Graph Queries in a Next-Generation Datalog System

Alexander Shkapsky
University of California, Los Angeles
shkapsky@cs.ucla.edu

Kai Zeng
University of California, Los Angeles
kzeng@cs.ucla.edu

Carlo Zaniolo
University of California, Los Angeles
zaniolo@cs.ucla.edu

ABSTRACT

Recent theoretical advances have enabled the use of special monotonic aggregates in recursion. These special aggregates make possible the concise expression and efficient implementation of a rich new set of advanced applications. Among these applications, graph queries are particularly important because of their pervasiveness in data intensive application areas. In this demonstration, we present our *Deductive Application Language* (*DeAL*) System, the first of a new generation of Deductive Database Systems that support applications that could not be expressed using regular stratification, or could be expressed using XY-stratification (also supported in *DeAL*) but suffer from inefficient execution. Using example queries, we will (i) show how complex graph queries can be concisely expressed using *DeAL* and (ii) illustrate the formal semantics and efficient implementation of these powerful new monotonic constructs.

1. INTRODUCTION

The recent revival of interest in Datalog [5] is driven by various developments that include the emergence of natural application areas, such as computer networking [13], parallel and distributed programming [10], and distributed data management [2], and the success of industrial-strength systems [9]. Additional drivers have been Datalog's uses in (i) advanced computational and semantic models [10, 7], (ii) the Big Data problem [3, 6], and (iii) Data Stream Management Systems [18]. Due to space limitations, this is a very incomplete list, which does not mention many significant contributions from the past, and the many new contributions that are emerging now, i.e., in a time that has been described with terms such as 'resurgence' [5], 'springtime' [10] and 'renaissance' [1] for Datalog[1].

[1]The last term is actually the most fitting, since the Renaissance is the era that, after the 'dark ages', revived arts and sciences, producing accomplishments that outshined and outlasted even the glorious ones of classical times.

Proceedings of the VLDB Endowment, Vol. 6, No. 12

The excitement with these new developments should not make us overlook the limitations and problems that have impaired the power and generality of Datalog in the past– problems that were clearly identified and motivated classical Datalog research [17, 19, 8, 15, 12, 16]. These seminal works have recognized a key problem as that of supporting aggregates in recursion, which are needed to support efficiently many applications, including graph queries. Many graph queries require recursive aggregation to compute an aggregate value for a vertex based on an aggregate value produced earlier for connected vertices. For example, PageRank iteratively calculates the probability for a document being visited by aggregating the transition probability for those documents linking to the document.

The previous solutions proposed were quite limited in their scope, and to the best of our knowledge, they were not supported in any Datalog system, forcing programmers to hand-code and manually tune procedural implementations. Even when expressed in XY-stratified rules [4], recursive programs with aggregates suffer from inefficient query evaluation performance. In this springtime for Datalog, a more general theory has sprouted to life [14], providing the formal basis for the very practical language and system extensions that are the focus of our demo. Space constraints force us to limit this presentation to a general overview, whereas details and formal proofs are given in [14].

In this paper, we present our *Deductive Application Language* (*DeAL*) System[2], a next-generation Datalog system we have developed at UCLA, building on earlier Deductive Database technology [4]. *DeAL* supports negation, built-in and user defined aggregates, and non-monotonic aggregation via XY-stratification, and more importantly, the new monotonic aggregate extensions discussed in Section 2. With support for many powerful constructs, a wide range of complex applications can be declaratively expressed and executed efficiently in our system. In this demonstration, we focus on *DeAL*'s support for graph queries due to their data intensive nature and importance in several application domains, including the web, social networks and computer networks. We show how *DeAL*'s monotonic aggregate syntax for recursive rules is natural and allows for the expression of powerful, yet elegant graph queries.

In the following section, we will review our monotonic aggregate extensions supported in *DeAL*. Section 3 provides an overview of the *DeAL* System architecture. Section 4 presents example advanced applications. Section 5 describes our demonstration and we conclude in section 6.

[2]http://wis.cs.ucla.edu/deals

2. MONOTONIC AGGREGATES

DeAL supports the standard SQL aggregates and user-defined aggregates. These are non-monotonic aggregates (w.r.t. set containment) and therefore can only be used in stratified programs, i.e., with the same constraints regulating the use of negation in programs. *DeAL* also supports XY-stratification that is a form of explicit (i.e., compile-time recognizable) local stratification that was first introduced by LDL++ [4] and recently proved quite useful in the parallelization of advanced analytics in MapReduce distributed execution environments [6]. The important novelty of *DeAL* however is that it introduces the two monotonic aggregates `fsmax` and `fscnt` that can be used freely in recursive definitions. The formal semantics of these two aggregates is based on the Datalog^{FS} work which proved that a least-fixpoint semantics, and its equivalent unique minimal model semantics, exist for these [14]. Given that their formal semantics has already received in-depth treatment [14], we will next concentrate on their intuitive appeal and expressivity, through a variety of examples, and then describe the basics of their efficient implementation. These topics are the focus of discussion for our demonstration.

