistic databases (PDBs) can answer queries for relational temporal models with uncertainties [7,8]. However, each query embeds a process behaviour, resulting in huge queries with possibly redundant information. In contrast to PDBs, we build more expressive and compact models including behaviour (offline) enabling efficient answering of more compact queries (online). For query answering, our approach performs deductive reasoning by computing marginal distributions at discrete time steps. In this paper, we study the problem of finding a most probable explanation (MPE) in temporal probabilistic relational models.

We [9] propose parameterised probabilistic dynamic models (PDMs) to represent probabilistic relational temporal behaviour and introduce the lifted dynamic

⋆ 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

junction tree algorithm (LDJT) to exactly answer multiple filtering and prediction queries for multiple time steps efficiently. LDJT combines the advantages of the interface algorithm [15] and the lifted junction tree algorithm (LJT) [3]. Specifically, this paper contributes (i) LDJT^{mpe} to efficiently solve the temporal MPE problem for relational temporal probabilistic models and (ii) a combination of LDJT and LDJT^{mpe} to efficiently answer assignment queries, that is to say the most probable assignment for a subset of random variable (randvar).

LDJT reuses an first-order junction tree (FO jtree) structure to answer multiple queries and reuses the structure to answer queries for all time steps $t > 0$. Additionally, LDJT ensures a minimal exact inter FO jtree information propagation between time steps. In the static case, LJT^{mpe} already solves the MPE problem for relational models efficiently. We propose to combine the advances in LDJT and LJT^{mpe} to also answer temporal assignment queries efficiently.

The remainder of this paper has the following structure: We begin by recapitulating PDMs as a representation for relational temporal probabilistic models and present LDJT, an efficient reasoning algorithm for PDMs. Afterwards, we present how LJT answers queries for the MPE and LDJT^{mpe} to answer queries for the MPE in relational temporal probabilistic models. Lastly, we investigate the advantages of LDJT and LDJT^{mpe} compared to LJT and LJT^{mpe} in more depth. We conclude by looking at possible extensions.

2 Related Work

We take a look at inference for temporal propositional models, static relational models, and give an overview about research on temporal relational models.

For exact inference on temporal propositional models, a naive approach is to unroll the temporal model for a given number of time steps and use any exact inference algorithm for static, i.e., non-temporal, models. Murphy [15] proposes the interface algorithm consisting of a forward and backward pass using temporal d-separation to apply static inference algorithms to the dynamic model. For propositional temporal models, the Viterbi algorithm is one approach to calculate the MPE for events over time. Additionally, Murphy [15] presents that given a variable elimination algorithm, one can solve the Viterbi problem by calculating max-products instead of sum-products.

First-order probabilistic inference leverages the relational aspect of a static model. For models with known domain size, it exploits symmetries in a model by combining instances to reason with representatives, known as lifting [18]. Poole [18] 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 [20], Milch et al. [14], and Taghipour et al. [23] extend LVE to its current form. Furthermore, there are versions of LVE to compute an MPE [21,2,5]. Lauritzen and Spiegelhalter [12] introduce the junction tree algorithm. To benefit from the ideas of the junction tree algorithm and LVE, Braun and Möller [3] present LJT, which efficiently performs exact first-order probabilistic inference on relational models given a set of queries. Additionally, Braun and Möller [5] present how to solve the MPE problem efficiently with LJT^{mpe} and how a combination of LJT and LJT^{mpe} can answer assignment queries for a subset of randvars.

To handle inference for temporal relational models most approaches are approximate. Additionally to being approximate, these approaches involve unnecessary groundings or are not designed to handle multiple queries efficiently. Ahmadi et al. [1] propose lifted belief propagation for dynamic Markov logic networks (DMLNs). Thon et al. [24] introduce CPT-L, a probabilistic model for sequences of relational state descriptions with a partially lifted inference algorithm. Geier and Biundo [11] present an online interface algorithm for DMLNs, similar to the work of Papai et al. [17]. Both approaches slice DMLNs to run well-studied MLN inference algorithms [19] on each slice. Two ways of performing online inference using particle filtering are described in [13,16]. Vlasselaer et al. [26,25] introduce an exact approach for temporal relational models, but perform inference on a ground knowledge base.

