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

Marcel Gehrke, Tanya Braun, and Ralf Möller

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

**Abstract.** The lifted dynamic junction tree algorithm (LDJT) answers *filtering* and *prediction* queries efficiently for probabilistic relational temporal models by building and then reusing a first-order cluster representation of a knowledge base for multiple queries and time steps. Unfortunately, a non-ideal elimination order can lead to groundings. We extend LDJT (i) to identify unnecessary groundings and (ii) to prevent groundings by delaying eliminations through changes in a temporal first-order cluster representation. The extended version of LDJT answers multiple temporal queries orders of magnitude faster than the original version.

## 1 Preventing Groundings in LJT

The elimination order in the lifted dynamic junction tree algorithm (LDJT) can lead to unnecessary groundings [2]. In this paper, we propose an approach to prevent unnecessary groundings and use the examples and definitions from [2].

A lifted solution to a query given a model means that we compute an answer without grounding a part of the model. Unfortunately, not all models have a lifted solution because lifted variable elimination (LVE), the basis for lifted junction tree algorithm (LJT), requires certain conditions to hold. Therefore, these models involve groundings with any exact lifted inference algorithm. Grounding a logical variable (logvar) is expensive and, during message passing, may propagate through all nodes. LJT has a few approaches to prevent groundings for a static first-order junction tree (FO jtree). On the one hand, some approaches originate from LVE. On the other hand, LJT has a fuse operator to prevent groundings, occurring due to a non-ideal elimination order [1].

### 1.1 General Grounding Prevention Techniques from LVE

One approach to prevent groundings is to perform lifted summing out. The idea is to compute VE for one case and exponentiate the result for isomorphic instances. Another approach in LVE to prevent groundings is count-conversion, which exploits that all random variables (randvars) of a parameterised randvar (PRV) $A$ evaluate to a value $v$ of $range(A)$. LVE forms a histogram by counting for each $v \in range(A)$ how many instances of $gr(A)$ evaluate to $v$.

* 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

**Definition 1.** *$\#_{X \in C}[P(\mathbf{X})]$ denotes a counting randvar (CRV) with PRV $P(\mathbf{X})$ and constraint $C$, where $lv(\mathbf{X}) = \{X\}$. Its range is the space of possible histograms. If $\{X\} \subset lv(\mathbf{X})$, the CRV is a parameterised CRV (PCRV) representing a set of CRVs. Since counting binds logvar $X$, $lv(\#_{X \in C}[P(\mathbf{X})]) = \mathbf{X} \setminus \{X\}$. We count-convert a logvar $X$ in a parametric factor (parfactor) $g = \mathbf{L} : \phi(\mathcal{A})|C$ by turning a PRV $A^i \in \mathcal{A}$, $X \in lv(A^i)$, into a CRV $A^{i'}$. In the new parfactor $g'$, the input for $A^{i'}$ is a histogram $h$. Let $h(a^i)$ denote the count of $a^i$ in $h$. Then, $\phi'(..., a^{i-1}, h, a^{i+1}, ...)$ maps to $\prod_{a^i \in range(A^i)} \phi(..., a^{i-1}, a^i, a^{i+1}, ...)^{h(a^i)}$.*

One precondition to count-convert a logvar $X$ in $g$, is that only one input in $g$ contains $X$. To perform lifted summing out PRV $A$ from parfactor $g$, $lv(A) = lv(g)$. For the complete list of preconditions for both approaches, see [3].

### 1.2 Preventing Groundings during Intra FO Jtree Message Passing

During message passing, LJT tries to eliminate PRVs by lifted summing out. However, the messages LJT passes via the separators restrict the elimination order, which can lead to groundings. LJT has three tests whether groundings occur during message passing, namely: (i) check whether LJT can apply lifted summing out, (ii) check whether count-conversion can prevent groundings, and (iii) check that count-converting will not lead to groundings in another parcluster.

A parcluster $\mathbf{C}^i = \mathcal{A}^i|C^i$ sends a message $m^{ij}$ containing the PRVs of the separator $\mathbf{S}^{ij}$ to parcluster $\mathbf{C}^j$. To calculate the message $m^{ij}$, LJT eliminates the PRVs not part of the separator, i.e., $\mathbf{E}^{ij} := \mathcal{A}^i \setminus \mathbf{S}^{ij}$, from the local model and all messages received from other nodes than $j$, i.e., $G' := G^i \cap \{m^{il}\}_{l \neq j}$. To eliminate a PRV from $G'$, LJT has to eliminate the PRV from all parfactors of $G'$. By combining all these parfactors, LJT only has to check whether a lifted summing out is possibile to eliminate the PRV. To eliminate $E \in \mathbf{E}^{ij}$ by lifted summing out from $G'$, we replace all parfactors $g \in G'$ that include $E$ with a parfactor $g^E = \phi(\mathcal{A}^E)|C^E$ that is the lifted product of these parfactors. Let $\mathbf{S}^{ij}_E := \mathbf{S}^{ij} \cap \mathcal{A}^E$ be the set of randvars in the separator that occur in $g^E$. For lifted message calculation, it necessarily has to hold $\forall S \in \mathbf{S}^{ij}_E$,

