# Archimedes: Efficient Query Processing over Probabilistic Knowledge Bases

Yang Chen*, Xiaofeng Zhou*, Kun Li†, Daisy Zhe Wang*
*Department of Computer and Information Science and Engineering, University of Florida
†Google, Inc.

## ABSTRACT

We present the ARCHIMEDES system for efficient query processing over probabilistic knowledge bases. We design ARCHIMEDES for knowledge bases containing incomplete and uncertain information due to limitations of information sources and human knowledge. Answering queries over these knowledge bases requires efficient probabilistic inference. In this paper, we describe ARCHIMEDES's efficient knowledge expansion and query-driven inference over UDA-GIST, an in-database unified data- and graph-parallel computation framework. With an efficient inference engine, ARCHIMEDES produces reasonable results for queries over large uncertain knowledge bases. We use the Reverb-Sherlock and Wikilinks knowledge bases to show ARCHIMEDES achieves satisfactory quality with real-time performance.

## 1 Introduction

Recent years have seen a drastic rise in the construction of web knowledge bases (KBs), e.g., DBPedia, Freebase, NELL, Probase, and YAGO. Meanwhile, due to the uncertainty of information extraction algorithms and the limitations of human knowledge, current knowledge bases are still incomplete and uncertain, resulting in suboptimal query results [5, 34]. The objective of this paper is to extend our previous research on knowledge expansion [5], query-driven inference [35], and the UDA-GIST computation framework [17] to build a prototype knowledge base system, ARCHIMEDES, to support efficient knowledge expansion with uncertain Horn clauses and query-driven probabilistic inference.

**Knowledge Expansion.** ARCHIMEDES applies large sets of Horn clauses to derive implicit knowledge from existing knowledge bases. The rules are constructed by state-of-the-art first-order mining algorithms [6, 4, 9, 25]. It employs a novel relational model [5] to apply batches of inference rules using relational operations and performs probabilistic inference by query-driven MCMC [18].

**Query-Driven Inference.** Observing that queries are often relevant to small parts of the knowledge graphs [26, 30], ARCHIMEDES applies MCMC only to the $K$-hop networks to achieve real-time performance. Specialized MLN inference algorithms [10, 14, 23] can improve inference quality over the $K$-hop network, but MCMC is more widely supported by existent big data frameworks, e.g., UDA-GIST [17] and GraphLab [19].

**UDA-GIST.** UDA-GIST [17] is an in-database analytics framework unifying data-parallel and graph-parallel computation. State-of-the-art big data frameworks support either data-parallel or graph-parallel computation. GraphLab [19], for example, supports only graph-parallel computation; Spark [32, 33] and MapReduce [7], on the other hand, support only data-parallel computation. UDA-GIST unifies these two types of parallel computation in a cohesive scalable system.

We evaluate ARCHIMEDES on Sherlock-Reverb [25, 8] and Wikilink [29]. These datasets contain large-scale, incomplete, and uncertain knowledge. We compare with Tuffy [22], the probabilistic inference engine of DeepDive, and GraphLab. We show that ARCHIMEDES produces competent result with efficient first-order reasoning and query-driven inference supported by the UDA-GIST in-database framework. We demonstrate the system with ARCHIMEDESONE [35], an interactive query interface. All our code and data are available online[1].

To summarize, we solve the problem of efficient query processing in probabilistic knowledge bases with three novel contributions:

- **Knowledge expansion:** Derive implicit knowledge from knowledge bases using large rule sets;
- **Query-driven inference:** Improve inference performance by focusing MCMC on the query variables;
- **Efficient computation:** Leverage the UDA-GIST unified data- and graph-parallel computation framework.

We organize the remainder of this paper as follows. Section 2 describes the overview of ARCHIMEDES system design. Sections 3 to 5 describe the system components in detail. Section 6 presents experimental evaluation with public knowledge bases. Section 7 discusses related work, and Section 8 concludes the paper.