However, by using efficient inference algorithms, we calculate exact solutions for temporal relational models. Therefore, we extend LDJT, which leverages the well-studied LVE and LJT algorithms, to also answer assignment queries.

3 Parameterised Probabilistic Models

Based on [6], we present parameterised probabilistic models (PMs) for relational static models. Afterwards, we extend PMs to the temporal case, resulting in PDMs for relational temporal models, which, in turn, are based on [9].

3.1 Parameterised Probabilistic Models

PMs combine first-order logic with probabilistic models, representing first-order constructs using logical variables (logvars) as parameters.

Definition 1. *Let $\mathbf{L}$ be a set of logvar names, Φ a set of factor names, and $\mathbf{R}$ a set of 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, people do research, attend conferences and publish in publications. From $\mathbf{R} = \{Hot, DoR\}$ and $\mathbf{L} = \{P, X\}$ with $\mathcal{D}(P) = \{p_1, p_2\}$ and $\mathcal{D}(X) = \{x_1, x_2, x_3\}$, we build the boolean PRVs Hot and $DoR(X)$. With $C = (X, \{x_1, x_2\})$, $gr(DoR(X)|C) = \{DoR(x_1), DoR(x_2)\}$.

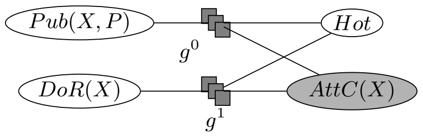

Fig. 1. Parfactor graph for G^{ex}

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 $\phi : \times_{i=1}^{n} range(A^i) \mapsto \mathbb{R}^+$ 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 $P(G) = \frac{1}{Z} \prod_{f \in gr(G)} \phi(\mathcal{A}_f)$ where Z is a normalisation constant.*

Adding boolean PRVs $Pub(X,P)$ and $AttC(X)$, $G_{ex} = \{g^i\}_{i=0}^{1}$, $g^0 = \phi^0(Pub(X,P), AttC(X), Hot)$, $g^1 = \phi^1(DoR(X), AttC(X), Hot)$ forms a model. All parfactors have eight input-output pairs (omitted). Constraints are $\top$, i.e., the ϕ's hold for all domain values. E.g., $gr(g^1)$ contains three factors with identical ϕ. Figure 1 depicts G^{ex} as a graph with four variable nodes for the PRVs and two factor nodes for g^0 and g^1 with edges to the PRVs involved.

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 $\mathbf{E}$, the expression $P(Q|\mathbf{E})$ denotes a query w.r.t. $P(G)$.*

3.2 Parameterised Probabilistic Dynamic Models

To define PDMs, we use PMs and the idea of how Bayesian networks give rise to dynamic Bayesian networks. We define PDMs based on the first-order Markov assumption, i.e., a time slice t only depends on the previous time slice $t-1$. Further, the underlining process is stationary, i.e., the model behaviour does not change over time. Now, we can define PDMs.

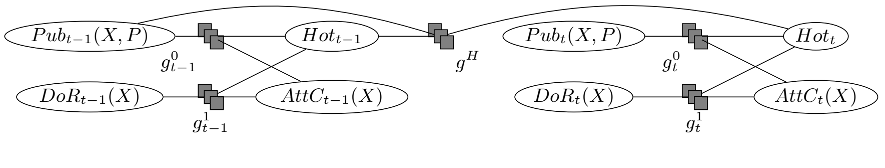

Fig. 2. $G^{ex}_{\rightarrow}$ the two-slice temporal parfactor graph for model G^{ex}

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 2 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.

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

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

4 Lifted Dynamic Junction Tree Algorithm

To provide means to answer queries for PMs, we recapitulate LJT, mainly based on [4]. Afterwards, we present LDJT [9] consisting of FO jtree constructions for a PDM and a filtering and prediction algorithm.