2.1 The `fsmax` Aggregate

We present the `fsmax` construct through an example using Bill of Materials (BOM), a classical recursive application for traditional Database Management Systems. In BOM, the database contains a large set of `assbl(Part, Subpart, Qty)` facts which describe how a given part is assembled using various subparts, each in a given quantity. Not all subparts are assembled, basic parts are instead supplied by external suppliers in a given number of days, as per the facts `basic(Part, Days)`. Simple assemblies, such as bicycles, can be put together the very same day in which the last basic part arrives. Thus, the time needed to deliver a bicycle is the maximum of the number of days that the various basic parts require to arrive.

EXAMPLE 1. *How many days until delivery?*

```
delivery(Part, fsmax⟨Days⟩) ← basic(Part, Days).
delivery(Part, fsmax⟨Days⟩) ← assbl(Part, Sub, _),
                              delivery(Sub, Days).
actualDays(Part, max⟨Days⟩) ← delivery(Part, Days).
```

We use the notation for aggregates that is used in LDL++ [4], and thus `max` denotes the usual non-monotonic aggregate that can also be expressed using negation. However `fsmax` denotes the new fs-aggregates introduced in [14] which are monotonic. For `fsmax` monotonicity is achieved by simply viewing this as a continuous function, which along with producing the maximum value of days required for a part to arrive also produces all the positive integers up to and including the max. In other words, where `max` might return the pair `(Bolt, 4)`, `fsmax` will instead return `(Bolt, 4)`, `(Bolt, 3)`, `(Bolt, 2)`, `(Bolt, 1)`. Thus if, the recursive computation also produces `(Bolt, 3)`, `(Bolt, 2)`, `(Bolt, 1)`, this can be discarded because w.r.t. set containment semantics, this is a subset of the previous. Thus only maxima are significant according to the value-based set semantics of Datalog. Therefore we now have maxima in recursion while preserving the declarative least-fixpoint semantics of Datalog [14].

The standard implementation and optimization techniques of Datalog, including magic sets and seminaive fixpoint remain valid [14]. Moreover when we consider the operational semantics of the seminaive fixpoint iteration, we see that because of the final `max` aggregate used in the `actualDays` only maxima are of interest for each part, and lesser values can be discarded during the recursive computation. This max-based optimization is applicable whenever the function used in the body to compute the `fsmax` argument is monotonically increasing for positive numbers. In the example at hand this is trivially true, since `Days` is transferred from the body to the head by the identity function, but the same would be true, if e.g., the argument of `fsmax` in the head is `Days` in the body multiplied by a positive constant. Therefore with `fsmax` used in recursion, both the declarative semantics of Datalog and its efficient implementation are preserved.

As discussed in [14] the `fsmax` aggregate can be used on arbitrary positive numbers not just integers. For instance, the following program computes the maximum probability path between two nodes in a network where `net(X, Y, P)` denotes the probability of reaching `Y` starting from `X` is `P`.

EXAMPLE 2. *Max Probability Path*

```
reach(X, Y, fsmax⟨P⟩) ← net(X, Y, P).
reach(X, Z, fsmax⟨P⟩) ← reach(X, Y, P1),
                        reach(Y, Z, P2), P = P1 * P2.
maxP(X, Y, max⟨P⟩)    ← reach(X, Y, P).
```

This program uses quadratic rules as in Floyd's shortest-path algorithm, but linear rules would also be sufficient. With regards to Floyd's algorithm, this can also be expressed in *DeAL*, but since we only support maxima, the inverses of the costs of the arcs was actually used in the computation to find shortest paths [14].

2.2 The Continuous `fscnt` Aggregate

This monotonic construct performs a continuous count of the number of distinct occurrences and allows us to express many queries that could not be expressed efficiently, or could not be expressed at all, in Datalog with stratified negation or aggregates. For instance, the following query that counts the paths between nodes in a directed graph was not expressible in Datalog with stratified aggregates [16].

