On the Completness and Complexity of the Lifted Dynamic Junction Tree Algorithm

Marcel Gehrke
Institute of Information Systems, University of Lübeck, Lübeck
{gehrke}@ifis.uni-luebeck.de

Abstract

Lifted inference allows to perform inference in polynomial time w.r.t. domain sizes. For a lifted algorithm, completeness investigates model classes for which the algorithm is guaranteed to compute a lifted solution. We contribute, to the best of our knowledge, the first completeness and complexity analysis for a temporal lifted algorithm, the so-called lifted dynamic junction tree algorithm (LDJT). To treat time as a first class citizen, LDJT introduces some constraints. Given these constraints, we analyse the classes of liftable models. Further, we show that LDJT has many advantages from a complexity point of view compared to a propositional temporal inference algorithm w.r.t. domain sizes. Therefore, LDJT advances the number of models for which inference tasks can be solved in reasonable time not only from a practically point of view, but also from a theoretical point of view.

Introduction

Temporal probabilistic relational models express relations between objects, modelling uncertainty as well as temporal aspects. Within one time step, a temporal model is considered static. When time advances, the current model state transitions to a new state. Performing inference on such models requires algorithms to efficiently handle the temporal aspect to be able to efficiently answer queries.

Reasoning in lifted representations has a complexity polynomial in domain sizes. However, to the best of our knowledge for temporal lifted algorithms no complexity and completeness results exist. Thus, we analyse the so-called lifted dynamic junction tree algorithm (LDJT) as a temporal lifted algorithm and provide theoretical guarantees for LDJT.

First-order probabilistic inference leverages the relational aspect of a static model, using representatives for groups of indistinguishable, known objects, also known as lifting (Poole 2003). Poole (2003) presents parametric factor graphs as relational models and proposes lifted variable elimination (LVE) as an exact inference algorithm on relational models. Taghipour et al. (2013a) extend LVE to its current form. To benefit from the ideas of the junction tree algorithm (Lauritzen and Spiegelhalter 1988) and LVE, Braun and Möller (2016) present the lifted junction tree algorithm (LJT) for exact inference given a set of queries. To answer multiple temporal queries, Gehrke, Braun, and Möller (2018) present the LDJT, which combines the advantages of the interface algorithm (Murphy 2002) and LJT. Other approaches for temporal relational models perform approximate inference. Ahmadi et al. (2013) propose a colour passing scheme to obtain a lifted representation of a dynamic Markov logic network (DMLN) using exact symmetries and extend lifted belief propagation (Singla and Domingos 2008) for temporal approximate inference. Further inference algorithms for DMLNs exist (Geier and Biundo 2011; Papai, Kautz, and Stefankovic 2012). But, to the best of our knowledge, none of these temporal approaches provide a completeness and complexity analysis.

For static lifted algorithms, completeness and complexity analysis exist. Taghipour et al. (2013b) introduce completeness results for LVE with generalised counting, which also hold for LJT (Braun 2020). The same completeness results also hold for other lifted inference algorithms (Van den Broeck 2011). Taghipour, Davis, and Blockeel (2013) present complexity results for LVE, which are extended to multiple queries and LJT by Braun (2020). However, these approaches do not account for temporal aspects, which introduce additional constraints.

Thus, we present completeness and complexity results for LDJT. Specifically, we analyse (i) the influence of treating time as a first class citizen on liftable models and (ii) how domain sizes influence the complexity of a LDJT compared to the propositional interface algorithm (Murphy 2002). We show that most of the completeness results of LJT also hold for LDJT. The main difference is that LDJT adds constraints to elimination orders and therefore, we have to remove one special case from the classes of liftable models for LDJT. Further, from a complexity perspective, we show that LDJT is highly adventurous compared to the propositional interface algorithm (Murphy 2002). Increasing the number of random variables (randvars) that directly influences the next time step, leads to an exponential growth in the interface algorithm, while increasing the domain size in LDJT leads to an logarithmic growth in the best case and with count-conversions to a polynomial factor.

In the following, we recapitulate parameterised probabilistic dynamic models (PDMs) as a formalism for specifying temporal probabilistic relational models and LDJT for efficient query answering in PDMs. Then, we show a model,

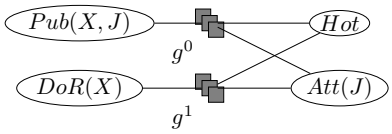

Figure 1: Parfactor graph for G^{ex}

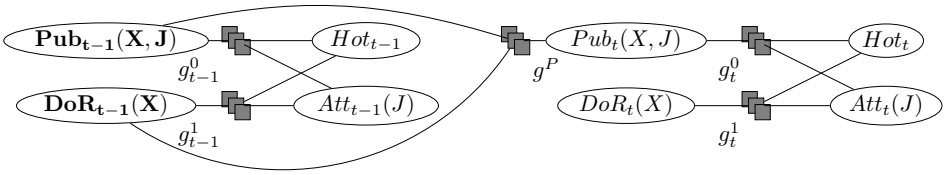

Figure 2: $G^{ex}_{\rightarrow}$ the two-slice temporal parfactor graph for model G^{ex}

for which LDJT induces groundings, due to the fact of treating time as first class citizen. Using this model, we present our completeness results for LDJT. Lastly, we investigate the complexity of LDJT and set it into relation to propositional algorithms.

