Fast Relational Probabilistic Inference and Learning: Approximate Counting via Hypergraphs

Mayukh Das
University of Texas, Dallas
mayukh.das1@utdallas.edu

Devendra Singh Dhami
University of Texas, Dallas
devendra.dhami@utdallas.edu

Gautam Kunapuli
University of Texas, Dallas
gautam.kunapuli@utdallas.edu

Kristian Kersting
Technical University of Darmstadt
kersting@cs.tu-darmstadt.de

Sriraam Natarajan
University of Texas, Dallas
sriraam.natarajan@utdallas.edu

Abstract

Counting the number of true instances of a clause is arguably a major bottleneck in relational probabilistic inference and learning. We approximate counts in two steps: (1) transform the fully grounded relational model to a large hypergraph, and partially-instantiated clauses to hypergraph motifs; (2) since the expected counts of the motifs are provably the clause counts, approximate them using summary statistics (in/out-degrees, edge counts, etc). Our experimental results demonstrate the efficiency of these approximations, which can be applied to many complex statistical relational models, and can be significantly faster than state-of-the-art, both for inference and learning, without sacrificing effectiveness.

Introduction

Significant advancements in research on Statistical Relational Learning (SRL) and AI (Getoor and Taskar 2007; Raedt et al. 2016) and in lifted inference (Poole 2003; Kersting, Ahmadi, and Natarajan 2009) have allowed for exploiting the symmetries of the data and model during learning and inference. The advantage of these algorithms is that they can succinctly represent and reason with dependencies among the attributes and relations of related objects. One of the key operations inside most, if not all, algorithms is *counting* the satisfied groundings (instances) of a partially instantiated relational rule (a first-order clause). For instance, when learning the parameters or structure of a Markov Logic Network (MLN) (Domingos and Lowd 2009; Khot et al. 2011), or when performing lifted inference (Poole 2003) one has to compute the expected/true counts in the model/data and inside a given cluster of objects.

Counting is a hard combinatorial search problem ($\#P$-complete). Consequently, algorithms for fast, approximate counting have been developed (Das et al. 2016; Sarkhel et al. 2016). *The key observation is that computing exact counts is not essential, particularly when the number of groundings for an object/relation is high*. For instance, knowing whether a Professor published 300 papers or 319 papers does not significantly change the belief over the popularity of the Professor. While reasonably successful, they make a few assumptions – including MLN-specific formulation and lack of support for partial groundings (Sarkhel et al. 2016) or restricted arity of relations (Das et al. 2016).

We relax these assumptions and present a *general approximate counting* technique that transforms the problem of counting partially instantiated clauses to motif-matching in equivalent hypergraph. Provably, counting in the original data corresponds to **computing *the expected counts* of the motif occurrences** in the transformed hypergraph: When this expectation is computed exactly, one can retrieve the true counts. In large data sets, this motivates the approximation of the expectation using summary statistics on the hypergraph. Our *experimental results across several domains on both learning and inference tasks* demonstrate clearly that this approximation indeed relaxes the assumptions of the previous methods and results in more efficient counting while exhibiting on-par performance to exact counting.

The paper makes the following contributions - (1) We present a method for converting structured data and relational/logic clauses to hypergraphs and motifs, respectively, and pose the problem of counting the number of groundings as counting over motifs-matches in the hypergraph. (2) We present and justify an approximation technique and outline the algorithm for counting the number of motifs based on the order of the variables that appear in the clauses. (3) Finally, we demonstrate the efficiency of the proposed approach without sacrificing the effectiveness on standard domains against state-of-the-art baselines on these tasks.

Background and Related Work

Our approach for efficient counting in SRL is inspired by cardinality estimation for query optimization in relational databases (Schiefer, Strain, and Yan 1998; Seputis 2000), that use incremental statistics (histograms etc.) across tables and frequently executed queries. While successful on standard benchmarks, these approaches are not directly applicable to our problem setting where we are interested in learning from adhoc databases. Probabilistic Database systems, such as *'Tuffy'* (Niu et al. 2011) exploit the power of relational databases for persistence and computations in SRL. Though successful, such systems may encounter challenges of relational databases including arbitrary join costs and may require carefully designed join caches for optimization. More importantly, they have not yet been applied to the challenging task of full model learning in SRL (only parameter-learning). Graph-based systems can potentially alleviate such limitations while allowing seamless representation of logical assertions. However, cardinality approximation is an open problem in the context of Graph databases (Neumann and Moerkotte 2011; Stocker et al. 2008).

Counting in SRL is akin to subgraph matching / enumeration, a *$\#P$-complete* (Valiant 1979; Vadhan 2001) problem. Our work relates to *approximate* sub-graph counting, in a graph theoretic context (Slota and Madduri 2013; Bhaskara et al. 2010). Most of these approaches focus, either on exploiting high performance architectures via parallelization, subject to resource availability, or on utilizing properties of restricted classes of graphs. *FACT* (Das et al. 2016) approximates counts via summarization of in-memory Property Graphs (Corby, Dieng, and Hébert 2000) encoded using RDFs [1] but suffer from certain fundamental limitations such as assuming binary relations and independence across relations sharing entities. ISMA (Demeyer et al. 2013) adopts a search tree optimization and pruning approach for subgraph enumeration in large biological networks. It has similar limitations and returns all subgraphs instead of count estimates. Approximate matching has been extensively studied in graph theory, recently in the context of hypergraphs (Dudek et al. 2014; 2013). While these approaches are interesting with theoretical guarantees, they apply to specific classes of simple hypergraphs with bounded degree and regularity which cannot model real-world multi-relational data.

Fürer and Kasiviswanathan (2014) propose a polynomial time sampling-based approximation strategy for counting isomorphic subgraphs matching a given template leveraging bounded-width *ordered bipartite decompositions* of the templates. A key feature of this approach is its provable generalization across varied classes of graphs. Ravkic et al. (2018) extend this principle for parameter learning in SRL via efficient computation of the decompositions. While potentially applicable in our context, these approaches have major limitations that include restricted arity of relations and requirement of decomposability of templates. *Our approach relaxes these limitations*.

Approximate Counting via Hypergraphs

Our goal is to compute the counts of a potentially partially instantiated clause given a database of ground assertions. We proceed by transforming a SRL model into a directed graph notation. Trivial conversion from a logic statement (essentially a *conjunctive rule*) to a simple directed graph has an important limitation of assumimng binary relations. Such graphs, however, cannot represent n-ary relations, which are very common. We employ hypergraphs (Berge and Minieka 1973), generalization of graphs in which a hyperedge joins an arbitrary number of nodes/vertices, in contrast to a graph in which an edge joins two vertices. Formally, a hypergraph is a pair of sets of vertices and hyperedges: $\mathcal{G} \equiv (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$. Since a hypergraph has hyperedges that connect an arbitrary number of nodes, a hyperedge itself is a set of nodes (making $\mathcal{E}_\mathcal{G}$ a set of sets of nodes). Our problem is.

Given: A set of grounded assertions (facts) $\mathcal{F}$, a conjunctive rule/clause C from a SRL model and a (possibly partial) substitution (instantiations) $\boldsymbol{\theta}$ of variables C,
To Do: Return the counts of true groundings $\#(\mathtt{C} \mid \boldsymbol{\theta})$ of the clause C,
Construct: A partially grounded structural hypergraph motif, $\mathcal{M} \equiv (\mathcal{V}_\mathcal{M}, \mathcal{E}_\mathcal{M})$ that represents the clause C and fully grounded hypergraph, $\mathcal{G} \equiv (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ that represents grounded assertions ($\mathcal{F}$), such that counting the number of instances of $\mathcal{M}$ in $\mathcal{G}$ yields an approximation to $\#(\mathtt{C} \mid \boldsymbol{\theta})$.

