Answering Multiple Conjunctive Queries with 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. We extend LDJT to answer conjunctive queries over multiple time steps, while keeping the complexity to answer a conjunctive query low, by avoiding eliminations. The extended version of saves computations compared to an existing approach to answer multiple lifted conjunctive queries.

1 Introduction

Areas like healthcare and logistics involve 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, healthcare systems involve electronic health records (relational) for many patients (objects), streams of measurements over time (temporal), and uncertainties [21] due to, e.g., missing information caused by data integration. Probabilistic databases (PDBs) can answer queries for relational temporal models with uncertainties [5,6]. However, each query possibly contains redundant information, resulting in huge queries. 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 exact inference for answering multiple conjunctive queries in temporal probabilistic models.

We propose the lifted dynamic junction tree algorithm (LDJT) to exactly answer multiple *filtering* and *prediction* queries for multiple time steps efficiently [7]. LDJT combines the advantages of the interface algorithm [13] and the lifted junction tree algorithm (LJT) [2]. Specifically, this paper presents LDJT^{con} to answer multiple conjunctive queries efficiently. In the static case, LJT answers conjunctive queries by merging a subtree of a first-order junction tree (FO jtree), which contains all query terms. For the temporal case, merging multiple time

* 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

steps, increases the complexity to answer multiple conjunctive query. Therefore, we propose to avoid eliminations of query terms to answer multiple conjunctive queries efficiently. Answering multiple conjunctive queries over different time steps can be used to perform probabilistic complex event processing (CEP) [25]. CEP is a hard problem and also for healthcare, a series of events is of interest.

The remainder of this paper has the following structure: We begin by recapitulating parameterised probabilistic dynamic models (PDMs) as a representation for relational temporal probabilistic models and LDJT. Afterwards, we present how LJT answers static conjunctive queries and propose an approach to answer temporal conjunctive queries. Lastly, we evaluate the computational savings of our approach and conclude by looking at possible extensions.

2 Related Work

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

For exact inference on propositional temporal 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 [13] proposes the interface algorithm consisting of a forward and backward pass using temporal d-separation to apply static inference algorithms to the dynamic model.

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 [16]. Poole [16] 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 [18], Milch et al. [11], and Taghipour et al. [20] extend LVE to its current form. Lauritzen and Spiegelhalter [9] 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.

To handle inference for relational temporal models most approaches are approximative. Additional to being approximative, these approaches involve unnecessary groundings or are not designed to handle multiple 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 [12], which is an approximate counterpart to the interface algorithm. Thon et al. [22] introduce CPT-L, a probabilistic model for sequences of relational state descriptions with a partially lifted inference algorithm. Geier and Biundo [8] present an online interface algorithm for dynamic Markov logic networks (DMLNs), similar to the work of Papai et al. [15]. Both approaches slice DMLNs to run well-studied MLN inference algorithms [17] on each slice. Two ways of performing online inference using particle filtering are described in [10,14]. Vlasselaer et al. [24,23] introduce an exact approach for relational dynamic models, but perform inference on a ground knowledge base.

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

3 Parameterised Probabilistic Models

Based on [4], 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 [7].

3.1 Parameterised Probabilistic Models

PMs combine first-order logic with probabilistic models, representing first-order constructs using logical variables (logvars) as parameters. Let us assume, we would like to remotely infer the condition of patients with regards to water retaining. To determine the condition of patients, we use the change of their weights. An increase in weight could either be caused by overeating or retaining water. Additionally, we use the change of weights of people living with the patient to reduce the uncertainty to infer conditions. In case both persons gain weight, overeating is more likely, while otherwise retaining water is more likely. If a water retention is undetected, it can be an acute life-threatening condition.

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.*

To model our scenario, we use the randvar names C, LT, S, and W for Condition, LivingTogether, ScaleWorks, and Weight, respectively, and the logvar names X and X'. From the names, we build PRVs $C(X)$, $LT(X, X')$, $S(X)$, and $W(X)$. The domain of X and X' is $\{alice, bob, eve\}$. The range of $C(X)$ is $\{normal$, $deviation$, $retains\ water\}$. $LT(X, X')$ and $S(X)$ have range $\{true$, $false\}$ and $W(X)$ has range $\{steady$, $falling$, $rising\}$. With $\kappa = (X, \{alice, bob\})$, $gr(C(X)|\kappa) = \{C(alice), C(bob)\}$. $gr(C(X)|\top)$ also contains $C(eve)$.