Preliminaries

We shortly present parameterised probabilistic models (PMs) (Braun and Möller 2018), then extend PMs to the temporal case, resulting in PDMs, and recapitulate LDJT (Gehrke, Braun, and Möller 2018; Gehrke, Braun, and Möller 2019), a *smoothing*, *filtering*, and *prediction* algorithm for PDMs.

Parameterised Probabilistic Models

PMs combine first-order logic with probabilistic models, using logical variables (logvars) as parameters to represent sets of indistinguishable constructs. As an example, we set up a PM to model the reputation of researchers, inspired by the competing workshop example (Milch et al. 2008), with a logvar representing researchers. A reputation is influenced by activities such as publishing, doing active research, and attending conferences. A randvar parameterised with logvars forms a parameterised randvar (PRV).

Definition 1. Let $\mathbf{R}$ be a set of randvar names, $\mathbf{L}$ a set of logvar names, Φ a set of factor names, and $\mathbf{D}$ a set of constants (universe). All sets are finite. Each logvar L has a domain $\mathcal{D}(L) \subseteq \mathbf{D}$. A *constraint* is a tuple $(\mathcal{X}, C_{\mathbf{X}})$ of a sequence of logvars $\mathcal{X} = (X^1, \ldots, X^n)$ and a set $C_{\mathcal{X}} \subseteq \times_{i=1}^n \mathcal{D}(X_i)$. The symbol $\top$ for C marks that no restrictions apply, i.e., $C_{\mathcal{X}} = \times_{i=1}^n \mathcal{D}(X_i)$. A *PRV* $R(L^1, \ldots, L^n), n \geq 0$ is a syntactical construct of a randvar $R \in \mathbf{R}$ possibly combined with logvars $L^1, \ldots, L^n \in \mathbf{L}$. If $n = 0$, the PRV is parameterless and forms a propositional randvar. A PRV A or logvar L under constraint C is given by $A_{|C}$ or $L_{|C}$, respectively. We may omit $|\top$ in $A_{|\top}$ or $L_{|\top}$. The term $\mathcal{R}(A)$ denotes the possible values (range) of a PRV A. An *event* $A = a$ denotes the occurrence of PRV A with range value $a \in \mathcal{R}(A)$.

We use the randvar names Att, DoR, Hot, and Pub for attends conference, does research, hot topic, and publishes at conference, respectively, and $\mathbf{L} = \{X, J\}$ with $\mathcal{D}(X) = \{x_1, x_2, x_3\}$ (people) and $\mathcal{D}(J) = \{j_1, j_2\}$ (conferences). We build boolean PRVs $Att(X)$, $DoR(X)$, Hot, and $Pub(X, J)$. A parametric factor (parfactor) describes a function, mapping argument values to real values (potentials), of which at least one is non-zero.

Definition 2. We denote a *parfactor* g by $\phi(\mathcal{A})_{|C}$ with $\mathcal{A} = (A^1, \ldots, A^n)$ a sequence of PRVs, $\phi : \times_{i=1}^n \mathcal{R}(A^i) \mapsto \mathbb{R}^+$ a function with name $\phi \in \Phi$, and C a constraint on the logvars of $\mathcal{A}$. We may omit $|\top$ in $\phi(\mathcal{A})_{|\top}$. The term $lv(Y)$ refers to the logvars in some element Y, a PRV, a parfactor or sets thereof. The term $gr(Y_{|C})$ denotes the set of all instances of Y w.r.t. constraint C. A set of parfactors forms a *model* $G := \{g^i\}_{i=1}^n$. The semantics of G is given by grounding and building a full joint distribution. With Z as the normalisation constant, G represents $P_G = \frac{1}{Z} \prod_{f \in gr(G)} f$.

Let us build the PM G_{ex}=$\{g^i\}_{i=0}^1$, shown in Fig. 1, with $g^0 = \forall j, x \in \mathcal{D}(J) \times \mathcal{D}(X) : \phi^0(Pub(x, j), Hot, Att(j))_{|\top}$ and $g^1 = \forall j, x \in \mathcal{D}(J) \times \mathcal{D}(X) : \phi^1(DoR(x), Hot, Att(j))_{|\top}$, each with eight input-output pairs (omitted). Next, we present a temporal extension of a PM.

Parameterised Probabilistic Dynamic Models

We define PDMs based on the first-order Markov assumption. Further, the underlying process is stationary.

Definition 3. A PDM G 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 $\mathbf{A}_{\pi}$ a set of PRVs from time slice π. The semantics of G is to instantiate G for a given number of time steps, resulting in a PM as defined above.

Figure 2 shows $G^{ex}_{\rightarrow}$ consisting of G^{ex} for time slice $t-1$ and t with *inter*-slice parfactors for the behaviour over time. The parfactor g^P is the *inter*-slice parfactor. For example, we can observe AAAI conference attendance, which changes over time as, unfortunately, getting papers accepted at consecutive conferences is difficult. Nonetheless, people often publish at similar venues again.