EXAMPLE 3. *Counting Paths*

```
cpaths(X, Y, fscnt⟨X⟩)          ← arc(X, Y).
cpaths(X, Z, fscnt⟨(Y, C)⟩)     ← cpaths(X, Y, C), arc(Y, Z).
maxC(X, Z, max⟨C⟩)              ← cpaths(X, Z, C).
```

The first rule computes that each arc counts as one path between its nodes. The second rule states that if there are `C` distinct paths from `X` to `Y`, then the presence of `arc(Y, Z)` establishes that there also are `C` occurrences of distinct paths from `X` to `Z` through `Y`. The `fscnt⟨(Y, C)⟩` aggregate in the head performs the continuous count of these occurrences[3].

Section 4 includes additional examples and discussion.

3. SYSTEM OVERVIEW

A high-level outline of the *DeAL* System architecture is shown in Figure 1. *DeAL* is implemented in Java and is currently deployed only in single-node server configuration.

[3]The final max value of this count can be simply derived by adding up the max value of each `C` for each `Y`–an observation that is used to greatly expedite the implementation, whereas at the formal semantic level we use continuous count since they are monotonic w.r.t. set containment.

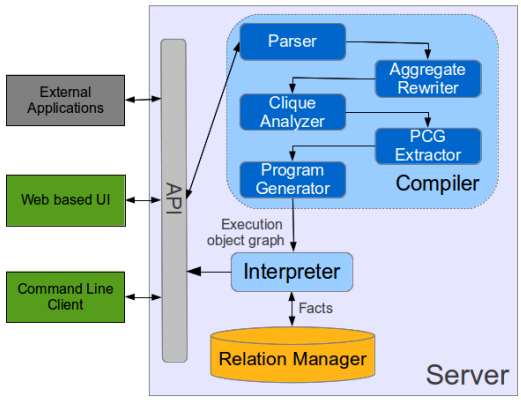

Figure 1: System Architecture

As shown in Figure 1, the system components supporting the execution of monotonic aggregate queries are the Compiler, Interpreter and the Relation Manager, which is the in-memory database for both the extensional database (EDB) and intensional database (IDB). The system supports both user-facing interfaces, such as our web based user interface shown in Figure 2 and our command line client, and external applications interfacing directly with the server's API.

As alluded to in Figure 1, there are five steps of compilation in *DeAL*. In step 1, rules are parsed. In step 2, rules with aggregates are expanded and rewritten using a generalization of the techniques used in [4]. In step 3, the Clique Analyzer ensures rules are properly grouped. In step 4, the rules are formed into a Predicate Connection Graph (PCG), which is an AND/OR graph describing the program. This AND/OR graph is then used in step 5 by the Program Generator to output an execution object graph, which are the instantiated Java objects of the program to be executed by the Interpreter. Should a query form be submitted with bound variables, the compiler will apply techniques such as magic sets to monotonic aggregates.

4. ADVANCED APPLICATIONS

In this section, we discuss advanced applications to illustrate the expressive power of our new monotonic aggregates.

4.1 Traditional DBMS Queries

Traditional Database Management System queries can be efficiently implemented using *DeAL*.

Company Control. In the Company Control program proposed in [15], companies can purchase ownership shares of other companies. In addition to the shares that company A owns directly, company A also controls the shares controlled by company B when A has a controlling majority ($> 50\%$) of B's shares.

EXAMPLE 4. *Company Control*

$$\begin{aligned}
&\texttt{cshares}(\texttt{A},\texttt{B},\texttt{dirct},\texttt{fsmax}\langle\texttt{P}\rangle) \leftarrow \texttt{ownedshares}(\texttt{A},\texttt{B},\texttt{P}).\\
&\texttt{cshares}(\texttt{A},\texttt{C},\texttt{indrct},\texttt{fscnt}\langle(\texttt{B},\texttt{P})\rangle) \leftarrow \texttt{bought}(\texttt{A},\texttt{B}),\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \texttt{cshares}(\texttt{B},\texttt{C},_,\texttt{P}).\\
&\texttt{bought}(\texttt{A},\texttt{B}) \leftarrow \texttt{cshares}(\texttt{A},\texttt{B},_,\texttt{P}),\texttt{P} > 50,\texttt{A} \neq \texttt{B}.
\end{aligned}$$

4.2 Graph Analytics

Many Data Mining and Machine Learning applications use graphs as the underlying model, such as a Markov chain; and many graph analytics queries use a probability model. As Example 2 shows, our semantics for monotonic aggregates naturally support probability. This makes many graph analytics queries easily expressible in *DeAL* and our optimizations makes query evaluation efficient as well.