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*

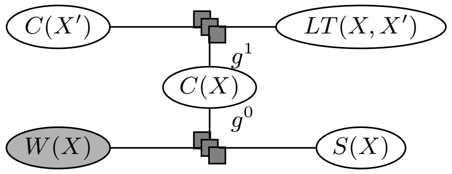

Fig. 1. Parfactor graph for G^{ex}

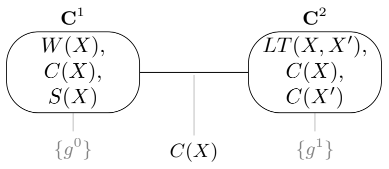

Fig. 2. FO jtree for G^{ex} (local models in grey)

ϕ. C is a constraint on $\mathbf{X}$. A PM $G := \{g^i\}_{i=0}^n$ 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.

Now, we build the model G^{ex} of our example with the parfactors:

$$g^0 = \phi^0(C(X), S(X), W(X))|\top \text{ and } g^1 = \phi^1(C(X), C(X'), LT(X,X'))|\kappa^1$$

We omit the concrete mappings of ϕ^0 and ϕ^1. Parfactor g^0 has the constraint $\top$, meaning it holds for *alice*, *bob*, and *eve*. The constraint κ^1 of g^1 ensures that $X \neq X'$ holds. Fig. 1 depicts G^{ex} as a parfactor graph and shows PRVs, which are connected via undirected edges to parfactors.

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.*

3.2 Parameterised Probabilistic Dynamic Models

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 underlying process is stationary, i.e., the model behaviour does not change over time.

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.*

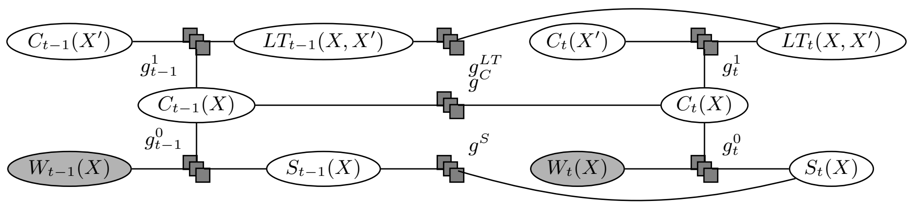

Fig. 3. $G^{ex}_\rightarrow$ the two-slice temporal parfactor graph for model G^{ex}

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 parfactors for the behaviour over time. In this example, g^{LT}, g^{C}, and g^{S} are the inter-slice parfactors.

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

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

4 Lifted Dynamic Junction Tree Algorithm

In this section, we recapitulate LJT [3] to answer queries for PMs and LDJT [7] a *filtering* and *prediction* algorithm to answer queries for PDMs.

4.1 Lifted Junction Tree Algorithm

LJT provides efficient means to answer queries $P(Q^i|\mathbf{E})$, with $Q^i \in \mathbf{Q}$ 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 define parameterised clusters (parclusters), 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 PM G is $J = (\mathbf{V}, \mathbf{P})$ where J is a cycle-free graph, the nodes $\mathbf{V}$ denote a set of parclusters, and $\mathbf{P}$ is a set of edges between parclusters. J 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 (running intersection). 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, enters evidence in the FO jtree, and to distribute local information of the nodes through the FO jtree, passes messages through an *inbound* and an *outbound* pass. To compute a message, LJT eliminates all non-separator 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 would like to know whether $S(bob)$ holds, we query $P(S(bob))$ for which LJT can use parcluster $\mathbf{C}^1$. LJT sums out $C(X)$, $W(X)$, and $S(X)$ where $X \neq bob$ from $\mathbf{C}^1$'s local model G^1, $\{g^0\}$, combined with the received messages.

