Approximate Counting for Fast Inference and Learning in Probabilistic Programming

Mayukh Das*, Devendra Singh Dhami*, Gautam Kunapuli*, Kristian Kersting†, Sriraam Natarajan*

*The University of Texas at Dallas, †TU Darmstadt;
*{mayukh.das1, devendra.dhami, gautam.kunapuli, sriraam.natarajan}@utdallas.edu, †kersting@cs.tu-darmstadt.de

Abstract

Inference and Parameter Learning inside Probabilistic Programming use an important operation that can be approximated: *counting*. We present an efficient approximation scheme that allows for fast counting and consequently faster inference and learning.

Keywords Scaling, Approximate Counting, Hypergraph

1 Introduction

Learning and inference in Probabilistic Programming and related Statistical Relational Learning (SRL) is difficult, particularly due to one major bottleneck – counting over supporting instances of partial programs (partially instantiated clauses/rules in probabilistic logic representation), [4, 7, 8]. Since counting is hard (#*P*-complete), learning and inferences become intractable for large data sets. But for most learning/inference tasks, **exact** counts are unnecessary in presence of relatively high number of instances and hence can be approximated. *For example, there will not be any significant change in the belief over the popularity of a Professor, based on whether (s)he has* 400 *or* 427 *publications.*

Several recent approaches for fast, approximate counting [3, 9], while reasonably successful, have major limitations due to some restrictive assumptions – including restricted arity [3] or MLN-specific algorithms and lack of support for partial groundings [9]. Our approach, ***M***otif-based ***A***pproximate ***C***ounting via ***H***ypergraphs (MACH), is a generalized count approximation technique enabling scalable counting of partial programs.

2 Approximate Counting via Hypergraphs

Our approach for computation of satisfied instances (true groundings) of rules (in First Order Logic), exploits it's equivalence with the problem of counting matching subgraphs. However, subgraph matching is hard problem in itself (#*P*-complete) and Das et al. [3] shows how pre-computed graph statistics followed by a message passing strategy, results in a reasonable approximation in large (dense) data sets. While effective, their formulation is empirical at best and lacks theoretical basis. MACH provides a fundamentally robust formulation using Hypergraphs.

Compilation: A hypergraph [1] is a generalization of a graph. A hyperedge can connect an arbitrary number of vertices and thus, can faithfully represent relations/predicates of any arity, as opposed to an ordinary edge (limited to binary). The key idea is to transform the data (set of facts/ground literals) into a hypergraph $\mathcal{G}$ and a FOL clause (whose #instances we are interested in) to a hypergraph motif $\mathcal{M}$, termed herein as a **P**artially **G**rounded **S**tructural **M**otif (*PGSM*). We then (approximately) count the sub(hyper)graphs in $\mathcal{G}$ that match the pattern $\mathcal{M}$. The transformation creates vertices from the arguments of a predicate and the predicate/relation itself forms a partially-directed labeled hyperedge. "*Partially Grounded*" indicates that some of the terms in $\mathcal{M}$, may be instantiated. However, all vertices/nodes in $\mathcal{G}$ are grounded.

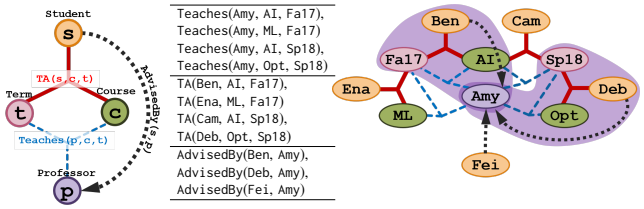

Figure 1. (**left**) Motif $\mathcal{M}_1$ for $\mathtt{C}_1$; (**middle**) Facts in the world; (**right**) Ground hypergraph, $\mathcal{G}_1$. Ternary predicates Teaches and TA are represented as hyperedges, via blue dashed lines and red solid lines resp., in $\mathcal{M}_1$ and $\mathcal{G}_1$.

