Efficient In-Database Analytics with Graphical Models

Daisy Zhe Wang, Yang Chen, Christan Grant and Kun Li
{daisyw,yang,cgrant,kli}@cise.ufl.edu
University of Florida, Department of Computer and Information Science and Engineering

Abstract

Due to recent application push, there is high demand in industry to extend database systems to perform efficient and scalable in-database analytics based on probabilistic graphical models (PGMs). We discuss issues in supporting in-database PGM methods and present techniques to achieve a deep integration of the PGM methods into the relational data model as well as the query processing and optimization engine. This is an active research area and the techniques discussed are being further developed and evaluated.

1 Introduction

Graphical models, also known as probabilistic graphical models (PGMs), are a class of expressive and widely adopted machine learning models that compactly represent the joint probability distribution over a large number of interdependent random variables. They are used extensively in large-scale data analytics, including information extraction (IE), knowledge base population (KBP), speech recognition and computer vision. In the past decade, many of these applications have witnessed dramatic increase in data volume. As a result, PGM-based data analytics need to scale to terabytes of data or billions of data items. While new data-parallel and graph-parallel processing platforms such as Hadoop, Spark and GraphLab [1, 36, 3] are widely used to support scalable PGM methods, such solutions are not effective for applications with data residing in databases or with need to support online PGM-based data analytics queries.

More specifically, large volumes of valuable data are likely to pour into database systems for many years to come due to the maturity of the commercial database systems with transaction support and ACID properties and due to the legal auditing and data privacy requirements in many organizations. Examples include electronic medical records (EMRs) in EPIC systems as well as financial transactions in banks and credit card companies. Unfortunately, lacking native in-database analytics support, these volumes of data have to be transferred in and out of the database for processing, which is of potentially huge overhead. It would be much more efficient to push analytic computation close to data.

Moreover, efficient support of *online queries* are required by many applications for interactive data analytics. Most parallel data processing frameworks only support off-line batch-oriented analytics, which is very in-efficient for online PGM-based analytics driven by ad-hoc queries. In these cases, PGM-based data analytics methods should be pushed into database systems as first-class citizens in the data model, query processing and query optimization. To achieve efficiency and scalability, a deep integration of graphical models and algorithms with a database system is essential.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

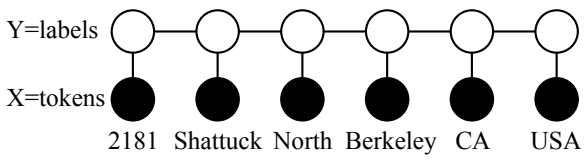
As a result, there is a need to extend database systems to perform large-scale in-database analytics based on probabilistic graphical models. Such an extension requires first-class modeling and implementation of probabilistic graphical models and algorithms to support query-time model-based inference and reasoning. New query optimization and view maintenance techniques are also needed to support queries with both relational and statistical analytics operations. The rest of the article describes our efforts to address three major challenges in supporting PGMs efficiently in a database system:

- *Model Representation and Model-Data Join*: First, we describe a relational representation of graphical models and model-data join algorithms that grounds a first-order PGM over a large number of data instances, resulting in a large propositional PGM [4].
- *Efficient In-Database Statistical Inference*: Second, we describe mechanisms that are required for efficient implementation of inference operations in a database system, including data-driven feature extraction and iterative inference computations over grounded PGMs [5].
- *Optimizing Queries with Inference*: Third, we describe new query optimization techniques to support online queries involving PGM-based analytics, which contain relational and inference operators over graphical models [6, 7].

Related Work Database systems and technologies lack the facilities to store probabilistic graphical models and perform statistical learning and inference tasks over PGMs. Three bodies of recent work have made progress in supporting efficient and scalable PGM methods in database for advanced data analytics. First, there are recent work on supporting in-database statistical analytics and machine learning methods such as linear algebra, matrix manipulation and convex optimization [8, 5, 9]. Second, graphical models and algorithms are used in probabilistic database literature to represent and manipulate uncertain data with high-dimensional joint distributions [10, 11, 12]. Third, recent work show that some algorithms, such as Viterbi, sum-product and Markov chain Monte Carlo (MCMC) sampling, commonly used for inference over graphical models can be efficiently supported and optimized in a relational query processing database system [13, 14, 15, 7].