4.1 Lifted Junction Tree Algorithm

LJT provides efficient means to answer queries $P(Q^i|\mathbf{E})$, with a set of query terms $\mathbf{Q}$, 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, 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-separator PRVs from the parcluster's local model and received messages.

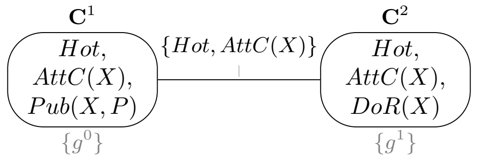

Fig. 3. FO jtree for G^{ex} (local models in grey)

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 3 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 $AttC(X)$ and $DoR(X)$ from $\mathbf{C}^2$'s local model G^2, $\{g^1\}$, combined with the received message.

4.2 LDJT: Overview

LDJT efficiently answers queries $P(\mathbf{Q}^i_\pi|\mathbf{E}_{0:t})$, 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 $Q^i_\pi \in \mathbf{Q}_0$, 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 $Q^i_\pi \in \mathbf{Q}_t$, 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.

4.3 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, i.e., the PRVs which have successors in the next slice.

Definition 7. *The forward interface is defined as:* $\mathbf{I}_{t-1} = \{A^i_t \mid \exists\phi(\mathcal{A})|C \in G : A^i_{t-1} \in \mathcal{A} \wedge \exists A^j_t \in \mathcal{A}\}$.

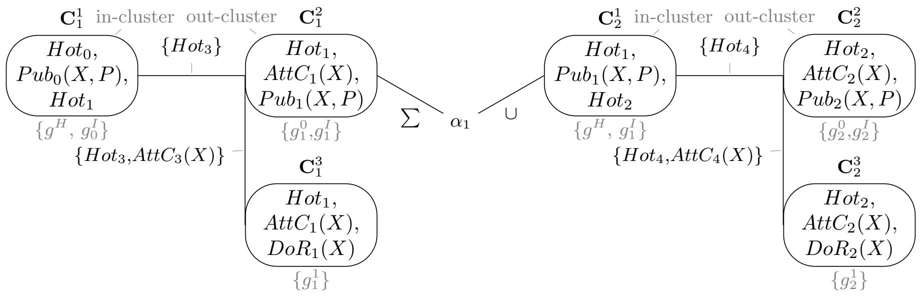
$\mathbf{C}_1^1$ in-cluster out-cluster $\mathbf{C}_1^2$
Hot_0, $Pub_0(X,P)$, Hot_1
$\{g^H, g_0^I\}$
$\{Hot_3\}$
Hot_1, $AttC_1(X)$, $Pub_1(X,P)$
$\{g_1^0, g_1^I\}$
$\{Hot_3, AttC_3(X)\}$
$\mathbf{C}_1^3$
Hot_1, $AttC_1(X)$, $DoR_1(X)$
$\{g_1^1\}$
$\sum$ α_1 $\cup$
$\mathbf{C}_2^1$ in-cluster out-cluster $\mathbf{C}_2^2$
Hot_1, $Pub_1(X,P)$, Hot_2
$\{g^H, g_1^I\}$
$\{Hot_4\}$
Hot_2, $AttC_2(X)$, $Pub_2(X,P)$
$\{g_2^0, g_2^I\}$
$\{Hot_4, AttC_4(X)\}$
$\mathbf{C}_2^3$
Hot_2, $AttC_2(X)$, $DoR_2(X)$
$\{g_2^1\}$

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

For $G_{\rightarrow}^{ex}$, which is shown in Fig. 2, 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, and constructs J_t. Further, LDJT labels the parcluster containing g_{t-1}^I as *in-cluster* and labels 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 for reusing the structure of J_t for all $t > 0$.

4.4 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 time 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 one and two. First, LDJT sums out the non-interface PRV $AttC_1(X)$ from $\mathbf{C}_1^2$'s local model and the received messages and saves the result in message α_1. After increasing t by one, LDJT adds α_1 to the *in-cluster* of J_2, $\mathbf{C}_2^1$. α_1 is then distributed by message passing and accounted for when calculating α_2.

