Lifted Most Probable Explanation

Tanya Braun and Ralf Möller

Institute of Information Systems, Universität zu Lübeck, Lübeck
{braun,moeller}@ifis.uni-luebeck.de

Abstract. Standard approaches for inference in probabilistic formalisms with first-order constructs include lifted variable elimination (LVE) for single queries, boiling down to computing marginal distributions. To handle multiple queries efficiently, the lifted junction tree algorithm (LJT) uses a first-order cluster representation of a knowledge base and LVE in its computations. Another type of query asks for a most probable explanation (MPE) for given events. The purpose of this paper is twofold: (i) We formalise how to compute an *MPE* in a lifted way with LVE and LJT. (ii) We present a *case study* in the area of IT security for risk analysis. A lifted computation of MPEs exploits symmetries, while providing a correct and exact result equivalent to one computed on ground level.

Keywords: Probabilistic Logical Model, Lifting, MPE, MAP, Abduction

1 Introduction

In recent years, IT security has become a major research area to perform tasks such as intrusion detection, risk analysis, or risk mitigation. These tasks need efficient and exact inference algorithms for large probabilistic models given sets of events and a multitude of queries to answer. Lifting uses symmetries in a model to speed up reasoning with known domain objects. Symmetries are bound to appear in networks at risk (e.g., several access points, user groups).

For single queries, researchers have sped up runtimes for inference significantly. Variable elimination (VE) decomposes a propositional model into subproblems to evaluate them in an efficient order [20], eliminating random variables (randvars) not present in a query. Lifted VE (LVE), also called FOVE, introduced in [12] and expanded in [13,8,19] to its current form GC-FOVE, exploits symmetries. For multiple queries, Lauritzen and Spiegelhalter present junction trees (jtrees), to represent clusters of randvars in a model, along with a reasoning algorithm [7]. In [1], we present the lifted junction tree algorithm (LJT) based on [7,19], using a first-order jtree (FO jtree). LJT imposes some static overhead but has a significant speed up compared to LVE for query answering (QA).

So far, queries concern marginal and conditional probability distributions. Another important inference task is to compute most probable explanations (MPEs), also known as abduction: We look for the most probable assignment to all randvars in a model. Pearl introduces the idea of MPEs and a propagation algorithm for singly-connected networks [11]. Dawid presents an algorithm to compute MPEs on jtrees [4], which allows for computing up to $k = 3$ MPEs [10]. Dechter formalises computing MPEs as a form of VE, replacing sums with max-out operations [5]. In [14], de Salvo Braz et al. adapt FOVE for MPEs, which does not incorporate GC-FOVE's advances. A related concept to MPE is maximum a posteriori assignment (MAP) for a most probable assignment to a subset of model randvars. Computing exact MAP solutions is intractable [5].

This paper extends LVE and LJT transferring the ideas of Dawid and Dechter, contributing the following. (i) We formalise how to compute an *MPE* with LVE and LJT and discuss MAPs and sets of queries. (ii) We present a *case study* in the area of IT security, specifically, on risk analysis, which our formalism easily models with probable assignments and likelihood of observations as common queries. LJT offers the advantage of clusters for local computations, identifying MAPs easily computable, and combining queries of different types easily.

Various research areas cover MPEs, from probabilistic logic programs [17] to probabilistic databases (DBs) [2]. Ceylan et al. show the complexity of computing most probable DBs and hypotheses jumping off Gribkoff et al.'s work on most probable DBs [6]. LJT offers compiling a DB into a compact model, even allowing for more expressiveness, for fast online QA w.r.t. varying query types. Schröder et al. study most probable state sequences in an agent scenario using lifting [15]. Muñoz-González et al. investigate exact inference for IT security, namely, for attack graphs (AGs), with a jtree algorithm performing best [9]. Similar scenarios appear, e.g., in healthcare [3].

The remainder of this paper is structured as follows: We introduce basic notations and recap LVE and LJT. Then, we present LVE and LJT for computing MPEs, followed by a discussion and a case study. We conclude with future work.

2 Preliminaries

This section introduces basic notations for models as well as the different query types and recaps LVE and LJT.

2.1 Parameterised Models

Parameterised models compactly represent propositional models with logical variables (logvars) as parameters in randvars. We set up a model for risk analysis with an AG. An AG models attacks and their targeted components in a network. A binary randvar holds if a component is compromised, which provides an attacker with privileges to further compromise a network towards a final target. Logvars represent users with certain privileges. We first denote basic blocks.

Definition 1. *Let* $\mathbf{L}$ *be a set of logvar names,* Φ *a set of factor names, and* $\mathbf{R}$ *a set of randvar names. A parameterised randvar* (PRV) $R(L_1, \dots, L_n), n \geq 0$, *is a syntactical construct with a randvar* $R \in \mathbf{R}$ *and logvars* $L_1, \dots, L_n \in \mathbf{L}$ *to represent a set of randvars. For PRV A, the term range*(A) *denotes possible values. A logvar L has a domain, denoted* $\mathcal{D}(L)$*. A* constraint $(\mathbf{X}, C_{\mathbf{X}})$ *is a tuple with a sequence of logvars* $\mathbf{X} = (X_1, \dots, X_n)$ *and a set* $C_{\mathbf{X}} \subseteq \times_{i=1}^{n} \mathcal{D}(X_i)$

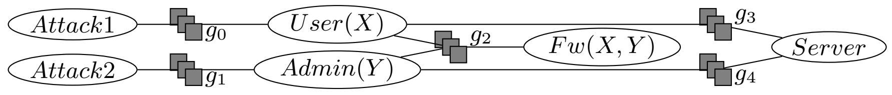

Fig. 1: Parfactor graph for G_{ex}

restricting logvars to certain values. The symbol $\top$ *marks that no restrictions apply and may be omitted. The term* $lv(P)$ *refers to the logvars in some* P, $rv(P)$ *to the PRVs with constraints, and* $gr(P)$ *denotes the set of instances of* P *with its logvars grounded w.r.t. constraints.* P *may be PRV, parfactor, or model.*

From $\mathbf{R} = \{Server, User\}$ and $\mathbf{L} = \{X, Y\}$ with $\mathcal{D}(X) = \{x_1, x_2, x_3\}$ and $\mathcal{D}(Y) = \{y_1, y_2\}$, we build the binary PRVs $Server$ and $User(X)$. With $C = (X, \{x_1, x_2\})$, $gr(User(X)|C) = \{User(x_1), User(x_2)\}$. $gr(User(X)|\top)$ also contains $User(x_3)$. Next, parametric factors (parfactors) combine PRVs.