Example. *A university domain has entity types (variables)* Professor (p), Student (s), Course (c), Term (t), ResearchProject (r) *and* Year (y). *We consider two conjunctive rules in this domain:*

$$\mathtt{AdvisedBy(s,p)} \wedge \mathtt{Teaches(p,c,t)} \wedge \mathtt{TA(s,c,t)} \quad (\mathtt{C}_1)$$

$$\mathtt{AdvisedBy(s,\,p)} \wedge \mathtt{WorksIn(p,\,r,\,y)} \wedge \mathtt{WorksIn(s,\,r,\,y)} \quad (\mathtt{C}_2)$$

The first clause states that s *is advised by* p *and is a TA for* c *that =*p *teaches in* t. *The second states that* s *advised by* p *works on* r *in a* y. *In SRL they are* soft *statements.*

A full grounding refers to a total substitution of values ($\mathbf{t}$) to a set of variables ($\mathbf{v}$), denoted $\boldsymbol{\theta} = \{\, v_1/t_1, \ldots, v_n/t_n \,\}$. A partial grounding is an incomplete substitution of values to some variables. With a slight abuse of notation, we do not distinguish between a partial and full grounding, denoting both by $\boldsymbol{\theta}$. A partial grounding will contain a mixture of constants and variables. It must be mentioned that while **we present conjunctive rules from a parameterized logical notation perspective, the same ideas can be easily extended to relational walks/paths in a relational database**. Our approach can easily be extended to clauses of any form by logical transformation. They can be used in discriminative models that use definite clauses (if-then rules), the coverage of the body (a conjunction) of the clause.

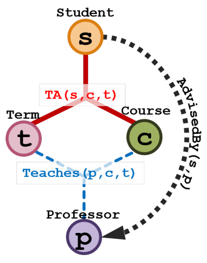
Definition 1 **(Counts).** *For* $\mathtt{Relation}(\mathtt{v}_1, \ldots, \mathtt{v}_t)$*, the predicate counts are the number of true instances of that predicate due to the assertions/facts* $\mathcal{F}$*, given the (partial) grounding* $\boldsymbol{\theta}$ *of the its variables. We denote the predicate counts of* Relation *(*R*) as* $n_\mathtt{R} = \#(\mathtt{R} \mid \boldsymbol{\theta})$.

For a clause C*, the clause counts are the number of true instances of* C *in database* $\mathcal{F}$*, given the partial groundings* $\boldsymbol{\theta}$ *of the variables in the clause. We denote the clause counts for a clause* C *as* $n_\mathtt{C} = \#(\mathtt{C} \mid \boldsymbol{\theta})$.

Example (continued). *Consider the facts in Figure 1 (center), where* Amy *teaches* 3 *courses* {AI, ML, Opt}*, teaching* AI *twice in the* Fa17 *and* Sp18 *terms.* Ben*, is a* TA *for* AI*, then the count for clause* $\mathtt{C}_1$ *given a partial grounding* $\boldsymbol{\theta}_1 = \{\,\mathtt{p/Amy},\, \mathtt{s/Ben}\,\}$ *is* $\#(C_1 \mid \boldsymbol{\theta}_1) = 1$*, since* Ben *is a* TA *for only one class. The count for* C_1 *given a partial grounding* $\boldsymbol{\theta}_2 = \{\,\mathtt{P/Amy},\, \mathtt{T/Fa17}\,\}$ *is* $\#(C_1 \mid \boldsymbol{\theta}_2) = 1$.

Das et al (2016) addressed a similar problem but they were restricted to binary (and unary) predicates. But, predicates in $\mathtt{C}_1$ and $\mathtt{C}_2$ are ternary, an issue that we can handle. Our approach has 3 steps – (i) convert the ground assertions ($\mathcal{F}$) to a hypergraph $\mathcal{G}$, (ii) convert a partially-grounded

[1] https://www.w3.org/RDF/



Teaches(Amy, AI, Fa17),
Teaches(Amy, ML, Fa17)
Teaches(Amy, AI, Sp18),
Teaches(Amy, Opt, Sp18)

TA(Ben, AI, Fa17),
TA(Ena, ML, Fa17)
TA(Cam, AI, Sp18),
TA(Deb, Opt, Sp18)

AdvisedBy(Ben, Amy),
AdvisedBy(Deb, Amy),
AdvisedBy(Fei, Amy)

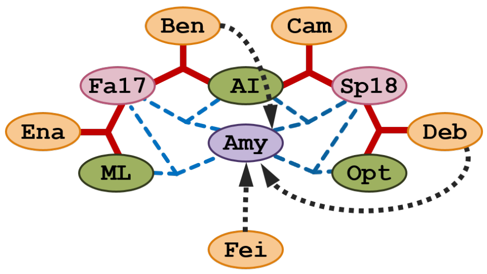
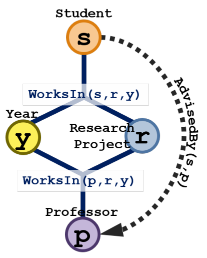


Figure 1: (**left**) Motif $\mathcal{M}_1$ for $\texttt{C}_1$; (**center**) Facts used to ground $\mathcal{M}_1$; (**right**) Ground graph, $\mathcal{G}_1$. Ternary predicates Teaches and TA are represented as hyperedges in both $\mathcal{M}_1$ and $\mathcal{G}_1$. The edges AdvisedBy(Deb, Amy) and AdvisedBy(Fei, Amy) also appear in the grounding of $\texttt{C}_2$ (see Fig. 2).



WorksIn(Amy, MLNs, 2017),
WorksIn(Amy, PRMs, 2017)
WorksIn(Hal, MLNs, 2018)

WorksIn(Ben, PRMs, 2017),
WorksIn(Fei, PRMs, 2017)
WorksIn(Fei, MLNs, 2018),

AdvisedBy(Ben, Amy),
AdvisedBy(Deb, Amy),
AdvisedBy(Fei, Amy),
AdvisedBy(Fei, Hal),

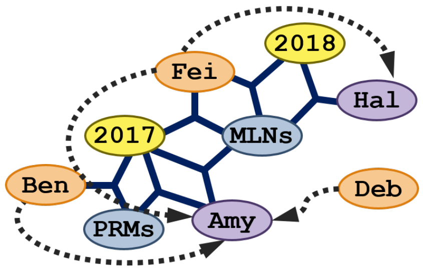


Figure 2: (**left**) Motif $\mathcal{M}_2$ for $\texttt{C}_2$; (**center**) Facts used to ground $\mathcal{M}_2$; (**right**) Ground graph, $\mathcal{G}$. The ternary predicate WorksIn are hyperedges in $\mathcal{M}_2$ and $\mathcal{G}_2$. The edges AdvisedBy(Deb, Amy) (Deb $\rightarrow$ Amy) and AdvisedBy(Fei, Amy) (Fei $\rightarrow$ Amy) also appear in the grounding of $\texttt{C}_1$ (see Fig. 1).

clause to a partially-grounded structural motif ($\mathcal{M}$), and (iii) count the number of subgraphs matching $\mathcal{M}$ in $\mathcal{G}$.