[1] http://dsr.cise.ufl.edu/projects/probkb-web-scale-probabilistic-knowledge-base.

| Entities $\mathcal{E}$ | Classes $\mathcal{C}$ | Relations $\mathcal{R}$ | Relationships $\Pi$ |
|---|---|---|---|
| Ruth Gruber,<br>New York City,<br>Brooklyn | $W$ (Writer) = {Ruth Gruber},<br>$C$ (City) = {New York City},<br>$P$ (Place) = {Brooklyn} | BornIn($W$, $P$), BornIn($W$, $C$),<br>LiveIn($W$, $P$), LiveIn($W$, $C$),<br>LocateIn($P$, $C$) | 0.96 BornIn(Ruth Gruber, New York City)<br>0.93 BornIn(Ruth Gruber, Brooklyn) |

| Rules $\mathcal{L}$ |
|---|
| 1.40 $\forall w \in W\ \forall p \in P$ (LiveIn($w$, $p$) $\leftarrow$ BornIn($w$, $p$)) |
| 1.53 $\forall w \in W\ \forall c \in C$ (LiveIn($w$, $c$) $\leftarrow$ BornIn($w$, $c$)) |
| 0.32 $\forall p \in P\ \forall c \in C\ \forall w \in W$ (LocateIn($p$, $c$) $\leftarrow$ LiveIn($w$, $p$) $\wedge$ LiveIn($w$, $c$)) |
| 0.52 $\forall p \in P\ \forall c \in C\ \forall w \in W$ (LocateIn($p$, $c$) $\leftarrow$ BornIn($w$, $p$) $\wedge$ BornIn($w$, $c$)) |
| $\infty$ $\forall c_1 \in C\ \forall c_2 \in C\ \forall w \in W$ (BornIn($w$, $c_1$) $\wedge$ BornIn($w$, $c_2$) $\rightarrow c_1 = c_2$) |

**Table 1: Example probabilistic knowledge base constructed from Reverb-Sherlock extractions.**

## 2 Probabilistic Knowledge Bases

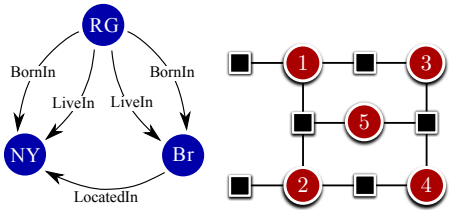
A probabilistic knowledge base extends first-order knowledge bases to support uncertain facts and rules. The primary goal of modeling uncertainty is to represent knowledge mined by probabilistic information extraction algorithms that contain uncertain facts and rules, as illustrated by the Reverb-Sherlock KB in Table 1. We formally define a probabilistic knowledge base below [5].

**Definition 1.** We define a *probabilistic knowledge base* to be a 5-tuple $\Gamma$=($\mathcal{E}$,$\mathcal{C}$,$\mathcal{R}$,$\Pi$,$\mathcal{L}$), where

1. $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$ is a set of *entities*. Each entity $e \in \mathcal{E}$ refers to a real-world object.
2. $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ is a set of *classes* (or *types*). Each class $C \in \mathcal{C}$ is a subset of $\mathcal{E}$: $C \subseteq \mathcal{E}$.
3. $\mathcal{R} = \{R_1, \ldots, R_{|\mathcal{R}|}\}$ is a set of *relations*. Each $R \in \mathcal{R}$ defines a binary relation on $C_i, C_j \in \mathcal{C}$: $R \subseteq C_i \times C_j$. We call $C_i, C_j$ the *domain* and *range* and use $R(C_i, C_j)$ to denote the relation with its domain and range.
4. $\Pi = \{(r_1, w_1), \ldots, (r_{|\Pi|}, w_{|\Pi|})\}$ is a set of *weighted facts* (or *relationships*). For each $(r, w) \in \Pi$, $r$ is a tuple $(R, x, y)$, where $R(C_i, C_j) \in \mathcal{R}$, $x \in C_i, y \in C_j$, and $(x, y) \in R$; $w \in \mathbb{R}$ is a weight indicating how likely $r$ is true. We also use $R(x, y)$ to denote the tuple $(R, x, y)$.
5. $\mathcal{L} = \{(F_1, W_1), \ldots, (F_{|\mathcal{L}|}, W_{|\mathcal{L}|})\}$ is a set of *weighted clauses* (or *rules*). It defines a *Markov logic network*. For each $(F, W) \in \mathcal{L}$, $F$ is a first-order logic clause, and $W \in \mathbb{R}$ is a weight indicating how likely $F$ holds.