Definition 2. *A* parfactor g *describes a function, mapping argument values to real values, i.e., potentials. We denote a parfactor by* $\forall\mathbf{X} : \phi(\mathcal{A}) \mid C$ *where* $\mathbf{X} \subseteq \mathbf{L}$ *is a set of logvars.* $\mathcal{A} = (A_1, \dots, A_n)$ *is a sequence of PRVs, each built from* $\mathbf{R}$ *and possibly* $\mathbf{X}$. *We omit* $(\forall\mathbf{X} :)$ *if* $\mathbf{X} = lv(\mathcal{A})$. $\phi : \times_{i=1}^{n} range(A_i) \mapsto \mathbb{R}^+$ *is a function with name* $\phi \in \Phi$. ϕ *is identical for all groundings of* $\mathcal{A}$. C *is a constraint on* $\mathbf{L}$. *A set of parfactors forms a* model $G := \{g_i\}_{i=1}^{n}$. G *represents the probability distribution* $P_G = \frac{1}{Z}\prod_{f \in gr(G)} f$, *where* Z *is the normalisation constant.*

We also build the binary PRVs $Attack1$, $Attack2$, $Admin(Y)$, and $Fw(X, Y)$ to model two different attacks, an admin user with more privileges than a regular user $User(X)$, and a firewall between admin and regular users. Model G_{ex} reads $\{g_i\}_{i=0}^{4}$, $g_0 = \phi_0(Attack1, User(X))$, $g_1 = \phi_1(Attack2, Admin(Y))$, $g_2 = \phi_2(User(X), Admin(Y), Fw(X, Y))$, $g_3 = \phi_3(Server, User(X))$, $g_4 = \phi_4(Server, Admin(Y))$. g_2 has eight, the others four input-output pairs (omitted). Constraints are $\top$, e.g., $gr(g_0)$ contains three factors with identical ϕ_0. Figure 1 depicts G_{ex} as a graph with six variable nodes for the PRVs and five factor nodes for g_0 to g_4 with edges to the PRVs involved.

A counting randvar (CRV) encodes for n interchangeable randvars how many have a certain value, exploiting symmetries in potentials. In $\phi(U_1, U_2, U_3)$ as below left, the potentials are the same for two U_i being true ($= 1$) and one false ($= 0$) or vice versa. Using a logvar N, a CRV, denoted as $\#_N[U(N)]$, and a histogram as range value, the mapping on the right carries the same information. A histogram specifies for each value of U how many of the n randvars have this value (first position $U = 1$, second $U = 0$).

$$\begin{array}{ll} (0,0,0) \to 1,\ (0,0,1) \to 2,\ (0,1,0) \to 2,\ (0,1,1) \to 3, & [0,3] \to 1,\ [1,2] \to 2, \\ (1,0,0) \to 2,\ (1,0,1) \to 3,\ (1,1,0) \to 3,\ (1,1,1) \to 2 & [2,1] \to 3,\ [3,0] \to 2 \end{array}$$

Definition 3. $\#_X[P(\mathbf{X})]$ *denotes a* CRV *with PRV* $P(\mathbf{X})$, *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, $lv(\#_X[P(\mathbf{X})]) = \mathbf{X} \setminus \{X\}$. A histogram h is a set of pairs $\{(v_i, n_i)\}_{i=1}^m$, $v_i \in range(P(\mathbf{X}))$, $m = |range(P(\mathbf{X}))|$, $n_i \in \mathbb{N}$, and $\sum_i n_i = |gr(P(\mathbf{X})|C)|$. A shorthand notation is $[n_1, \dots, n_m]$. $h(v_i)$ returns n_i. Adding two histograms of a PRV yields $\{(v_i, n_i + n_i')\}_{i=1}^m$, multiplying one with a value c yields $\{(v_i, c \cdot n_i)\}_{i=1}^m$.*

The *semantics* of a model is given by grounding and building a full joint distribution. QA asks for a likelihood of an event, a marginal distribution of a set of randvars, or a conditional distribution given events, all types boiling down to computing marginals w.r.t. a model's joint distribution. Formally, $P(\mathbf{Q}|\mathbf{E})$ denotes a query with $\mathbf{Q}$ a set of grounded PRVs and $\mathbf{E}$ a set of events (grounded PRVs with range values). For G_{ex}, $P(Admin(y_1)|Server = 1)$ asks for the conditional distribution of $Admin(y_1)$ with $Server = 1$ (compromised) as an event.

MPE and MAP are assignment queries. An MPE for a model G and evidence $\mathbf{E}$ is the most probable assignment to all remaining randvars, i.e., $mpe(G, \mathbf{E}) = \arg\max_{\mathbf{a} \in range(rv(G) \setminus rv(\mathbf{E}))} P(\mathbf{a}|\mathbf{E})$. A MAP for a set of randvars $\mathbf{V}$ is the most probable assignment to $\mathbf{V}$ given $\mathbf{E}$ summing over the remaining randvars, i.e., $map(G, \mathbf{V}, \mathbf{E}) = \arg\max_{\mathbf{v} \in range(\mathbf{V})} \sum_{\mathbf{a}} P(\mathbf{v}, \mathbf{a}|\mathbf{E})$, $\mathbf{a} \in range(rv(G) \setminus \mathbf{V} \setminus rv(\mathbf{E}))$. While MPE is relatively easy to compute, with $\arg\max$ operations only, MAP is not as it involves sums and $\arg\max$ operations, which are not commutative.

2.2 Query Answering Algorithms

LVE and LJT answer queries for probability distributions. LJT uses an FO jtree with LVE as a subroutine. We briefly recap LVE and LJT.

Lifted Variable Elimination: LVE exploits symmetries that lead to duplicate calculations. In essence, it computes VE for one case and exponentiates its result for isomorphic instances (lifted summing out). GC-FOVE implements LVE through an operator suite (cf. [19] for details). Its main operator *sum-out* realises lifted summing out. An operator *absorb* handles evidence in a lifted way. The remaining operators (*count-convert*, *split*, *expand*, *count-normalise*, *multiply*, *ground-logvar*) aim at enabling a lifted summing out, transforming part of a model. All operators have pre- and postconditions to ensure computing a result equivalent to one computed on $gr(G)$. Algorithm 1 shows an outline. To answer a query, LVE eliminates all non-query randvars. For a new query, LVE starts over.