2 Probabilistic Graphical Models

Probabilistic graphical models use a graph-based representation as the basis for compactly encoding a complex distribution over a high-dimensional space [16]. In this graphical representation, the nodes correspond to the variables in our domain, and the edges correspond to direct probabilistic interactions between them. As an example, Figure 1(a) shows a graphic model for text analysis.



(a)

X_1	Y_1	$\Phi(X_1, Y_1)$
2181	street number	22
2181	street name	5
2181	zip code	8
...	...	...

(b)

Figure 1: (a) An example linear-chain CRF model. (b) Factor $\Phi(X_1, Y_1)$ showing the probabilistic correlation between variables X_1 and Y_1.

The variables are an observed token sequence $\mathbf{X}$ and its label sequence $\mathbf{Y}$ to be predicted. The labels can be a street number, a city, a zip code, etc. The edges represent the probabilistic correlations among the variables, defined by *factors*. In this example, there are two types of edges. The vertical edges represent the factors between token and corresponding label. For instance, token "2181" is more likely to be a street number than a street name or a zip code. Hence, we have a factor in Figure 1(b) to reflect this correlation. As shown in the table, the values of $\Phi(X_1, Y_1)$ can be any real numbers that indicate the relevant likelihood of different assignments and do not

need to be probabilities. Likewise, the horizontal edges represent factors between adjacent labels. For instance, if the previous label is a street number, then a street name is likely to follow.

Together, these factors define a probability distribution $P(\mathbf{Y}|\mathbf{X})$ that can be used to make label predictions. This particular kind of linear graphical models for text analysis is studied extensively and is called linear-chain *conditional random field* (CRF) [17]. The task of computing or approximating the probability distribution $P(\mathbf{Y}|\mathbf{X})$ is called *inference*. Hence, inference is a key step toward predicting the labels $\mathbf{Y}$.

More recently, statistical relational learning (SRL) models combine PGMs with first-order logic to model the uncertainty and probabilistic correlations in relational domains. SRL models can be considered as first-order PGMs, which are *grounded* into propositional PGMs at inference time. As an example, Markov logic networks (MLNs) is a state-of-the-art SRL model that supports inference and querying over a probabilistic knowledge base (KB) defined by sets of entities, classes, relations, uncertain facts and uncertain first-order rules [4]. Table 1 shows an example probabilistic KB constructed from web extractions. The rule set $\mathcal{L}$ defines an MLN, allowing us to infer facts that are not explicitly stated in the original KB. These facts are uncertain, and their joint probability distribution is represented by a ground PGM (factor graph), as shown in Figure 2. The task of grounding and probabilistic inference is a key challenge we address in this paper. Based on the inference result, queries such as "Return all writers lived in Brooklyn" over this probabilistic KB would produce a list of person names ranked by the probabilities.

Entities $\mathcal{E}$	**Classes** $\mathcal{C}$	**Relations** $\mathcal{R}$	**Facts** Π
Ruth Gruber, New York City, Brooklyn	W (Writer) = {Ruth Gruber}, C (City) = {New York City}, P (Place) = {Brooklyn}	born in(W, P), born in(W, C), live in(W, P), live in(W, C), locate in(P, C)	0.96 born in(Ruth Gruber, New York City) 0.93 born in(Ruth Gruber, Brooklyn)

Rules $\mathcal{L}$
1.40 $\forall x \in W\ \forall y \in P$ (live in$(x, y) \leftarrow$ born in(x, y))
1.53 $\forall x \in W\ \forall y \in C$ (live in$(x, y) \leftarrow$ born in(x, y))
0.32 $\forall x \in P\ \forall y \in C\ \forall z \in W$ (locate in$(x, y) \leftarrow$ live in$(z, x) \wedge$ live in(z, y))
0.52 $\forall x \in P\ \forall y \in C\ \forall z \in W$ (locate in$(x, y) \leftarrow$ born in$(z, x) \wedge$ born in(z, y))
∞ $\forall x \in C\ \forall y \in C\ \forall z \in W$ (born in$(z, x) \wedge$ born in$(z, y) \rightarrow x = y$)

Table 1: A probabilistic graph data model representation of a sample probabilistic KB.