As in [5], we confine $\mathcal{L}$ to Horn clauses with binary predicates. Horn clauses prove useful in various knowledge base inference tasks [21, 5, 25]. Their similar structures facilitate efficient inference engines leveraging the KB relational model in Section 3.

**Example 1.** Table 1 shows an example probabilistic KB constructed from Reverb [8] extractions and Sherlock [25] rules. The knowledge base describes the birth place and city of a writer Ruth Gruber with relations "BornIn," "LiveIn," and "LocateIn." The extracted facts state that Ruth Gruber was born in New York City and Brooklyn with weights 0.96 and 0.93, respectively, assigned by IE algorithms. The weighted rules infer Ruth Gruber's living place based on his birth place, and a hard rule with an infinite positive weight states that a person was born in only one city. □




**(a) Knowledge graph.** **(b) Factor graph.**

| ID | Fact |
|---|---|
| 1 | BornIn(Ruth Gruber, New York City) |
| 2 | BornIn(Ruth Gruber, Brooklyn) |
| 3 | LiveIn(Ruth Gruber, New York City) |
| 4 | LiveIn(Ruth Gruber, Brooklyn) |
| 5 | LocatedIn(Brooklyn, New York City) |

**(c) Variables 1-5 in the factor graph (b).**

**Figure 1: Factor graph representation of the Reverb-Sherlock knowledge base.**

We view a probabilistic knowledge base as a template for constructing ground factor graphs [24]. A *factor graph* is a set of factors $\Phi = \{\phi_1, \ldots, \phi_N\}$, where each factor $\phi_i$ is a function $\phi_i(\mathbf{X}_i)$ over a random vector $\mathbf{X}_i$ indicating the probabilistic correlations among the random variables in $\mathbf{X}_i$. These factors together determine a joint probability distribution over the random vector $\mathbf{X}$ consisting of all the random variables in the factors [16]. Factor graphs are visually represented as graphs. In the graph representation, each node is a fact $X_i$ (circle) or factor $\phi_i(\mathbf{X}_i)$ (square) with variables $\mathbf{X}_i$ as its neighbors. Figure 1(b) shows an example factor graph representation of the knowledge graph Figure 1(a). Each factor in Figure 1(b) is defined by a ground fact, e.g., BornIn(Ruth Gruber, New York City) with weight 0.96, or a ground rule, e.g., LiveIn(Ruth

Gruber, New York City) ← BornIn(Ruth Gruber, New York City) with weight 1.40. We use factor graphs to describe these correlations among the facts.

In a factor graph $\Phi = \{\phi_1, \ldots, \phi_N\}$, the factors together determine a joint probability distribution over the random vector $\mathbf{X}$ consisting of all the random variables in the factor graph:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_i \phi_i(\mathbf{X}_i) = \frac{1}{Z} \exp\left(\sum_i W_i n_i(\mathbf{x})\right), \quad (1)$$

where $n_i(\mathbf{x})$ is the number of true groundings of rule $F_i$ in $\mathbf{x}$, $W_i$ is its weight, and $Z$ is the partition function, i.e., normalization constant. ARCHIMEDES answers user queries by computing the marginal probability $P(X = x)$, the probability distribution of a query node $X$ defined by (1). The computation of marginal probabilities is called *marginal inference* in probabilistic graphical models literature. Exact inference is tractable for only limited families of graphical models [16], and state-of-the-art MLN inference engines use sampling algorithms including Markov chain Monte Carlo (MCMC) and MC-SAT [24, 23, 22]. Observing the ground factor graphs of real knowledge bases are large [5] while user queries often focus on small parts of the knowledge graph [35], ARCHIMEDES employs a query-driven approach to focus MCMC on the query nodes to avoid computation over the entire factor graph.