4.2 LDJT: Overview

LDJT efficiently answers queries $P(Q^i_\pi|\mathbf{E}_{0:t})$, with $Q^i_\pi \in \mathbf{Q_t}$ and $\mathbf{Q_t} \in \{\mathbf{Q}_t\}_{t=0}^T$, 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 $t = 0$, enter $\mathbf{E}_0$ in J_0, pass messages, answer each query term $Q^i_\pi \in \mathbf{Q}_0$, and preserve the state in message α_0. (iii) For $t > 0$, instantiate J_t for the current time step t, recover the previous state from α_{t-1}, enter $\mathbf{E}_t$ in J_t, pass messages, answer each query term $Q^i_\pi \in \mathbf{Q}_t$, and preserve the state in message α_t.

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.

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

For $G^{ex}_\rightarrow$, which is shown in , PRVs $C_{t-1}(X)$, $LT_{t-1}(X, X')$, and $S_{t-1}(X)$ 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^I_0 over $\mathbf{I}_0$ to G_0, builds an FO jtree J_0 and labels the parcluster with g^I_0 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^I_{t-1} and g^I_t, constructs J_t, and labels the parcluster containing g^I_{t-1} as *in-cluster* and the parcluster containing g^I_t 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 $t > 0$.

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

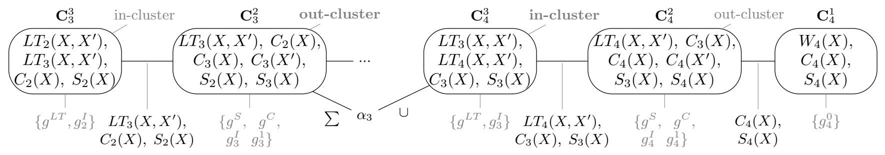
$\mathbf{C}_3^3$
in-cluster
$LT_2(X,X'),$ $LT_3(X,X'),$ $C_2(X),\ S_2(X)$
$\mathbf{C}_3^2$
out-cluster
$LT_3(X,X'),\ C_2(X),$ $C_3(X),\ C_3(X'),$ $S_2(X),\ S_3(X)$
...
$\mathbf{C}_4^3$
in-cluster
$LT_3(X,X'),$ $LT_4(X,X'),$ $C_3(X),\ S_3(X)$
$\mathbf{C}_4^2$
out-cluster
$LT_4(X,X'),\ C_3(X),$ $C_4(X),\ C_4(X'),$ $S_3(X),\ S_4(X)$
$\mathbf{C}_4^1$
$W_4(X),$ $C_4(X),$ $S_4(X)$
$\{g^{LT}, g_2^I\}$
$LT_3(X,X'),$ $C_2(X),\ S_2(X)$
$\{g^S,\ g^C,$ $g_3^I\ \ g_3^1\}$
$\sum$
α_3
$\cup$
$\{g^{LT}, g_3^I\}$
$LT_4(X,X'),$ $C_3(X),\ S_3(X)$
$\{g^S,\ g^C,$ $g_4^I\ \ g_4^1\}$
$C_4(X),$ $S_4(X)$
$\{g_4^0\}$

Fig. 4. Forward pass of LDJT without $\mathbf{C}_3^1$ (local models and labeling in grey)

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 PRVs from $\mathbf{C}_3^2$'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, $\mathbf{C}_4^3$. α_3 is then distributed by message passing and accounted for during calculating α_4.

5 Conjunctive Queries

We begin with recapitulating how LJT answers conjunctive queries in the static case [4]. Afterwards, we introduce LDJT^{con} to efficiently answer multiple conjunctive queries with query terms from different time steps.

5.1 Conjunctive Queries in LJT

We extend Def. 3 to allow for multiple query terms in a static query.