In general, a query asks for a probability distribution of a randvar given fixed events as evidence.

Definition 4. Given a PDM G, a query term Q (ground PRV), and events $\mathbf{E}_{0:t} = \{E^i_t = e^i_t\}_{i,t}$, the expression $P(Q_t|\mathbf{E}_{0:t})$ denotes a *query* w.r.t. P_G.

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

Query Answering Algorithm: LDJT

The important property of LDJT (Gehrke, Braun, and Möller 2018) for this paper is that LDJT constructs first-order junction trees (FO jtrees) to efficiently answer multiple queries

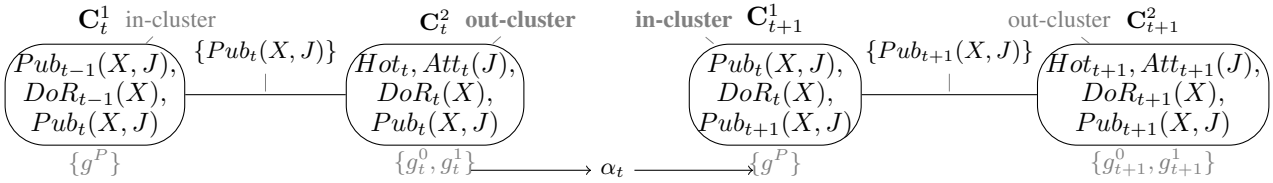

Figure 3: Groundings LDJT cannot prevent

using LVE. The FO jtrees in LDJT contain a minimal set of PRVs to α-separate time steps, which means that information about these PRVs renders FO jtrees independent from each other. Let us now define an FO jtree, with parameterised clusters (parclusters) as nodes, and present how LDJT proceeds in time.

Definition 5. Let $\mathbf{X}$ be a set of logvars, $\mathbf{A}$ a set of PRVs with $lv(\mathbf{A}) \subseteq \mathbf{X}$, and C a constraint on $\mathbf{X}$. Then, $\mathbf{A}_{|C}$ denotes a *parcluster*. We omit $|C$ if $C = \top$ and $lv(\mathbf{A}) = \mathbf{X}$. An *FO jtree* for a model G is a cycle-free graph $J = (V, E)$, where V is the set of nodes, i.e., parclusters, and E the set of edges. J must satisfy three 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 parameterised set $\mathbf{S}^{ij}$, called *separator* of edge $\{i, j\} \in E$, is defined by $\mathbf{C}^i \cap \mathbf{C}^j$. Each $\mathbf{C}^i \in V$ has a *local model* G^i and $\forall g \in G^i$: $rv(g) \subseteq \mathbf{C}^i$. The G_i's partition G.

Querying a minimal set of PRVs with LVE in an FO jtree combines all information to α-separate time steps. To obtain the minimal set, LDJT uses interface PRVs $\mathbf{I}_t$ of $G_{\rightarrow}$.

Definition 6. The forward interface $\mathbf{I}_{t-1}$ is given by

$$\mathbf{I}_{t-1} = \{A_t^i \mid \exists \phi(\mathcal{A})_{|C} \in G : A_{t-1}^i \in \mathcal{A} \wedge \exists A_t^j \in \mathcal{A}\}.$$

PRVs $Pub_{t-1}(X, J)$ and $DoR_{t-1}(X)$ from $G_{\rightarrow}^{ex}$, shown in Fig. 2, make up $\mathbf{I}_{t-1}$. While constructing FO jtree structures, LDJT ensures that the FO jtree J_t for time step t has a parcluster containing $\mathbf{I}_{t-1}$, which is called *in-cluster*, and a parcluster containing $\mathbf{I}_t$, which is called *out-cluster*,. The *in-* and *out-clusters* allow for reusing the FO jtree structures.

To proceed in time, LDJT calculates a forward message α_t over $\mathbf{I}_t$ using the *out-cluster* of J_t. Hence, α_t contains exactly the necessary information, as a set of parfactors, to be able to answer queries in the next time step. Afterwards, LDJT adds α_t to the local model of the *in-cluster* of J_{t+1}.

Figure 3 depicts passing on the current state. To capture the state at t, LDJT sums out the non-interface PRVs Hot_t, $Att_t(J)$ from the local model and received messages of $\mathbf{C}_t^2$ and saves the result in message α_t. Increasing t by one, LDJT adds α_3 to $\mathbf{C}_{t+1}^1$'s local model.

Let us now have a look at a model, where treating time as a first class citizen leads to groundings.

Algorithm Induced Groundings

In the following, we have a look at a case where LDJT grounds a temporal model. The case can be described as a PRV that depends on it predecessors. Later, we use the example as a counter example in our completeness analysis.