Algorithm 1 Outline of LVE

```
LVE(Model $G$, Query $\mathbf{Q}$, Evidence $\mathbf{E}$)
  Absorb $\mathbf{E}$ in $G$
  while $G$ has non-query PRVs do
    if PRV $A$ fulfils sum-out prec. then
      Eliminate $A$ using sum-out
    else
      Apply transformator
  return MULTIPLY($G$)          ▷ normalise
```

Algorithm 2 Outline of LJT

```
LJT(Model $G$, Queries $\{\mathbf{Q}_i\}_{i=1}^m$, Evid. $\mathbf{E}$)
  Construct FO jtree $J$
  Enter $\mathbf{E}$ into $J$
  Pass messages on $J$
  for each query $\mathbf{Q}_i$ do
    Find subtree $J_i$ for $\mathbf{Q}_i$
    Extract submodel $G_i$ from $J_i$
    LVE($G_i$, $\mathbf{Q}_i$, $\emptyset$)
```

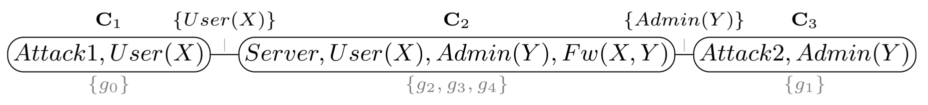

Fig. 2: FO jtree for G_{ex} (local parcluster models in grey)

Lifted Junction Tree Algorithm: Algorithm 2 outlines LJT for a set of queries. LJT first constructs an FO jtree with parameterised clusters (parclusters) as nodes, which are sets of PRVs connected by parfactors, defined as follows.

Definition 4. *An* FO jtree *for a model G is a cycle-free graph $J = (V, E)$, where V is the set of nodes and E the set of edges. Each node in V is a parcluster $\mathbf{C}_i$. A* parcluster *is denoted by $\forall \mathbf{L} : \mathcal{A} \mid C$. $\mathbf{L}$ is a set of logvars. $\mathcal{A}$ is a set of PRVs with $lv(\mathcal{A}) \subseteq \mathbf{L}$. We omit $(\forall \mathbf{L} :)$ if $\mathbf{L} = lv(\mathcal{A})$. Constraint C restricts $\mathbf{L}$. $\mathbf{C}_i$ has a local model G_i of assigned parfactors. An assigned parfactor $\phi(\mathcal{A}_\phi)|C_\phi$ must fulfil (i) $\mathcal{A}_\phi \subseteq \mathcal{A}$, (ii) $lv(\mathcal{A}_\phi) \subseteq \mathbf{L}$, and (iii) $C_\phi \subseteq C$.*
An FO jtree must satisfy three properties: (i) A parcluster $\mathbf{C}_i$ is a set of PRVs from G. (ii) For every parfactor $g = \phi(\mathcal{A})|C$ in G, $\mathcal{A}$ appears in some $\mathbf{C}_i$. (iii) If a PRV from G appears in $\mathbf{C}_i$ and $\mathbf{C}_j$, it must appear in every parcluster on the path between nodes i and j in $\mathcal{J}$. The parameterised set $\mathbf{S}_{ij}$, called separator *of edge i—j in $\mathcal{J}$, contains the shared randvars of $\mathbf{C}_i$ and $\mathbf{C}_j$, i.e., $\mathbf{C}_i \cap \mathbf{C}_j$.*

For G_{ex}, Fig. 2 depicts an FO jtree with three parclusters, $\mathbf{C}_1 = \forall X{:}\{Attack1, User(X)\}|\top$, $\mathbf{C}_2 = \forall X, Y{:}\{Server, User(X), Admin(Y), Fw(X,Y)\}|\top$, $\mathbf{C}_3 = \forall Y{:}\{Attack2, Admin(Y)\}|\top$. Separators are $\mathbf{S}_{12} = \mathbf{S}_{21} = \{User(X)\}$ and $\mathbf{S}_{23} = \mathbf{S}_{32} = \{Admin(Y)\}$. Each parcluster has a non-empty local model.

After constructing an FO jtree, LJT enters evidence for the local models to absorb it. Message passing propagates local information through the FO jtree in two passes: First, LJT sends messages from the periphery towards the center. Second, LJT sends messages the opposite way. A message is a set of parfactors over separator PRVs. For a message from node i to neighbour j, LJT eliminates all PRVs not in separator $\mathbf{S}_{ij}$ from G_i and the messages from other neighbours using LVE. Afterwards, each parcluster holds all information of the model in its local model and messages. LJT answers a query by finding a subtree whose parclusters cover the query randvars, extracting a submodel of local models and messages, and using LVE to answer the query on the submodel. Next, we adapt LVE in the form of GC-FOVE to compute an MPE.

3 Lifted Algorithms for Most Probable Explanations

We adapt LVE and LJT to compute MPEs. We use the GC-FOVE operator suite as a basis for LVE$^{\mathrm{mpe}}$, which becomes a subroutine in LJT$^{\mathrm{mpe}}$.

3.1 Most Probable Explanation with LVE

For LVE$^{\mathrm{mpe}}$, we replace lifted summing out with a lifted maxing out that picks a maximum potential and retains the argument values that lead to that potential

Operator 1 Lifted Maxing-out

Operator MAX-OUT
Inputs:
(1) $g = \phi(\mathcal{A})|C$: a parfactor in G
(2) A_i: an atom in $\mathcal{A}$, to be summed out from g
Preconditions:
(1) For all PRVs $\mathcal{V}$, other than A_i, in G: $rv(\mathcal{V}) \cap rv(A_i|C) = \emptyset$
(2) A_i contains all the logvars $X \in lv(\mathcal{A})$ for which $\pi_X(C)$ is not singleton
(3) $\mathbf{X}^{excl} = lv(A_i) \setminus lv(\mathcal{A} \backslash A_i)$ count-normalised w.r.t. $\mathbf{X}^{com} = lv(A_i) \cap lv(\mathcal{A} \backslash A_i)$ in C
Output: $\phi'(\mathcal{A}')|C'$, such that
(1) $\mathcal{A}' = \mathcal{A} \backslash A_i$
(2) $C' = \pi_{\mathbf{X}^{com}}(C)$
(3) for each valuation $\mathbf{a}' = (\dots, a_{i-1}, a_{i+1}, \dots)$ of $\mathcal{A}'$,
given $\phi(\dots, a_{i-1}, a_i, a_{i+1}, \dots) = (p_i, \bigcup_{j=1}^{l} h_{i,V_j})$ and $r = \text{COUNT}_{\mathbf{X}^{excl}|\mathbf{X}^{com}}(C)$,
$\phi'(\mathbf{a}') = \Big(p', \{h_{A_i}\} \cup \bigcup_{j=1}^{l} h'_{i,V_j}\Big)$,
$p' = (\max_{a_i \in range(A_i)} p_i)^r$,
$h_{A_i} = r \cdot h$, $h = \arg\max_{a_i \in range(A_i)} p_i$ if A_i PCRV,
$h_{A_i} = \{(a_i, n_i)\}_{i=1}^{|range(A_i)|}$ otherwise.
$n_i = r$ for $a_i = \arg\max_{a_i \in range(A_i)} p_i$,
$n_i = 0$ otherwise.
$h'_{i,V_j} = c_j \cdot h_{i,V_j}$, $c_j = \text{COUNT}_{\mathbf{X}^{excl} \cap lv(V_j)|\mathbf{X}^{com}}(C)$
Postcondition: $mpe(G \setminus \{g\} \cup \{\text{MAX-OUT}(g, A_i)\}) = \arg\max_{rv(A_i|C)} G$

for a set of interchangeable randvars, extending parfactors as a consequence, and adapt GC-FOVE. At the end, we look at G_{ex} to compute an MPE. We assume familiarity with operations from relational algebra such as renaming ρ, join $\bowtie$, selection σ, and projection π. We need a count function, the notion of count normalisation, and alignments for the operators.