Following this procedure, LDJT can answer *filtering* and *prediction* queries. However, currently LDJT cannot answer *assignment* queries.

5 Most Probable Assignments in LJT

Let us first have a look at assignment queries and then we provide an intuition how LJT^{mpe} solves the problem based on [5].

Definition 8. *Given a PM G and evidence $\mathbf{E}$, we are interested in the most probable assignment for all PRVs $\mathbf{V}$ in G without evidence. Thus, to solve an MPE, LJT^{mpe} computes $\arg\max_{\mathbf{V}} P(\mathbf{V}|\mathbf{E})$.*

The basic idea of calculating an MPE, compared to answering marginal distribution queries, is to use a maximisation instead of a summation to eliminate PRVs. To efficiently calculate a maximisation for relational probabilistic static models, Braun and Möller [5] propose a lifted maximisation for the current version of LVE [23] and also apply it to LJT [3].

As LJT calculates a lifted solution to the MPE problem, LJT^{mpe} uses the fact that instances behave the same for assignments. Thus, for a PRV, LJT^{mpe} only needs to store the number of instances for the range values of that PRV, which maximise the potential. Assume a MPE is that two people are doing research and one does not do research. Then, it is the same if either *alice* and *bob*, *alice* and *eve*, or *bob* and *eve* do research. Hence, LJT^{mpe} only stores a histogram encoding that two map to true and one maps to false.

To compute an MPE, LJT^{mpe} does not only need to store assignments for PRVs, but also potentials, as we want the assignment that maximises the potential. Thus, to calculate an MPE, the function ϕ in a parfactor $\phi(\mathcal{A})|C$ maps arguments to a pair of a potential and a set of histograms for already maxed out PRVs. By storing the potential and a set of histograms, LJT^{mpe} can directly read out the MPE after the last maxing out. For example after the last maxing out, a parfactor could like like this: $\phi() \rightarrow (p, [6,0]_P, [3,0]_D, [3,0]_A, [1,0]_H)$.

Knowing what parfactors store computing an MPE, we now take a look at how LJT^{mpe} answers MPE queries efficiently. Algorithm 1 outlines the steps LJT^{mpe} performs. The first two steps are the same steps as for marginal queries. LJT^{mpe} first builds a corresponding FO jtree J from a PM G and enters evidence afterwards. The main difference compared to marginal queries is that LJT^{mpe} only performs an *inbound* message pass. LJT performs a complete message pass for marginal queries, to efficiently answer multiple queries. However, there only is one MPE query for a given set of evidence, for which LJT^{mpe} needs to max out all PRVs without evidence. Thus, as long as one parcluster has all the information, which a root has after an *inbound* message pass, that parcluster can answer an MPE query. The other important difference is that LJT performs a maxing out compared to a summing out to eliminate a PRV. After the *inbound* message pass, LJT^{mpe} maxes out the remaining PRVs from the root cluster of the *inbound* message passing. Lastly, LJT^{mpe} can directly read out the MPE, which lead to the highest potential.

Algorithm 1 LJT^{mpe} for PM G and Evidence $\mathbf{E}$

```
procedure LJT$^{mpe}$($G, \mathbf{E}$)
    Build FO jtree $J$ using $G$
    Enter $\mathbf{E}$ in $J$
    Perform an inbound message pass on $J$
    Answer MPE query
```

Figure 3 depicts an FO jtree J for our example PM G^{ex}. After the evidence entering, LJT^{mpe} performs an *inbound* message pass with, e.g., $\mathbf{C}^1$ as root. Thus, LJT^{mpe} calculates the message m^{21}: LJT^{mpe} eliminates $DoR(X)$ from $\mathbf{C}^2$ by applying lifted maxing out. For each case, where the range values of Hot and $AttC(X)$ are the same and only the range value of $DoR(X)$ differs, LJT^{mpe} keeps the higher potential of the range value and saves the assignment to max out $DoR(X)$. Hence, m^{21} contains a parfactor with 4 rows and each row contains the maximum potential and whether a true or false assignment of $DoR(X)$ leads to the potential. After LJT^{mpe} sends m^{21} to $\mathbf{C}^1$, $\mathbf{C}^1$ holds all the information necessary to answer an MPE query. In $\mathbf{C}^1$, LJT^{mpe} still has to eliminate Hot, $AttC(X)$, and $Pub(X,P)$ by applying lifted maxing out. Having eliminated all PRVs, LJT^{mpe} can return the most probable assignment given the evidence.

