Graph-based Approximate Counting for Relational Probabilistic Models

Mayukh Das
School of Informatics & Computing
Indiana University
Bloomington, IN

Yuqing Wu
School of Informatics & Computing
Indiana University
Bloomington, IN

Tushar Khot
Allen Institute of AI
Seattle, WA

Kristian Kersting
Dept. of Knowledge Discovery
Technical University of Dortmund
Germany

Sriraam Natarajan
School of Informatics & Computing
Indiana University
Bloomington, IN

Abstract

One of the key operations inside most relational probabilistic models is counting - be it for parameter/structure learning or for efficient inference. However, most approaches use the logical structure for counting and do not exploit any fast counting methods. In this work-in-progress, we explore the closer connections to graph data bases and propose methods that obtain both exact and approximate counts effectively. We demonstrate the efficiency in the task of parameter learning.

Introduction

Statistical Relational AI (Getoor and Taskar 2007) deals with the problems of learning and inference in the presence of rich, structured multi-relational data. A key operation required by most if not all StaRAI models is *counting*. For instance, Markov Logic networks (MLNs) (Domingos and Lowd 2009) use counting as their fundamental operation in computing posterior probabilities. Similarly, most directed models that employ a combination function such as mean or weighted mean (Natarajan et al. 2009) use counts. Finally, most lifted inference methods (Singla and Domingos 2008; Van den Broeck et al. 2011; Poole 2003; Kersting, Ahmadi, and Natarajan 2009; Ahmadi, Kersting, and Hadiji 2010; de Salvo Braz et al. 2009; Milch et al. 2008) that aim to reason at a high-level as much as possible by exploiting symmetries also consider counts of the objects in each group to compute the final posteriors.

However, since the counts are primarily used in exponents of different functions (be it the log-linear function of MLNs or the products in lifted inference methods), computing exact counts appears to be an overkill. This is particularly true because most systems are logic-based and counting is one of the most expensive operations, more so, since counting all possible words is a major bottleneck. In many cases, the exact count of the groundings is not necessary. For instance, intuitively, it should not matter (to an inference algorithm) whether a particular Professor has co-authored approximately 500 publications or exactly 519 publications when answering a query about the success of this Professor since the number is high anyway.

Our hypothesis (which we verify empirically) is that performing approximate counting can allow for high efficiency gains with a small loss of performance. To this effect, we exploit the progress in the graph databases and graph theory (Metzler and Miettinen 2015; Castellana et al. 2015; Lichtenwalter and Chawla 2012; Pan 2009; Zou et al. 2011; Sun et al. 2012) to perform fast, approximate counting over query structures that can be performed in relatively short time. More specifically, we compile our logical model to a graph/network represented in *resource description framework* (RDF) format. This equivalent model allows for both approximate and exact counts to be performed in a fraction of time that is required by the original logical model. We first show how the logical model can be converted to an equivalent graph representation. Then, we present the exact computation algorithm that simply counts sub-graphs via queries. We then outline an approximation method (similar to messaging passing methods for probabilistic graphical models) that uses summary statistics (expected values) based on in-degrees and out-degrees to estimate the counts. Our approach, conceptually, bears resemblance to the work by (Venugopal, Sarkhel, and Gogate 2015), which achieves efficiency via counting only the satisfied groundings (similar to counting paths in an equivalent graph); however, the difference lies in the ability of our method to count non-existent paths (unsatisfied groundings) as well. Finally, we demonstrate the effectiveness and efficacy of the proposed counting approach on a learning method employed in several standard benchmark data sets.

To summarize, we make the following key contributions: (1) We first present a compilation strategy based on graph data bases for relational probabilistic models. (2) In addition to compilation, we show how to perform counting (from exact to approximate) on such graph structured data. (3) Finally, we perform an initial evaluation on several standard data sets in the task of learning parameters (that involve parameter tying across instances). Extending this evaluation rigorously and showing the potential for speeding up standard lifted inference techniques remains an interesting direction of future research.

Background & Related Work

Approximation of counts via summaries is closely related to cardinality/selectivity estimation and has been deeply studied in relational databases (Schiefer, Strain, and Yan 1998) using histogram based summaries (Matias, Vitter, and Wang 1998; Seputis 2000) or VC dimension based meth-

ods (Riondato et al. 2011). It has also been studied in the context of Graph databases (Neumann and Moerkotte 2011; Stocker et al. 2008). Some key definitions:

Property Graph (Model) (Corby, Dieng, and Hébert 2000) is a model of representing graph structured data effectively and efficiently, where every edge $\mathcal{E} = \langle v_1, v_2 \rangle$ is denoted as a triple $\langle subject, predicate, object \rangle$, such that, $subject = v_1, object = v_2$ and $predicate = label(\mathcal{E})$. Conceptually, predicate is *property* of the subject, and object is its value.