Definition 5. *Given a constraint $C = (\mathbf{X}, C_{\mathbf{X}})$, for any $\mathbf{Y} \subseteq \mathbf{X}$ and $\mathbf{Z} \subseteq \mathbf{X} \setminus \mathbf{Y}$, the function* $\text{COUNT}_{\mathbf{Y}|\mathbf{Z}} : C_{\mathbf{X}} \to \mathbb{N}$ *tells us how many values of $\mathbf{Y}$ co-occur with the value of $\mathbf{Z}$ in $C_{\mathbf{X}}$, defined by* $\text{COUNT}_{\mathbf{Y}|\mathbf{Z}}(t) = |\pi_{\mathbf{Y}}(\sigma_{\mathbf{Z}=\pi_{\mathbf{Z}}(t)}(C_{\mathbf{X}}))|$. *We define* $\text{COUNT}_{\mathbf{Y}|\mathbf{Z}}(t) = 1$ *for $\mathbf{Y} = \emptyset$. $\mathbf{Y}$ is* count-normalised *w.r.t. $\mathbf{Z}$ in C iff $\exists n \in \mathbb{N}$ s.t. $\forall t \in C_{\mathbf{X}}$:* $\text{COUNT}_{\mathbf{Y}|\mathbf{Z}}(t) = n$, *denoted by* $\text{COUNT}_{\mathbf{Y}|\mathbf{Z}}(C)$.

Definition 6. *A* substitution $\theta = \{X_i \to t_i\}_{i=1}^{n} = \{\mathbf{X} \to \mathbf{t}\}$ *replaces each occurrence of logvar X_i with term t_i (a logvar or domain value). An* alignment *θ between two parfactors $\phi_1(\mathcal{A}_1)|C_1$ and $\phi_2(\mathcal{A}_2)|C_2$ is a one-to-one substitution $\{\mathbf{X}_1 \to \mathbf{X}_2\}$ with $\mathbf{X}_1 \subseteq lv(\mathcal{A}_1)$ and $\mathbf{X}_2 \subseteq lv(\mathcal{A}_2)$ s.t. $\rho_\theta(\pi_{\mathbf{X}_1}(C_1)) = \pi_{\mathbf{X}_2}(C_2)$.*

Operator 1 defines *max-out* to replace *sum-out*. The inputs (parfactor g, PRV A_i) and preconditions are identical. The composition of output ϕ' and the postcondition differ. To eliminate A_i, *max-out* selects for each valuation $(\dots, a_{i-1}, a_{i+1}, \dots)$ the maximum potential p_i in $range(A_i)$ and its corresponding $\arg\max$ value a. The new ϕ' maps $(\dots, a_{i-1}, a_{i+1}, \dots)$ to p_i exponentiated with $r = \text{COUNT}_{\mathbf{X}^{excl}|\mathbf{X}^{com}}(C)$. r denotes the number of instances that the logvars only appearing in A_i ($\mathbf{X}^{excl}$) stand for w.r.t. the remaining logvars ($\mathbf{X}^{com}$).

To retain the range value a, we map arguments in parfactors not only to a potential but also to range values for maxed out PRVs. We store the range values in histograms to have an identical representation for both maxed out PRVs and PCRVs. Histograms also enable us to encode how many instances $\mathbf{X}^{excl}$ represent so as to not carry around constraints for eliminated logvars. Thus, function ϕ in a parfactor $\phi(\mathcal{A})|C$ maps arguments to a pair of a potential and a set of histograms $\{h_{V_1}, \dots, h_{V_l}\} = \bigcup_{j=1}^{l} h_{V_j}$ for already maxed out PRVs V_j.

The logvars in $\mathbf{X}^{excl}$ disappear from the arguments of g with the elimination of A_i and they need to be accounted for in the histograms as well: If A_i is a PRV, assignment a is stored in a peak-shaped histogram h_{A_i} where a maps to r and the other values to 0. If A_i is a (P)CRV, a already is a histogram h to multiply with r, i.e., $h_{A_i} = r \cdot h$. The histograms are multiplied with $\textsc{Count}_{\mathbf{X}^{excl} \cap lv(V_j) | \mathbf{X}^{com}}(C)$, to account for logvars in $\mathbf{X}^{excl}$ appearing in any maxed out PRV V_j. h_{A_i} is added to the set of histograms. The postcondition reflects the max-out operation.

The operator *absorb* is only applied at the beginning to absorb evidence. The sets of histograms are empty. The operators *split*, *expand*, *count-normalise*, and *ground-logvar* all involve duplicating a parfactor and partitioning constraints. The operators *split* and *expand*, which implement splitting operations for PRVs and CRVs respectively, do so s.t. sets of randvars do not overlap (ensure first precondition of *max-out*). The operator *count-normalise* duplicate and partition to count-normalise a set of logvars w.r.t. another set (ensure third precondition of *max out*). Operator *ground-logvar* implements grounding through duplication and partitioning as well. Duplicating a parfactor also duplicates its histograms. Assignments are unaffected as we only change constraints. If a constraint changes for a logvar that appears in a maxed out PRV, the change also applies to it.

The operators *multiply* and *count-convert* remain. Operator 2 shows how to multiply two parfactors. Their histograms concern different PRVs, else LVE would have multiplied them earlier. For each new input, histogram sets unify.