### 2.1 System Architecture

To efficiently process queries, we design three key components of ARCHIMEDES: an inference engine for efficient knowledge expansion to derive implicit knowledge from existing KBs [5], query-driven inference to compute probabilities of the query facts [35], and the UDA-GIST framework for in-database data-parallel and graph-parallel analytics [17]. We provide a user interface for load, search, and update queries, as described in [35]. The system architecture is shown in Figure 2.

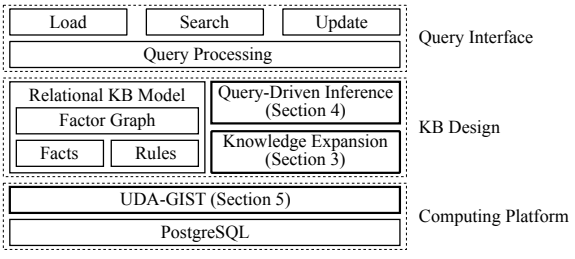



**Figure 2: ARCHIMEDES System Components.**

ARCHIMEDES models facts, rules, and the factor graph in relational tables. The relational model enables it to efficiently perform knowledge expansion by joining the facts and rules tables. The knowledge expansion and query-driven inference using MCMC exemplify applications requiring both data-parallel and graph-parallel computation. They are efficiently supported by the UDA-GIST in-database analytics framework by unifying the UDAs from relational databases and GIST from graph analytics with a shared in-memory state. We describe the details of each component in Sections 3 to 5.

## 3 Knowledge Expansion

To efficiently apply the inference rules, we represent a knowledge base as relational tables. This relational model is first introduced by ProbKB [5] and proves efficient in rule mining [4] by applying rules in batches using join queries. The main challenge with inference rules is that they have flexible structures. To adapt for different structures, we define structural equivalence to divide rules into equivalent classes so that each equivalent class has a fixed table format. In particular, we call two first-order clauses *structurally equivalent* if they differ only in entities, types, and predicates.

**Example 2.** Consider the following inference rules:

1. isMarriedTo($x$, $y$) ← isMarriedTo($y$, $x$);
2. isInterestedIn($x$, $y$) ← influences($y$, $x$) ;
3. influences($x$, $y$) ← directed($x$, $z$), actedIn($y$, $z$);
4. influences($x$, $y$) ← worksAt($x$, $z$), worksAt($y$, $z$).

Rules 1 and 2 are structurally equivalent since their only differences are the predicates (isMarriedTo, influences, isInterestedIn). Similarly, Rules 3 and 4 are structurally equivalent. Therefore, we store Rules 1 and 2 in one table with the columns specifying the predicates of the head and body, as shown in Table 2 (left). We store Rules 3 and 4 in Table 2 (right), its columns storing the head and first, second predicates of the rule body. □

| Head | Body |
|---|---|
| isMarriedTo | isMarriedTo |
| isInterestedIn | influences |

| Head | Body1 | Body2 |
|---|---|---|
| influences | directed | actedIn |
| influences | worksAt | worksAt |

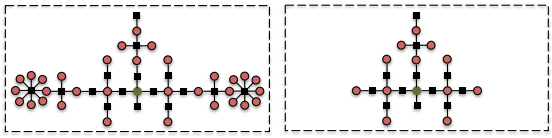
**Table 2: (Left) Relational table for rules 1 and 2. (Right) Relational table for rules 3 and 4.**