$$lv(S) \subseteq lv(E). \tag{1}$$

Otherwise, $E$ does not include all logvars in $g^E$. LJT may induce Eq. (1) for a particular $S$ by count-conversion if $S$ has an additional, count-convertible logvar:

$$lv(S) \setminus lv(E) = \{L\},\ \text{L count-convertible in } g^E. \tag{2}$$

In case Eq. (2) holds, LJT count-converts $L$, yielding a (P)CRV in $m^{ij}$, else, LJT grounds. Unfortunately, a (P)CRV can lead to groundings in another parcluster. Hence, count-conversion helps in preventing a grounding if all following messages can handle the resulting (P)CRV. Formally, for each node $k$ receiving $S$ as a (P)CRV with counted logvar $L$, it has to hold for each neighbour $n$ of $k$ that

$$S \in \mathbf{S}^{kn} \vee\ \text{L count-convertible in } g^S. \tag{3}$$

LJT fuses two parclusters to prevent groundings if Eqs. (1) to (3) checks determine unnecessary groundings would occur by message passing between these parcluster.

## 2 Preventing Groundings in LDJT

LDJT has an intra and inter FO jtree message passing phase. Intra FO jtree message passing takes place inside of an FO jtree. Inter FO jtree message passing takes place between two FO jtrees. In both cases unnecessary groundings can occur. To prevent groundings during intra FO jtree message passing, LJT successfully proposes to fuse parclusters. Additionally, LDJT performs inter FO jtree message passing using two instantiations of an FO jtree structure. Unfortunately, having two FO jtrees, LDJT cannot fuse parclusters from different FO jtrees. Hence, LDJT requires a different approach to preventing unnecessary groundings during inter FO jtree message passing. In the following, we present how LDJT prevents grounding and discuss preventing of groundings during intra and inter FO jtree message passing as well as the implications for a lifted run.

### 2.1 Preventing Groundings during Inter FO Jtree Message Passing

As we desire a lifted solution, LDJT also needs to prevent unnecessary groundings induced during inter FO jtree message passes. Therefore, LDJT's *expanding* performs two steps: (i) check whether inter FO jtree message pass induced groundings occur, (ii) prevent groundings by extending the set of interface PRVs, and prevent possible intra FO jtree message pass induced groundings.

**Checking for Groundings** To determine whether an inter FO jtree message pass induces groundings, LDJT also uses Eqs. (1) to (3). For the forward pass, LDJT applies the equations to check whether the $\alpha_{t-1}$ message from $J_{t-1}$ to $J_t$ leads to groundings. More precisely, LDJT needs to check for groundings for the inter FO jtree message passing between $J_0$ and $J_1$ as well as between two temporal FO jtree copy patters, namely $J_{t-1}$ to $J_t$ for $t > 1$.

Thus, LDJT checks all PRVs $E \in \mathbf{E}^{ij}$, where $i$ is the *out-cluster* from $J_{t-1}$ and $j$ is the *in-cluster* from $J_t$, for groundings. In case Eq. (1) holds, no additional checks for $E$ are necessary as eliminating $E$ does not induce groundings. In case Eq. (2) holds, LDJT has to test whether Eq. (3) holds in $J_t$ at least on the path from *in-cluster* to *out-cluster*. Hence, if Eqs. (2) and (3) both hold, eliminating $E$ does not lead to groundings, but if Eq. (2) or Eq. (3) fail groundings occur.

**Expanding Interface Separators** In case eliminating $E$ leads to groundings, LDJT delays the elimination to a point where the elimination does no longer lead to groundings. Therefore, LDJT adds $E$ to the *in-cluster* of $J_t$, which results in $E$ also being added to the inter FO jtree separator . Hence, LDJT does not need to eliminate $E$ in the *out-cluster* of $J_{t-1}$ anymore. Based on the way LDJT constructs the FO jtree structures, the FO jtrees stay valid. Lastly, LDJT prevents groundings in the extended *in-cluster* of $J_t$ as described in Section 1.2.