Definition 2 (Partially Grounded Structural Motif). *A motif $\mathcal{M}$ is a Partially Grounded Structural Motif (PGSM) if it is a hypergraph representation of a clause* C*, where some of the nodes are parameterized, while others are instantiated to their respective values. That is, a PGSM is a structural motif arising from a partial grounding.*

Conversion to Hypergraphs

Given a clause C, we construct a hypergraph motif $\mathcal{M}$ as follows. Each variable in every predicate of C is added as a vertex to $\mathcal{V}_{\mathcal{M}}$, the vertex set of $\mathcal{M}$. Next, all the arguments of a predicate are connected by a hyperedge, which is added to $\mathcal{E}_{\mathcal{M}}$, the edge set of $\mathcal{M}$. Directed edges connect variables appearing in binary predicates in the order in which they appear, while an n-hyperedge connects the n variables appearing in a n-ary predicate. The nodes of $\mathcal{M}$ correspond to the variables in C (which can be partially grounded) and the edges correspond to the predicates that contain the respective variables. Given facts ($\mathcal{F}$), a fully-grounded hypergraph $\mathcal{G}$ is similarly constructed. Each constant in $\mathcal{F}$ is added as a vertex to $\mathcal{V}_{\mathcal{G}}$, vertex set of $\mathcal{G}$. Then, all constants appearing in a fact in $\mathcal{F}$ are connected by a hyperedge, and added to $\mathcal{E}_{\mathcal{G}}$, the edge set of $\mathcal{G}$. The construction of $\mathcal{G}$ essentially amounts to parsing assertions (in predicate logic), indexing and insertion into a hypergraph database (Iordanov et al. 2010).

Example (continued). *Figs. 1 and 2 (left) show motifs $\mathcal{M}_1$ and $\mathcal{M}_2$ for $\texttt{C}_1$ and $\texttt{C}_2$ respectively.* Amy *advises three students:* Deb*, who is a teaching assistant,* Fei*, who is a research assistant and* Ben *who is both. Thus, $\mathcal{G}_1$, the ground graph for the given assertions (Fig. 1, middle & right) Similarly, $\mathcal{G}_2$ (Fig. 2) based on the given assertions. Note that,* Ben *is both a teaching and a research assistant and appears in both $\mathcal{G}_1$ and $\mathcal{G}_2$. Also, a student may have more than one advisors (Ex:* Fei *– 2 advisors:* Amy *and* Hal*). This highlights various complex interactions among entities and attributes; counting these is critical in inference and learning.*

An important aspect of our work is that, we exploit the notion of *'Partially-Ordered' Hypergraphs* (Feng et al. 2018). In SRL, relations can be interpreted to have an implicit directionality based on the argument order. While translation from argument ordering to normal directed graphs is conceptually straightforward, it is non-trivial in the case of hypergraphs. In directed graphs with loops, unary/binary relations are represented as directed loops/edges from the vertex denoting the first argument to the vertex denoting the second one. Naively, directed hyperedge may be conceived based on the strict ordering of the arguments of an N-ary relation. But, semantically, it is hard to justify. In some domains such strict ordering may make sense while in others it may not.

Example. *In the ternary relation '*teaches(Amy, AI, FA17)*' the ordering* professor(p) $\prec$ course(c) $\prec$ term(t) *seems most intuitive, considering "*Amy *teaches the course* AI *in term* FA17*". However, the statement*

"The course that Amy *teaches in* FA17 *is* AI*" is equally valid, hence the ordering:* p $\prec$ t $\prec$ c. *A relation such as CoAuthorIn(p, student(s), paper(pa)) is even more ambiguous. Here the orderings, (1)* p $\prec$ s $\prec$ pa*, (2)* s $\prec$ p $\prec$ pa*, (3)* p $\prec$ pa $\prec$ s *etc., are all semantically similar. Thus strict total ordering is not reasonable.*

A completely undirected representation is also not advantageous since directions allow us to leverage certain properties in our formulation. Thus, we use '*partially-ordered hypergraphs*', where the loops and binary edges have the same protocols as a normal directed graphs. However, for a hyperedge with n nodes $(x_1, \ldots, x_n)$, where the argument order in the original n-ary predicate is $1 \prec 2 \prec \ldots \prec n$, we assume the last node x_n to be the sink node (i.e., hyperedge ends here) and all others as source nodes (hyperedge starts here). It is partial-ordered since ordering exist only between the sink node and a source node and not among the source nodes themselves ($x_{\setminus n} \prec x_n$). It applies to both the ground hypergraph of assertions as well as the PGSMs.

Approximate Counting via PGSMs

We consider the case of counts of conjunctive clauses. Partial grounding in a clause C has 2 scenarios based on number of substitutions. Let number of variables in C be $\ell_{\mathtt{C}}$.

(**Case 1**) When $|\boldsymbol{\theta}| = \ell_{\mathtt{C}}$, that is, when the clause is fully grounded, $\#(\mathtt{C} \mid \boldsymbol{\theta}) = 1$ if $\mathcal{M} \in \mathcal{G}$ else 0. Thus counting, here, is equivalent to checking that the grounded motif $\mathcal{M}$ is a subgraph of $\mathcal{G}$.

(**Case 2**) When $0 \le |\boldsymbol{\theta}| < \ell_{\mathtt{C}}$, that is, when the clause is either fully lifted ($\ell_{\mathtt{C}} = 0$) or is partially grounded ($|\boldsymbol{\theta}| < \ell_{\mathtt{C}}$), counting is considerably harder. This is the case we address in the rest of this paper.

Now, for clause C with $\ell_{\mathtt{C}}$ variables, we assume that $m_{\mathtt{C}}$ of these variables are *not grounded*, and $\ell_{\mathtt{C}} - m_{\mathtt{C}}$ variables are grounded. Then, the task is to count the number of groundings of $m_{\mathtt{C}}$, termed as *query* variables, given assignments ($\boldsymbol{\theta}$) to the $\ell_{\mathtt{C}} - m_{\mathtt{C}}$ ground variables. *Ex: For clause* $\mathtt{C}_1$*,* $\ell_{\mathtt{C}_1} = 4$ *and given a partial grounding* $\boldsymbol{\theta} = \{\mathtt{p/Amy}, \mathtt{s/Ben}\}$*, we have* $m_{\mathtt{C}_1} = 2$ *lifted variables (courses* c *and terms* t*) which we want to count over.*

Without loss of generality, let the first $m_{\mathtt{C}}$ variables in C be the the *ungrounded* or *query* variables; that is, $v_i, i = 1, \ldots, m_{\mathtt{C}}$. Given a motif $\mathcal{M}$ constructed from C, the maximum number of possible subgraphs that match $\mathcal{M}$ in $\mathcal{G}$ is $\prod_{i=1}^{m_{\mathtt{C}}} n_i$, where n_i is the number of possible groundings for *query* variable v_i. Let $P(e \mid \mathcal{G})$ denote the probability of a hyperedge being present in $\mathcal{G}$, then the count of the number of matches of $\mathcal{M}$ in $\mathcal{G}$ is also the clause count, $\#(\mathtt{C} \mid \boldsymbol{\theta})$:

$$P(\mathcal{M}) \prod_{v \,\in\, \mathcal{V}_{\mathcal{M}}} n_v = \left[\prod_{e \,\in\, \mathcal{E}_{\mathcal{M}}} P(e \mid \mathcal{G})\right] \cdot \left[\prod_{v \,\in\, \mathcal{V}_{\mathcal{M}}} n_v\right] \quad (1)$$

Intuitively, $P(e \mid \mathcal{G})$ is the *fractional predicate count* that is the ratio of the predicate count given the partial substitutions, to number of groundings of non-substituted variables.