Based on the relational model, we express the knowledge expansion algorithm as join queries between the facts and rules tables, one join for each rules table. The details of the join queries are described in [5]. Our experiments show that applying rules in batches results in a 200-300 times of speedup over the state-of-the-art approaches. The result of knowledge expansion is a ground factor graph $\Phi = \{\phi_1, \ldots, \phi_N\}$, where each factor $\phi_i(\mathbf{X}_i)$ represents a ground rule. The factor graph is modeled by a relational table, the columns storing predicate IDs of variables $X \in \mathbf{X}$ and weights of the factors. Performing probabilistic inference on this factor graph yields marginal probabilities of the query facts.

## 4 Query-Driven Inference

ARCHIMEDES uses query-driven inference to speed up MLN inference algorithms by focusing computation on the query facts. The query-driven inference algorithm is designed with the UDA-GIST analytics framework [17] to achieve efficient inference in a relational database system. Furthermore, we use $K$-hop approximation to focus computation on the query facts.

**$K$-hop approximation.** To achieve real-time response, we approximate the inference by extracting $K$-hop sub-networks of the ground factor graph, consisting of nodes within $K$ hops from the query nodes. The $K$-hop approximation is based on the observation that neighbors of the query nodes have more influence than distant nodes. In Figure 3(a), for example, to compute the probability of the central node, we use the 2-hop sub-network in Figure 3(b) for approximation. To achieve real-time response, we use an additional network limit parameter to control the expansion of $K$-hop sub-networks as $K$ increases. In our evaluation, we achieve an 18 times of speedup compared to inference over the entire factor graph by choosing $K = 2$, with an acceptable error of 0.04 in the computed probabilities.



(a) (b)

**Figure 3: (a) The original factor graph. (b) 2-hop network.**

**UDA-GIST.** We use the MCMC algorithms to compute the probabilities defined by the $K$-hop network. We optimize MCMC on the UDA-GIST in-database analytics framework [17]: we build the factor graph by relational operations with User Defined Aggregates (UDAs) and compute probabilities of query facts by MCMC with General Iterative State Transition (GIST). The combined UDA-GIST framework extends relational database systems to support algorithms requiring both data- and graph-parallel computation, including MCMC and MC-SAT. We describe the design of UDA-GIST in Section 5.

## 5 UDA-GIST

Most major DBMSes support User-Defined Aggregates (UDAs) for parallel data analytics. UDAs are suitable for data-parallel analytics where data are naively partitioned and computation is performed on the partitions in parallel. In the context of query processing over large probabilistic KB graphs, such data-parallel operators implement efficient propositional KB graph materialization, subgraph matching, and result generation.

However, UDAs do not support efficient statistical inference algorithms that perform iterative transitions over a large state, where the state is a graph-like data structure. The computation is not naively partitioned due to data dependency within the state (e.g., dependencies between nodes and edges in a graph) as DBMSes are fundamentally data driven and computation is tied to the processing of tuples. We refer to these iterative processing algorithms as graph parallel algorithms. MCMC and random walk over large probabilistic graphical graph are examples of such algorithms. The fundamental question is: *Can graph-parallel inference algorithms be efficiently implemented in DBMSes?*

**The General Iterative State Transition (GIST) Operator.** To answer the demand of supporting in-database graph-parallel analytics, we propose the GIST abstraction to generalize the GraphLab API [17]. GIST defines four abstract data types to describe state-transition algorithms: an in-memory *state* representing the state space, a *task* encoding the state transition task for each iteration, a *scheduler* responsible for the generation and scheduling of tasks, and a *convergence UDA (cUDA)* imposing the stopping condition of the GIST operations. An efficient GIST implementation also supports optimizations including (1) asynchronous parallelization of state transitions, (2) efficient and flexible state implementation, and (3) code generation.