LJT and LDJT each have a preprocessing step to prevent algorithm introduced groundings. While creating an FO jtree, the introduced elimination order with the clustering could lead to unnecessary or algorithm induced groundings. Therefore, LJT has a *fusion* step (Braun and Möller 2017), which more or less checks if groundings occur while calculating a message between clusters. In case the groundings could be prevented by another elimination order, the *fusion* step merges the clusters where calculating a message would otherwise result in unnecessary groundings. Thereby, the *fusion* step might enlarge clusters, but prevents algorithm induced groundings. For LDJT an *extension* step exists (Gehrke, Braun, and Möller 2018b,a), which checks for unnecessary grounding while computing temporal messages. The *extension* step tries to delay the elimination of PRVs to prevent unnecessary groundings. Nonetheless, there are groundings that cannot be prevented. Next, we explain a general pattern, when there are algorithm induced groundings. Afterwards, we use our example from 3 to illustrate the problem even further.

More or less, fusing the *in-cluster* and *out-cluster* during *extension* is a case for which LDJT cannot prevent groundings. For such a case to happen, LDJT cannot eliminate a PRV A in the *out-cluster* of J_{t-1} without grounding. Thus, LDJT adds A to the *in-cluster* of J_t. The checks for testing whether LDJT can eliminate A on the path from the *in-cluster* to the *out-cluster* of J_t fail. Thereby, LDJT fuses all parclusters on the path between the two parclusters, but LDJT still cannot eliminate A. LDJT has not been able to eliminate A in the *out-cluster* of J_{t-1} without groundings as well as on the path from the *in-cluster* to the *out-cluster* in J_t. Hence, LDJT also cannot eliminate A in the fused parcluster on the subtree from *in-cluster* to the *out-cluster* without grounding. Even worse, LDJT cannot eliminate A from time step $t-1$ and t in the *out-cluster* to calculate α_t without grounding. For an unrolled model, a lifted solution might be possible, however, with many PRVs in a single parcluster since, in addition to other PRVs, a single parcluster contains A for all time steps. Depending on domain sizes and the maximum number of time steps, either using LDJT with groundings or using the unrolled model with LJT is advantageous as we show in the empirical evaluation.

Example 1 (Groundings LDJT cannot prevent)**.** *Figure 3 depicts FO jtrees for two time steps of $G_{\rightarrow}^{ex}$. To calculate α_t, LDJT could count-covert X in $Pub_t(X, J)$ from g_t^0 and count-covert X in $DoR_t(X)$ from ϕ^1. Afterwards,*

LDJT can multiply the parfactors and eliminate $Att_t(J)$ using lifted summing out. Now, LDJT can count-convert J in $Pub_t(X, J)$, leading to both variables being count-converted, and finally, summing out H_t. Unfortunately, the count-conversions lead to groundings in $\mathbf{C}^1_{t+1}$ as LDJT cannot count-convert X and J in g^P. To eliminate $Pub_t(X, J)$ for the message from $\mathbf{C}^1_{t+1}$ to $\mathbf{C}^2_{t+1}$, LDJT first needs to multiply α_t and g^P. However, in α_t $Pub_t(X, J)$ is completely count-converted and g^P contains two PRVs both with the logvars X and J. Therefore, LDJT cannot count-convert X and J in g^P. Hence, preparing g^P for the multiplication with α_t leads to grounding g^P.

The extension *step of LDJT now would try to delay the eliminations of Hot_t and $Att_t(J)$ to prevent the groundings. However, trying to eliminate Hot_t and $Att_t(J)$ at $\mathbf{C}^1_{t+1}$ leads to a similar problem. LDJT can eliminate $Pub_t(X, J)$ by multiplying g^I and g^0_t. Afterwards, LDJT has a parfactor that includes Hot_t, $Att_t(J)$, $DoR_t(X)$, and $Pub_{t+1}(X, J)$. From that parfactor, LDJT needs to eliminate Hot_t, $Att_t(J)$, and $DoR_t(X)$, which leads to groundings because X as well as J cannot be count-converted without groundings.*

LJT fuses $\mathbf{C}^1_{t+1}$ and $\mathbf{C}^2_{t+1}$. Unfortunately, LDJT also does not eliminate $Pub_{t+1}(X, J)$ in the fused parcluster to calculate α_{t+1}. After multiplying g^I and g^0_t and eliminating $Pub_t(X, J)$, LDJT has the same problem as before fusing. *Therefore, LDJT also cannot eliminate the PRVs here without inducing groundings, even after applying all techniques to prevent algorithm-induced groundings.*

Knowing how treating time as a first class citizen, influences the liftable models of LDJT, we now have a look at the completeness of LDJT.

Completeness

In the following, we show completeness results for LDJT. Braun (2020) show that LJT is complete for 2-logvar models and Taghipour et al. (2013b) show that LVE with generalised counting is complete for 2-logvar models. The same completeness results also hold for other exact static lifted inference algorithms (Van den Broeck 2011). A 2-logvar model has at most two logvars in each parfactor. In general, with completeness, one can classify the models for which a probabilistic inference algorithm runs in polynomial time w.r.t. the domain size.

Definition 7 (2-logvar models)**.** Let $\mathcal{M}^{2lv}$ be the model class of 2-logvar models.