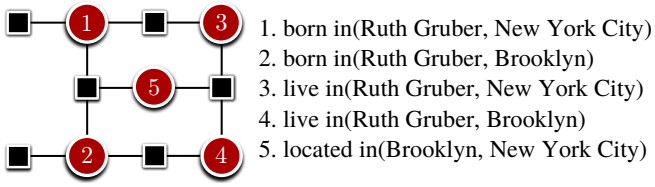


Figure 2: Factor graph representation of probabilistic KB in Table 1.

3 Model Representation and Model-Data Join

In a probabilistic knowledge base (KB), as the example in Table 1, queries need to be performed over the propositional form of the first-order PGM (MLN). The process of generating the propositional KB graph is called *grounding* in the SRL literature. The grounding process replaces variables in the first-order rules with constants, *materializing* it into a propositional PGM (factor graph), as shown in Figure 2, for further inference and querying.

The main challenge in KB graph materialization is efficiency and scalability due to the fast-growing sizes of current knowledge bases. For instance, the YAGO2s knowledge base [18] has 10 million entities and 120 million facts, and Freebase [19] has 45 million entities and 2.77 billion facts. To effectively manage these knowledge bases, recent works, PROBKB [4] and TUFFY [9], use bottom-up relational implementations to take the best advantage of query execution and optimization in a data-parallel processing system such as a parallel database. New techniques are also invented to further reduce the propositional KB graph based on deterministic semantic constraints [4].

3.1 Materialization via Model-Data Joins

As shown in [20], A naive materialization of the propositional KB graph enumerates all possible constants for every free variable in each first-order rule, which results in an exponential blow-up in the number of ground atoms and clauses. However, previous work [21, 22] have shown that this is both impractical and unnecessary in an extremely sparse relational domain, where only a very small fraction of the ground atoms are true and a vast majority of ground clauses are trivially satisfied given evidence and the closed world assumption (CWA). Based on these ideas, [9] implements a lazy closure grounding algorithm to generate a reduced grounded factor graph from an MLN program using a bottom-up grounding strategy.

While such a bottom-up approach greatly improves efficiency and scalability compared to top-down inference engines such as Alchemy [23], evaluation results show that the bottleneck lies in the number of first-order rules. This is because [9] represents each predicate as a table and translates each first-order rule into a recursive SQL query over the corresponding predicate tables. Given tens of thousands of rules over a KB, this implies tens of thousands of SQL queries in each iteration, incurring an unnecessarily heavy workload.

In our work on knowledge base expansion [4], we tackled this problem by exploiting the structure of first-order rules, partitioning them into tables, each representing a set of structurally equivalent rules. Instead of storing the MLNs in normal text files, the rules are stored as relational tables inside the database, and as a consequence, the grounding algorithm can be efficiently expressed as join queries among the facts and rules tables that apply inference rules *in batches*. Our experiments used the REVERB-SHERLOCK Wikipedia dataset [24, 25], which contains more than $400K$ facts, $270K$ entities and $30K$ rules. We categorized the first-order Horn clauses of lengths 2 and 3 and all functional dependency constraints into structurally equivalent rule tables [4]. During grounding, we ran one SQL query for each unique rule structure. The experiment results showed orders-of-magnitude speed-up compared to the state-of-the-art MLN inference engine. As a SQL-based algorithm, our approach allows easy integration of future improvements using query optimization and parallel databases like Greenplum.