**The UDA-GIST Data Processing Framework.** We integrate the GIST operator into DBMSes with UDAs and User-Defined Functions [2]. From the relational representation of a probabilistic KB graph, SQL queries and UDAs generate a large in-memory state representing the propositional KB graph. The GIST operator then runs parallel inference algorithms on the in-memory state. The query results are extracted from the converged state using an independent UDA function. UDA-GIST unifies data-parallel (e.g., graph materialization) and graph-parallel computation (e.g., inference) into an integrated in-database analytics framework.

## 6 Experiments

We evaluate ARCHIMEDES using Reverb-Sherlock [8, 25] Wikipedia KB with 407,247 facts and 30,912 first-order inference rules, a synthetic knowledge base with varying numbers of facts and rules ranging from 10K to 10M, and Wikilinks for cross-document coreference on UDA-GIST. We run the experiments on a 32-core machine with 64GB of RAM running Red Hat Linux 4.

### 6.1 Result of Knowledge Expansion

To evaluate knowledge expansion, we use Tuffy [22] as the baseline. Figures 4(a)(b) compare performance of ARCHIMEDES with Tuffy on the synthetic knowledge base with varying numbers of facts and rules. We see that ARCHIMEDES achieves more than 200 times of

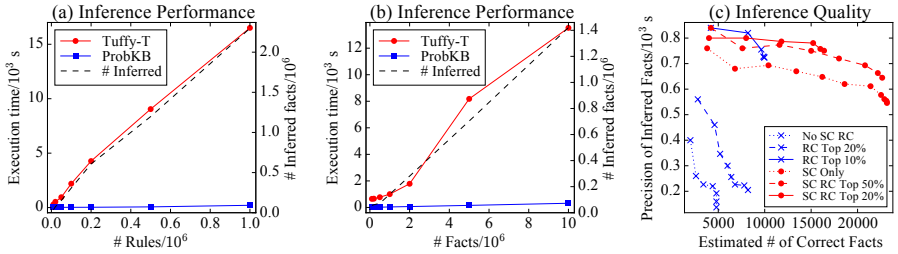



**Figure 4: Knowledge expansion results. (a)(b) Performance comparison with Tuffy. (c) Quality improvement on Reverb-Sherlock.**

speedup over Tuffy for $10^7$ facts. The speedup benefits from the batch application of rules with join operations supported by the relational knowledge base model.

The precision of the inferred facts is shown in Figure 4(c). We use semantic constraints and rule cleaning to improve precision of the inferred facts [5]. As shown in the figure, both semantic constraints and rule cleaning improve precision. The raw Reverb-Sherlock dataset infers 4800 new correct facts at a precision of 0.14. The precision drops quickly when we generate new facts since unsound rules and ambiguous entities result in many erroneous facts. On the contrary, the precision significantly improves with our quality control methods: with top 10% rules we infer 9962 facts at a precision of 0.72; with semantic constraints, we infer 23,164 new facts at precision 0.55. Combining these two methods, we are able to infer 22,654 new facts at precision 0.65 using top 50% rules, and 16,394 new facts at precision 0.75 using top 20% rules.

### 6.2 Result of Query-Driven Inference

Figures 5(a)-(c) report the runtime results for query-driven inference by $K$-hop approximation with different numbers of hops from large, medium, and small clusters. We see that in all the networks, as the number of hops and size of the retrieved networks grow, it takes longer for inference. As a result, query-driven inference achieves a speedup of more than one order of magnitude compared to using the entire factor graph for computation. Meanwhile, we observe that the error rate in the computed probabilities drops to 0.04 with only 3000 neighboring nodes in the MCMC computation. Thus, query-driven inference efficiently answers user queries by focusing computation on the relevant neighbors with acceptable error rates in the computed probabilities.