Example. *The probability* $P(\mathtt{Teaches}(\mathtt{Amy}, \mathtt{c}, \mathtt{t}) \mid \mathcal{G})$ *can be computed as the number of courses that* Amy *has taught across all terms divided by the cross-product of the free variables (* c *and* t*) (total number of possible courses times the number of possible terms). Using the partial substitution* $\boldsymbol{\theta} = \{\mathtt{p/Amy}\}$ *we have,* $P(\mathtt{Teaches}(\mathtt{Amy}, \mathtt{c}, \mathtt{t}) \mid \mathcal{G}) = \frac{\#(\mathtt{Teaches(p,c,t)}|\boldsymbol{\theta})}{\#(\mathtt{c}) \cdot \#(\mathtt{t})}$. *In Fig. 1, this is* $4/(3 \cdot 2) = 2/3$.

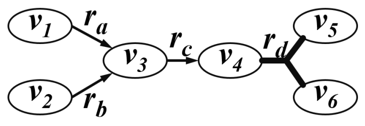


Figure 3: The motif $\mathcal{M} \equiv r_a(v_1, v_3) \wedge r_b(v_2, v_3) \wedge r_c(v_3, v_4) \wedge r_d(v_4, v_5, v_6)$. Note, r_d is a hyperedge.

Expression (1) presents the expected count. If $P(\mathcal{M})$ is computed exactly, we retrieve the true counts. Since that is intractable we find an approximation.

To understand the intuition behind approximating P, consider the motif in Figure 3. The maximum number of times this motif can be present in the ground graph is $\prod_{v \,\in\, \mathcal{V}_{\mathcal{M}}} n_v$. The presence of each hyperedge $e \in \mathcal{G}$ is a Boolean concept (i.e., present or absent in $\mathcal{G}$). Without loss of generality, the joint distribution $P(r_a, r_b, r_c, r_d)$ for Fig 3 is,

$$\prod_{e \in \mathcal{M}} P(e \mid \mathcal{G}) = P(r_a) \cdot P(r_b) \cdot P(r_c \mid r_a, r_b) \cdot P(r_d \mid r_c)$$

Thus, we now view identifying a motif in a graph as a *search in a directed model with Boolean variables*, where finding an edge depends on the previous edge being found. Note that when any of the edges is absent, the motif will not be present in the grounded graph, and this joint probability is automatically driven to 0. The above expression resembles estimating local models, which is commonly done in standard graphical model estimation. In our case, we further *approximate each of these conditional distributions using summary statistics*.

Reverting to (1), the first term is the product of the individual edge distributions conditioned on the incoming edge, and the second term is the cross-product of the total number of groundings of the query (ungrounded) variables in the motif. The second term is computed a single pre-processing step, followed by caching. For the first term, we employ *summary statistics to approximate this distribution*.

Approximation of P

Summary Computation: In order to approximate the joint distribution P, we employ *graph summaries* similar to the ones used in relational database query engines for cardinality estimation (Schiefer, Strain, and Yan 1998; Neumann and Moerkotte 2011; Seputis 2000). Note that these summaries must be selected at the appropriate level of granularity. For instance, finely-grained summaries will correspond to searching the entire database, while highly-aggregated summaries, on the other hand, such as average in-degree and out-degree can lead to poor approximations. Thus, we compute three summaries: (1) node and edge frequencies, (2) node in- and out-degrees, and (3) dependency summaries from hypergraph $\mathcal{G} = (\mathcal{V}_{\mathcal{G}}, \mathcal{E}_{\mathcal{G}})$:

TYPESUM: The **type summary** captures frequencies of node and edge types. Recall that every node in $\mathcal{G}$ corresponds to an entity type, and every edge to a relation type (*edge label* or predicate name). Let $\mathcal{T}$ be the set of entity types, and $\mathcal{R}$ the set of predicate types. With the $\mathsf{TypeOf}(\cdot)$, which returns the node (variable) and edge (relation) types, type summaries $\forall\, \mathtt{T}_i \in \mathcal{T}$ and $\forall\, \mathtt{R}_j \in \mathcal{R}$ are:

$$\#(\mathtt{T}_i) = \#\{\, v \mid v \in \mathcal{V}_{\mathcal{G}},\ \mathsf{TypeOf}(v) = \mathtt{T}_i \,\}, \quad (2)$$
$$\#(\mathtt{R}_j) = \#\{\, e \mid e \in \mathcal{E}_{\mathcal{G}},\ \mathsf{TypeOf}(e) = \mathtt{R}_i \,\}. \quad (3)$$

DEGREESUM: **Degree summaries** are the frequencies of incoming and outgoing edges of every node grouped by edge labels. $\mathsf{InEdges}_{\mathcal{G}}(v)$ and $\mathsf{OutEdges}_{\mathcal{G}}(v)$ represent the sets of incoming and outgoing edges of v in $\mathcal{G}$. The degree summaries $\forall\, v \in \mathcal{V}_{\mathcal{G}}$ and $\forall\, \mathtt{R}_j \in \mathcal{R}$ are:

$$\mathsf{In}_{\mathcal{G}}(v \mid \mathtt{R}_j) = \#\{\, e \mid e \in \mathsf{InEdges}_{\mathcal{G}}(v),\ \mathsf{TypeOf}(e) = \mathtt{R}_j \,\}, \quad (4)$$

$$\mathsf{Out}_{\mathcal{G}}(v \mid \mathtt{R}_j) = \#\{\, e \mid e \in \mathsf{OutEdges}_{\mathcal{G}}(v),\ \mathsf{TypeOf}(e) = \mathtt{R}_j \,\}. \quad (5)$$

DEPENDENCYSUM: Most broadly, we seek to summarize all the **possible dependencies** for a relation $\mathtt{R}_j$: $P(\mathtt{R}_j \mid \mathcal{R} \setminus \mathtt{R}_j)$, that is, the dependency of a R_j on all other relations $\mathcal{R} \setminus \mathtt{R}_j$, which is computationally expensive. Given z relation types in $\mathcal{R}$, we instead construct a $z \times z$ *pairwise dependency matrix*, Δ, whose $(i,\, j)$-th element is $\delta_{ij} = P(\mathtt{R}_i \mid \mathtt{R}_j),\ \forall\, i, j = 1, \ldots, z$. For a pair of relations $\mathtt{R}_i$ and $\mathtt{R}_j$, $P(\mathtt{R}_i \mid \mathtt{R}_j)$ is estimated by sampling paths of length 2 from $\mathtt{R}_j \rightarrow \mathtt{R}_k$. We use these paiwise dependency values for approximation. This matrix is *not symmetric*, since $P(\mathtt{R}_i \mid \mathtt{R}_j)$ will not be the same as $P(\mathtt{R}_j \mid \mathtt{R}_i)$, except under some specific circumstances.