Theorem 1. *LVE and LJT are complete for PDMs that are in $\mathcal{M}^{2lv}$.*

Proof. LVE and LJT are complete for $\mathcal{M}^{2lv}$ and if a PDM G is a from $\mathcal{M}^{2lv}$, then the unrolled version of G is also from $\mathcal{M}^{2lv}$ and thus, LVE and LJT are complete for G. Therefore, LVE and LJT answer queries in polynomial time w.r.t. the domain size by computing a lifted solution. □

Theorem 2. *LDJT is not complete for all models in $\mathcal{M}^{2lv}$.*

Proof. Example 1 shows a model from $\mathcal{M}^{2lv}$ and LDJT cannot compute a lifted solution for that model. Hence, LDJT is not complete for all models in $\mathcal{M}^{2lv}$.

With the counterexample from Example 1, we have shown that LDJT is not complete for all 2-logvar models. Similar counter examples can also be build with other inter-slice parfactors. For example with the inter-slice parfactors

$$\forall j, x \in \mathcal{D}(J) \times \mathcal{D}(X) : \phi^P(DoR_t(x), Pub_{t+1}(x, j))_{|\top}$$
$$\forall j, x \in \mathcal{D}(J) \times \mathcal{D}(X) : \phi^D(Pub_t(x, j), DoR_{t+1}(x))_{|\top},$$

we can construct a similar counterexample. The main problem here is that in the inter-slice parfactors there is at least one PRV with two logvars for time slice t and at least one PRV with two logvars for time slice $t + 1$. Such a pattern leads to the fact that we cannot eliminate PRVs with fewer logvars without eliminating PRVs with two logvars first. □

By unrolling the corresponding model and using LJT, it builds a parcluster containing the PRV $Pub_t(X, J)$ for all time steps. Most likely, the FO jtree consists of not much more than one parcluster, which basically results in performing LVE on the unrolled model. Further, by clustering a PRV for all time steps in one parcluster, the model is not time separated anymore. We also could adjust the *extension* of LDJT to allow for such parclusters and therefore, be also complete for 2-logvar models. However, with LDJT, we aim at handling temporal aspects efficiently, which is not given anymore by performing LVE on the unrolled model. Therefore, we trade in completeness to handle temporal aspects efficiently.

Theorem 3. *LDJT is complete for models from $\mathcal{M}^{2lv}$ with inter-slice parfactors that do not have PRVs with two logvars for time slice t and $t + 1$.*

Proof. For the proof, we consider the three cases that remain, namely:

i) Only PRVs with at most one logvar in inter-slice parfactors,
ii) only PRVs with two logvars for time slice t in inter-slice parfactors, and
iii) only PRVs with two logvars for time slice $t + 1$ in inter-slice parfactors,

Case i) means that an *out-cluster* can have PRVs with two logvars, but all of them can be eliminated at the *out-cluster*. Therefore, there cannot be any algorithm-induced groundings while calculating an α message. Additionally, the same argumentation also holds for β messages (backward messages while answering hindsight queries) during backward passes with an *in-cluster*. Inside a time step, LJT ensures that it is complete for 2-logvar models. Thus, LDJT is complete for case i).

Case ii) means that at least one PRV with two logvars is in the interface. Therefore, *out-clusters* and *in-clusters* have at least one PRV p with two logvars. As trying to eliminate the non-interface PRVs could lead to count-converting the two logvars of p, these count-conversions then could lead to groundings in an *in-cluster*. Therefore, all descriptions

about PRVs from an *out-cluster* need to be sent to an *in-cluster*. However, between an *in-cluster* and an *out-cluster*, LDJT can eliminate p and afterwards the remaining PRVs from time-slice t: In the inter-slice parfactors, p only occurs for time slice t, but not for time-slice $t+1$. On the path from the *in-cluster* to the *out-cluster*, α_t is multiplied with the inter-slice parfactors. Multiplying α_t with the inter-slice parfactors ensures that LDJT can eliminate p. Further, p cannot occur for time slice $t+1$ in the inter-slice parfactors. Thus, LDJT can eliminate the remaining PRVs from time-slice t with generalised counting. Hence, all PRVs that need to be eliminated for α_{t+1} can be eliminated using lifted operations. The argumentation is valid for a forward pass as well as for a backward pass. Hence, LDJT is complete for case ii).

Case iii) is similar to case i). To calculate temporal messages there are no algorithm-induced groundings. The difference is that on the path from an *in-cluster* to an *out-cluster*, LJT might need to prevent algorithm-induced groundings by fusion, which in the worst case leads to merging *in-cluster* and *out-cluster*. However, when calculating a temporal message, LDJT eliminates all two logvar PRVs, and then generalised counting ensures that LDJT does not have to ground. Therefore, LDJT is complete for all three cases, which, in turn, means that LDJT is complete for 2-logvar models with inter-slice parfactors that do not have PRVs with two logvars for time slice t and $t+1$. □

Taghipour (2013, subsection 6.7) conjectures that one could easily generalise the counting operation to also allow counting of multiple logvars. Thus, from a theoretical point of view, by generalising the counting operation even further, one could solve our grounding problem, making LDJT also complete for all 2-logvar models. However, Taghipour (2013, subsection 6.7) also mentions that additional research on this counting problem is needed.