Let us now have a look at Fig. 1 to understand the central idea of preventing inter FO jtree message pass induced groundings. Fig. 1 shows $J_t$ instantiated for time step 3 and 4. Using these instantiations, LDJT checks for groundings during

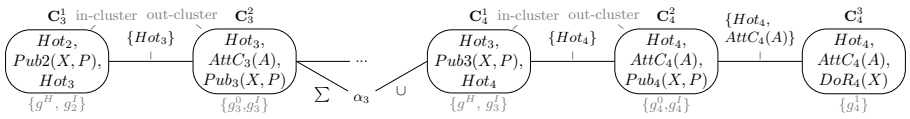


**Fig. 1.** Forward pass of LDJT without $\mathbf{C}_3^3$ (local models and labeling in grey)

inter FO jtree message passing for the temporal copy pattern. To compute $\alpha_3$, LDJT eliminates $AttC_3(A)$ from $\mathbf{C}_3^2$'s local model. Hence, LDJT checks whether the elimination leads to groundings. In this example, Eq. (1) does not hold, since $AttC_3(A)$ does not contain all logvars, $X$ and $P$ are missing. Additionally, Eq. (2) is not applicable, as the expression $lv(S) \setminus lv(E) = \{X, P\} \setminus \{C\} = \{X, P\}$, which contains more than one logvar and therefore is not count-convertible.

As eliminating $AttC_3(A)$ leads to groundings, LDJT adds $AttC_3(A)$ to the parcluster $\mathbf{C}_4^1$. Additionally, LDJT also extends the inter FO jtree separator with $AttC_3(A)$ and thereby changes the elimination order. By doing so, LDJT does not need to eliminate $AttC_3(A)$ in $\mathbf{C}_3^2$ anymore and therefore calculating $\alpha_3$ does not lead to groundings. However, LDJT has to check whether adding the PRV leads to groundings in $\mathbf{C}_4^1$. For the extended parcluster $\mathbf{C}_4^1$, LDJT needs to eliminate the PRVs $Hot_3$, $AttC_3(A)$, and $Pub3(X, P)$. To eliminate $Pub3(X, P)$, LDJT first count-converts $AttC_3(A)$ and then Eq. (1) holds for $Pub3(X, P)$. Afterwards, it can eliminate the count-converted $AttC_3(A)$ and the PRV $Hot_3$ as Eq. (1) holds for both of them. Thus, by adding the PRV $AttC_{t-1}(A)$ to the *in-cluster* of $J_t$ and thereby to the inter FO jtree separator, LDJT can prevent unnecessary groundings. Additionally, as LDJT uses this FO jtree structure for all time steps $t > 0$, i.e., the changes to the structure also hold for all $t > 0$.

**Theorem 1.** *LDJT's* expanding *is correct and produces a valid FO jtree.*

*Proof.* In the initial FO jtree structures, the separator between FO jtree $J_{t-1}$ and $J_t$ consists of exactly $\mathbf{I}_{t-1}$. Thus, by taking the intersection of the PRVs contained in $J_{t-1}$ and $J_t$, we get the set of PRVs from $\mathbf{I}_{t-1}$. While LDJT calculates $\alpha_{t-1}$, it only needs to eliminate PRVs $\mathbf{E}$ not contained in the separator and thereby $\mathbf{I}_{t-1}$. Therefore, all $E \in \mathbf{E}$ are not contained in any parcluster of $J_t$. Hence, by adding $E$ to the *in-cluster* of $J_t$, LDJT does not violate any FO jtree properties. Further, LDJT does not even have to validate properties like the running intersection property, since it could not have been violated in the first place. Additionally, LDJT extends the set of interface PRVs, resulting in an over-approximation of the required PRVs for the inter FO jtree communication to be correct.

### 2.2 Discussion

In the following, we start by discussing workload and performance aspects of the intra and inter FO jtree message passing. Afterwards, we present model constellations where LDJT cannot prevent groundings.

**Performance** The additional workload for LDJT introduced by handling unnecessary groundings is moderate. In the best case, LDJT checks Eqs. (1) to (3) for calculating two messages, namely for the $\alpha_{t-1}$ message and for the message LDJT passes from in *in-cluster* of $J_t$ in the direction of the *out-cluster* of $J_t$. In the worst case, LDJT needs to check $1 + (m - 1)$ messages, where $m$ is the number of parclusters on the path from the *in-cluster* to the *out-cluster* in $J_t$.

From a performance point of view, increasing the size of the $\alpha$ messages and of a parcluster is not ideal, but always better than the impact of groundings, which would result in ground calculations for each time step. By applying the intra FO jtree message passing check, LDJT may fuse the *in-cluster* and *out-cluster*, which most likely results in a parcluster with many model PRVs. Increasing the number of PRVs in a parcluster, increases LDJT's workload for query answering. But even with the increased workload a lifted run is faster than grounding. However, in case the checks determine that a lifted solution is not obtainable, using the initial model with the local clustering is the best solution.