The summary statistics ($\mathbb{S}$) are pre-computed for a ground hypergraph $\mathcal{G}$, and approximate the local distributions:

$$\mathbb{S} = [\{\, \#(\mathtt{T}_i)\}_{\mathtt{T}_i \in \mathcal{T}},\ \#(\mathtt{R}_j)\}_{\mathtt{R}_j \in \mathcal{R}},\ \{\, \mathsf{In}_{\mathcal{G}}(v \mid \mathtt{R}_j)\,\}_{\mathtt{R}_j \in \mathcal{R}},\ \{\, \mathsf{Out}_{\mathcal{G}}(v \mid \mathtt{R}_j)\,\}_{\mathtt{R}_j \in \mathcal{R}},\ \Delta \equiv \{\, P(\mathtt{R}_i \mid \mathtt{R}_j)\,\}_{i,j=1}^{z}].$$

The computation of $\mathbb{S}$ is performed once as a pre-processing step. The counts for any subsequent query motif $\mathcal{M}$ are computed by estimating local edge distributions using $\mathbb{S}$, without explicitly searching through $\mathcal{G}$. We now explain these cases.
Computation of local distributions: The local distribution $P(e)$ of an edge e *with no dependencies* in a motif $\mathcal{M}$, is computed based on grounding of the nodes connected by the edge. For a unary relation or a directed loop such as in Fig. 4(a), we have $P(e) = \frac{\#(\mathtt{R}_e)}{\#(\mathtt{T}_x)}$ when fully lifted. Here, $\mathtt{T}_x = \mathsf{TypeOf}(x)$, the entity-class of node x, and

$$P(e \mid \boldsymbol{\theta}_x) = \begin{cases} 1, & \text{if } e \in \mathcal{G}, \text{ when } \boldsymbol{\theta}_x = \{\mathtt{T}_x/x\}, \\ 0, & \text{otherwise.} \end{cases}$$

For the binary case, fully lifted and fully grounded scenarios are computed similarly, with $P(e) = \frac{\#(\mathtt{R}_e)}{\#(\mathtt{T}_x)\cdot\#(\mathtt{T}_y)}$, and

$$P(e \mid \boldsymbol{\theta}_{xy}) = \begin{cases} 1, & \text{if } e \in \mathcal{G},\ \boldsymbol{\theta}_{xy} = \{\mathtt{T}_x/x, \mathtt{T}_y/y\}, \\ 0, & \text{otherwise.} \end{cases}$$

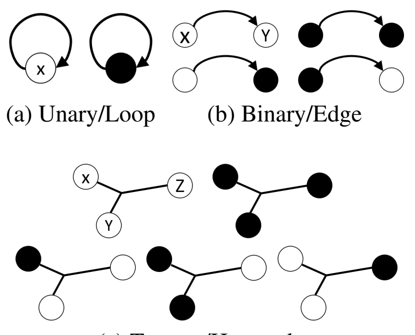


Figure 4: Partial grounding in edges (Solid=ground).

Partial groundings, as shown in Fig. 4(b) (bottom), are computed via degree summaries:

$$P(e \mid \boldsymbol{\theta}_y) = \frac{\mathsf{In}_{\mathcal{G}}(y \mid e)}{\#(T_x)\cdot 1}, \text{ when } \boldsymbol{\theta}_y = \{\mathtt{T}_y/y\},$$
$$P(e \mid \boldsymbol{\theta}_x) = \frac{\mathsf{Out}_{\mathcal{G}}(y \mid e)}{1\cdot\#(T_y)}, \text{ when } \boldsymbol{\theta}_x = \{\mathtt{T}_x/x\}.$$

While the above distributions can be computed exactly, it is not as straightforward for n-ary relations, which result in hyperedges as shown in Fig. 4(c). The fully lifted and fully grounded scenarios are similar to the above cases, but handling partial grounding needs several assumptions:

1. Since, edge directions in an n-hyperedge are not always intuitively clear and may be domain dependent, we follow a partial ordering protocol described earlier.
2. Since for $n > 2$ there can be multiple combinations of partial grounding (shown in Fig. 4(c)) and it is intractable to summarize wrt. each, we assume independence among distributions with respect to each grounded node.

When one or more source nodes are grounded $\boldsymbol{\theta}_{x,y} = \{\mathtt{T}_x/x,\, T_y/y\}$ (bottom middle in Fig. 4(c))

$$P(e \mid \boldsymbol{\theta}_{x,y}) = \left(\frac{\mathsf{Out}_{\mathcal{G}}(x \mid e)}{1\cdot 1\cdot\#(\mathtt{T}_z)}\right)\cdot\left(\frac{\mathsf{Out}_{\mathcal{G}}(y \mid e)}{1\cdot 1\cdot\#(T_z)}\right).$$

However, if the sink node is grounded $\boldsymbol{\theta}_Z = \{Z/z\}$, as show bottom right in Figure 4(c): $[P(e \mid \boldsymbol{\theta}_z) = \mathsf{In}_{\mathcal{G}}(z \mid e)/\,(\#(\mathtt{T}_x)\cdot\#(\mathtt{T}_y)\cdot 1)]$.

The conditional distribution of an edge preceded by other incoming edges is accessed directly from the conditional dependency matrix, Δ. Note that Δ only captures pairwise dependencies and computation of arbitrary dependencies can happen under certain assumptions and properties.
[Independence among incoming edges on same source]. In the motif $\mathcal{M}$, if multiple incoming edges $\mathtt{R}_1, \ldots, \mathtt{R}_j$ are incident on the source node v of the edge $\mathtt{R}_k$, then the dependency $P(\mathtt{R}_k \mid \mathtt{R}_1, \ldots, \mathtt{R}_j)$ factorizes into $\prod_{i=1}^{j} P(\mathtt{R}_k \mid \mathtt{R}_i)$, making the effect of *all preceding edges independent of each other*. This assumption is justified, from a database perspective where *relational joins using the join same attribute are processed independently.*

Algorithm 1 MACH: **M**otif-based **A**pproximate **C**ounting via **H**ypergraphs

```
1: procedure MACH(Clause C, Hypergraph $\mathcal{G}$, Substitution $\boldsymbol{\theta}$, Summary statistics $\mathbb{S}$)
2:     $\mathcal{M}(\mathcal{V}_\mathcal{M}, E_\mathcal{M}) \leftarrow$ ConstructMotif(C | $\boldsymbol{\theta}$)
3:     for $v \in \mathcal{V}_\mathcal{M}$ do                ▷ Count node groundings, $n_v$
4:         if $v \in \boldsymbol{\theta}$ then              ▷ $v$ is grounded in $\boldsymbol{\theta}$
5:             $n_v \leftarrow 1$
6:         else                                  ▷ $v$ is a variable in $\boldsymbol{\theta}$
7:             $n_v \leftarrow \#(\text{TypeOf}(v))$             ▷ All values of $v$
8:         end if
9:     end for
10:    $\chi(\mathcal{M}) \leftarrow \prod_{v \in \mathcal{V}_\mathcal{M}} n_v$                  ▷ Cartesian product
11:    $P(\mathcal{M}) \leftarrow$ ApproxJoint($\mathcal{M}, \boldsymbol{\theta}, \mathcal{G}, \mathbb{S}$)      ▷ See Alg 2
12:    return $\chi(\mathcal{M}) \cdot P(\mathcal{M})$            ▷ As given by Eqn (1)
13: end procedure
```

[Independence due to grounded node]. Two edges in a *PGSM* sharing a common node are rendered independent of each other when the shared node is already instantiated.

[Independence among incoming on different source]. For an n-ary hyperedge, where multiple incoming edges are incident on different source nodes, we assume a similar factorization though the independence property no longer holds, since, constructing summaries for arbitrary dependencies is intractable. While this is a *strong assumption*, in many practical domains, this leads to significant efficiency gains while preserving performance (see experiments).

[Incoming edge to sink is not considered for conditional dependency]. Figure 3 clearly shows, that r_b being incoming on the sink node v_3 of r_a $P(r_a \mid r_b)$. However, r_c considers both r_a and r_b as incoming, resulting in $P(r_c \mid r_a, r_b)$. Note that, the idea generalizes to any arity.