Definition 8 (1-logvar PRV models)**.** Let $\mathcal{M}^{1prv}$ be the model class where each PRV has at most 1 logvar.

Corollary 1. *LDJT is complete for* $\mathcal{M}^{1prv}$.

Proof. The proof directly follows from Theorem 3 and Taghipour (2013, Thm. 7.2). □

In general, completeness results for relational inference algorithms assume liftable evidence. In case evidence breaks symmetries, query answering might not run in polynomial time but in exponential time w.r.t. domain sizes. Also, even if an algorithm is not complete for a certain class, the algorithm might still be able compute a lifted solution for some models of that class. For example, LDJT can calculate a lifted solution for a 3-logvar model, even though LDJT is not complete for 3-logvar models. Knowing the boundaries of completeness and LDJT, we now take a look at the complexity of LDJT.

Complexity

For the complexity analysis, we first have a look at the complexity of each step of LDJT, before we compile the overall complexity of LDJT. Each of these steps can also be found in the complexity analysis for LJT (Braun 2020). Therefore, we also often compare the complexity of LDJT to LJT.

For LDJT, the complexity of the FO jtree construction is negligible. Compared to LJT (Braun 2020), the complexity of the FO jtree construction of LDJT only differs in constant factors. The construction of the FO jtree structures J_0 and J_t is actually the very same as for LJT. The difference is that two FO jtrees are constructed. For *extension*, LDJT additionally performs checks on two messages and twice the *fusion* step of LJT. The additional checks are constant factors and therefore, do not change the complexity of the FO jtree construction, which we can neglect (Braun 2020).

For the complexity analysis of LDJT, we assume that the FO jtree structures of LDJT are minimal and do not induce groundings. Further, we slightly change the definition of *lifted width* (Taghipour et al. 2013b; Braun 2020), as we now consider a PDM G and two FO jtrees, J_0 and J_t.

Definition 9. Let $w_{J_0} = (w_g^0, w_\#^0)$ be the *lifted width* of J_0 and let $w_{J_t} = (w_g^t, w_\#^t)$ be the *lifted width* of J_t. The *lifted width* w_J of a pair (J_0, J_t) is a pair $(w_g, w_\#)$, where $w_g = \max(w_g^0, w_g^t)$ and $w_\# = \max(w_\#^0, w_\#^t)$.

Further, T is the maximum number of time steps, n is the largest domain size among $lv(G)$, $n_\#$ is the largest domain size of the counted logvars, r is the largest range size in a G, $r_\#$ is the largest range size among the PRVs in the Counted RVs, and n_J being the $max(n_{j_0}, n_{j_t})$. The largest possible factor is given by $r^{w_g} \cdot n_\#^{w_\# \cdot r_\#}$. Hence, we always look at the highest number that occurs either in J_0 or J_t.

Evidence entering consists of absorbing evidence at each applicable node.

Lemma 1. *The complexity of absorbing an evidence parfactor is*

$$O(T \cdot n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n_\#^{w_\# \cdot r_\#}). \tag{1}$$

The difference to LJT (Braun 2020) is that LDJT does not only enter evidence in each parcluster of an FO jtree, but enters evidence in each instantiated FO jtree. Overall, there is evidence for up to T time steps. Therefore, LDJT enters evidence in T FO jtrees.

Passing messages consists of calculating messages with LJT for every time step. Here, we consider the worst case, i.e., for each time step querying the first and last time step, the average case, i.e., *hindsight* and *prediction* queries with a constant offset, and the best case, i.e., only *filtering* queries.

Lemma 2. *The worst case complexity of passing messages is*

$$O(T^2 \cdot n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n_\#^{w_\# \cdot r_\#}). \tag{2}$$

The average case complexity of passing messages is

$$O(T \cdot n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n_\#^{w_\# \cdot r_\#}). \tag{3}$$

The best case complexity of passing messages is

$$O(T \cdot n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n_\#^{w_\# \cdot r_\#}). \tag{4}$$

The complexity of one complete message pass in an FO jtree, consists of calculating $2 \cdot (n_J - 1)$ messages and

each message has a complexity of $O(\log_2 n \cdot r^{w_g} \cdot n_{\#}^{w_{\#} \cdot r_{\#}})$ (Braun 2020). One difference in LDJT compared to LJT is that LDJT needs to calculate $2 \cdot (n_J - 1) + 2$ messages for the current FO jtree, because LDJT calculates an α and a β message in addition to the normal message pass. For the FO jtree used to answer *prediction* or *hindsight* queries, LDJT calculates $2 \cdot (n_J - 1) + 1$ messages, as LDJT calculates either an α or β message respectively. Additionally, LDJT computes at least one message pass for each time step and at most a message pass for all time steps for each time step. Therefore, we investigate the worst, average, and best case complexity of message passing in LDJT.

The worst case for LDJT is that for each time step there is a query for the first and the last time step. Therefore, for each of the T time steps, LDJT would need to perform a message pass in all T FO jtrees, leading to $T \cdot T$ message passes. Hence, LDJT would perform a message pass for the current time step t, a backward pass from t to the first time step, which includes a message pass on each FO jtree on the path, and a forward pass from t to the last time step, which include a message pass on each FO jtree on the path. These message passes are then executed for each time step. Thus, LDJT performs overall $T \cdot T$ message passes. The complexity of Eq. (2) is also the complexity of LJT given an unrolled FO jtree constructed by LDJT and evidence for each time step. However, the complexity of message passing for LDJT is normally much lower, as one is hardly ever interested in always querying the first and the last time step.