Resource Description Framework (RDF) (Pan 2009; Corby, Dieng, and Hébert 2000) is a language/framework to represent a property graph where everything – subject, predicate or object is a resource, with a namespace binding. The namespace has to be valid URI. Example: ⟨*http://example.org/Anna*⟩ which could be a subject or and object or ⟨*http://example.org/Friends*⟩ which is a predicate.

SPARQL (PrudHommeaux, Seaborne, and others 2008) is a query language for querying on RDF data. It is different from SQL, in that the subgraph we wish to query is encoded in the WHERE part of the query as connected triples.

*SELECT * FROM* G_f *WHERE* {*?a Friends ?b. ?b Hates ?c*}

Graph-Based Approximate Counting

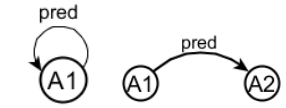
We now present how statements/formulas in first order predicate logic (FOL) can be represented, both intuitively and easily, as a "property graph/network". The underlying observation is that satisfiability of formulas in FOL can be posed as determination of the existence of a particular path in the graph. We first present the equivalent representation before outlining the counting algorithms.

Equivalent Representation

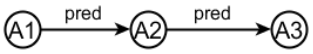
Predicates (either partially/fully grounded or ungrounded) are used to construct a property graph ($\mathcal{G}$). The arguments of the predicate become the nodes of G and the predicate itself becomes a directed edge in G with the name of the predicate as its label. The direction of the edge is determined by the order of the arguments of the predicate. This is motivated by the fact that predicates express relation between two or more entities or at times a property of an entity, viewed as a reflexive relation of an entity with itself.

Consider, any predicate $pred(A_1, A_2)$. The arguments $A_1 \& A_2$ are added to the set of nodes, $\mathcal{N}$ in $\mathcal{G}$. $\mathcal{E} = A_1 \rightarrow A_2$ is a directed edge from A_1 to A_2 in the set of edges, E and $label(\mathcal{E}) = pred$. Thus, $\mathcal{G} = (\mathcal{N}, E)$. We now present different cases of $pred$.

- *Unary Predicates:* For unary predicates ($pred(A_i)$), the directed edge will be a 'self-loop', i.e., $\mathcal{E} = A_i \rightarrow A_i \in E$. This is illustrated in Figure 1(a).
- *Binary Predicates:* This is the straightforward case. For $pred(A_1, A_2)$, the two nodes A_1 and A_2 have a directed edge $\mathcal{E} = A_1 \rightarrow A_2$. See Figure 1(b).
- *N-ary Predicates:* The more general case can be handled by converting the N-ary to $N - 1$ binary predicates as is done with most formalisms. Thus, for predicate $pred(A_1, A_2, A_3)$, we construct two edges $\mathcal{E}_1 = A_1 \rightarrow A_2$ and $\mathcal{E}_2 = A_2 \rightarrow A_3$ (Figure 1(c)), both the edges having the same label. Note that this can introduce some spurious relations (Kersting and De Raedt 2007) but with large amounts of data, this serves as a reasonable approximation (as we show later).



(a) Unary (b) Binary



(c) N-ary

Figure 1: Handling Arity

To summarize, given a set of facts (evidence) $F_{ev} = \{pred_i(A_{i1}, A_{i2}, \ldots, A_{ij})\}_{i=1}^{K}$ where K is the size of the evidence (number of given facts) and $j \geq 1$ (value of j is the Arity of predicate $pred_i$), we construct a corresponding evidence graph G_{ev} of size $O(K * |A|)$. We use the standard *"Smokes-Friends-Cancer"* problem (Domingos and Lowd 2009) and demonstrate the graph construction in Figure 2. As can be observed, $Smokes(Anna)$ and $Cancer(Gary)$ are both unary predicates and hence are self-loops while others are directed edges. As another example, for the facts

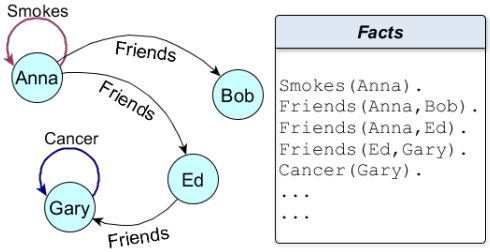


Figure 2: Smokes Friends Graph

presented below, the equivalent graph is shown in Figure 3.

author("class_7", "author_blum_a_").
author("class_8", "author_blum_a_").
author("class_9", "author_blum_a_").
title("class_7", "TITLE_A").
title("class_8", "TITLE_A").
title("class_9", "TITLE_B").
venue("class_7", "venue1").
venue("class_8", "venue2").
venue("class_9", "venue3").
haswordauthor("author_blum_a_","word_a").
haswordauthor("author_blum_a_","word_blum").