Definition 8. *Given a PM G, grounded PRVs $\mathcal{Q}$ and grounded PRVs with fixed range values $\mathbf{E} = \{E^i{=}e^i\}_i$, the expression $P(\mathcal{Q}|\mathbf{E})$ denotes a query w.r.t. $P(G)$.*

Each query of the set of queries $\mathbf{Q}$ that LJT answers can be a conjunctive query. Since the query terms are not necessarily contained in a single parcluster, LJT builds for that conjunctive query a parcluster containing all query terms to leverage its default query answering behaviour. Therefore, LJT identifies a subtree containing all query terms. LJT merges the subtree into one parcluster to answer the query. Further, LJT can still use the messages calculated during the initial message pass, which enter the subtree from the outside. Thus, after merging the subtree, LJT can directly use LVE on the local model of the merged subtree with the messages to answer a conjunctive query.

5.2 Conjunctive Queries in LDJT

Now, we introduce LDJT^{con} to answer multiple conjunctive queries. In case LDJT answers conjunctive *filtering* queries, meaning that all query terms are from the same time step, LDJT can just use LJT's merging approach. However, in case the query terms of a conjunctive query are from time step t up to time step $t+\delta$, LDJT would need to instantiate FO jtrees for δ time steps and identify a subtree for the combination of δ FO jtrees. The subtree contains at least $(\delta - 2) \times m + 2$ parclusters, where m is the number of parclusters on the path between *in-* and *out-cluster*. Thus, merging the parclusters of the subtree, leads to a parcluster with many PRVs. Further, the asymptotic complexity of LVE is exponential in the number of PRVs [19]. Hence, we propose an approach to answer temporal conjunctive queries, which merges fewer PRVs in a parcluster. First, we extend Def. 5 to allow for multiple query terms in a temporal query.

Definition 9. *Given a PDM G, grounded PRVs $\mathcal{Q}_t$ and grounded PRVs with fixed range values $\mathbf{E}_{0:t} = \{E^i_t = e^i_t\}_{i,t}$, $P(\mathcal{Q}_t|\mathbf{E}_{0:t})$ denotes a query w.r.t. $P(G)$.*

Now, each query that LDJT^{con} answers can be a conjunctive query. To answer a conjunctive query, LDJT^{con} needs a parcluster containing all query terms. We construct this parcluster without over-approximating the number of PRVs as much as merging a subtree. Thus, we develop an approach to avoid eliminations of query terms to obtain one parcluster with all query terms. To send a message from parcluster $\mathbf{C}^1$ to $\mathbf{C}^2$, LDJT eliminates all PRVs from $\mathbf{C}^1$ that are not included in the separator $\mathbf{S}^{12}$. Hence, LDJT extends separators with query terms. A PRV is in a separator iff the PRV is contained in both parclusters, which the separator connects. Therefore, to avoid the elimination of a PRV, LDJT^{con} adds the PRV to all parclusters on the path from the parcluster, where the PRV would be eliminated, to a designated parcluster. By extending parclusters with the query PRVs, LDJT can avoid the elimination of the query terms to answer conjunctive queries by leveraging LDJT's behaviour to answer a query.

A naive approach to extend parclusters is to add the query PRVs to all parclusters of the relevant time steps. Unfortunately by over-approximating the extension of parclusters, LDJT increases the number of PRVs in each parcluster. However, the complexity of LVE depends on the PRVs parclusters. Thus, we propose to add the query PRVs on demand, which is outlined in Alg. 1. Basically, LDJT adds all query PRVs to a designated parcluster. Therefore, the number of PRVs in parclusters is only extended by the necessary number of PRVs.

Using Alg. 1 LDJT^{con} ensures that one parcluster contains all query terms. Then LDJT performs a message pass, and answers the conjunctive query $\mathcal{Q}$. To answer a conjunctive query, LDJT^{con} instantiates FO jtree $\mathcal{J}$ for the time steps t to $t+\delta$ of $\mathcal{Q}$. From $\mathcal{J}$ LDJT^{con} selects a *root* parcluster, which contains most of the query terms from $\mathcal{Q}$ and is from the last time step of $\mathcal{J}$, as designated receiver of all query PRVs. Now, LDJT^{con} needs to avoid the elimination of the query terms of $\mathcal{Q}$ to the *root* parcluster. Therefore, starting from each leaf parcluster, LDJT^{con} traverses the path to the *root* parcluster. As FO jtrees are cycle-free graphs, there is exactly one path from each leaf parcluster to the *root* parcluster. While traversing the paths, LDJT^{con} checks whether a parcluster contains query PRVs and adds the query PRVs to all parclusters on the path to the *root* parcluster. Thereby, LDJT delays the elimination of query terms to the *root*