The Algorithm: Algorithm 1 presents MACH: **M**otif-based **A**pproximate **C**ounting via **H**ypergraphs. As pre-processing, grounded hypergraphs are constructed and summarized (TypeSum, DegreeSum, DependencySum). Subsequently, for every clause, a motif is constructed and counted (using the detailed procedure ApproxJoint presented in Algorithm 2). Algorithm 2 outlines the computation of the joint distribution $P(\mathcal{M})$ for a given PGSM $\mathcal{M}$. The ApproxJoint procedure accepts a motif $\mathcal{M}$, the partial substitution $\boldsymbol{\theta}$ of the motif, the ground hypergraph of assertions/facts $\mathcal{G}$ and the summaries $\mathbb{S}$ constructed in the pre-processing step as the arguments and then iteratively analyzes each (hyper)edge in $\mathcal{M}$ ($\forall e \in \mathcal{E}_\mathcal{M}$) **[lines 3-38]**. At any iteration with respect to a given (hyper)edge e, containers $\mathcal{Q}$ and $\mathcal{V}$ store the nodes (of e) which are lifted/query or instantiated/grounded, respectively, based on $\boldsymbol{\theta}$ **[lines 4-5]**.

The 3 important cases – e (1) is fully grounded, (2) fully lifted or (3) partially grounded, are handled explicitly in this procedure. [Case 1:] When e is fully grounded (*i.e.* container $\mathcal{Q}$ is empty), probability of edge $P(e)$ is set to 0/1 based on whether $e|\boldsymbol{\theta} \in \mathcal{E}_\mathcal{G}$ **[lines 7-12]**. [Case 2:] When e is fully lifted (*i.e.* container $\mathcal{V}$ is empty), $P(e)$ is computed as a ratio of summarized frequency of the relation type R_e and the Cartesian product of the query/lifted nodes in $\mathcal{Q}$, if e has no dependencies. The frequency estimates are

Algorithm 2 Product of local distributions

```
1: procedure ApproxJoint(Motif $\mathcal{M}$, Substitution $\boldsymbol{\theta}$, Hypergraph $\mathcal{G}$, Summary statistics $\mathbb{S}$)
2:     $P(\mathcal{M}) \leftarrow 1.0$                    ▷ Initialize joint distribution
3:     for $e \in \mathcal{E}_\mathcal{M}$ do                  ▷ For each edge in the motif
4:         $\mathcal{Q} \leftarrow$ QueryNodesOf($e \mid \boldsymbol{\theta}$)
5:         $\mathcal{V} \leftarrow$ GroundNodesOf($e \mid \boldsymbol{\theta}$)
6:         $P(e) \leftarrow 1$                      ▷ Initialize edge probability
7:         if $\mathcal{Q} = \varnothing$ then                ▷ $e \mid \boldsymbol{\theta}$ is fully grounded
8:             if $e \in \mathcal{E}_\mathcal{G}$ then
9:                 $P(e) \leftarrow 1$           ▷ $e$ exists in both $\mathcal{M}$ and $\mathcal{G}$
10:            else
11:                $P(e) \leftarrow 0$                ▷ $e$ does not exist in $\mathcal{G}$
12:            end if
13:        else if $\mathcal{V} = \varnothing$ then               ▷ $e \mid \boldsymbol{\theta}$ is fully lifted
14:            if $\mathcal{E}_\mathcal{Q} = \text{IncomingEdges}_\mathcal{M}(v) = \varnothing$ then
                                                           ▷ $v \in \mathcal{Q}$
15:                $P(e) \leftarrow \#(\text{R}_e) / \prod_{v \in \mathcal{Q}} n_v$             ▷ From $\mathbb{S}$
16:            else
17:                $P(e) \leftarrow \prod_\mathcal{Q} \prod_{f \in \mathcal{E}_\mathcal{Q}} P(\text{R}_e \mid \text{R}_f)$    ▷ From $\mathbb{S}$
18:            end if
19:        else                          ▷ $e \mid \boldsymbol{\theta}$ is partially grounded
20:            $\mathcal{O}, i \leftarrow$ Order($\mathcal{V} \cup \mathcal{Q}$, $\boldsymbol{\theta}$)
21:            for $v \in \mathcal{V}$ do           ▷ For each grounded node
22:                if $v \in \mathcal{O}$ then              ▷ Source nodes, $\mathcal{O}$
23:                    $p \leftarrow \text{Out}_\mathcal{G}(v) / \prod_{v \in \mathcal{Q}} n_v$        ▷ From $\mathbb{S}$
24:                else                                 ▷ Sink node, $i$
25:                    $p \leftarrow \text{In}_\mathcal{G}(v) / \prod_{v \in \mathcal{Q}} n_v$          ▷ From $\mathbb{S}$
26:                end if
27:                $P(e) \leftarrow P(e) \cdot p$
28:            end for
29:            for $q \in \mathcal{Q}$ do               ▷ For each query node
30:                $\mathcal{E}_q \leftarrow \text{IncomingEdges}_\mathcal{M}(q)$
31:                if $\mathcal{E}_q \neq \varnothing$ and $q \in \mathcal{O}$ then
                                                     ▷ Source nodes, $\mathcal{O}$
32:                    $p \leftarrow \prod_{f \in \mathcal{E}_q} P(\text{R}_e \mid \text{R}_f)$         ▷ From $\mathbb{S}$
33:                end if
34:                $P(e) \leftarrow P(e) \cdot p$
35:            end for
36:        end if
37:        $P(\mathcal{M}) \leftarrow P(\mathcal{M}) \cdot P(e)$
38:    end for
39:    return $P(\mathcal{M})$
40: end procedure
```

accessible from TypeSum. In case there are dependencies (incoming edges on nodes of $\mathcal{Q}$) conditionals are accessed from DependencySum $\forall v \in \mathcal{Q}$ and multiplied together[2], $\prod_{f \in \mathcal{E}_\mathcal{M} \setminus e} P(\text{R}_e \mid \text{R}_f)$, **[lines 13-18]**. Finally, Case 3, e being partially grounded ($\mathcal{Q} \neq \emptyset$ & $\mathcal{V} \neq \emptyset$), requires a more involved analysis **[lines 20-35]**. Determining the partial-order of the nodes of e is important when computing probability of the partially-grounded (hyper)edge e. $\mathcal{O}$ and i maintains the index over all *source* nodes and sink node respectively **[line 20]**. First we inspect all grounded nodes of e (*i.e.* $\mathcal{V}$). Since a grounded node nullifies any possible dependency, we need not worry about conditionals. If a grounded node $v \in \mathcal{V}$ is a source node $v \in \mathcal{O}$, temporary distribution p *w.r.t.* v is given by the ratio of the out-degree of v, $Out_\mathcal{G}(v)$, to the Carte-

[2]Due to assumptions on independent pairwise conditionals.

sian product of the type frequencies of all the lifted/query nodes. In case of a sink node, $v \in i$, the ratio is with the in-degree of the node, since the (hyper)edge e is assumed to be partially directed from the source nodes to the sink node. Note that v here is grounded, *i.e.* and actual entity is in $\mathcal{G}$ and the degree estimates are accessible from $\mathbb{S}$, specifically DEGREESUM. $P(e)$ is then updated with the product of the temporary distribution of all $v \in \mathcal{V}$ **[lines 21-28]**. Finally, we inspect all lifted/query nodes of e ($q \in \mathcal{Q}$). For any query node q, **iff it is a source node** (refer to properties & assumptions), the temporary distribution p is computed as the product of the conditionals with respect to all incoming edges on q ($\prod_{f \in \mathcal{E}_q} P(\mathtt{R}_e \mid \mathtt{R}_f)$, where $f \in \mathcal{E}_q$ is an incoming edge on q and R_f is its relation type). Again, $P(e)$ is updated with the product of temporary distributions of all $q \in \mathcal{Q}$ **[lines 29-35]**. $P(\mathcal{M})$ is thus the product of all the factors *i.e.* the edge distributions.