3.2 Propositional KB Graph Reduction

Even with the lazy grounding techniques, the resulting propositional KB graph can still be prohibitively large, especially with large numbers of recursive first-order rules. However, due to the ambiguous entities and uncertain facts and rules, many of the inferred facts are erroneous. To make it worse, these erroneous facts propagate rapidly in the inference chain without proper constraints, as shown in Figure 3.

To improve the accuracy, we use probability thresholds to prune the rules with low credibility and a set of deterministic semantic constraints to detect erroneous facts and ambiguous entities. Before grounding starts, we eliminate rules with low probabilities from the rule tables. This avoids unsound rules applied repeatedly to multiple facts. Then, in each grounding iteration, we remove facts violating the deterministic semantic constraints to prevent them from further propagation.

Our experiments using REVERB-SHERLOCK and LEIBNIZ datasets [24, 25, 26] show that such probability and constraint based pruning produces a propositional KB graph with more than twice as many correct inferred facts. The accuracy of the inferred facts is improved to 60% from the 15% generated from the baseline. However,

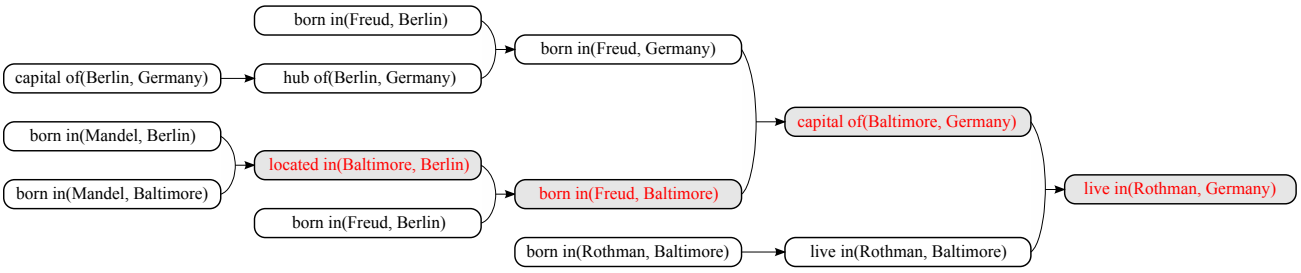


Figure 3: Error propagation: erroneous facts inferring other erroneous facts in the inference chain (shaded).

it is a nontrivial task to make further improvements since the extracted facts and rules are inherently noisy. Thus, instead of performing the pruning during inference, we propose, as a future work, to apply our constraining techniques to the rule learning phase to improve rule quality.

4 Efficient In-Database Statistical Inference

In many domains, structured data and unstructured text are both important assets for data analytics. For example, electronic medical record (EMR) systems use transactional databases to store both structured data such as lab results and monitor readings, and unstructured data such as notes from physicians. PGM-based methods such as linear-chain CRF, the example shown in Figure 1(a), are used for information extraction (IE) from text. With our goal to push PGM-based analytics close to the data, it is important to support basic statistical methods, such as feature extraction and inference for in-database text analytics tasks.

Basic text analytics tasks include part-of-speech (POS) tagging, named entity extraction (NER), and entity resolution (ER) [27]. Different statistical models and algorithms are implemented for each of these tasks with different runtime-accuracy tradeoffs. For example, an entity resolution task could be to find all mentions in a text corpus that refer to a real-world entity X. Such a task can be done efficiently by approximate string matching to find all mentions in text that approximately match the name of entity X. However, such a method is not as accurate as the state-of-the-art collective entity resolution algorithms based on statistical models, such as conditional random fields (CRFs).

4.1 Statistical Text Analytics in MADlib

Based on the MADlib [5] framework, we set out to implement statistical methods in SQL to support various text analytics tasks. We use CRFs as the basic statistical model to perform more advanced text analytics. Similar to Hidden Markov Models (HMM), linear-chain CRFs are a leading probabilistic model for solving many text analytics tasks, including POS and NER. To support sophisticated text analytics, we first implement key methods for text feature extraction including approximate string matching.