Algorithm 1 presents the creation of the evidence graph and Algorithm 2 presents the summarization of the evidence graph. The key idea is to use these summary statistics (in-degrees and out-degrees) to estimate the counts of the queries as demonstrated later.

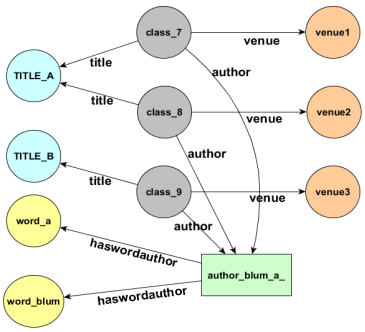


Figure 3: Equivalent Property Graph

Algorithm 1: *createEvidenceGraph()*

Data: Evidence File $\mathcal{F}$
Result: Evidence Graph $\mathcal{G}_{ev}$
Initialization: Empty Evidence Graph $\mathcal{G}_{ev} = \{\}$
  Empty Triple Store τ
for *each grounded predicate* $\mathcal{P} = p(A_1, A_2, \ldots)$ *in* $\mathcal{F}$ **do**
  Parse $\mathcal{P}$
  Edge $\mathcal{E} \leftarrow p$
  if $\mathcal{P}$ *is unary* **then**
    Subject Node $\mathcal{N}_s \leftarrow A_1$
    Object Node $\mathcal{N}_o \leftarrow \mathcal{N}_s$ [Self loop]
  else
    if $\mathcal{P}$ *is binary* **then**
      Subject Node $\mathcal{N}_s \leftarrow A_1$
      Object Node $\mathcal{N}_o \leftarrow A_2$
    else
      Split n-ary into n binary predicates and process
    end
  end
  New Triple $\mathfrak{T} \leftarrow \langle \mathcal{N}_s, \mathcal{E}, \mathcal{N}_o \rangle$
  Add $\mathfrak{T}$ to $\mathcal{G}_{ev}$ and commit $\mathcal{G}_{ev}$ to τ
end
return $\mathcal{G}_{ev}$

Obtaining Counts

We now explain how to perform counting given the summary statistics of the graph. We observe that this is equivalent to counting subgraphs in a heterogeneous network while satisfying certain constraints. To understand this better, consider the Smokes-Friends-Cancer example Figure 2. For simplicity, let us assume the following clause:

$$Smokes(a1) \wedge Friends(a1, a2)$$

Now to calculate the number of satisfiable groundings of this clause, we count the subgraphs(motifs) that have the structure presented in Figure 4(a). Hence, the goal is to count the number of subgraphs/motifs in the Smokes-Friends-Cancer network given in figure 2. The given structure becomes the constraint on the counting task, i.e., the goal is to count subgraphs that satisfy this constraint. Figure 4(b) presents an example of this structure given the two groundings of the

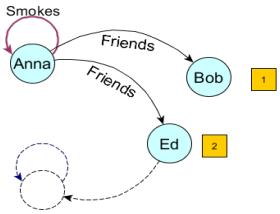
Algorithm 2: *summarize()*

Data: Evidence Graph $\mathcal{G}_{ev}$
Result: Summary statistics in a set of Hash data structures $\{H_{in}, H_{out}\}$
Initialize $\{H_{in}, H_{out}\}$ as empty structures;
/* ---- Calculate and store in-degree summary of every Node ----- */
Query $\leftarrow$ *"SELECT* $\{$*count(?s) as ?cnt*$\}$ *?p ?o from* $\mathcal{G}_{ev}$ *WHERE* $\{$*?s ?p ?o*$\}$ *GROUP BY ?p ?o"*;
ResultSet $R_s \leftarrow execute(Query, G_{ev})$;
for *each* $\langle cnt, p, o \rangle \in R_s$ **do**
  $put(H_{in}, \langle o, \langle p, cnt \rangle \rangle)$;
end
/* --- Calculate and store out-degree summary of every Node --- */
Query $\leftarrow$ *"SELECT ?s ?p* $\{$*count(?o) as ?cnt*$\}$ *from* $\mathcal{G}_{ev}$ *WHERE* $\{$*?s ?p ?o*$\}$ *GROUP BY ?s ?p"*;
ResultSet $R_s \leftarrow execute(Query, G_{ev})$;
for *each* $\langle s, p, cnt \rangle \in R_s$ **do**
  $put(H_{out}, \langle s, \langle p, cnt \rangle \rangle)$;
end
return H_{in}, H_{out}

above clause (with, Bob and Ed being friends of Anna).