Another assignment query is a maximum a posteriori (MAP) query, where we are interested in the most probable assignment only for a subset of PRVs.

Definition 9. *Given a PM G and evidence $\mathbf{E}$, we are interested in the most probable assignment for some PRVs $\mathbf{V}$ in G without evidence. Let $\mathbf{S}$ be the remaining PRVs. To solve a MAP, LJT and LJT^{mpe} compute* $\arg\max_{\mathbf{V}} \sum_{\mathbf{S}} P(\mathbf{V}|\mathbf{E})$.

The non-commutativity of summing out and maxing out leads to a restriction of the elimination order as it forces an algorithm to sum out PRVs before maxing out query PRVs [5]. Hence, the problem of solving a MAP is in general harder [22]. To compute a lifted solution to the MAP problem, lifting imposes additional restriction on the elimination order, making the problem even harder.

However, LJT can identify harmless MAP queries. In case, a MAP queries is over all PRVs from a parcluster or from connected parclusters, LJT can use the message pass from marginal queries, in which LJT sums out PRVs. Based on the elimination order induced by separators, all other PRVs are summed out and LJT^{mpe} can calculate an MPE for the PRVs from the (connected) parcluster(s).

Assume we are interested in the assignment of Hot, $AttC(X)$, and $Pub(X,P)$. Hence, LJT needs to sum out $DoR(X)$ and LJT^{mpe} needs to max out Hot, $AttC(X)$, and $Pub(X,P)$. To answer the query, LJT^{mpe} can use $\mathbf{C}^1$. During the message pass for marginal queries, LJT calculates the message m^{21} for which it eliminates $DoR(X)$. With m^{21}, $\mathbf{C}^1$ holds all the information necessary to answer the assignment query. Thus, LJT^{mpe} can simply calculate an MPE on $\mathbf{C}^1$.

Unfortunately, LJT^{mpe} does not efficiently handle temporal aspects of PDMs. Thus, we now introduce LDJT^{mpe} to efficiently answer assignment queries for relational temporal models.

6 Most Probable Assignments in LDJT

In this section, we investigate how MPE and MAP queries can be solved efficiently for temporal relational models. Further, we discuss why an efficient handling of temporal aspects is necessary.

6.1 MPE Queries

We look at temporal assignment queries and introduce how LDJT^{mpe} efficiently answers temporal assignment queries.

Definition 10. *Given a PDM G and evidence $\mathbf{E}_{0:T}$ for all time steps, we are interested in the most probable assignment for all PRVs $\mathbf{V}$ in G without evidence. Thus, to solve an temporal MPE, LDJT^{mpe} computes* $\arg\max_{\mathbf{V}} P(\mathbf{V}|\mathbf{E}_{0:T})$.

The basic idea of solving the temporal lifted MPE problem is also to use a maximisation instead of summation, which LDJT uses for marginal queries. The Viterbi algorithm is one approach to solve the temporal propositional MPE problem, which is already successfully applied to propositional temporal models by applying a max-product instead of a sum-product [15].

Algorithm 2 outlines how LDJT^{mpe} efficiently solves the temporal lifted MPE problem. The basic idea for LDJT^{mpe} is a combination from LDJT and LJT^{mpe}. The first step in Alg. 2 is to construct FO jtree structures J_0 and J_t as described in Section 4.3. Using the structures, LDJT^{mpe} enters a loop where for every time step, it recovers the previous state, enters evidence for the current time steps, performs an *inbound* message pass with the *out-cluster* as root, and calculates γ_t to preserve the current time step. The main differences are that LDJT^{mpe} calculates γ_t by maxing out instead of α_t by summing out and that LDJT^{mpe} only performs an *inbound* message pass with the *out-cluster* as root. After LDJT^{mpe} propagated all the information to the last time step and thereby left the loop, it answers the MPE query.