Example. *A FOL clause* $\mathtt{C}_1 = \mathtt{AdvisedBy(s,p)} \wedge \mathtt{TA(s,c,t)} \wedge \mathtt{Teaches(p,c,t)}$ *transforms into a PGSM* $\mathcal{M}_1$ *as shown in Figure 1. Hypergraph* $\mathcal{G}_1$ *is created from the facts. Note, there are only 2 instances in* $\mathcal{G}_1$ *that match* $\mathcal{M}_1$ *(purple shaded regions)*

Approximation: The number of subgraphs matching a motif $\mathcal{M}$ can be expressed as the ***expected*** vertex product of $\mathcal{M} = (\mathcal{V}_\mathcal{M}, \mathcal{E}_\mathcal{M})$ given $\mathcal{G}$.

$$P(\mathcal{M}|\mathcal{G})\left(\prod_{v\in\mathcal{V}_\mathcal{M}} n(v)\right) = \left(\prod_{e\in\mathcal{E}_\mathcal{M}} P(e|\mathcal{G})\right).\left(\prod_{v\in\mathcal{V}_\mathcal{M}} n(v)\right) \quad (1)$$

where, $n(v)$, the *typecount*, is the #entities in the domain of the variable v. *Ex: In Figure 1,* $n(s) = 5$ *as there are 5 students.* If it is grounded ($v = \mathbb{C}$), $n(v) = 1 \vee 0$ based on existence of $\mathbb{C}$ in $\mathcal{G}$. $P(e|G)$ represents the **local model of a hyperedge** (regarded as a random variable) in $\mathcal{M}$. Thus, the joint model of the motif $\mathcal{M}$ is factorizable into a product over the local models. The second part $\prod_{v\in\mathcal{V}_\mathcal{M}} n(v)$ of equation 1 is straightforward and is the Cartesian product over all the variables in a clause. The first part $P(\mathcal{M}|\mathcal{G})$, is what we are interested in estimating. If the distribution can be computed exactly, eqn. 1 will return the exact counts. However, exact

estimation of the local edge distributions is inefficient and we approximate them. For further clarity, consider the motif $\mathcal{M}$

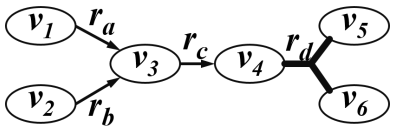

Figure 2. Motif $\mathcal{M} \equiv r_a(v_1, v_3) \wedge r_b(v_2, v_3) \wedge r_c(v_3, v_4) \wedge r_d(v_4, v_5, v_6)$. Hyperedge r_d represents ternary relation $\mathsf{r_d}()$. in Figure 2. Here, $\prod_{e \in \mathcal{E}_\mathcal{M}} P(e|\mathcal{G}) = P(r_a)\ P(r_b)\ P(r_c|r_a, r_b)\ P(r_d|r_c)$. Note, r_a and r_b have no dependencies, whereas models of r_c and r_d are distributions ***conditioned*** on incoming edges on the source vertices. For any directed (hyper)edge, source vertex is where the edge starts and sink is where it ends. Since, for any n-hyperedge ($n > 2$) direction is *ambiguous*, we use 'Partially-Ordered' Hypergraphs[5] where the last argument in an n-ary predicate is considered as the sink vertex and the rest as sources. The joint model for $\mathcal{M}$ is thus akin to a directed probabilistic graphical model over edge random variables.

To efficiently compute local models, we use graph summary statistics (of $\mathcal{G}$) of 3 types – (1) TypeSum: cardinality of object and relation types, (2) DegreeSum: out and in-degrees of all objects, and their type averages (3) DependencySum: **pairwise conditionals** among all relation types estimated via sampling $\mathcal{G}$