The best case for LDJT is that it only needs to answer *filtering* queries. That it to say it needs to calculate $2 \cdot (n_J - 1) + 1$ messages for each FO jtree, as LDJT calculates an α for each FO jtree. Further, LDJT needs to perform exactly one message pass on each instantiated FO jtree. Therefore, LDJT needs to pass messages on T FO jtrees.

The average case for LDJT is that for each time step LDJT answers a constant number of *hindsight* and *prediction* queries. Assume for each time step, LDJT has a query for $t - 10$ and $t + 15$. Then, LDJT needs to perform 25 message passes to answer all queries for one time step. Therefore, LDJT passes messages $25 \cdot T$ times. In general, *prediction* and *hindsight* queries are often close to the current time step and T can be huge.

Under the presence of *prediction* and *hindsight* queries, LDJT does not always need to calculate $2 \cdot (n_J - 1)$ messages for each FO jtree. In case LDJT has no query for time step t, but only needs J_t to calculate an α or β message, then calculating $(n_J - 1)$ messages suffice for J_t. By selecting the *out-cluster* for a *prediction* queries and respectively the *in-cluster* for a *hindsight* queries as root, then all required messages are present at the *out-cluster* or the *in-cluster* to calculate an α or β message. Hence, an efficient query answering plan can reduce the complexity of message passing with constant factors.

The last step is *query answering*, which consists of finding a parcluster and answering a query on an assembled submodel. For query answering, we combine all queries in one set instead of having a set of queries for each time step.

Lemma 3. *The complexity of answering a set of queries* $\{Q_k\}_{k=1}^{m}$ *is*

$$O(m \cdot \log_2(n) \cdot r^{w_g} \cdot n_{\#}^{w_{\#} \cdot r_{\#}}). \tag{5}$$

The complexity for query answering in LDJT does not differ from the complexity of LJT. Nonetheless, the number of queries m for LDJT is often higher than the number of queries m for LJT.

We now combine the stepwise complexities to arrive at the complexity of LDJT by adding up the complexities in Eqs. (1) to (5).

Theorem 4. *The worst case complexity of LDJT is*

$$O(((T^2 + T) \cdot n_J + m) \cdot \log_2(n) \cdot r^{w_g} \cdot n_{\#}^{w_{\#} \cdot r_{\#}}). \tag{6}$$

The average case complexity of LDJT is

$$O((T \cdot n_J + m) \cdot \log_2(n) \cdot r^{w_g} \cdot n_{\#}^{w_{\#} \cdot r_{\#}}). \tag{7}$$

The best case complexity of LDJT is

$$O((T \cdot n_J + m) \cdot \log_2(n) \cdot r^{w_g} \cdot n_{\#}^{w_{\#} \cdot r_{\#}}). \tag{8}$$

Comparison to the Ground Interface Algorithm

In this subsection, we show that lifting is crucial when it comes to temporal models, where multiple instances influence the next time step. Therefore, we show that the complexity of the ground interface algorithm is exponential in the number of instances influencing the next time step, while the runtime complexity of LDJT without count-conversions is independent of the domain sizes (except for the $log_2(n)$ part). The implication of this result is that even for small domain sizes lifting is necessary, otherwise, the problem becomes infeasible.

For LJT compared to a ground junction tree algorithm, the speed up is twofold. The first speed up is that LJT has fewer nodes in an FO jtree than the corresponding ground jtree has, i.e., $n_{gr(J)} >> n_J$. The other speed up originates from the counted part, $n_{\#}^{w_{\#} \cdot r_{\#}}$. Under the presence of counting, the lifted width is smaller than the ground width. Further, Taghipour, Davis, and Blockeel (2013) shows that in models that do not require count-conversions, the lifted width is equal to the ground width. Therefore, the factor r^{w_g} of the complexity of LJT is the same as the ground width of a junction tree algorithm without count-conversions.

From a complexity perspective, LDJT has another advantage over the interface algorithm. The w_g part of lifted width w_J from LDJT is much lower compared to the ground tree width of the interface algorithm. The interface algorithm ensures that all randvars that have successors in the next time step are grouped in one cluster of an jtree. Therefore, the corresponding jtree has at least $|gr(\mathbf{I}_t)|$ randvars in a cluster. Thus, the ground width w_g of the interface algorithm depends on the domain sizes of the interface PRVs. The lifted width w_J with its part w_g of LDJT is independent of the domain sizes of the interface PRVs. Hence, the lifted width, even without count-conversions, is much smaller compared to the ground width. Therefore, LDJT can answer queries for large domain sizes, which is often infeasible for the interface algorithm due to a high complexity.