To calculate γ_t, LDJT^{mpe} maxes out all non-interface PRVs from the *out-cluster* of J_t instead of summing them out for α_t. As LDJT^{mpe} only needs to answer one MPE query, over all time steps, it suffices to always perform an *inbound* message pass with the *out-cluster* as root. This way the *out-cluster* has all the information to calculate γ_t and for the last time step, LDJT^{mpe} uses the *out-cluster* to answer the MPE query, by maxing out the interface PRVs and reading out the most probable assignment to all PRVs without evidence.

For our example PDM G^{ex}, LDJT^{mpe} first builds FO jtree structures J_0^{ex} and J_t^{ex}, which are the same structures as for marginal queries. For time step

Algorithm 2 LDJT^{mpe} for PDM G and Evidence $\mathbf{E}_{0:T}$

```
procedure LDJT^{mpe}($G, \mathbf{E}_{0:T}$)
    Build FO jtree $J_0$ and $J_t$ using $G$
    $t := 0$
    while $t \neq T + 1$ do
        Recover previous state from $\gamma_{t-1}$
        Enter $\mathbf{E}_t$ in $J_t$
        Perform an inbound message pass on $J_t$ with out-cluster as root
        Calculate $\gamma_t$
    Answer MPE query using out-cluster of $J_T$
```

0, LDJT^{mpe} does not have a previous state to recover. Thus, LDJT^{mpe} directly enters evidence for time step 0 in J_0^{ex}. On J_0^{ex}, LDJT^{mpe} performs an *inbound* message pass with the *out-cluster* as root and calculates γ_0 by maxing out all non-interface PRVs from the *out-cluster*. For time step 1, LDJT^{mpe} instantiates J_1^{ex} from J_t^{ex}. Figure 4 depicts the FO jtree instantiations for time step 1 and 2. LDJT^{mpe} adds γ_0 to the *in-cluster*, $\mathbf{C}_1^1$, of J_1^{ex}, enter evidence for time step 1, and performs an *inbound* message pass with $\mathbf{C}_1^2$ as root. Hence, LDJT^{mpe} calculates two messages, namely m_1^{12} and m_1^{32}. With m_1^{12} and m_1^{32}, $\mathbf{C}_1^2$ holds all the information necessary to calculate γ_1, by maxing out $AttC_1(X)$. LDJT^{mpe} proceeds in this fashion until it reaches the last time step T. In J_T^{ex} the *out-cluster* $\mathbf{C}_T^2$ holds all the information to answer the MPE query. After maxing out Hot_T, $AttC_T(X)$, and $Pub_T(X, P)$, LDJT^{mpe} can directly read out the assignments for all PRVs from each time step.

Now, we show that LDJT^{mpe} calculates the most probable assignments.

Theorem 1. *LDJT^{mpe} is sound, i.e., computes an MPE for PDM G equivalent to an MPE computed for $gr(unrolled(G))$, where unrolled returns an unrolled model for T time steps.*

Proof. LJT^{mpe} is sound. Basically, LDJT^{mpe} unrolls an FO jtree for T time steps and performs an *inbound* message pass. Thus, LDJT^{mpe} produces the same result as unrolling an FO jtree, providing it to LJT^{mpe}, and LJT^{mpe} performing an inbound message pass with one special parcluster as root. The calculations are equivalent. Hence, as LJT^{mpe} is sound, LDJT^{mpe} is also sound.

6.2 MAP Queries

Let us first define temporal MAP queries and then we introduce how a combination of LDJT, for summing out, and LDJT^{mpe}, for maxing out, can efficiently solve certain temporal MAP queries.