Text Feature Extraction: Text feature extraction is a first step in most statistical text analytics methods, and it can be an expensive operation. To achieve high quality, CRF models often assign hundreds of features to each token in the document. Examples of such features include: (1) dictionary features: *does this token exist in a provided dictionary?* (2) regex features: *does this token match a provided regular expression?* (3) edge features: *is the label of a token correlated with the label of a previous token?* (4) word features: *does this the token appear in the training data?* and (5) position features: *is this token the first or last in the token sequence?* The right combination of the features depends on the application. We implement a common set of text feature extractors with exactly one SQL query for the extraction of one type of features from text data [28]. For example, the following SQL query extract all the features that are matches of one of the regular expressions present in the `regextable` table:

```
SELECT start_pos, doc_id, 'R_' || r.name, ARRAY[-1,label]
FROM regextbl r, segmenttbl s
WHERE s.seg_text ~ r.pattern;
```

Approximate String Matching: A recurring primitive operation in text processing applications is the ability to match strings approximately. The technique we use is based on q-grams [29]. Specifically, Approximate string matching between two named entities text strings (e.g., George W. Bush vs. George Bush) is used as one of the features in a PGM such as conditional random field (CRF) to perform entity resolution (i.e., co-reference). We use the trigram module in PostgreSQL to create and index 3-grams over text [30]. Given a string "Tim Tebow" we create a 3-gram by using a sliding window of 3 characters over the text string. Using the 3-gram index, we create an approximate matching user-defined function (UDF) that takes a query string and returns all documents in the corpus that contain at least one approximate match.

4.2 In-Database Inference over PGMs

Once we have the features, the next step is to perform inference on the model. We implement two types of statistical inference within the database: the Viterbi algorithm to compute the MAP (maximum-a-priori) labels over a liear-chain PGM over sequence data, and Markov chain Monte Carlo (MCMC) to compute the marginal probability distribution over a general PGM.

Viterbi Inference: The Viterbi dynamic programming algorithm is a popular algorithm to find the top-k most likely labelings of a document for linear chain CRF models [27].

Like any dynamic programming algorithm, the Viterbi algorithm is recursive. We experiment with two different implementations. First, we implement it with a combination of recursive SQL and window aggregate functions. We discuss this implementation at length in an earlier work [15]. Our initial recursive SQL implementation only runs over PostgreSQL versions 8.4 and later; it does not run in Greenplum. Second, we implement a Python UDF that uses iterations to drive the recursion in the Viterbi algorithm. This iterative implementation runs on both PostgreSQL and Greenplum. In Greenplum, Viterbi runs in parallel over different subsets of the document on a multi-core machine. We also implement a CRF learning algorithm based on convex optimization in parallel using Python-UDF interface. Our evaluation uses CoNLL2000 dataset containing 8936 tagged sentences for learning, which achieves a 0.9715 accuracy consistent with the state-of-the-art POS models with 45 POS tags. To evaluate the inference performance, we extract 1.2 million sentences from the New York Times dataset [31]. Results show that the runtime is sublinear to and improves with an increase in the number of cores [28].

MCMC Inference: Markov chain Monte Carlo (MCMC) algorithms are classical sampling methods that are used to estimate probability distributions. We implement two MCMC methods in MADlib: Gibbs sampling and Metropolis-Hastings (MCMC-MH).

The MCMC algorithms involve iterative procedures where the current iteration depends on previous iterations. We use SQL window aggregates to carry "states" across iterations to perform the Markov-chain process. This window function based implementation runs on PostgreSQL version 8.4 and later. We discuss this implementation at length in our recent work [7]. We are currently working on integrating MCMC algorithms into Greenplum DBMS. We also plan to implement MCMC using Python UDFs and extend user-defined aggregates (UDAs) to support a new class of general iterative state transformation (GIST) operations and compare the performance between the two new in-database implementations of MCMC sampling based inference algorithms over PGMs.