Theoretical properties: MACH is aimed at efficient and performance-preserving count approximations of satisfied groundings of logical clauses in Statistical Relational models. Efficiency is the key aspect of the approximation strategy. Note, there are two distinct phases in our approach - (1) Pre-processing phase including construction of the ground hypergraph of assertions $\mathcal{G}$ and summary $\mathbb{S}$, and (2) Computing APPROXJOINT phase. While, pre-processing has an asymptotic time complexity of $O(n^2.|R| + n)$, where n is the number of assertions/facts given and $|R|$ is the number of distinct relation types in the domain, it is computed only once for a data set and is thus inconsequential.

Complexity of APPROXJOINT, however, is crucial since it is called arbitrary number of times during structure / parameter learning and inference. Assuming that the summary statistics $\mathbb{S}$ can be accessed efficiently, if we denote the maximum length of a conjunctive clause/motif as k, maximum arity of the clause/motif as $\mathcal{A}$ and maximum in-degree of a node in the motif as d, then the asymptotic time complexity of APPROXJOINT for any given motif $\mathcal{M}$ is $O(k.\mathcal{A}.d)$. Most SRL systems work with reasonably small clause lengths for tractability, making k a constant. Hence, the effective complexity reduces to $O(\mathcal{A}.d)$. Cartesian product of the nodes in $\mathcal{M}$ ($\prod_{v \in \mathcal{V}_\mathcal{M}} n_v$) is a single operation requiring *constant* time and is inconsequential. Performance-preserving approximation requires negligible change (deterioration) in the predictive performance of an SRL model compared to its original performance with exact counts. Our evaluation results corroborates our claim. A *PAC* analysis for approximation error bounds is an interesting future research direction.

Experimental Evaluation

We investigate the following questions: **(Q1)** Is MACH effective and efficient in full model learning with n-ary relations compared to a robust baseline? **(Q2)** Is modeling n-ary relations faithfully crucial when learning relational model? and **(Q3)** How does MACH compare (scaling vs. performance) to a state-of-the-art database-centric MLN system?

System: We implemented MACH[3] using Java-based *HypergraphDB* architecture (Iordanov et al. 2010), which facilitates construction, operations and persistence of hypergraphs. All database operations related to MACH are done via carefully designed Java methods using the APIs. MACH is a *pluggable* independent module that can be seamlessly integrated with any Java-based SRL system that uses counts.

Baselines: To evaluate the effectiveness and efficiency of MACH on full structure and parameter learning of MLNs, we compared against the following baselines: (1) *FACT* (Das et al. 2016), which approximates counts via message-passing on normal multi-graphs, integrated with *MLN-Boost* and (2) basic *MLN-Boost*, the vanilla implementation of boosted structure and parameter learner for MLNs (Khot et al. 2011), the state-of-the-art in MLN (full) model learning without count approximations. It is reasonable to expect comparison against scalable learning of MLNs (Venugopal, Sarkhel, and Gogate 2015; Sarkhel et al. 2016). However, they cannot handle partial groundings without an exponential blow-up in model / database size since they resort to creating new predicates for every grounding in a clause.

Data Sets: We used three standard SRL data sets: *UW-CSE, Citeseer and WebKB*, a biomedical data set *Carcinogenesis* (Srinivasan et al. 1997), and an NLP/Information-Extraction(IE) data set *NELL-Sports* for evaluation. *UWCSE* is a relation-extraction/link-prediction task over an anonymized representation of staff, students and faculty of 5 different computer science departments; the target is to predict the `AdvisedBy` relation between faculty and students. *Citeseer* is a citation data set for IE, where the target is to predict which field a paper belongs to based on the title. *WebKB* is a consolidated data set of links among the departmental web pages of 4 universities, and the target is to predict if a web page is a faculty page. *Carcinogenesis* is biomedical data set of the structures of chemical compounds (drugs), and the task is to predict if they are carcinogenic. NELL (Carlson et al. 2010) is a system that extracts information from online text, and converts them into a probabilistic knowledge base. *NELL-Sports* is NELL data from the sports domain consisting of information about players and teams. The task is to predict whether a team plays a particular sport.

Measures: We computed AUC-ROC, AUC-PR, CLL, F1 and running times averaged over 5 random train/test splits.

(Q1: full model learning) Table 1 summarizes the performance and efficiency results of MACH against the baselines for structure and parameter learning of MLNs. It is clearly evident that there is no real deterioration in predictive performance due to count approximation in MACH (across all data sets) compared to MLN-Boost. However, MACH substantially beats the baselines on efficiency (learning time), especially on larger data sets such as *Citeseer*, where it is about 4 times faster. Even on smaller data sets such as *UW-CSE* and *WebKB*, the difference in learning times, though not as large, is still sizable. This answers **(Q1)** affirmatively.

(Q2: n-ary relations) All data sets except *WebKB* and *NELL-Sports* contain several ternary relations. On such data sets we run FACT by decomposing n-ary relations into $\binom{n}{2}$ binary relations with the order of the arguments aligned with the order in the original relation. We decompose a relation

[3]Code @ https://github.com/mayukhdas/MACH

Table 1: Results: Performance vs. Efficiency (running time for **L**earning and **I**nference in seconds). ** indicates n-ary predicates.

		Performance				Efficiency	
Data Sets	**Methods**	**AUC–ROC**	**AUC–PR**	**CLL**	**F1**	**L-Time [s]**	**I-Time [s]**
UWCSE**	MACH	0.981	0.337	-0.133	0.217	**13.2**	5.1
	FACT	0.500	0.0068	-0.061	NaN	7.48	2.8
	MLN-Boost	0.998	0.361	-0.134	0.227	27.5	9.4
Citeseer**	MACH	0.998	0.989	-0.173	0.973	**10837**	12.52
	FACT	0.97	0.92	-0.256	0.934	11042	12.45
	MLN-Boost	0.999	0.998	-0.059	0.977	42499	27.42
Carcinogenesis**	MACH	0.525	0.568	-0.811	0.328	108.48	1.8
	FACT	0.500	0.550	-0.704	NaN	102.5	1.5
	MLN-Boost	0.587	0.572	-0.902	0.489	153.84	2.37
WebKB	MACH	1.0	1.0	-0.049	1.0	**5.8**	0.757
	FACT	1.0	1.0	-0.076	1.0	5.97	0.797
	MLN-Boost	1.0	1.0	-0.075	1.0	8.13	0.896
NELL-Sports	MACH	0.78	0.65	-1.43	0.65	253.92	1.03
	FACT	0.76	0.64	-1.38	0.65	238.07	2.01
	MLN-Boost	0.78	0.66	-0.55	0.68	396.24	2.20

Table 2: Performance vs. Efficiency (running time for **I**nference in seconds) compared against *Tuffy*.

Data Sets	**System**	**AUC-ROC**	**I-Time [s]**
UW-CSE	MACH	0.892	**20.06**
	Tuffy	0.877	37.11
Carcinogenesis	MACH	0.524	**199.8**
	Tuffy	0.560	944.98

$\mathtt{R}(\mathtt{v}_1, \ldots, \mathtt{v}_n)$, $n > 2$ as $\{\mathtt{R}_{ij}(\mathtt{v}_i, \mathtt{v}_j)\}$, $(i < j)$, that is $i \in [1, n)$, $j \in (i, n]$. We modify the assertions based on the new relations as well. Observe that, though the efficiency gain is equivalent to (and at times better than) MACH, there is a significant deterioration in performance, especially on the relatively smaller data sets. In the case of a large data set such as *Citeseer*, the performances are relatively similar since the number of spurious assertions introduced by the decomposition of n-ary relations are negligible compared to the sheer size of the original data set. In contrast, this size difference is not prevalent in smaller data sets, which results in worse predictive performance compared to MLN-Boost and MACH. Thus, it is critical that n-ary relations are represented faithfully, which answers **(Q2)** affirmatively.