Algorithm 1 Answer Conjunctive Query for Unrolled FO Jtrees for Time Steps t to $t + \delta$ $\mathcal{J}$ and Conjunctive Query $\mathcal{Q}$

```
procedure AnswerConjunctiveQuery($\mathcal{J}$, $\mathcal{Q}$)
    $root$ := Parcluster with the most query terms from time step $t + \delta$
    for all Leaf parcluster $p \in \mathcal{J}$ do
        $current := p$
        while $current \neq root$ do
            $qt := \mathcal{Q} \cap current$
            $next$ := next parcluster on the path to $root$
            $next := next + qt$
            $current := next$
    $\mathcal{J}$ := LJT.PassMessages($\mathcal{J}$)
    LVE.AnswerQuery($root$, $\mathcal{Q}$)
```

parcluster. Another way of interpreting the extension of the *root* parcluster is to add all the query terms of $\mathcal{Q}$ to the *root* parcluster and then to ensure the running intersection property of an FO jtree. After *root* is extended, LDJT^{con} has to repeat a message pass, as the PRVs in parclusters changed. Lastly, LDJT^{con} can use LVE to answer the conjunctive query with the *root* parcluster's local model, which contains at least the query terms, and the incoming messages.

Unfortunately, by avoiding eliminations of query terms, LDJT needs to perform an extra message pass as outlined in Alg. 1. Nonetheless, the approach is still advantageous over identifying a subtree and merge the subtree into one parcluster for conjunctive queries over multiple time steps. Even though the work to answer one conjunctive query is the same, our approach is parallelisable and the search space for the elimination order is smaller. Further, for a second conjunctive query with the same query PRVs but different grounding, the work of the message pass can be reused and thereby redundant computations prevented.

To perform CEP, events from different time steps are queried. For example, we are interested whether there is an influence from $LT_t(x_1, x_2)$, $C_{t+2}(x_1)$, and $C_{t+2}(x_2)$. Figure 4 shows our example model unrolled for time step 3 and 4, without parcluster $\mathbf{C}_3^1$. Assuming, we have the conjunctive query $P(LT_2(eve, bob),$ $C_4(bob), C_4(eve))$, then LDJT^{con} can apply the steps of Alg. 1 to answer the query. First, LDJT^{con} selects $\mathbf{C}_4^1$ as *root* parcluster, because $\mathbf{C}_4^1$ is from the latest time step and is a parcluster containing most of the query terms. Afterwards, LDJT^{con} extends the parclusters on the path from the leaf parclusters $\mathbf{C}_3^1$ and $\mathbf{C}_3^3$ to *root*. $\mathbf{C}_3^3$ includes the query term $LT_2(eve, bob)$. Hence, LDJT adds $LT_2(X, X')$ to all parclusters on the path to the *root* parcluster, namely $\mathbf{C}_3^2$, $\mathbf{C}_4^3$, $\mathbf{C}_4^2$, and to the *root* parcluster $\mathbf{C}_4^1$. No additional parcluster on the path from $\mathbf{C}_3^3$ to *root* contain any query terms. The same holds for the path from $\mathbf{C}_3^1$ to *root*. Second, LDJT^{con} performs a message pass on the extended FO jtree. Last, LDJT^{con} uses *root* to answer the conjunctive query. LDJT^{con} increases the maximum number of PRVs in a parcluster from 6 to 7, allowing us to efficiently answer multiple conjunctive query, e.g., also for *alice* and *bob*. By performing merging, all parclusters would be merged in a parcluster with 12 PRVs.