First, applying LJT's *fusion* is more efficient since fusing the *out-cluster* with another parclusters could increase the number of its PRVs. In case of changed PRVs, LDJT has to rerun the *expanding* check. Therefore, LDJT first applies the intra and then the inter FO jtree message passing checks.

**Groundings LDJT Cannot Prevent** Fusing the *in-cluster* and *out-cluster* due to the inter FO jtree message passing check is one case for which LDJT cannot prevent groundings. In this case, LDJT cannot eliminate $E$ in the *out-cluster* of $J_{t-1}$ without groundings. Thus, LDJT adds $E$ to the *in-cluster* of $J_t$. The checks whether LDJT can eliminate $E$ 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 and LDJT still cannot eliminate $E$. Even worse, LDJT cannot eliminate $E$ from time step $t - 1$ and $t$ in the *out-cluster* to calculate $\alpha_t$. In theory, for an unrolled model, a lifted solution might be possible, but with many PRVs in a parcluster, since, in addition to other PRVs, one parcluster contains $E$ for all time steps. Depending on the domain size and the maximum number of time steps, either grounding or using the unrolled model is advantageous.

If $S$ occurs in an inter-slice parfactor for both time steps, then another source of groundings is a count-conversion of $S$ to eliminate $E$. In such a case, LDJT cannot count-convert $S$ in the inter-slice parfactor, which leads to groundings.

## 3 Evaluation

For the evaluation, we use the example model $G^{ex}$ with the set of evidence being empty, for $|\mathcal{D}(X)| = 10$, $|\mathcal{D}(P)| = 3$, $|\mathcal{D}(C)| = 20$, and the queries $\{Hot_t, AttC_t(c_1), DoR_t(x_1)\}$ for each time step. We compare the runtimes on commodity hardware with 16 GB of RAM of the extended LDJT version against the original version and then also against LJT for multiple maximum time steps.

Figure 2 shows the runtime in seconds for each maximum time step. We can see that the runtime of the extended LDJT (diamond) and the original LDJT

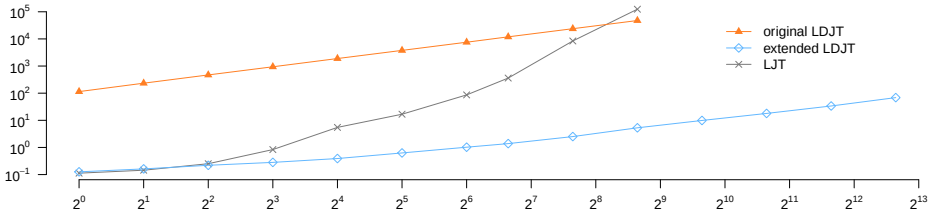


**Fig. 2.** Y-axis: runtimes [seconds], x-axis: maximum time steps, both in log scale

(filled triangle) is, as expected, linear, while the runtime of LJT (cross) roughly is exponential, to answer queries for changing maximum number of time steps. Further, we can see how crucial preventing groundings is. Due to the FO jtree construction overhead, the extended version is about a magnitude of three faster for first time steps, but the construction overhead becomes negligible with more time steps. Overall, the extended LDJT is up to four orders of magnitude faster.

Additionally, we see the runtimes of LJT. LJT is faster for the initial time steps, especially in case grounding are prevented by unrolling. Nonetheless, after several time steps, the size of the parclusters becomes a big factor, which also explains the exponential behaviour [3]. To summarise the evaluation results, on the one hand, we see how crucial the prevention of groundings is and, on the other hand, how crucial the dedicated handling of temporal aspects is.

## 4 Conclusion

We present how LDJT can prevent unnecessary groundings by delaying eliminations to the next time step and thereby changing the elimination order. To delay eliminations, LDJT increases the *in-cluster* of the temporal FO jtree structure and the separator between *out-cluster* and *in-cluster* with PRVs, which lead to the groundings. First results show that the extended LDJT significantly outperforms the orignal version and LJT if unnecessary groundings occur.

We currently work on extending LDJT to calculate the most probable explanation. Other interesting future work includes a tailored automatic learning for parameterised probabilistic dynamic models and parallelisation of LJT.

## References

1. Braun, T., Möller, R.: Preventing Groundings and Handling Evidence in the Lifted Junction Tree Algorithm. In: Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (Künstliche Intelligenz). pp. 85–98. Springer (2017)
2. Gehrke, M., Braun, T., Möller, R.: Towards Preventing Unnecessary Groundings in the Lifted Dynamic Junction Tree Algorithm. In: Proceedings of KI 2018: Advances in Artificial Intelligence. Springer (2018)
3. Taghipour, N.: Lifted Probabilistic Inference by Variable Elimination. Ph.D. thesis, KU Leuven (2013)