$$1: \quad Smokes(Anna) \wedge Friends(Anna, Bob)$$
$$2: \quad Smokes(Anna) \wedge Friends(Anna, Ed)$$

(a) First Order Clause



(b) SubGraphs satifying the clause

Figure 4: FO-Clause and equivalent subgraph counting

Exact Counting Given that we have mapped the counting of satisfied groundings to subgraph counting, exact counting can be performed in a relatively straightforward manner. To this effect, we represent the property graph as an RDF (mentioned earlier). Now, exact counting requires simply retrieving all the subgraphs matching the motif or the pattern induced by a clause as shown above in Figure 4 and then enumerating and counting them thereafter. This can be achieved by a straightforward SPARQL query (an SQL-like query language designed for querying on Graph data represented via RDF) as shown below.

For the clause $Smokes(a1) \wedge Friends(a1, a2)$, the query to return all the subgraphs is as follows:

```
PREFIX namespace:<http://example.org/>
SELECT ?a1 ?a2 FROM Evidence_Graph
WHERE {?a1 namespace:Smokes ?a1.
       ?a1 namespace:Friends ?a2}
```

This query will return all the subgraphs present in the evidence graph G_{ev} which can then be counted. In the SPARQL query, the variables are denoted with a ? before its name. All the constraints (here parameterised first order predicates) are encoded in the query in the "WHERE" part of the query. Once all possible subgraphs in the evidence graph are returned into a result set $R_s = \{g_s^{(1)}, g_s^{(2)}, \ldots\}$, the size of result set $n(R_s)$ is the count value.

While this is straightforward, we have just essentially converted one exponential problem to another one, since sub-graph matching is already a hard problem, which might be relatively faster to compute due to progress database technology, especially graph databases. Hence, we now present an approximation method that can be more tractable for queries that employ large evidence.

Approximate counting Our key intuition remains the same – a clause is equivalent to a pattern and our goal is to search for that pattern in the evidence graph G_{ev} as shown in Figure 4. The key strategy to obtain this count approximately is to use the summary statistics collected via the *summarize()* procedure in Algorithm 2.

Inspired by the success of message passing algorithms in probabilistic graphical models, we develop an algorithm that uses augmented count values from summary statistics as messages. To understand this, consider the clause:

$$pr1(a_1, a_2) \wedge pr2(a_3, a_2) \wedge pr3(a_2, a_4) \Rightarrow h(a_2) \quad (1)$$

The equivalent graph for the body of the clause is shown in Figure 5

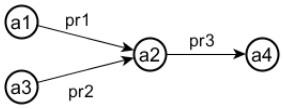
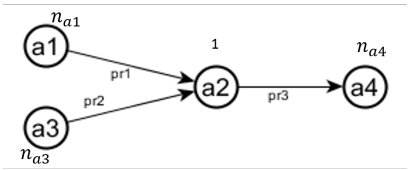