Operator 2 Lifted Multiplication

Operator MULTIPLY
Inputs:
(1) $g_1 = \phi_1(\mathcal{A}_1)|C_1$: a parfactor in G
(2) $g_2 = \phi_2(\mathcal{A}_2)|C_2$: a parfactor in G
(3) $\theta = \{\mathbf{X}_1 \to \mathbf{X}_2\}$: an alignment between g_1 and g_2
Preconditions:
(1) for $i = 1, 2 : \mathbf{Y}_i = lv(\mathcal{A}_i) \setminus \mathbf{X}_i$ is count-normalised w.r.t. $\mathbf{X}_i$ in C_i
Output: $\phi(\mathcal{A})|C$, with
(1) $\mathcal{A} = \mathcal{A}_1\theta \cup \mathcal{A}_2$, and
(2) $C = \rho_\theta(C_1) \bowtie C_2$
(3) for each valuation $\mathbf{a}$ of $\mathcal{A}$, with $\mathbf{a}_1 = \pi_{\mathcal{A}_1\theta}(\mathbf{a})$ and $\mathbf{a}_2 = \pi_{\mathcal{A}_2}(\mathbf{a})$:
given $\phi_1(\mathbf{a}_1) = (p_1, \bigcup_{j_1=1}^{l_1} h_{V_{j_1}})$, $\phi_2(\mathbf{a}_2) = (p_2, \bigcup_{j_2=1}^{l_2} h_{V_{j_2}})$,
$\phi(\mathbf{a}) = \left(p_1^{\frac{1}{r_2}} \cdot p_2^{\frac{1}{r_1}}, \bigcup_{j_1=1}^{l_1} h_{V_{j_1}} \cup \bigcup_{j_2=1}^{l_2} h_{V_{j_2}}\right)$, with $r_i = \textsc{Count}_{\mathbf{Y}_i|\mathbf{X}_i}(C_i)$
Postcondition: $G \equiv G \setminus \{g_1, g_2\} \cup \{\textsc{multiply}(g_1, g_2, \theta)\}$

Operator 3 Count Conversion

Operator COUNT-CONVERT
Inputs:
(1) $g = \phi(\mathcal{A})|C$: a parfactor in G
(2) X: a logvar in $lv(\mathcal{A})$
Preconditions:
(1) there is exactly one atom $A_i \in \mathcal{A}$ with $X \in lv(A_i)$
(2) X is count-normalised w.r.t. $lv(\mathcal{A}) \setminus \{X\}$ in C
(3) for all counted logvars $X^{\#}$ in g: $\pi_{X,X^{\#}}(C) = \pi_X(C) \times \pi_{X^{\#}}(C)$
Output: $\phi'(\mathcal{A}')|C$, such that
(1) $\mathcal{A}' = \mathcal{A} \setminus A_i \cup A'_i$ with $A'_i = \#_X[A_i]$
(2) for each valuation $\mathbf{a}'$ to $\mathcal{A}'$ with $a'_i = h$: $\phi'(\dots, a_{i-1}, h, a_{i+1}, \dots) = \left(p', \bigcup_{j=1}^{l} h'_{V_j}\right)$,
given $\phi(\dots, a_{i-1}, a_i, a_{i+1}, \dots) = (p_i, \bigcup_{j=1}^{l} h_{i,V_j})$
$p' = \prod_{a_i \in range(A_i)} p_i^{h(a_i)}$, $h'_{V_j} = \sum_{a_i \in range(A_i)} h_{i,V_j}$ if $X \in lv(V_j)$, $h'_{V_j} = h_{V_j}$ oth.
Postcondition: $G \equiv G \setminus \{g\} \cup \{\text{COUNT-CONVERT}(g, X)\}$

Operator 3 shows a count conversion of a logvar X in a parfactor $g = \phi(\mathcal{A})|C$ yielding a (P)CRV. While no other PRV in g may contain X, maxed out PRVs may contain X. Each new valuation $(\dots, a_{i-1}, h, a_{i+1}, \dots)$, where h is a histogram for the PCRV, maps to a new potential calculated as before and a set of histograms that are updated. If a maxed out PRV V_j contains the newly counted logvar X, it makes the number of instances of X explicit in V_j: It adds the histograms mapped to by $\phi(\dots, a_{i-1}, a_i, a_{i+1}, \dots)$ $h(a_i)$ times for each a_i.

Let us take a look at an example for *max-out*, *multiply*, and *count-convert*. Consider parfactor $g_2 = \phi_2(User(X), Admin(Y), Fw(X,Y))$ in G_{ex}. PRV $Fw(X,Y)$ contains all logvars and does not appear further in G_{ex}. No logvars disappear after its elimination ($r = 1$). Assume the following mapping with random potentials:

$$(0,0,0) \to (0.9, \emptyset),\ (0,0,1) \to (0.1, \emptyset),\ (0,1,0) \to (0.4, \emptyset), (0,1,1) \to (0.6, \emptyset),$$
$$(1,0,0) \to (0.55, \emptyset), (1,0,1) \to (0.45, \emptyset), (1,1,0) \to (0.2, \emptyset), (1,1,1) \to (0.8, \emptyset)$$

The new parfactor reads $g'_2 = \phi'_2(User(X), Admin(Y))$. For input $(0,0)$, we build the output from $(0,0,0) \to 0.9$ and $(0,0,1) \to 0.1$. The maximum potential is 0.9 with assignment 0 for $Fw(X,Y)$, i.e., $(0,0) \to (0.9^1, [0,1])$. The other pairs are:

$$(0,1) \to (0.6^1, [1,0]), \quad (1,0) \to (0.55^1, [0,1]), \quad (1,1) \to (0.8^1, [1,0]).$$

To max out $User(X)$, we need to multiply g_0, g'_2, and g_3, leading to a parfactor $g_m = \phi_m(Attack1, User(X), Admin(Y), Server)$. For new input $(0,0,0,0)$, the output is the product of the potentials for the potential part. For the histogram set, the output is the union of $\emptyset$, $[0,1]$, and $\emptyset$, to which $\phi_0(0,0)$, $\phi'_2(0,0)$, and $\phi_3(0,0)$ map. Next, we count-convert Y, yielding parfactor $g_{\#} = \phi_{\#}(Attack1, User(X), \#_Y[Admin(Y)], Server)$ with $range(\#_Y[Admin(Y)]) = \{[0,2],[1,1],[2,0]\}$. Since Y appears in $Fw(X,Y)$, we account for $|gr(Y|\top)| = 2$ Y values in the histograms for $Fw(X,Y)$. Given original pairs $\phi_m(0,0,0,0) \to$

$(2, \{[0,1]\})$ and $\phi_m(0,0,1,0) \to (1, \{[1,0]\})$, new pairs are:

$$\phi'_{12}(0,0,[0,2],0) \to (1^0 \cdot 2^2, \{[0,1]+[0,1]\}) = (4, \{[0,2]\})$$
$$\phi'_{12}(0,0,[1,1],0) \to (1^1 \cdot 2^1, \{[1,0]+[0,1]\}) = (2, \{[1,1]\})$$
$$\phi'_{12}(0,0,[2,0],0) \to (1^2 \cdot 2^0, \{[1,0]+[1,0]\}) = (1, \{[2,0]\})$$

Now, we can max out $User(X)$ (includes multiplying the histograms for $Fw(X,Y)$ with $\text{COUNT}_{\{X\}\cap\{X,Y\}|\emptyset}(\top) = 3$). Next, we set up LVE$^{\text{mpe}}$.