In addition to the above techniques, there are a host of other features of both PostgreSQL and MADlib that are valuable for statistical inference in text analytics. Extension libraries for PostgreSQL and Greenplum provide text processing features such as inverted indexes, trigram indexes for approximate string matching, and array data types to model parameters. Existing modules in MADlib, such as Naive Bayes and Sparse/Dense Matrix

manipulations are building blocks to implement statistical text analytics methods. Leveraging this diverse set of tools and techniques that are already in the database allows us to build a sophisticated text analytics engine with statistical inference that has comparable performance to off-the-shelf implementations, but runs natively in a DBMS close to the data [15, 7, 28].

5 Optimizing Queries with Inference

While most of the current systems perform PGM-based data analytics offline in a batch mode, more applications require online data analytics based on PGM methods including inference. As an example, a query over EMR data can be: *return all patients who had a heart operation in the past 3 years and who lives in Berkeley city*. Processing of this query requires linear-chain CRF-based extraction over address text fields because a simple inverted index lookup would return patient records who live on *"Berkeley"* street as well as those who live in *"Berkeley"* city.

Another query example is over the probabilistic KB in Figure 2: *Q: return all writers who lived in Brooklyn*. This query requires inference over the probabilistic KB graph, however, only on a very small portion of the KB graph that is relevant to the query. This observation applies for queries over linear-chain PGM-based text extractions as well: the inference incurred from a query can be applied to a small fraction of PGMs to get the desired answer.

The intuition is to focus the inference computation on subparts of a grounded PGM that are most correlated with the query results. First query-driven inference algorithms are developed [15, 32], which adapt batch-oriented inference algorithms to prune computation given relational query (SQL) conditions.

5.1 Query-Proportional Sampling-Based Inference

In probabilistic KB literature, query-driven inference is done via a top-down theorem proving algorithm [33]. There are two issues with this state-of-the-art approach. First, top-down theorem proving is very expensive as shown in previous work [33, 9]. Incremental theorem proving bounded by time, on the other hand, may miss variables that are influential (i.e., strongly correlated) to the variables in result. Second, the resulting proof tree can be prohibitively large.

To solve the first problem, the materialized propositional KB graph from Section 3 can be used to quickly identify all the variables correlated to the result. To solve the second problem, we can use query-proportional inference techniques. In a *query-proportional sampling algorithm*, the sampling focuses on nodes having strongest influence on the query nodes. An intuitive example of such influential nodes are the query nodes' direct neighbors. In this section, we introduce a quantitative approach to measure such *influence*.

Given a query Q, it is reasonable to choose a sample distribution p that more frequently selects the query variables for sampling. Clearly, the query variable marginal depends on the remaining variables, so we must tradeoff sampling between query and non-query variables. Observing that not all non-query variables influence the query variables equally, we need to answer a key question: *How to pick a variable selection distribution p for a query Q to obtain the highest fidelity answer under a finite time budget?*

Previous work proposes to sample variables based on their *influence* on a query variable according to the graphical model defined by generalizing mutual information [32]. The mutual information measures dependence as a distance between the full joint distribution and its independent approximation. If variables x and y are independent, then this distance is zero and so is their mutual information. Such a query-proportional sampling algorithm based on the influence function not only increases the efficiency of sampling-based inference algorithms, but also computes best-effort approximate result given a time budget.

Our current work has shown promising results applying query-proportional sampling in a cross-document coreference application. We perform query-proportional sampling for coreference queries where only one or few

entities need to be resolved. The result shows that query-driven inference converges for low-selectivity queries in seconds to minutes compared to hours over the New York Times dataset [31].

5.2 Constrained Inference Optimizations

Apart from query-proportional sampling algorithms, another class of query optimization techniques are based on constrained inference. Existing inference algorithms can be optimized by pushing both *query-specific* and *model-specific* constraints to restrict the possible worlds sampled and returned by the inference algorithms over probabilistic graphical models.

Query-specific constraints include the selection, subgraph matching and join conditions. Query-specific constraints limit the space of possible results. Prior work have pushed selection conditions into a Viterbi inference algorithm for information extraction from text [6]. In that case, Viterbi stops early if the selection condition cannot be satisfied in the top-k results, leading to faster query answering [6]. The selection condition over text extraction can be `WHERE token = 'apple' and label = 'company'`. Such selection conditions over text would be translated into the positions of specific `apple` tokens with their doc ID's and label ID's.