Markov Chains and Page Rank. A Markov chain is represented by the transition matrix W of $s \times s$ components where w_{ij} is the probability to go from state i to state j in one step. A Markov chain is called *irreducible* if for each pair of states i, j, the probabilities to go from i to j and from j to i in one or more steps is greater than zero.

Computing stabilized probabilities of a Markov chain has many real-world applications, such as estimating the distribution of population in a region, and determining the Page Rank of web nodes. Let P be a vector of stabilized probabilities of cardinality s, the equilibrium condition in terms of matrices is: $P = W \cdot P$. Although computing this fixpoint is far from trivial, irreducible chains can be modeled quite naturally in *DeAL*. If $\texttt{p\_state}(\texttt{X},\texttt{K})$ denotes that $\texttt{K}$ is the rank of node $\texttt{X}, 1 \leq \texttt{X} \leq \texttt{s}$, and $\texttt{w\_matrix}(\texttt{Y},\texttt{X},\texttt{W})$ denotes that there is an arc from $\texttt{Y}$ to $\texttt{X}$ with weight $\texttt{W}$. Then, we compute the fixpoint as follows:

$$\begin{aligned}
&\texttt{p_state}(\texttt{X},\texttt{fscnt}\langle\texttt{K}\rangle) \leftarrow \texttt{p_state}(\texttt{Y},\texttt{C}),\texttt{w_matrix}(\texttt{Y},\texttt{X},\texttt{W}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \texttt{K} = \texttt{C} * \texttt{W}.\\
&\texttt{rank}(\texttt{X},\texttt{max}\langle\texttt{K}\rangle) \leftarrow \texttt{p_state}(\texttt{X},\texttt{K}).\\
&\texttt{w_matrix}(1,1,w_{11}).\\
&\texttt{w_matrix}(1,2,w_{12}).\\
&\vdots\\
&\texttt{w_matrix}(\texttt{s},\texttt{s},w_{ss}).
\end{aligned}$$

Note that each fixpoint of such a program is an equilibrium $P = W \cdot P$ of the Markov Chain represented by matrix W. We add $\texttt{p\_state}(1, 0.1).\texttt{p\_state}(2, 0.1)...\texttt{p\_state}(\texttt{s}, 0.1)$, which provided the set of baseline facts of the least fixpoint.

T_{Pbl}. For each irreducible Markov chain there exists a non trivial fixpoint, therefore T_P has one that is not null at every node, and there exists a finite fixpoint for T_{Pbl}. Therefore, the least fixpoint for T_{Pbl} is finite. Additionally, in [14] we have proven the following properties:

- The least fixpoint of the baseline *DeAL* program that models an irreducible Markov chain is finite.
- Every non-null solution of an irreducible Markov chain can be obtained by scaling the least fixpoint solution of the baseline model produced by *DeAL*.

In summary, although there has been a significant amount of previous work on Markov chains, *DeAL*'s monotonic aggregates provide us with a new method and a simple algorithm which are valid for all irreducible Markov chains, including periodic ones.

Social Networks. Social Networks is an area rich with example applications that can utilize *DeAL*'s monotonic aggregates. Shortest Path queries identify the minimum cost path from a node to other nodes in the graph. Connected Component queries are used to identify communities within a network. The diffusion of innovation, information and behaviors in a social network can be modeled as a Jackson-Yariv Diffusion Model (JYDM) [11] and implemented with monotonic aggregates in *DeAL*. The approach behind the efficient execution of a JYDM is discussed in [14].

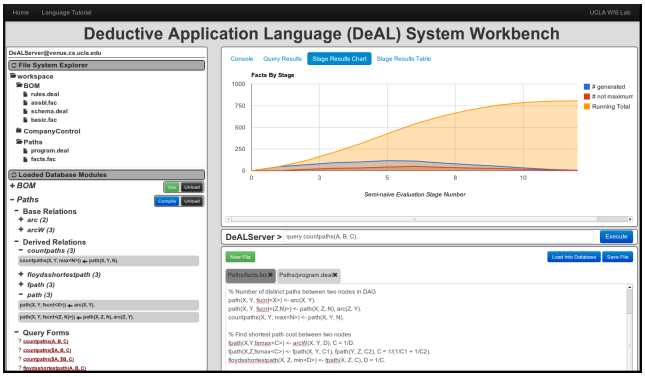

Figure 2: Web Based UI

5. DEMONSTRATION

The main goals of our demonstration were to (i) expose the user to *DeAL*'s graph query capabilities, (ii) familiarize the user with application domains *DeAL* is intended for, and (iii) present the user with an understanding of how recursive aggregate graph queries are efficiently executed in our system. To achieve (i) and (ii) we allowed the user to execute prepared queries over our datasets using the web based user interface pictured in Figure 2. The prepared queries included our example queries from Sections 2 and 4, as well as programs for Shortest Path, PageRank and JYDM. Additionally, our user interface allowed conference attendees to write, compile and execute queries.