LVE^{mpe}: Algorithm 3 outlines LVE$^{\text{mpe}}$ with model G and evidence $\mathbf{E}$ as input. Having absorbed $\mathbf{E}$, it maxes out all PRVs in G, applying transformators if necessary. The result is a parfactor with no arguments, a potential, and histograms for each PRV in $rv(G) \setminus rv(\mathbf{E})$ stating how many instances have a specific value.

Consider an MPE for G_{ex} without evidence. LVE$^{\text{mpe}}$ maxes out $Fw(X,Y)$ and $User(X)$ as given above, yielding $g'_{\#} = \phi'_{\#}(Attack1, \#_Y[Admin(Y)], Server)$. It multiplies g_1 and g_4, count-converts Y in the product, and multiplies the result into $g'_{\#}$. It then maxes out the CRV and the remaining randvars, which results in a parfactor with an empty argument mapping to a potential and a set of histograms, e.g., $\phi() \to (p, \{[0,6]_{Fw}, [0,3]_{Us}, [0,2]_{Ad}, [0,1]_{Se}, [0,1]_{At1}, [0,1]_{At2}\})$. Now, we argue why LVE$^{\text{mpe}}$ is sound.

Theorem 1. *LVE^{mpe} is sound, i.e., computes an MPE for model G equivalent to an MPE computed for $gr(G)$.*

Proof. We assume that LVE as specified by GC-FOVE is correct. Thus, potentials are handled correctly. Replacing $\sum$ with $\arg\max$ produces a correct MPE in the ground case. For a set of interchangeable randvars, the $\arg\max$ assignment is identical given each possible valuation of the remaining randvars. So, assigning one value for all instances of a PRV, as in *max-out*, is correct. Storing the assignments in histograms based on counts is a different representation to avoid storing constraints for logvars only appearing in maxed out PRVs. The enabling operators, except count conversion, do not affect histograms of maxed out PRVs. When count-converting a logvar, we need to count the instances of the same logvar in maxed out PRVs accumulating assignments based on the histogram of the new CRV. With correct eliminations and count representation in histograms, LVE$^{\text{mpe}}$ computes an MPE equivalent to one computed on a ground level. □

Algorithm 3 Outline of LVE$^{\text{mpe}}$

```
LVE^MPE(Model $G$, Evidence $\mathbf{E}$)
  Absorb $\mathbf{E}$ in $G$
  while $G$ has not maxed out PRVs do
    if PRV $A$ fulfils max-out prec. then
      Eliminate $A$ using MAX-OUT
    else
      Apply transformator
  return MULTIPLY($G$)
```

Algorithm 4 Outline of LJT$^{\text{mpe}}$

```
LJT^MPE(Model $G$, Evidence $\mathbf{E}$)
  Construct FO jtree $J$
  Enter $\mathbf{E}$ into $J$
  Pass max messages on $J$
  Get local model $G_i$ from node $i$ that received messages from all neighbours
                                                      ▷ includes messages
  return LVE^MPE($G_i$, $\emptyset$)
```

3.2 Most Probable Explanation with LJT

We adapt LJT to compute an MPE by calculating messages using LVE$^{\mathrm{mpe}}$ to max out non-separator PRVs. Messages carry over the assignments of maxed out PRVs. As outlined in Alg. 4, LJT$^{\mathrm{mpe}}$ constructs an FO jtree J for an input model G, enters evidence $\mathbf{E}$ into J, and passes messages in J, which only needs an inward pass. At the innermost node, it maxes out the remaining PRVs and returns an MPE. For new evidence, LJT$^{\mathrm{mpe}}$ starts over at entering evidence.

Computing an MPE for G_{ex} without evidence starts with constructing an FO jtree as seen in Fig. 2. Without evidence, message passing commences. Nodes 1 and 3 prepare a message for node 2 using LVE$^{\mathrm{mpe}}$. At both nodes, the logvar is count-converted to max out the ground PRV leading to a message from node 1 over $\#_X(User(X))$ with an assignment for $Attack1$ and a message from node 3 over $\#_Y(Admin(Y))$ with an assignment for $Server$. Node 2 receives both messages and maxes out its PRVs to complete the MPE. After maxing out $Fw(X,Y)$ in g_2, it multiplies the result with g_3 and g_4 and count-converts X and Y to multiply each message in. Then, it maxes out $\#_X(User(X))$ and $\#_Y(Admin(Y))$ as well as the remaining randvars producing a parfactor with an empty argument that maps to a potential and a set of histograms identical to the one LVE$^{\mathrm{mpe}}$ yields for G_{ex}. Next, we argue why LJT$^{\mathrm{mpe}}$ is sound.

Theorem 2. *LJT^{mpe} is sound, i.e., computes an MPE for model G equivalent to an MPE computed for $gr(G)$.*

Proof. With a correct LJT, LJT$^{\mathrm{mpe}}$ constructs a valid jtree, the basis for local computations [16]. The *max-out* and *multiply* operators in the roles of marginalisation and combination fulfil the axioms for local computations in a probability propagation [16] that allow us to compute assignments locally and distribute them. Assuming that LVE$^{\mathrm{mpe}}$ is sound, computing assignments is sound. After one pass, the innermost node holds assignments for all model PRVs without evidence, which LJT returns. □

3.3 Discussion

This section looks at MAP queries, a set of queries of different types as well as data and runtime performance.

Maximum A Posterior Queries: MAP, i.e., a query for a most probable assignment to a subset of model PRVs, is a more general case of MPE. The non-commutativity of summing out and maxing out leads to a restriction of the elimination order as it forces an algorithm to sum out randvars before maxing out query randvars. Logvars complicate matters further. Consider

$$\operatorname*{arg\,max}_{(f,u)\in range(Fw(X,Y),User(X))} \sum_{a\in range(Admin(Y))} \phi(u,a,f).$$

We need to sum out $Admin(Y)$ before maxing out $Fw(X,Y)$ and $User(X)$. As $Admin(Y)$ does not contain X and X is not count-convertible (it appears in two PRVs), we need to ground X to sum out $Admin(Y)$.