Example 1: As an example of how we push a selection condition into a Viterbi matrix V in Figure 4, consider the condition: return text string d with the street num as the label of token #2 (counting from 0). Then in the second recursive step, only the partial segmentation in $V(1, streetnumber)$ satisfies the condition. In the third recursive step, because no partial segmentations in $V(2, y)$, $y \in Y$ come from cell $V(1, streetnumber)$ as shown in Figure 4, token #2 is the smallest pruning position. Thus, we stop the dynamic programming algorithm and conclude that d does not satisfy the condition.

pos	street num	street name	city	state	country
0	5	1	0	1	1
1	2	15	7	8	7
2	12	24	21	18	17
3	21	32	34	30	26
4	29	40	38	42	35
5	39	47	46	46	50

Figure 4: Viterbi matrix V for the linear-chain CRF model in Figure 1(a). First row are all the possible labels and first column are the token positions in the text string "2181 Shattuck North Berkeley CA USA" starting from 0. Each cell $V(i, y)$ stores a ranked list of *entries* $e = \{score, prev(label, idx)\}$ ordered by *score*. Each entry in $V(i, y)$ contains: (1) score of a top-k (partial) segmentation ending at position i with label y; and, (2) a pointer to the previous entry $prev$ on the path that led to top-k scores in $V(i, y)$.

Model-specific constraints refer to the deterministic or near deterministic correlations between variables in a PGM. Such constraints limit the space of possible worlds for sampling, which can be explored to reduce inference computation. Inference algorithms such as MCSAT have been proposed to effectively handle deterministic constraints in relational domain [34] for first-order PGMs such as MLN.

A final query optimization approach is a cost-based heuristic to choose among different PGM inference algorithms, either exact or approximate based on the query and the statistics of the grounded PGM model. Prior work has shown promising results of a hybrid inference algorithm for skip-chain CRF over text and probabilistic database queries over extraction results [7]. The evaluation shows that a simple cost-based optimizer can achieve significant speed-up by choosing the more efficient inference algorithms for different data, model and query combinations.

6 Conclusion

In this article, we discuss challenges and approaches in extending database systems for efficient and scalable in-database analytics based on probabilistic graphical models. First, we discuss our recent work in database representation of probabilistic KBs and an efficient SQL-based grounding algorithm for first-order PGMs. Second, to support in-database statistical methods and inference, we discuss the implementation of statistical text analysis methods and PGM inference algorithms as part of the MADlib project, a library with native implementations of statistical methods in database. Finally, for efficient query processing with both relational and statistical inference operations, we discuss query-driven inference and constraint inference techniques as well as a cost-based optimizer to choose among different inference algorithms based on data, model and query.

Supporting PGM methods in database and online queries with statistical inference is a new and active research area. PGM methods are used for advanced modeling and analysis over very different types of data, including text data, relational data and graph data. We expect much more progress made and techniques developed in this area in the near future.

References

[1] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, p. 21, 2007.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI*, 2012.

[3] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new framework for parallel machine learning," in *UAI*, 2010.

[4] Y. Chen and D. Z. Wang, "Knowledge expansion over probabilistic knowledge bases," in *SIGMOD Conference*, pp. 649–660, 2014.

[5] J. Hellerstein, C. Re, F. Schoppmann, D. Wang, E. Fratkin, A. Gorajek, K. Ng, C. Welton, X. Feng, K. Li, and A. Kumar, "The madlib analytics library or mad skills, the sql," *To appear in VLDB*, 2012.

[6] D. Z. Wang, M. J. Franklin, M. Garofalakis, and J. M. Hellerstein, "Querying probabilistic information extraction," *VLDB*, 2010.

[7] D. Z. Wang, M. J. Franklin, M. Garofalakis, J. M. Hellerstein, and M. L. Wick, "Hybrid in-database inference for declarative information extraction," in *SIGMOD*, 2011.