Definition 11. *Given a PDM G and evidence $\mathbf{E}_{0:T}$ for all time steps, we are interested in the most probable assignment for some PRVs $\mathbf{V}$ in G without evidence. Let $\mathbf{S}$ be the remaining PRVs. Thus, to solve an temporal MAP, LDJT and LDJT^{mpe} compute* $\arg\max\limits_{\mathbf{V}} \sum\limits_{\mathbf{S}} P(\mathbf{V}|\mathbf{E}_{0:T})$.

A combination of LDJT and LDJT^{mpe} can also answer MAP queries. Also for the combination, a MAP query over a parcluster is harmless. One feature of LDJT is that an α message separates one time slice from the next. Thus, LDJT and LDJT^{mpe} can efficiently answer MAP queries over complete time steps and reuse α messages computed while answering marginal queries. For example, we are only interested in the assignment of all PRVs from the last 20 time steps. Then, LDJT^{mpe} can instantiate J_{T-20}, add α_{T-21}, start solving an MPE, and read out at the *out-cluster* of J_T the most probable assignments for the last 20 time steps. The α_{T-21} message includes all necessary information for the summing out of all PRVs from time step 0 to $T-21$. Therefore, in the case of MAP queries over complete time steps, the non-commutativity of summing out and maxing out does not lead to a restriction of the elimination order. Hence, using LDJT and LDJT^{mpe}, we can easily identify that MAP queries over complete time steps are harmless. Additionally, LDJT^{mpe} reuses computations from marginal queries to answer MAP queries over complete time steps.

Given both LDJT and LDJT^{mpe} are sound, soundness of the combination of both for queries over time steps follows.

6.3 Discussion

Now, we discuss how LDJT^{mpe} efficiently handels temporal aspects compared to LJT^{mpe} for MPE and MAP queries.

MPE Queries: In case we unroll a PDM into a PM and simply use LJT^{mpe}, the constructed FO jtree would not necessarily be constructed in a way to handle the temporal aspects efficiently and thus, would have an impact on the performance. However, if we unroll an FO jtree based on the structures J_0 and J_t from LDJT^{mpe}, then from a computational perspective, LJT^{mpe} and LDJT^{mpe} would perform the same calculations. Both would only perform one *inbound* message pass and by selecting the *out-cluster* of the last time step as the root also for LJT^{mpe}, then both algorithms would compute exactly the same messages.

Nonetheless, there still is a difference in the memory consumption. On the one hand, LJT^{mpe} would need to store the complete unrolled FO jtree. Thus, all messages, evidence, and local models for all time steps need to be stored in memory, which might not always be feasible for a high number of maximal time steps. On the other hand, out of the box, LDJT^{mpe} only needs to store one FO jtree, including messages, with a γ message, evidence, and local models, for the current time step. Thus, LJT^{mpe} can only perform similarly by using the FO jtree construction of LDJT^{mpe} and the memory consumption of LDJT^{mpe} is significantly lower due to the efficient handling of temporal aspects.

MAP Queries: As we have already mentioned, assignment queries over complete time steps are safe for a combination of LDJT and LDJT^{mpe}. Additionally to being safe, these queries could also be of high interest as often one is probably not interested in the assignment of all PRVs, but only in the PRVs of the last few time steps. Further, while answering marginal queries with LDJT, it can simply store the calculated α messages for assignment queries. Thus, for MAP queries over complete time steps, a combination of LDJT and LDJT^{mpe} can efficiently reuse computations and only needs to store one FO jtree at a time.

To answer MAP queries over complete time steps with a combination of LJT and LJT^{mpe}, we again could provide them with an unrolled FO jtree, analogus to the MPE queries. Here, we would again have the overhead on the memory consumption. Nonetheless, also with LJT and LJT^{mpe}, we could use the message pass of LJT, which includes a complete message pass with *inbound* and *outbound* phase, and apply LJT^{mpe} on the time slices for which we want to know the assignment. However, the combination of LJT and LJT^{mpe} does not support out of the box to dynamically add new time steps nor the separation with the α messages. Hence, a combination of LDJT and LDJT^{mpe} significantly outperforms LJT and LJT^{mpe} for MPE and MAP queries.