While MAP is harder to compute than MPE, FO jtrees allow for identifying harmless MAPs, namely, over whole parclusters. After message passing using LVE, a parcluster has all information for its PRVs with outside PRVs summed out. Using LVE$^{\mathrm{mpe}}$, one can compute an MPE for a parcluster, producing an answer to a MAP over that parcluster. If a set of parclusters build a subtree in the FO jtree, LJT$^{\mathrm{mpe}}$ computes a MAP answer over the subtree.

Set of Queries: For a set of probability and assignment queries, LJT shows its particular strength. LVE computes an answer for each query starting with the original model and given evidence, using LVE$^{\mathrm{mpe}}$ for MPE queries. In contrast, LJT constructs an FO jtree for the input model and enters the given evidence. After message passing with LVE, all probability queries can be answered on smaller parcluster models. If an MPE query occurs, LJT does a message pass with LVE$^{\mathrm{mpe}}$. For further probability queries, we either need to cache the old messages or repeat message passing with LVE. For MAP queries, LJT is able to use the LVE messages.

New evidence or a new model does not change the procedure for LVE as it starts with the full model and evidence. For LJT, new evidence means it needs to enter evidence anew and pass messages. A new model leads to constructing a new FO jtree followed by evidence entering and message passing.

Storage and Runtime: Regarding *storage*, switching from a probability to an assignment query requires storing assignments. This additional space requirement comes inherently with the new query type. Making the assignments part of the parfactors allows for dropping assignments no longer necessary during maxing out, setting space free as well. At the end, the MPE is directly encoded in the remaining parfactors to read out. Storing assignments as histograms allows for handling existential and universal quantification as one, reading out assignment counts directly from the histograms, and reducing constraints when logvars disappear through elimination. Of course, LJT requires more space for its FO jtree (independent of query types) trading off space with runtime.

Regarding *runtime*, MPE and probability queries for single randvars given evidence do not differ w.r.t. the magnitude of eliminations to perform. For MPEs, each remaining PRV has to be eliminated. For probability queries, each remaining PRV except the query randvar has to be eliminated. With just one MPE query, LJT$^{\mathrm{mpe}}$ incurs static overhead for constructing an FO jtree without trading it off further. Allowing a set of queries for MPEs, MAPs over parclusters, and probability distributions given a set of evidence, LJT trades off its overhead for construction and message passing easily. LJT re-uses the LVE messages to answer MAP queries, spending effort on one message pass for an MPE query.

In the worst case, MPE as well as probability queries have an exponential runtime even in the lifted case [18]. In such a case, LVE grounds all logvars, yielding computations identical to those performed in the propositional case.

4 Case Study: Risk Analysis

We present a case study on risk analysis of a network. The inherent uncertainties about attacks or causes within detecting if a network is compromised combined with dependencies and influences between different aspects of a network lend itself to model the scenario with a probabilistic model. A network also contains symmetries for various access points, hardware and software components, or users with different affiliations or permissions. Thus, parameterised models provide features to easily capture an attack scenario without an exploding number of randvars and thus, combinatorial effort during computations.

We expand our running example to model an AG close to [9] extending their case study with first-order constructs. In our scenario, logvars represent employees equipped with different permissions depending on their tasks (technician vs. accountant, admin vs. user). CRVs allow for modelling vulnerabilities where a certain number of components needs to be compromised. For the altered AG, Fig. 3 shows a corresponding FO jtree with seven nodes (without local models and separators). The grey node contains all servers in the network. The nodes with thick lines contain the three attacks.

Based on an AG, we can perform a risk analysis. A static analysis assess vulnerabilities in a network at rest. A dynamic analysis aims at identifying possible attack paths after an intrusion, which helps selecting countermeasures. Both analyses generate various queries, which makes LJT a suitable algorithm.

For a *static analysis*, one is interested in the probability of each network component to be compromised, which requires to compute probabilities for each PRV in the AG without evidence. To answer the queries, LJT passes messages on the FO jtree and answers queries for each PRV using a corresponding parcluster. Logvars allow for querying the likelihood of a certain number of users being compromised in the form of a conjunctive query. Changing employee numbers requires a new message pass but no new FO jtree.

Consider an intrusion detection system (IDS) detecting an intrusion. With the information from the IDS as evidence, a *dynamic analysis* generates various queries: (i) MPE to get the current most likely state of the system to find the nodes that are currently most likely compromised as well (undetected by the IDS), requiring one message pass; (ii) Probability of a specific attack being the source or the probability of specific nodes being compromised such as the firewall,

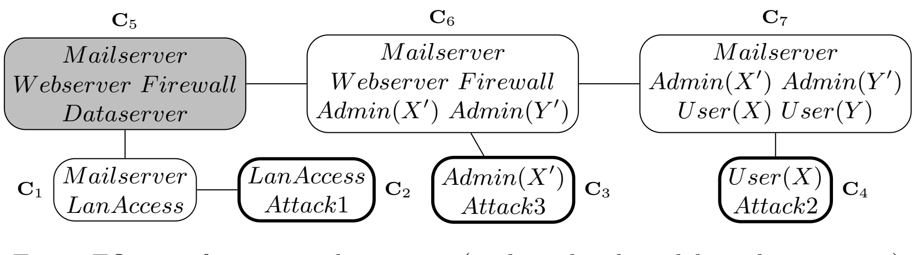

Fig. 3: FO jtree for an attack scenario (without local models and separators)

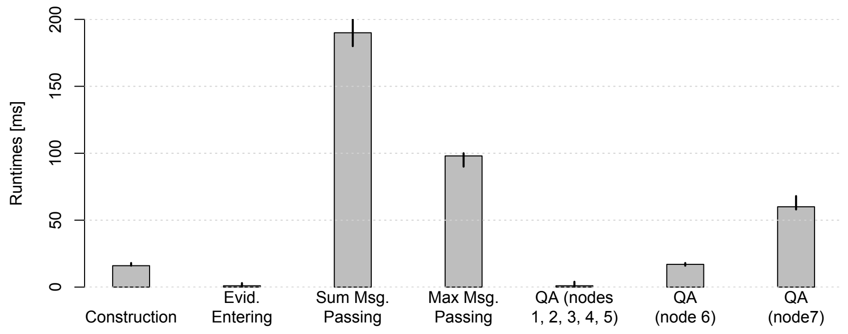

Fig. 4: Runtimes [ms] for the steps of LJT for the AG (whiskers min/max)

requiring passing messages to answer the probability queries; (iii) MAP of, e.g., the servers parcluster provides the most likely state of the servers; (iv) Testing different attack paths by setting different nodes to compromised to track which nodes are now most likely to be compromised or by setting a suspected target to compromised to identify the nodes most likely compromised to reach the target, requiring passing messages for varying evidence sets. With only incremental changes to evidence, only one pass (instead of two) is necessary to propagate the changed evidence from its entry point to the remaining parts of the FO jtree.