(Q3: HypergraphDB versus DB) In order to evaluate against a database-centric MLN system, we compared MACH with *Tuffy* (Niu et al. 2011). While *Tuffy* is a robust MLN engine integrated with PostgreSQL RDBMS, an unbiased comparison with our system is challenging. This is because, although *Tuffy* has no restrictions on the type (discriminative vs. generative) of MLNs (unlike MLN-Boost which can only represent discriminative MLNs via Horn clauses), it does not support structure learning (which MLN-Boost can perform, simultaneously with parameter learning). In order to keep the comparison fair, we learn MLNs via MLN-Boost, convert it into *Tuffy* format, and then compare the inference performance and efficiency with MACH. MACH was integrated with the inference engine of MLN-Boost directly. Table 2 summarizes the results. Observe how MACH shows significant gain in efficiency, while being at par with *Tuffy*'s performance. Since *Tuffy* reports time inclusive of setup and post-processing, we did the same for MACH. Note that inference time of MACH for *UW-CSE* is 20.06 sec., which is almost twice as fast as *Tuffy*, includes the setup and hypergraph construction and summarization time of 13.26 sec. On *Carcinogenesis* we observe that MACH is approximately 5 times as fast (w/ 192.2 s. setup time). The performances are almost equivalent, though the inference algorithms are different. This allows us to answer **(Q3)** affirmatively. Finally, note that *Tuffy* infers on all possible instances of the target. For fair comparison we deactivate sub-sampling of negative examples in MLN-Boost.

Conclusion

We present a method for approximating counts of logical rules in SRL models via transforming both the clauses and the full set of groundings into a hypergraph. We showed how counting the number of subgraphs in these hypergraphs correspond to counting the number of groundings in the SRL model. We presented the use of summary statistics to approximate this count. There are several possible directions to extend our work - first is more rigorous evalation on large data sets across other models. Second is that is while in our work we consider only conjunctions, extending the idea to count over arbitrary clauses. Finally, combining our method with methods such as FROG (Shavlik and Natarajan 2009) that eliminates trivial groundings of clauses can potentially allow our algorithm to be employed in real large data sets.

Acknowledgements

MD, GK & SN gratefully acknowledge the support of CwC Program Contract W911NF-15-1-0461 with the US Defense Advanced Research Projects Agency (DARPA) and the Army Research Office (ARO). DSD acknowledges the National Institute of Health (NIH) grant no. R01 GM097628. Any opinions, findings and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the DARPA, ARO or the US government. Parts of this work grew out of discussions of KK within the DFG Research Training Group GRK 1994.

References

Berge, C., and Minieka, E. 1973. *Graphs and hypergraphs*. North-Holland Publishing Company Amsterdam.

Bhaskara, A.; Charikar, M.; Chlamtac, E.; Feige, U.; and Vijayaraghavan, A. 2010. Detecting high log-densities: An $O(N^{\frac{1}{4}})$ approximation for densest K-subgraph. In *SOTC*.

Carlson, A.; Betteridge, J.; Kisiel, B.; Settles, B.; Hruschka Jr, E.; and Mitchell, T. 2010. Toward an architecture for never-ending language learning. In *AAAI*.

Corby, O.; Dieng, R.; and Hébert, C. 2000. A conceptual graph model for W3C resource description framework. In *ICCS*.

Das, M.; Wu, Y.; Khot, T.; Kersting, K.; and Natarajan, S. 2016. Scaling Lifted Probabilistic Inference and Learning Via Graph Databases. In *SDM*.

Demeyer, S.; Michoel, T.; Fostier, J.; Audenaert, P.; Pickavet, M.; and Demeester, P. 2013. The index-based subgraph matching algorithm (ISMA): Fast subgraph enumeration in large networks using optimized search trees. *PLOS One*.

Domingos, P., and Lowd, D. 2009. *Markov Logic: An Interface Layer for AI*. M & C.

Dudek, A.; Frieze, A.; Ruciński, A.; and Šileikis, M. 2013. Approximate counting of regular hypergraphs. *Information Processing Letters*.

Dudek, A.; Karpinski, M.; Ruciński, A.; and Szymańska, E. 2014. Approximate counting of matchings in (3,3)-hypergraphs. In *Algorithm Theory – SWAT 2014*.

Feng, F.; He, X.; Liu, Y.; Nie, L.; and Chua, T.-S. 2018. Learning on partial-order hypergraphs. In *WWW*.

Fürer, M., and Kasiviswanathan, S. P. 2014. Approximately counting embeddings into random graphs. *Combinatorics, Probability and Computing*.

Getoor, L., and Taskar, B. 2007. *Introduction to Statistical Relational Learning*. MIT Press.

Iordanov, B.; Vandev, K.; Costa, C.; Marinov, M.; Saraiva de Queiroz, M.; Holsman, I.; Picard, A.; and Bogdahn, I. 2010. HyperGraphDB 1.3.

Kersting, K.; Ahmadi, B.; and Natarajan, S. 2009. Counting Belief Propagation. In *UAI*.

Khot, T.; Natarajan, S.; Kersting, K.; and Shavlik, J. 2011. Learning Markov logic networks via functional gradient boosting. In *ICDM*.

Neumann, T., and Moerkotte, G. 2011. Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In *ICDE*.

Niu, F.; Ré, C.; Doan, A.; and Shavlik, J. W. 2011. Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS. *PVLDB*.

Poole, D. 2003. First-Order Probabilistic Inference. In *IJCAI*.

Raedt, L. D.; Kersting, K.; Natarajan, S.; and Poole, D. 2016. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lect. on AI & ML*.

Ravkic, I.; Znidarsic, M.; Ramon, J.; and Davis, J. 2018. Graph sampling with applications to estimating the number of pattern embeddings and the parameters of a statistical relational model. *Data Min. Knowl. Discov.* 32.

Sarkhel, S.; Venugopal, D.; Pham, T.; Singla, P.; and Gogate, V. 2016. Scalable training of Markov logic networks using approximate counting. In *AAAI*.

Schiefer, B.; Strain, L. G.; and Yan, W. P. 1998. Method for estimating cardinalities for query processing in a relational database management system. US Patent 5,761,653.

Seputis, E. A. 2000. Database system with methods for performing cost-based estimates using spline histograms. US Patent 6,012,054.

Shavlik, J., and Natarajan, S. 2009. Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *IJCAI*.

Slota, G. M., and Madduri, K. 2013. Fast approximate subgraph counting and enumeration. In *Intl. Conf. on Parallel Processing*.

Srinivasan, A.; King, R. D.; Muggleton, S. H.; and Sternberg, M. 1997. Carcinogenesis predictions using ILP. *Inductive Logic Programming*.

Stocker, M.; Seaborne, A.; Bernstein, A.; Kiefer, C.; and Reynolds, D. 2008. Sparql basic graph pattern optimization using selectivity estimation. In *WWW*.

Vadhan, S. P. 2001. The complexity of counting in sparse, regular, and planar graphs. *SICOMP*.

Valiant, L. G. 1979. The complexity of enumeration and reliability problems. *SICOMP*.

Venugopal, D.; Sarkhel, S.; and Gogate, V. 2015. Just count the satisfied groundings: Scalable local-search and sampling based inference in MLNs. In *AAAI*.