[8] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton, "Mad skills: new analysis practices for big data," *Proc. VLDB Endow.*, vol. 2, pp. 1481–1492, 2009.

[9] F. Niu, C. Ré, A. Doan, and J. Shavlik, "Tuffy: scaling up statistical inference in markov logic networks using an rdbms," *VLDB*, 2011.

[10] P. Sen and A. Deshpande, "Representing and querying correlated tuples in probabilistic databases," in *ICDE*, pp. 596–605, 2007.

[11] D. Z. Wang, E. Michelakis, M. Garofalakis, and J. M. Hellerstein, "Bayesstore: managing large, uncertain data repositories with probabilistic graphical models," *In Proceedings of VLDB Endowment*, vol. 1, no. 1, pp. 340–351, 2008.

[12] M. Wick, A. McCallum, and G. Miklau, "Scalable probabilistic databases with factor graphs and mcmc," *In Proceedings of VLDB Endowment*, 2010.

[13] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas, "Mcdb: a monte carlo approach to managing uncertain data," in *Proceedings of the 2008 ACM SIGMOD*, pp. 687–700, 2008.

[14] H. C. Bravo and R. Ramakrishnan, "Optimizing mpf queries: Decision support and probabilistic inference," in *SIGMOD*, 2007.

[15] D. Wang, E. Michelakis, M. Franklin, M. Garofalakis, and J. Hellerstein, "Probabilistic Declarative Information Extraction," in *ICDE*, 2010.

[16] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[17] C. Sutton and A. McCallum, "Introduction to conditional random fields for relational learning," in *Introduction to Statistical Relational Learning*, 2008.

[18] J. Biega, E. Kuzey, and F. M. Suchanek, "Inside yago2s: a transparent information extraction architecture," in *Proceedings of the 22nd international conference on World Wide Web companion*, International World Wide Web Conferences Steering Committee, 2013.

[19] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ACM, 2008.

[20] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1, pp. 107–136, 2006.

[21] J. Shavlik and S. Natarajan, "Speeding up inference in markov logic networks by preprocessing to reduce the size of the resulting grounded network," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pp. 1951–1956, Morgan Kaufmann Publishers Inc., 2009.

[22] P. Singla and P. Domingos, "Memory-efficient inference in relational domains," in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pp. 488–493, AAAI Press, 2006.

[23] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, J. Wang, and P. Domingos, "The alchemy system for statistical relational AI," tech. rep., 2009.

[24] "ReVerb dataset, http://reverb.cs.washington.edu/,"

[25] S. Schoenmackers, O. Etzioni, D. Weld, and J. Davis, "Learning first-order horn clauses from web text," in *EMNLP*, 2010.

[26] T. Lin and O. Etzioni, "Identifying functional relations in web," *In Proceeding of Conference on Empirical Methods in Natural Language Processing*, 2010.

[27] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*. MIT Press, 1999.

[28] K. Li, C. Grant, D. Z. Wang, S. Khatri, and G. Chitouras, "Gptext: Greenplum parallel statistical text analysis framework," in *Proceedings of the Second Workshop on Data Analytics in the Cloud*, pp. 31–35, ACM, 2013.

[29] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, L. Pietarinen, and D. Srivastava, "Using q-grams in a dbms for approximate string processing," 2001.

[30] C. E. Grant, J. dan Gumbs, K. Li, D. Z. Wang, and G. Chitouras, "Madden: query-driven statistical text analytics," in *CIKM*, pp. 2740–2742, 2012.

[31] "New York Times dataset, https://catalog.ldc.upenn.edu/ldc2008t19,"

[32] M. L. Wick and A. McCallum, "Query-aware mcmc," in *NIPS*, 2011.

[33] S. Schoenmackers, O. Etzioni, and D. Weld, "Scaling textual inference to the web," in *EMNLP*, 2008.

[34] H. Poon, P. Domingos, and M. Sumner, "A general method for reducing the complexity of relational inference and its application to mcmc," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, pp. 1075–1080, AAAI Press, 2008.