TypeSum allows us to compute the Cartesian product and prior distributions of edges in $\mathcal{M}$ ($P(r) = \frac{|r|}{n(u) \times n(v) \times \ldots}$, r is a hyperedge $\langle u, v, \ldots \rangle$). If one or more vertices in r are grounded, we use DegreeSum. For instance, if r is binary $\langle u, v \rangle$, and v is grounded ($v = \mathbb{C}$) then $P(r) = \frac{In_r(v=\mathbb{C}|\mathcal{G})}{n(u) \times n(v)}$, using the in-degree since v is sink. n-hyperedges ($n > 2$) follow the partial ordering protocol described earlier. Conditional distributions are critical and require us to leverage several properties and assumptions – (1) independence among incoming parents incident on same source *[Ex: $P(r_c|r_a, r_b) = P(r_c|r_a)P(r_c|r_b)$]*, (2) independence due to grounded shared vertex *[Ex: If v_3 was grounded $P(r_c|r_a, r_b) = P(r_c)$]* and (3) independence among incoming parents on different source vertices of a n-hyperedge *[Ex: If there was another incoming edge r_e on v_4, assume $P(r_d|r_c, r_e) \approx P(r_d|r_c)P(r_d|r_e)$]*. While the last one is a restrictive assumption, it allows for using pairwise DependencySum, since efficient computation of all possible conditionals is intractable. *In summary, as preprocessing, grounded hypergraph $\mathcal{G}$ is constructed, summarized and stored. Then, every given clause is transformed into motif $\mathcal{M}$ and approximately counted as described above.*

3 Experiments

We evaluate MACH based on 2 experimental questions, **(Q1)** Is MACH effective and efficient in full model learning with n-ary relations? and **(Q2)** Is faithful modeling of n-ary relations crucial? MACH is implemented as a pluggable module using Java-based *HypergraphDB* architecture [6]. We integrated it into the state-of-the-art Boosted MLN framework [7] for full model (structure+parameters) learning. MACH is compared against the following baselines, (1) FACT [3] and (2) MLN-Boost (Boosted MLN learner w/o approximate counting).

We evaluate with 2 data sets, *UWCSE*, standard link—prediction data set over staff, faculty and students and *NELL-Sports*, a sports domain data set extracted from NELL [2]. Table 1 shows how MACH allows significant reduction in learning time with no deterioration in predictive performance (AUC-ROC & PR) answering **(Q1)** affirmatively. Note that, UWCSE contains ternary predicates which were not modeled correctly by *FACT* leading to significantly worse performance, corroborating the need for faithful representation of n-ary relations **(Q2)**.

		Performance		Efficiency
Data Sets	**Methods**	**ROC**	**PR**	**L-Time [s]**
UWCSE**	MACH	0.981	0.337	**13.2**
	FACT	0.500	0.0068	7.48
	MLN-Boost	0.998	0.361	27.5
NELL-Sports	MACH	0.78	0.65	253.92
	FACT	0.76	0.64	238.07
	MLN-Boost	0.78	0.66	396.24

Table 1. Results: Performance (AUC) vs. Efficiency (**L**earning time in seconds). ** denotes presence of n-ary predicates.

4 Conclusion

We present a generalized and robust count approximation approach *MACH* for scalable and efficient probabilistic relational learning. Theoretical guarantees on error as well as a unifying framework integrating domain reduction and approximation are some interesting future research directions.

References

[1] C. Berge and E. Minieka. 1973. *Graphs and hypergraphs.* North-Holland publishing company Amsterdam.

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

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

[4] P. Domingos and D. Lowd. 2009. *Markov Logic: An Interface Layer for AI.* Morgan & Claypool.

[5] F. Feng, X. He, Y. Liu, L. Nie, and T.S. Chua. 2018. Learning on Partial-Order Hypergraphs. In *WWW.*

[6] B. Iordanov, K. Vandev, C. Costa, M. Marinov, M. Saraiva de Queiroz, I. Holsman, A. Picard, and I. Bogdahn. 2010. HyperGraphDB 1.3. (2010).

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

[8] D. Poole. 2003. First-Order Probabilistic Inference. In *IJCAI.*

[9] S. Sarkhel, D. Venugopal, T. Pham, P. Singla, and V. Gogate. 2016. Scalable Training of Markov Logic Networks Using Approximate Counting. In *AAAI.*