Theorem 1. *LDJT^{con}'s answering of conjunctive queries is correct.*

Proof. While extending a parcluster P to contain at least all query terms, LDJT ensures the running intersection property of FO jtrees. Thus, after the extension, the FO jtree is still valid, only with a changed elimination order. Further, LDJT performs a complete message pass after the FO jtree structure is changed to distribute information. Therefore, LDJT still has a valid FO jtree with P containing all query terms and the local model of P received the incoming messages. Hence, given that LVE is correct, using LVE to answer the conjunctive query with P's local model produces a correct answer to the conjunctive query.

Algorithm 1 still has room for improvement, e.g., currently, in case paths to the *root* parclusters merge, they are traversed multiple times. Further, LDJT could directly perform the message passing, while extending the parclusters and in case one only wants to use the unrolled FO jtree to answer conjunctive query with different grounding of the query PRVs, an *inbound* pass to the *root* parcluster would suffice to answer the conjunctive query. Furthermore, instead of unrolling FO jtrees, LDJT could also always only instantiate an FO jtree for one time step and proceed in time as described in Section 4.4, and one could increased parclusters to prevent groundings [3]. Nonetheless, Alg. 1 in the current form allows for answering conjunctive queries from time step $t - \pi$ to $t + \delta$ in case one extends LDJT to answer hindsight queries by performing smoothing.

6 Evaluation

For the evaluation, we use the example model G^{ex} and evaluate computations LDJT^{con} can save. Therefore, we compare the maximum number of PRVs in a parcluster for LDJT^{con} against merging a subtree containing all query terms. We evaluate the influence the number of PRVs and the time interval in a conjunctive query have on the maximum number of PRVs in a parcluster. An example query is $P(W_{t-\delta}(eve), C_t(eve))$, which has two PRVs and the time interval is δ.

Figure 5 shows the maximum number of PRVs in a parcluster for different time intervals dependent on the maximum number of PRVs queried in a time step. The line for 2 PRVs (filled diamond) shows the parcluster size for conjunctive queries with at most 2 different PRVs queried in a time step, analogous for 1,3, 4, and 5. For example our query $P(W_{t-\delta}(eve), C_t(eve))$ has 1 PRV in each time step, relating to the 1 PRV line.

In Fig. 5 the 5 PRVs line (filled triangle) correspond to merging a subtree. Further, with merging one merges all time step in the time interval. Therefore, for our example query with a δ of 10, the size of the maximum parcluster grows to 55 PRVs. For LDJT^{con} there are only two different time steps involved with only one PRV for each time step involved. Therefore, the size of the largest parcluster only grows from 5 to 6 PRVs. Overall the size of the largest parcluster is always smaller by using LDJT^{con} compared to merging a subtree.

We desire small parclusters, as the complexity of LVE is exponential to the number of PRVs [19]. For example with our query, with LDJT^{con}, the largest

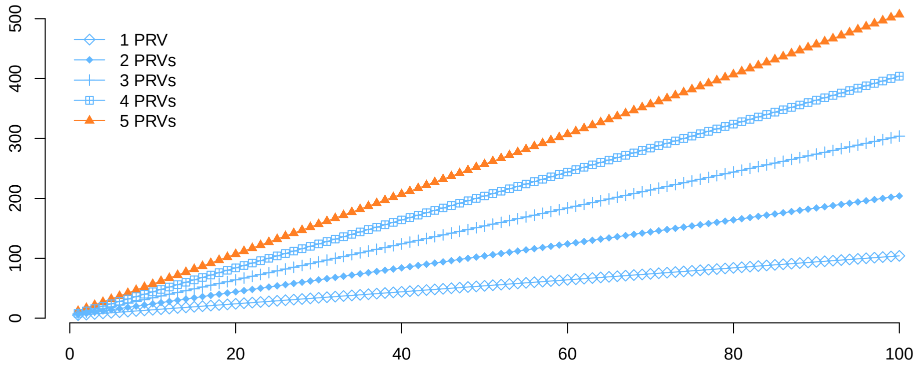

Fig. 5. Y-axis: maximum number of PRVs in a parcluster, x-axis: δ

parcluster has 6 PRVs and with merging a subtree has 55 PRVs. Further, performing CEP could lead to asking the same conjunctive query at least for a subset of our individuals. Hence, starting with a second query only with different groundings, LDJT^{con} saves the elimination of 49 PRVs, by reusing the computations performed during message passing by LDJT^{con}.