We had several ways to achieve (iii). First, our user interface provided conference attendees with high-level visualizations of the intermediate results produced during recursive monotonic aggregation. Second, to gain further insight into the efficiency of our approach, attendees were able to explore execution traces of the computations produced during execution that lead to the final results. Lastly, we presented live performance comparisons of *DeAL* programs written with the new monotonic extensions against both stratified *DeAL* programs and XY-stratified *DeAL* programs, as the system supports all three paradigms.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented the *DeAL* System, a next-generation Datalog system which features new monotonic aggregates that efficiently support of a wide range of important graph queries. We reviewed our new monotonic aggregates `fsmax` and `fscnt` using example queries and we discussed advanced applications from several domains to show the general applicability of these extensions. We provided a high-level overview of the *DeAL* System architecture. Lastly, we reviewed the details of our demonstration.

Our future work includes investigating the effectiveness and performance of the parallel fixpoint computation techniques proposed [3] in new *DeAL* applications. In fact, the parallelized seminaive fixpoint techniques investigated in [3], after minor extensions, can now be used for the many advanced applications supported in *DeAL*.

Acknowledgements

This work was supported by NSF under grant: IIS 1218471.

7. REFERENCES

[1] S. Abiteboul. Datalog: La renaissance. `http://www.college-de-france.fr/site/serge-abiteboul/course-2012-05-09-10h00.htm`, 2012.

[2] S. Abiteboul, M. Bienvenu, A. Galland, and E. Antoine. A rule-based language for web data management. In *PODS*, pages 293–304, 2011.

[3] F. N. Afrati, V. R. Borkar, M. J. Carey, N. Polyzotis, and J. D. Ullman. Map-reduce extensions and recursive queries. In *EDBT*, pages 1–8, 2011.

[4] F. Arni, K. Ong, S. Tsur, H. Wang, and C. Zaniolo. The deductive database system ldl++. *TPLP*, 3(1):61–94, 2003.

[5] P. Barceló and R. Pichler, editors. *Datalog in Academia and Industry–2nd International Workshop, Datalog 2.0, Vienna, Austria, September 11-13*, volume 7494 of *LNCS*. Springer, 2012.

[6] Y. Bu, V. R. Borkar, M. J. Carey, J. Rosen, N. Polyzotis, T. Condie, M. Weimer, and R. Ramakrishnan. Scaling datalog for machine learning on big data. *CoRR*, abs/1203.0160, 2012.

[7] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *ICDE*, pages 2–13, 2011.

[8] S. Greco and C. Zaniolo. Greedy algorithms in datalog. *TPLP*, 1(4):381–407, 2001.

[9] T. J. Green, M. Aref, and G. Karvounarakis. Logicblox, platform and language: A tutorial. In *Datalog in Academia and Industry*, pages 1–8, 2012.

[10] J. M. Hellerstein. Datalog redux: experience and conjecture. In *PODS*, pages 1–2, 2010.

[11] M. O. Jackson and L. Yariv. Diffusion on social networks. *Economie Publique*, 16:3–16, 2005.

[12] P. G. Kolaitis. The expressive power of stratified logic programs. *Information and Computation*, 90(1):50–66, 1991.

[13] B. T. Loo, T. Condie, M. N. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, 2009.

[14] M. Mazuran, E. Serra, and C. Zaniolo. Extending the power of datalog recursion. *The VLDB Journal*, 22(4):471–493, 2013.

[15] I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The magic of duplicates and aggregates. In *VLDB*, pages 264–277, 1990.

[16] I. S. Mumick and O. Shmueli. How expressive is stratified aggregation? *Annals of Mathematics and Artificial Intelligence*, 15:407–435, 1995.

[17] K. A. Ross and Y. Sagiv. Monotonic aggregation in deductive databases. *Journal of Computer and System Sciences*, 54(1):79–97, 1997.

[18] C. Zaniolo. Logical foundations of continuous query languages for data streams. In *Datalog in Academia and Industry*, pages 177–189, 2012.

[19] C. Zaniolo, S. Ceri, C. Faloutsos, R. T. Snodgrass, V. S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann, 1997.