We have implemented a prototype of LJT for probability and assignment queries using Taghipour's LVE implementation as a subroutine. LVE has faster runtimes with a single query. But with a second query, LJT has traded off its overhead and provides answers faster. A propositional version of the junction tree algorithm takes minutes with smaller domain sizes and runs into memory problems with the domain sizes given above. We use the AG in Fig. 3 with 20 parfactors and domain sizes $|\mathcal{D}(X)| = 90$, $|\mathcal{D}(X')| = 10$, $|\mathcal{D}(Y)| = 15$, and $|\mathcal{D}(Y')| = 3$ as input, yielding a grounded model size of 2599. LJT constructs the FO jtree in Fig. 3 in 16 ms.

We have tested the following queries

- MPE for the whole model
- MAP at parcluster $\mathbf{C}_5$: $Mailserver, Webserver, Dataserver$
- Probability query for $Attack1$, $Attack2$, $Mailserver$, $Webserver$, $Firewall$, $Dataserver$, $User(x_1)$, $User(x'_1)$, $Admin(y_1)$, $Admin(y'_1)$

We have tested the following evidence

- $Attack2 = true$
- $DataServer = true$
- $User(X) = true$ for 30 instances of X
- $User(X') = true$ for 3 instances of X'

Figure 4 shows runtimes for the LJT steps averaged over several runs. Evidence entering with varying evidence does not take much time even if evidence affects each parcluster. Message passing takes the most time. For an MPE, this step is faster as only one pass is necessary. The time spent on messages is offset during QA. If increasing the domain sizes, the runtimes for inference rise linearly. The domain sizes do not affect the time for construction and evidence entering. Message passing and QA take up slightly more time. The time for answering a query varies between 0.1 and 60 ms, depending on the parclusters used for QA. The times for MAP versus a probability query do not differ on average.

If setting up new evidence and asking a query, LJT outputs an answer after 200-250 ms, which is in average the time LVE needs to answer a single query. After the first query, answers take between 0.1 and 60 ms again. So, even after setting up new evidence, runtimes are reasonably low for new queries.

5 Conclusion

This paper formalises computing MPEs with LVE and LJT, providing a formalism to compute assignment queries in a lifted way. LVE in the form of GC-FOVE and LJT are now able to compute probability queries (likelihood, marginals, conditionals) as well as assignment queries (MPE, MAP). An FO jtree allows for identifying harmless MAP assignment queries that are computable reusing the messages passed for probability queries. The case study on risk analysis in the area of IT security shows LVE and LJT in an applied context where different types of queries occur regularly. Using their formalism allows for capturing large scenarios while providing algorithms for fast query answering.

Future work includes adapting LJT to incrementally changing environments as to not restart FO jtree construction from scratch and salvage as much work as possible. Additionally, we look at ways to further optimise runtimes with parallelisation, caching, and indexing. Currently, we work on setting up a dynamic version that models the temporal or sequential aspect found in many applications.

References

1. Braun, T., Möller, R.: Lifted Junction Tree Algorithm. In: Proceedings of KI 2016: Advances in Artificial Intelligence. pp. 30–42 (2016)
2. Ceylan, İ.İ., Borgwardt, S., Lukasiewicz, T.: Most Probable Explanations for Probabilistic Database Queries. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (2017)
3. Chen, H., Erol, Y., Shen, E., Russell, S.: Probabilistic Model-based Approach for Heart Beat Detection. Physiological Measurement 37(9) (2016)
4. Dawid, A.P.: Applications of a General Propagation Algorithm for Probabilistic Expert Systems. Statistics and Computing 2(1), 25–36 (1992)
5. Dechter, R.: Bucket Elimination: A Unifying Framework for Probabilistic Inference. In: Learning and Inference in Graphical Models., pp. 75–104. MIT Press (1999)
6. Gribkoff, E., van den Broeck, G., Suciu, D.: The Most Probable Database Problem. In: Proceedings of the 1st International Workshop on Big Uncertain Data (2014)
7. Lauritzen, S.L., Spiegelhalter, D.J.: Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. Journal of the Royal Statistical Society. Series B: Methodological 50, 157–224 (1988)

8. Milch, B., Zettelmoyer, L.S., Kersting, K., Haimes, M., Kaelbling, L.P.: Lifted Probabilistic Inference with Counting Formulas. In: AAAI-08 Proceedings of the 23rd Conference on Artificial Intelligence. pp. 1062–1068 (2008)
9. Muñoz-González, L., Sgandurra, D., Barrère, M., Lupu, E.C.: Exact Inference Techniques for the Analysis of Bayesian Attack Graphs. IEEE Transactions on Dependable and Secure Computing PP(99), 1–14 (2017)
10. Nilsson, D.: An Efficient Algorithm for Finding the M Most Probable Configurations in Probabilistic Expert Systems. Statistics and Computing 8(2), 159–173 (1998)
11. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
12. Poole, D.: First-order Probabilistic Inference. In: IJCAI-03 Proceedings of the 18th International Joint Conference on Artificial Intelligence (2003)
13. de Salvo Braz, R.: Lifted First-order Probabilistic Inference. Ph.D. thesis, University of Illinois at Urbana Champaign (2007)
14. de Salvo Braz, R., Amir, E., Roth, D.: MPE and Partial Inversion in Lifted Probabilistic Variable Elimination. In: AAAI-06 Proceedings of the 21st Conference on Artificial Intelligence (2006)
15. Schröder, M., Lüdtke, S., Bader, S., Krüger, F., Kirste, T.: LiMa: Sequential Lifted Marginal Filtering on Multiset State Descriptions. In: Proceedings of KI 2017: Advances in Artificial Intelligence. pp. 222–235 (2017)
16. Shenoy, P.P., Shafer, G.R.: Axioms for Probability and Belief-Function Propagation. Uncertainty in Artificial Intelligence 4 9, 169–198 (1990)
17. Shterionov, D., Renkens, J., Vlasselaer, J., Kimmig, A., Meert, W., Janssens, G.: The most probable explanation for probabilistic logic programs with annotated disjunctions. In: Revised Selected Papers of the 24th International Conference on Inductive Logic Programming - Volume 9046. pp. 139–153 (2015)
18. Taghipour, N., Davis, J., Blockeel, H.: First-order Decomposition Trees. In: Advances in Neural Information Processing Systems 26, pp. 1052–1060. Curran Associates, Inc. (2013)
19. 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)
20. Zhang, N.L., Poole, D.: A Simple Approach to Bayesian Network Computations. In: Proceedings of the 10th Canadian Conference on Artificial Intelligence. pp. 171–178 (1994)