Figure 5: Graph for body of clause in eqn 1

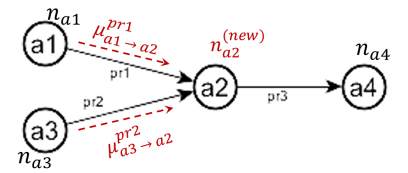
We define a few notations first,

- $In_{pr}(c)$ is the in-degree of a node with constant c present in G_{ev} with respect to edges with predicate pr. (Eg: In Figure 2, $In_{Friends}(Ed) = 1$).
- $Out_{pr}(c)$ is the out-degree of a node with constant c present in G_{ev} with respect to edges with predicate pr. (Eg: In Figure 2, $Out_{Friends}(Anna) = 2$).
- $In_{pr}^{(avg)}$ is the average in-degree of all nodes that have incoming edge $\mathcal{E}$ with $label(\mathcal{E}) = pr$. Hence $In_{pr}^{(avg)} = \frac{\sum_i^N In_{pr}(v_i)}{N}$.
- $Out_{pr}^{(avg)}$ is the average out-degree of all nodes that have incoming edge $\mathcal{E}$ with $label(\mathcal{E}) = pr$. Hence $Out_{pr}^{(avg)} = \frac{\sum_i^N Out_{pr}(v_i)}{N}$.
- $\Theta(v)$ or type Count is the number of constants possible (given by the evidence) for variable v. So for parameterized predicate $pr(v)$, if the structure of the predicate is $pr(type_A)$, i.e., the argument of the predicate is of type $type_A$ then $\Theta(v) = n(type_A)$.
- $\mu_{v_i \to v_j}^{pr}$ is the message transmitted from variable v_i to variable v_j over edge pr.
- $\mathcal{G}_q$ is the query graph, or the graph formed by the formula/clause for which we are counting.

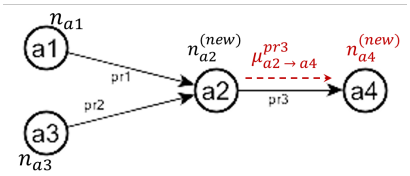
Next we will first describe the process informally, and present the algorithm.



(a) Initialization



(b) Next step



(c) Final step

Figure 6: Approx Counting

We start with the assumption that at least one of the variables is tied to a constant, that is, the counts are obtained against one example (in our case let us assume that the variable $a2$ is tied to an appropriate constant). With this assumption, we first initialize the counts of each variable, in the graph formed by the body of the clause that we have in Figure 5. As the variable $a2$ is tied to a constant its count is assumed to be 1 and all other variables are initialized to their type counts, $\Theta(v)$. This is illustrated in figure 6(a). At the initial state, $count(a1) = n_{a1} = \Theta(a1)$, $count(a2) = 1$ [$\because$ $a2$ *is a constant*], $count(a3) = n_{a3} = \Theta(a3)$ and $count(a4) = n_{a4} = \Theta(a4)$ (Constant tying is not essential, and has been done to demonstrate how the counts vary with examples. However, if not required, then all variables are simply initialized to their "typeCounts").

Given that the graph is initialized, we will demonstrate how the message passing occurs here. One important factor here is that, the graph is directional hence the order of the variables (for variable elimination) is a trivial choice. The variables (nodes) that have no incoming edges in the query graph $\mathcal{G}_q$ are the starting points and the rest of the order is immaterial. Thus, in our example we start with $a1$ and $a3$ and messages are passed from these nodes to the node $a3$ [$\mu^{pr1}_{a1\to a2}$ from $a1$ to $a2$ and $\mu^{pr2}_{a3\to a2}$ from $a3$ to $a2$] as displayed in Figure 6(b). The messages are augmented counts:

$$\mu^{pr1}_{a1\to a2} = \frac{Out^{(avg)}_{pr1}}{\Theta(a2)}.\frac{In_{pr1}(C)}{\Theta(a1)}.n_{a1} \quad (2)$$

The expression $\frac{Out^{(avg)}_{pr1}}{\Theta(a2)}$ gives us the ratio of the average outgoing edge to the maximum number of possible outgoing edges, with predicate $pr1$ in this case. $\frac{In_{pr1}(C)}{\Theta(a1)}$ is a similar expression for the case of incoming. Their product gives us a heuristic that is representative of our belief about the presence of this predicate in the G_{ev}, which we then use to augment the count. Again the message $\mu^{pr2}_{a3\to a2}$ is obtained as,

$$\mu^{pr2}_{a3\to a2} = \frac{Out^{(avg)}_{pr2}}{\Theta(a2)}.\frac{In_{pr2}(C)}{\Theta(a1)}.n_{a3} \quad (3)$$

Now, the count value of $a2$ is updated. Note how the mean/average of in-degree is not considered here, since the variable $a2$ is tied to an example (constant C).

$$n^{(new)}_{a2} = \prod_{(v,pr)\in\{(a1,pr1),(a3,pr2)\}} \mu^{pr}_{v\to a2}.(n^{(old)}_{a2} = 1) \quad (4)$$

Finally another message $\mu^{pr3}_{a2\to a4}$ is passed from $a2$ to $a4$ as shown in figure 6(c). The message is as shown below:

$$\mu^{pr3}_{a2\to a4} = \frac{Out_{pr3}(C)}{\Theta(a4)}.\frac{In^{(avg)}_{pr3}}{\Theta(a2)}.n^{(new)}_{a2} \quad (5)$$

Updating the count of $a4$ works in a similar fashion as shown in Equation 4. Due to reasons mentioned earlier, mean/average out-degree is not considered (in Eqn 5), instead the out-degree of the exact constant node is used.