### 6.3 Result of UDA-GIST

We evaluate the performance and scalability of UDA-GIST by cross-document coreference using the Wikilinks datasets [29]. The dataset contains about 40 millions mentions over 3 millions entities. We extract two datasets: Wikilink 1.5 (first 565 1.5M mentions from the 40M dataset) and Wikilink 40 (all 40M mentions in the dataset) from this Wikilink dataset. The Wikilink 40 dataset is 27 times larger than used in the current state-of-the-art [28]. The result is reported in Figures 5(d)(e). For the entire dataset, the state building takes approximately 10 minutes. We run 20 iterations each with $10^{11}$ pairwise mention comparisons. Each iteration takes approximately 1 hour and we see the graph converges at iteration 10 with precision 0.79, recall 0.83 and $F_1$ 0.81. Using our solution, within a manageable 10-hour computation in a single system the coreference analysis can be performed on the entire Wikilink dataset, 27 times larger than achieved by the current state-of-the-art [28].

## 7 Related Work

**Knowledge Base Construction.** Knowledge bases are receiving increasing research and industrial interest, e.g.: DBpedia [1], Freebase [3], NELL [21], ProBase [31], and YAGO [20]. However, they are often incomplete and uncertain due to limitations of information sources and human knowledge. To model the correlations among uncertain facts, NELL [21] and DeepDive [34] use inference rules. We extend this approach to large rule sets with similar structures [6, 4, 9, 25] by modeling the rules as relational tables, enabling the applications of batches of inference rules with relational operators [5]. Our approach is scalable; it has been applied to large knowledge bases including Freebase [6, 4].

**Probabilistic Inference.** To compute the probabilities, general probabilistic inference algorithms–MCMC [27], Gibbs sampling [11], belief propagation [12]–or specialized MLN inference algorithms [10, 14, 23] are viable options. All the inference algorithms benefit from query-driven inference by avoiding computation on the entire graph [35, 26, 30]. In ARCHIMEDES, we use MCMC because of its wide use and existing support in UDA-GIST [18] and other state-of-the-art big data analytics frameworks we describe below.

**Parallel computing.** In recent years, various analytics frameworks have been developed to facilitate large-scale data analytics: MADlib [15], Spark [33, 32], MapReduce [7], GraphLab [19], and GraphX [13]. These frameworks support either data-parallel or graph-parallel computation. For example, the data-driven UDA operations in MADlib, Spark, and MapReduce provide data-parallel

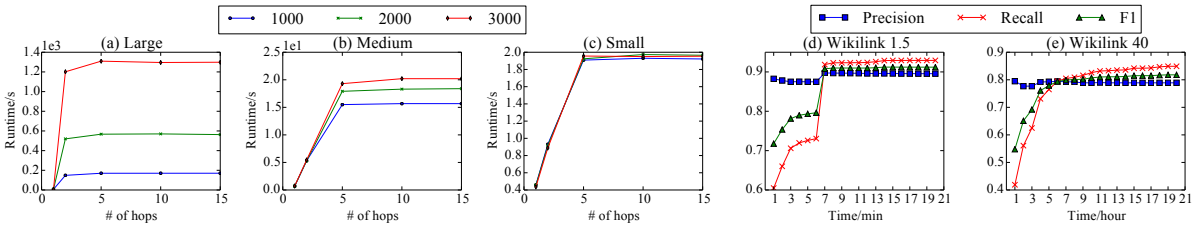



**Figure 5: (a)-(c) Performance of query-driven inference. (d)(e) Performance improvement of UDA-GIST compared to GraphLab.**

computation, but are inefficient for asynchronous graph-parallel computation like MCMC. GraphX, built on Spark, is based on a synchronous computation engine, making MCMC less efficient than GraphLab [17]. GraphLab, however, requires sequential graph construction and result extraction. The UDA-GIST framework improves on these works by integrating UDA and GIST with a shared in-memory state, thus unifying data- and graph-parallel computation frameworks in a DBMS.

## 8 Conclusion

In this paper, we present ARCHIMEDES for query processing over probabilistic knowledge bases. We extend the state-of-the-art query processing and optimization techniques to knowledge base systems by knowledge expansion and query-driven inference, supported by the UDA-GIST framework. UDA-GIST is an in-database analytics framework that unifies data-parallel and graph-parallel computation. We evaluate ARCHIMEDES with public knowledge bases including Reverb-Sherlock and Wikilink. We show ARCHIMEDES achieves real-time performance with satisfactory quality. In future work, we plan to improve the query processing algorithm and supporting framework with performance optimizations.