In general, for MPE queries the runtime results from [5] also hold for LDJT^{mpe}, if we use an unrolled FO jtree for LJT^{mpe}, as they would perform the same computations. Additionally, for MAP queries, the runtime results from [9,10] also hold for LJT^{mpe} as the assignments only induce a small overhead [5] and LJT^{mpe} efficiently handels temporal aspects of PDMs.

7 Conclusion

We present LDJT^{mpe} to efficiently solve the temporal MPE problem for temporal probabilistic relational models. The basic idea is to use lifted maxing out instead of lifted summing out, which LDJT uses, to eliminate PRVs. Additionally, we show that a combination of LDJT and LDJT^{mpe} can efficiently answer MAP queries over the last x time steps. Further, by comparing LDJT and LDJT^{mpe} against LJT and LJT^{mpe}, we show that an efficient handling of temporal aspects is necessary and that LDJT and LDJT^{mpe} significantly outperforms LJT and LJT^{mpe} for temporal models.

We currently work on extending LDJT to handle incremental changes in a PDM efficiently. 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. Apsel, U., Brafman, R.I.: Exploiting Uniform Assignments in First-order MPE. In: Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence. pp. 74–83. AUAI Press (2014)
3. 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)
4. 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)
5. Braun, T., Möller, R.: Lifted Most Probable Explanation. In: International Conference on Conceptual Structures. pp. 39–54. Springer (2018)
6. Braun, T., Möller, R.: Parameterised Queries and Lifted Query Answering. In: Proceedings of IJCAI 2018. pp. 4980–4986 (2018)
7. Dignös, A., Böhlen, M.H., Gamper, J.: Temporal Alignment. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 433–444. ACM (2012)
8. Dylla, M., Miliaraki, I., Theobald, M.: A Temporal-Probabilistic Database Model for Information Extraction. Proceedings of the VLDB Endowment **6**(14), 1810–1821 (2013)

9. Gehrke, M., Braun, T., Möller, R.: Lifted Dynamic Junction Tree Algorithm. In: Proceedings of the 23rd International Conference on Conceptual Structures. pp. 55–69. Springer (2018)
10. Gehrke, M., Braun, T., Möller, R.: Preventing Unnecessary Groundings in the Lifted Dynamic Junction Tree Algorithm. In: Proceedings of the AI 2018: Advances in Artificial Intelligence. pp. 556–562. Springer (2018)
11. 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)
12. 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) **50**(2), 157–224 (1988)
13. Manfredotti, C.E.: Modeling and Inference with Relational Dynamic Bayesian Networks. Ph.D. thesis, Ph. D. Dissertation, University of Milano-Bicocca (2009)
14. 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)
15. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. Ph.D. thesis, University of California, Berkeley (2002)
16. Nitti, D., De Laet, T., De Raedt, L.: A particle Filter for Hybrid Relational Domains. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2764–2771. IEEE (2013)
17. 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)
18. Poole, D.: First-order probabilistic inference. In: Proceedings of IJCAI. vol. 3, pp. 985–991 (2003)
19. Richardson, M., Domingos, P.: Markov Logic Networks. Machine learning **62**(1), 107–136 (2006)
20. de Salvo Braz, R.: Lifted First-Order Probabilistic Inference. Ph.D. thesis, Ph. D. Dissertation, University of Illinois at Urbana Champaign (2007)
21. de Salvo Braz, R., Amir, E., Roth, D.: MPE and Partial Inversion in Lifted Probabilistic Variable Elimination. In: AAAI. vol. 6, pp. 1123–1130 (2006)
22. Sharma, V., Sheikh, N.A., Mittal, H., Gogate, V., Singla, P.: Lifted Marginal MAP Inference. In: UAI-18 Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence. pp. 917–926. AUAI Press (2018)
23. 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)
24. Thon, I., Landwehr, N., De Raedt, L.: Stochastic relational processes: Efficient inference and applications. Machine Learning **82**(2), 239–272 (2011)
25. 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)
26. 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)