Overall, we show that LDJT can handle temporal models with large domain sizes. Thus, lifting greatly matters

and already for small domain sizes LDJT with all its parts, including ensuring preconditions of lifting, is necessary to compute solutions for models. For large domain sizes, the propositional interface algorithm becomes infeasible from a runtime complexity, while LDJT can obtain a solution with a low runtime complexity in comparison to the propositional interface algorithm.

Conclusion

In this paper, we present completeness and complexity analysis for LDJT. To the best of our knowledge, this is the first such analysis for a temporal lifted inference algorithm. We show that treating time as a first class citizen, LDJT has to restrict the class of liftable models by one special case in the inter-slice parfactors. For nearly all (realistic) scenarios, LDJT has a complexity linear to the maximum number of time steps, which is the desired behaviour for an exact temporal inference algorithm. Additionally, also from a complexity point of view, we show how adventurous LDJT and thereby lifting is compared to a temporal propositional inference algorithm, allowing to compute solutions to liftable models in reasonable time more or less independent of domain sizes.

Knowing the benefits of a lifted temporal inference algorithm, next steps contain investigating how other sequential problems can benefit from lifting.

References

Ahmadi, B.; Kersting, K.; Mladenov, M.; and Natarajan, S. 2013. Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. *Machine learning* 92(1): 91–132.

Braun, T. 2020. *Rescued from a Sea of Queries: Exact Inference in Probabilistic Relational Models*. Ph.D. thesis.

Braun, T.; and Möller, R. 2016. Lifted Junction Tree Algorithm. In *Proceedings of KI 2016: Advances in Artificial Intelligence*, 30–42. Springer.

Braun, T.; and Möller, R. 2017. Preventing Groundings and Handling Evidence in the Lifted Junction Tree Algorithm. In *Proceedings of KI 2017: Advances in Artificial Intelligence*, 85–98. Springer.

Braun, T.; and Möller, R. 2018. Parameterised Queries and Lifted Query Answering. In *IJCAI-18 Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 4980–4986. International Joint Conferences on Artificial Intelligence Organization.

Gehrke, M.; Braun, T.; and Möller, R. 2018. Lifted Dynamic Junction Tree Algorithm. In *Proceedings of the 23rd International Conference on Conceptual Structures*, 55–69. Springer.

Gehrke, M.; Braun, T.; and Möller, R. 2018a. Preventing Unnecessary Groundings in the Lifted Dynamic Junction Tree Algorithm. In *Proceedings of the AI 2018: Advances in Artificial Intelligence*, 556–562. Springer.

Gehrke, M.; Braun, T.; and Möller, R. 2018b. Towards Preventing Unnecessary Groundings in the Lifted Dynamic Junction Tree Algorithm. In *Proceedings of KI 2018: Advances in Artificial Intelligence*, 38–45. Springer.

Gehrke, M.; Braun, T.; and Möller, R. 2019. Relational Forward Backward Algorithm for Multiple Queries. In *Proceedings of the 32nd International Florida Artificial Intelligence Research Society Conference (FLAIRS-32)*, 464–469. AAAI Press.

Geier, T.; and Biundo, S. 2011. Approximate Online Inference for Dynamic Markov Logic Networks. In *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence*, 764–768. IEEE.

Lauritzen, S. L.; and Spiegelhalter, D. J. 1988. 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.

Milch, B.; Zettlemoyer, L. S.; Kersting, K.; Haimes, M.; and Kaelbling, L. P. 2008. Lifted Probabilistic Inference with Counting Formulas. In *AAAI08 Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, 1062–1068. AAAI Press.

Murphy, K. P. 2002. *Dynamic Bayesian Networks: Representation, Inference and Learning*. Ph.D. thesis, University of California, Berkeley.

Papai, T.; Kautz, H.; and Stefankovic, D. 2012. Slice Normalized Dynamic Markov Logic Networks. In *NIPS12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, 1907–1915. Curran Associates Inc.

Poole, D. 2003. First-order probabilistic inference. In *IJCAI03 Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 985–991. Morgan Kaufmann Publishers Inc.

Singla, P.; and Domingos, P. M. 2008. Lifted First-Order Belief Propagation. In *AAAI08 Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, 1094–1099. AAAI Press.

Taghipour, N. 2013. *Lifted Probabilistic Inference by Variable Elimination*. Ph.D. thesis, Ph. D. Dissertation, KU Leuven.

Taghipour, N.; Davis, J.; and Blockeel, H. 2013. First-order Decomposition Trees. In *NIPS13 Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, 1052–1060. Curran Associates Inc.

Taghipour, N.; Fierens, D.; Davis, J.; and Blockeel, H. 2013a. Lifted Variable Elimination: Decoupling the Operators from the Constraint Language. *Journal of Artificial Intelligence Research* 47(1): 393–439.

Taghipour, N.; Fierens, D.; Van den Broeck, G.; Davis, J.; and Blockeel, H. 2013b. Completeness Results for Lifted Variable Elimination. In *Artificial Intelligence and Statistics*, 572–580.

Van den Broeck, G. 2011. On the Completeness of First-Order Knowledge Compilation for Lifted Probabilistic In-

ference. In *NIPS11 Proceedings of the 24th International Conference on Neural Information Processing Systems*, 1386–1394. Curran Associates Inc.