**Acknowledgments.** We acknowledge the support of NSF under IIS Award # 1526753, DARPA under FA8750-12-2-0348-2 (DEFT/CUBISM), and a gift from Google.

## 9 References

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 2007.

[2] T. Bain, L. Davidson, R. Dewson, and C. Hawkins. User defined functions. In *SQL Server 2000 Stored Procedures Handbook*, pages 178–195. Springer, 2003.

[3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*. ACM, 2008.

[4] Y. Chen, S. Goldberg, D. Z. Wang, and S. S. Johri. Ontological pathfinding. In *SIGMOD*. ACM, 2016.

[5] Y. Chen and D. Z. Wang. Knowledge expansion over probabilistic knowledge bases. In *SIGMOD*. ACM, 2014.

[6] Y. Chen, D. Z. Wang, and S. Goldberg. Scalekb: Scalable learning and inference in large knowledge bases. *The VLDB Journal*, 2016.

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 2008.

[8] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *EMNLP*, 2011.

[9] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 2015.

[10] W. Gatterbauer and D. Suciu. Dissociation and propagation for approximate lifted inference with standard relational database management systems. *The VLDB Journal*, 2016.

[11] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin. Parallel gibbs sampling: From colored fields to thin junction trees. In *AISTATS*, 2011.

[12] J. Gonzalez, Y. Low, and C. Guestrin. Residual splash for optimally parallelizing belief propagation. In *AISTATS*, 2009.

[13] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. Graphx: Graph processing in a distributed dataflow framework. In *OSDI*, 2014.

[14] E. Gribkoff and D. Suciu. Slimshot: in-database probabilistic inference for knowledge bases. *Proceedings of the VLDB Endowment*, 2016.

[15] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, et al. The madlib analytics library: or mad skills, the sql. *VLDB*, 2012.

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

[17] K. Li, D. Z. Wang, A. Dobra, and C. Dudley. Uda-gist: An in-database framework to unify data-parallel and state-parallel analytics. *VLDB*, 2015.

[18] K. Li, X. Zhou, D. Z. Wang, C. Grant, A. Dobra, and C. Dudley. In-database batch and query-time inference over probabilistic graphical models using uda-gist. *The VLDB Journal*, 2016.

[19] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *VLDB*, 2012.

[20] F. Mahdisoltani, J. Biega, and F. Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*, 2014.

[21] T. Mitchell and et. al. Never-ending learning. In *AAAI*, 2015.

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

[23] H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI*, 2006.

[24] M. Richardson and P. Domingos. Markov logic networks. *Machine learning*, 2006.

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

[26] J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré. Incremental knowledge base construction using deepdive. *VLDB*, 2015.

[27] S. Singh. *Scaling MCMC Inference and Belief Propagation to Large, Dense Graphical Models*. PhD thesis, University of Massachusetts Amherst, 2014.

[28] S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Large-scale cross-document coreference using distributed inference and hierarchical models. In *Proceedings of ACL-HLT*, 2011.

[29] S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Wikilinks: A large-scale cross-document coreference corpus labeled via links to wikipedia. *University of Massachusetts, Amherst, Tech. Rep.*, 2012.

[30] M. L. Wick and A. McCallum. Query-aware mcmc. In *NIPS*, 2011.

[31] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *SIGMOD*, 2012.

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

[33] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.

[34] C. Zhang. *DeepDive: A Data Management System for Automatic Knowledge Base Construction*. PhD thesis, UW-Madison, 2015.

[35] X. Zhou, Y. Chen, and D. Z. Wang. Archimedesone: Query processing over probabilistic knowledge bases. *VLDB*, 2016.