Given $n^{(new)}_{a4}$ is the final count we want since that is the only variable left after eliminating the rest, which is an approximate estimate of the number of subgraphs present in G_{ev}. Formal algorithm is given in Algorithm 3[1].

Pre-process: Call methods $createEvidenceGraph(\mathcal{F})$ (*Algorithm 1*) to get the evidence graph G_{ev} and call $summarize(G_{ev})$ (*Algorithm 2*) to get the summary data structures $H = \{H_{in}, H_{out}, H^{(avg)}_{in}, H^{(avg)}_{out}\}$ at the beginning of any inference or learning system.

Discussion There are a few important things to note. Firstly, with *Closed World* assumption, absence of an edge denotes, the negation of a predicate. Otherwise, negated form of the predicate becomes the label of an edge in the evidence graph. Secondly, for a self loop, degree summary is either 1 or 0, while N-ary predicates that have been converted to multiple binary ones behave as ordinary directed edges in the graph. Finally, and most importantly, to prove that the values returned by our message-passing/variable-elimination based system can be considered as counts, we can argue as follows: (1) If we multiply the initial count values of the variables in a query graph, such as in Figure 6(a), we will get the size of the full cross-product. (2) Instead, the messages passed are these counts, "augmented" by the belief about the presence of particular predicate/edge, based on G_{ev}, hopefully moving the value away from the cross-product and bringing it closer to the true count. (3) Though, counts returned by our method, are usually slight over-estimations, however it works well as demonstrated in the experiments.

[1]Note: In Algorithm 3 *parse()* is just a name given (for the ease of representation) to the operation of parsing an FOL clause into a query graph as shown in Equation 1 and Figure 5

Algorithm 3: *approxCount()*

Data: Clause $\mathfrak{C}$,H, example constant C for one of the variables
Result: Approximate count cnt

Initialize: Build $G_q \leftarrow parse(\mathfrak{C})$;
  Start with nodes with no incoming edge in G_q;
$V \leftarrow \{v : v \in G_q\}$;
for *each variable* $v \in G_q$ **do**
  for *each* $x \in G_q$, *s.t.* $\exists pr(x, v)$ **do**
    if $v = C$ *(constant)* **then**
      $\mu^{pr}_{x\to v} \leftarrow \frac{Out^{(avg)}_{pr}}{\Theta(v)}.\frac{In_{pr}(C)}{\Theta(x)}.n_x$;
    end
    else if $x = C$ *(constant)* **then**
      $\mu^{pr}_{x\to v} \leftarrow \frac{Out_{pr}(C)}{\Theta(v)}.\frac{In^{(avg)}_{pr}}{\Theta(x)}.n_x$;
    end
    else
      $\mu^{pr}_{x\to v} \leftarrow \frac{Out^{(avg)}_{pr}}{\Theta(v)}.\frac{In^{(avg)}_{pr}}{\Theta(x)}.n_x$;
    end
  end
  $n^{(new)}_v \leftarrow \prod_{\{(x_i,pr(x_i,v))\}_i} \mu^{pr}_{x\to v}.n_v$;
  if $sizeOf(V) = 1$ **then**
    $cnt \leftarrow n_v$;
    break;
  end
  $V \leftarrow V - \{v\}$;
end
return cnt

Implementation & Experiments

We employ a powerful graph representation language, RDF (Corby, Dieng, and Hébert 2000; Pan 2009) (Resource Description Framework), which is being studied and improved continuously by the Graph Database community, enabling us to, effortlessly, handle multi-relational graphs. In addition, we employ the SPARQL query language for querying on Graph structured data represented via RDF. Since off-the-shelf Graph Database systems have their internal optimizers, it is hard to benchmark the effectiveness of the proposed ap-

Table 1: Results of approximate counting on Combining Rules

			w/ Approx Counting	Original	Variances in approx. counting time
yeast	819 Facts	Counting time (secs)	2.804624383	7.896000303	0.01458
		MSE	0.257421138	0.236180698	
		CLL	-0.716749575	-0.665557672	
IMDb	264 facts	Counting time (secs)	0.186358918	0.357694597	1.00E-04
		MSE	0.168238687	0.06813761	
		CLL	-0.584496553	-0.230262177	
Cora	1498 Facts	Counting Time (secs)	0.442818746	1.283442534	5.62708E-05
		MSE	0.22335702	0.217540997	
		CLL	-0.639157011	-0.626396615	
WebKB	12284 Facts	Counting Time (secs)	4.844581832	8.410465749	6.67E-11
		MSE	0.291384968	0.291211276	
		CLL	-0.638317395	-0.587	

proach by simply employing them. Hence, we implemented our approach in Java using the Apache Jena library, which supports both RDF-based graph representation as well as usage of SPARQL.