7 Conclusion

We present how LDJT^{con} answers conjunctive queries by avoiding eliminations. To avoid eliminations, LDJT^{con} increases parclusters with query PRVs until all query PRVs are in one parcluster. Results show that extending significantly reduces computations for multiple conjunction queries compared to merging.

We are currently working on 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: Proc. of the Joint German/Austrian Conference on Artificial Intelligence. pp. 30–42. Springer (2016)
3. Braun, T., Möller, R.: Preventing Groundings and Handling Evidence in the Lifted Junction Tree Algorithm. In: Proc. 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. Dignös, A., Böhlen, M.H., Gamper, J.: Temporal Alignment. In: Proc. of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 433–444. ACM (2012)

6. Dylla, M., Miliaraki, I., Theobald, M.: A Temporal-Probabilistic Database Model for Information Extraction. Proc. of the VLDB Endowment **6**(14), 1810–1821 (2013)
7. Gehrke, M., Braun, T., Möller, R.: Lifted Dynamic Junction Tree Algorithm. In: Proc. of the 23rd International Conference on Conceptual Structures. Springer (2018)
8. Geier, T., Biundo, S.: Approximate Online Inference for Dynamic Markov Logic Networks. In: Proc. of the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI). pp. 764–768. IEEE (2011)
9. 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)
10. Manfredotti, C.E.: Modeling and Inference with Relational Dynamic Bayesian Networks. Ph.D. thesis, Ph. D. Dissertation, University of Milano-Bicocca (2009)
11. Milch, B., Zettlemoyer, L.S., Kersting, K., Haimes, M., Kaelbling, L.P.: Lifted Probabilistic Inference with Counting Formulas. In: Proc. of AAAI. vol. 8, pp. 1062–1068 (2008)
12. Murphy, K., Weiss, Y.: The Factored Frontier Algorithm for Approximate Inference in DBNs. In: Proc. of the Seventeenth conference on Uncertainty in artificial intelligence. pp. 378–385. Morgan Kaufmann Publishers Inc. (2001)
13. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. Ph.D. thesis, University of California, Berkeley (2002)
14. Nitti, D., De Laet, T., De Raedt, L.: A particle Filter for Hybrid Relational Domains. In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2764–2771. IEEE (2013)
15. Papai, T., Kautz, H., Stefankovic, D.: Slice Normalized Dynamic Markov Logic Networks. In: Proc. of the Advances in Neural Information Processing Systems. pp. 1907–1915 (2012)
16. Poole, D.: First-order probabilistic inference. In: Proc. of IJCAI. pp. 985–991 (2003)
17. Richardson, M., Domingos, P.: Markov Logic Networks. Machine learning **62**(1), 107–136 (2006)
18. de Salvo Braz, R.: Lifted First-Order Probabilistic Inference. Ph.D. thesis, Ph. D. Dissertation, University of Illinois at Urbana Champaign (2007)
19. Taghipour, N., Davis, J., Blockeel, H.: First-order Decomposition Trees. In: Proc. of the Advances in Neural Information Processing Systems. pp. 1052–1060 (2013)
20. 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)
21. Theodorsson, E.: Uncertainty in measurement and total error: tools for coping with diagnostic uncertainty. Clinics in laboratory medicine **37**(1), 15–34 (2017)
22. Thon, I., Landwehr, N., De Raedt, L.: Stochastic relational processes: Efficient inference and applications. Machine Learning **82**(2), 239–272 (2011)
23. 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)
24. Vlasselaer, J., Meert, W., Van den Broeck, G., De Raedt, L.: Efficient Probabilistic Inference for Dynamic Relational Models. In: Proc. of the 13th AAAI Conference on Statistical Relational AI. pp. 131–132. AAAIWS'14-13, AAAI Press (2014)
25. Wang, Y., Cao, K., Zhang, X.: Complex Event Processing over Distributed Probabilistic Event Streams. Computers & Mathematics with Applications **66**(10), 1808–1821 (2013)