Note that, our system is not specific to any particular inference or learning algorithm, and thus, it must seamlessly integrate with existing learning/inference systems. Finally, our system requires that the input is in predicate logic format.

Experiments

We consider the problem of parameter learning with mean and weighted-mean combining rule (Natarajan et al. 2009) using the EM-algorithm. Extending this analysis to existing lifted inference methods is our current focus and our active research direction. In the original EM-algorithm, we replaced the step where the counting of satisfied instances of the rule was performed using counting inside a logical formalism with a function call to our approximation procedure. The counting times and other metrics have been compared between the original code and the modified code that uses approximate counting.

Original System: The counting mechanism of the prior work (Natarajan et al. 2009) simply counts the exact number of literals that satisfy the body of a given clause for every example. Since there could be multiple clauses with the same head, this procedure is repeated for all the clauses, over all the examples. We refer to this as *CombRulesOriginal*.

Modification: We replaced the original counting method with the approximate counting method presented above. We repeat the approximate counting for every example and for every rule. We refer to this as *CombRulesApprox*.

Datasets: We have used 4 standard SRL datasets:

1. *Yeast*: This data set is about interaction among proteins, protein complexes and enzymes in yeasts. This dataset is essentially used in prediction problems where the target is the class of a protein. This is a mid-size data set with 819 facts.
2. *IMDB*: We used a smaller version of this with 264 facts, mainly about actors, directors and movies.
3. *Cora*: This is again a medium size data set of about 1500 facts. Target predicate is the $samevenue(venue, venue)$.
4. *WebKB*: This is comparatively large data set, 12284 facts. Target is the $departmentOf()$ predicate.

For both the approaches, we measure the following - (1) *Counting time*: Time taken for counting groundings that satisfy a clause for every example and every sense combination and finally storing the counts. (2) *MSE*: Mean Squared Error. (3) *CLL*: Conditional Log Likelihood. The last two metrics are essential to prove the effectiveness of our system. Our hypothesis is that our approximate counting method sacrifices MSE minimally while drastically reducing the counting time.

Table 1 presents the results of our experiments. It can be observed that there is at least a 2-3 times reduction in counting time over all the data sets when using our approximate counting approach. The MSE $mse(CombRulesApprox)$ is comparable to the original MSE $mse(CombRulesOriginal)$, i.e., MSE does not suffer much because of approximation. Similarly, except for the IMDB domain, the CLL values of the two algorithms do not differ significantly.

For the *IMDB* dataset, the performance seems to be much worse than the original approach. On deeper analysis, it can be observed that the approximate counting mechanism is based on average values of in-degree and out-degree summaries and then carefully augmenting the summary values of one variable with that of another, based on their interaction. With a large dataset, our sample mean would approach the true distribution. Hence, in our experiments, with reasonable sized datasets approximations based on average degree estimates works well. But, the IMDB dataset is very small (264 facts). This will generate an extremely sparse evidence graph, which will not reflect the true distribution of the edge degrees. Hence, we hypothesize that our method will be more effective for medium to large data sets and not necessarily for smaller ones.

Conclusion

We presented our work-in-progress on approximating counting of satisfied instances - a crucial operation in several relational probabilistic models. Our proposed approach converted the predicate logic format to an equivalent graph database format that can be queried efficiently. We also developed an approximate counting method that is inspired by probabilistic message passing. Our initial results demon-

strated that this technique is effective in learning from moderate to reasonably large data sets.

More rigorous evaluation of the proposed approach is warranted. Particularly, we will focus on scaling these to very large ($>$ 1M) data sets. Comparisons with other database systems such as Deepdive (Wu et al. 2015) is essential and a comparison to theta-subsumption (Maloberti and Sebag 2004) could be interesting. Finally, integrating our approach with powerful lifted inference techniques remains an interesting and fruitful research direction.

Acknowledgments

SN gratefully acknowledges the support of the DARPA DEFT Program under the Air Force Research Laboratory (AFRL) prime contract no. FA8750-13-2-0039. KK acknowledges the German-Israeli Foundation (GIF) for Scientific Research and Development project 1180-218.6/2011. 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, AFRL, GIF or the US government.

References

Ahmadi, B.; Kersting, K.; and Hadiji, F. 2010. Lifted belief propagation: Pairwise marginals and beyond. In P. Myllymaeki, T. Roos, T. J., ed., *Proceedings of the 5th European Workshop on Probabilistic Graphical Models (PGM–10)*.

Castellana, V. G.; Morari, A.; Weaver, J.; Tumeo, A.; Haglin, D.; Villa, O.; and Feo, J. 2015. In-memory graph databases for web-scale data. *Computer* (3):24–35.

Corby, O.; Dieng, R.; and Hébert, C. 2000. A conceptual graph model for w3c resource description framework. In *Conceptual Structures: Logical, Linguistic, and Computational Issues*. Springer. 468–482.

de Salvo Braz, R.; Natarajan, S.; Bui, H.; Shavlik, J.; and Russell, S. 2009. Anytime lifted belief propagation. In *Statistical Relational Learning Workshop*.

Domingos, P., and Lowd, D. 2009. *Markov Logic: An Interface Layer for AI*. San Rafael, CA: Morgan & Claypool.

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

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

Kersting, K., and De Raedt, L. 2007. Bayesian logic programming: Theory and tool. In *An Introduction to Statistical Relational Learning*.

Lichtenwalter, R. N., and Chawla, N. V. 2012. Vertex collocation profiles: subgraph counting for link analysis and prediction. In *Proceedings of the 21st international conference on World Wide Web*, 1019–1028. ACM.

Maloberti, J., and Sebag, M. 2004. Fast theta-subsumption with constraint satisfaction algorithms. *Machine Learning* 55(2):137–174.

Matias, Y.; Vitter, J. S.; and Wang, M. 1998. Wavelet-based histograms for selectivity estimation. In *ACM SIGMoD Record*, volume 27, 448–459. ACM.

Metzler, S., and Miettinen, P. 2015. Join size estimation on boolean tensors of rdf data. In *Proceedings of the 24th International Conference on World Wide Web Companion*, 77–78. International World Wide Web Conferences Steering Committee.

Milch, B.; Zettlemoyer, L.; Kersting, K.; Haimes, M.; and Pack Kaelbling, L. 2008. Lifted Probabilistic Inference with Counting Formulas. In *AAAI*.

Natarajan, S.; Tadepalli, P.; Dietterich, T. G.; and Fern, A. 2009. Learning first-order probabilistic models with combining rules. *AMAI*.

Neumann, T., and Moerkotte, G. 2011. Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, 984–994. IEEE.

Pan, J. Z. 2009. Resource description framework. In *Handbook on Ontologies*. Springer. 71–90.

Poole, D. 2003. First-Order Probabilistic Inference. In *IJCAI*, 985–991.

PrudHommeaux, E.; Seaborne, A.; et al. 2008. Sparql query language for rdf. *W3C recommendation* 15.

Riondato, M.; Akdere, M.; Çetintemel, U.; Zdonik, S. B.; and Upfal, E. 2011. The vc-dimension of sql queries and selectivity estimation through sampling. In *Machine Learning and Knowledge Discovery in Databases*. Springer. 661–676.

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.

Singla, P., and Domingos, P. 2008. Lifted first-order belief propagation. In *AAAI*, 1094–1099.

Stocker, M.; Seaborne, A.; Bernstein, A.; Kiefer, C.; and Reynolds, D. 2008. Sparql basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th international conference on World Wide Web*, 595–604. ACM.

Sun, Z.; Wang, H.; Wang, H.; Shao, B.; and Li, J. 2012. Efficient subgraph matching on billion node graphs. *Proceedings of the VLDB Endowment* 5(9):788–799.

Van den Broeck, G.; Taghipour, N.; Meert, W.; Davis, J.; and De Raedt, L. 2011. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI*.

Venugopal, D.; Sarkhel, S.; and Gogate, V. 2015. Just count the satisfied groundings: Scalable local-search and sampling based inference in mlns. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Wu, S.; Zhang, C.; Wang, F.; and Ré, C. 2015. Incremental knowledge base construction using deepdive. *arXiv preprint arXiv:1502.00731*.

Zou, L.; Mo, J.; Chen, L.; Özsu, M. T.; and Zhao, D. 2011. gstore: answering sparql queries via subgraph matching. *Proceedings of the VLDB Endowment* 4(8):482–493.