6-2012

Querying Provenance for Ranking and Recommending

Zachary G. Ives
University of Pennsylvania, zives@cis.upenn.edu

Andreas Haeberlen
University of Pennsylvania, ahae@cis.upenn.edu

Tao Feng
University of Pennsylvania

Wolfgang Gatterbauer
Carnegie Mellon University

Querying Provenance for Ranking and Recommending

Abstract
As has been frequently observed in the literature, there is a strong connection between a derived data item's provenance and its authoritativeness, utility, relevance, or probability. A standard way of obtaining a score for a derived tuple is by first assigning scores to the "base" tuples from which it is derived — then using the semantics of the query and the score measure to derive a value for the tuple. This "provenance-enabled" scoring has led to a variety of scenarios where tuples' intrinsic value is based on their provenance, independent of whatever other tuples exist in the data set.

However, there is another class of applications, revolving around sharing and recommendation, in which our goal may be to rank tuples by their "importance" or the structure of their connectivity within the provenance graph. We argue that the most natural approach is to exploit the structure of a provenance graph to rank and recommend "interesting" or "relevant" items to users, based on global and/or local provenance graph structure and random walk-based algorithms. We further argue that it is desirable to have a high-level declarative language to extract portions of the provenance graph and then apply the random walk computations. We extend the ProQL provenance query language to support a wide array of random walk algorithms in a high-level way, and identify opportunities for query optimization.

Disciplines
Computer Sciences

Comments
Ives, Z., Haeberlen, A., Feng, T., & Gatterbauer, W., Querying Provenance for Ranking and Recommending, *4th USENIX Workshop on the Theory and Practice of Provenance (TaPP'12)*, June 2012, https://www.usenix.org/conference/tapp12/querying-provenance-ranking-and-recommending

Querying Provenance for Ranking and Recommending

Zachary G. Ives Andreas Haeberlen Tao Feng
Computer and Information Science Department
University of Pennsylvania
{zives,ahae,fengtao}@cis.upenn.edu

Wolfgang Gatterbauer
Tepper School of Business
Carnegie Mellon University
gatt@cmu.edu

1 Introduction

As has been frequently observed in the literature [4, 5, 7], there is a strong connection between a derived data item's provenance and its authoritativeness, utility, relevance, or probability. A standard way of obtaining a score for a derived tuple is by first assigning scores to the "base" tuples from which it is derived — then using the semantics of the query and the score measure to derive a value for the tuple. For instance, a probabilistic database may use provenance to compute a probabilistic event expression for a result, and, given probability values or distributions over the base data values, it can use this to compute probability values or distributions over the resulting value [4, 5]. Similarly, a keyword search system over databases, which typically finds "join trees" relating tuples matching the search terms, can assign a score to each join tree by looking at the base scores and composing the scores through the operators [18].

Green et al. [8] have shown that evaluation of scores under many of these models is a special case of computing over data with annotations following the *provenance semiring* model — in which tuples are annotated with polynomial expressions from a *commutative semiring* describing their direct derivations from other tuples. The commutative semiring is an abstract algebraic structure that can be "specialized" to a variety of scoring models, such as constructing probabilistic event expressions, counting the number of derivations, or determining the "minimal witness" for the existence of a tuple.

The set of provenance annotation expressions can equivalently be represented as a *graph* relating tuples in the database and their connections via derivations. This graph representation can be directly stored, manipulated, and operated upon using extensions to standard database techniques [7, 12]. A wide array of applications can then be built over a provenance graph storage system, ranging from keyword search [18] (where each result is annotated with a score depending on the sources and mappings) to incremental view maintenance [7] (where each tuple in a view is annotated with the conditions under which it remains derivable) to security [6] (where tuples are annotated with visibility levels). All of these applications can directly operate over the same provenance graph: given an assignment of annotation values to "base" tuples, plus a specification for how these annotations compose (a particular instantiation of the semiring) the application can traverse the provenance graph and assign an annotation to every derived result.

Clearly, "provenance-enabled" scoring has led to a variety of new applications and usage scenarios where tuples' intrinsic value is based on their provenance, independent of whatever other tuples exist in the data set. However, there is another class of applications, revolving around sharing and *recommendation*, in which our goal may be to rank tuples by their "importance" or the structure of their connectivity within the provenance graph. More concretely, consider two problems that are commonly encountered in collaborative settings:

Ranking usage or influence based on the structure of provenance. Consider a community portal for sharing data, code, or queries, along the lines of SourceForge, or a platform for sharing, analyzing, and visualizing structured data along the lines of Google Fusion Tables or myexperiment.org. Intuitively, if we track the provenance of every derived object, we should be able to rank the "best" contributors to the site in terms of their overall influence on what was shared. Observe that this problem is quite different from assigning a score based purely on an individual object's provenance. It more closely resembles the *link analysis* problem for Web pages.

Measuring relatedness based on close connectivity, which may be useful in clustering and recommendation. In a variety of recommendation applications, it is useful to be able to cluster users together based on how many data items and/or tools they commonly use. Again, in terms of the structure of the provenance graph, if two different users have a high degree of overall connec-

tivity, we may wish to cluster them. This resembles the clustering and recommendation problem addressed in YouTube [3], which again exploits link analysis.

Our interest in these problems is motivated by application scenarios in which we must rank the influence and overall utilization of data and/or code. In the **ieeg.org** project [13] for sharing neuroscience data and tools, we seek to rate user contributions based on their overall impact. In the TrustForge project (`rtg.cis.upenn.edu/TrustForge`) we seek to rank code modules for trustworthiness, largely based on how frequently the code modules are incorporated and tested in other projects. Ideally, we would like to be able to rank of modules and users, as well as cluster them, in both a general and a context-sensitive way.

In this paper, we argue that the most natural approach is to exploit the structure of a provenance graph to rank and recommend "interesting" or "relevant" items to users, based on provenance (sub)graph structure and **random walk-based algorithms**. Random walk algorithms has been well-studied over Web and social network graphs, but to our knowledge have not been considered in the provenance space. We further argue that it is desirable to have a high-level **declarative language** to extract portions of the provenance graph and then apply the random walk computations. We build upon the ProQL provenance query language and make the following specific contributions:

- We propose the use of link analysis via *random walks over provenance graphs*, as a way of measuring impact or similarity. (Section 2.)
- We extended the ProQL provenance query language to support a wide array of random walk algorithms. (Section 3.2.)
- We suggest avenues of exploration for optimizing the computation of such queries. (Section 3.3.)

2 Ranking and Recommendation

Many methods have been proposed for ranking and making recommendations in graphs. The most popular schemes, and the ones we focus on, are based on the notion of a *random walk* in a graph, with Google's PageRank [15] as the most popular example. Space constraints prevent a full survey of the literature, but we briefly summarize the key ideas, starting with the PageRank algorithm and then discussing a number of popular variants.

The majority of link analysis algorithms are based on the notion of a "random surfer" who starts at a random node in the graph, and then randomly follows outgoing links to other nodes (with some probability of instead jumping to another random node). The PageRank of a node n corresponds to the proportion of the time that the random surfer spends at n in the limit. Its computation can be expressed iteratively using matrix operations.

Suppose we are given n pages $p_1 \dots p_n$ where each p_i has $L(p_i)$ outgoing links. Initialize a vector representing PageRank at iteration 0, $\mathbf{R^0}$, to have the value $1/n$ in each element. Then we iteratively recompute $\mathbf{R^i}$ for iteration i, given the previous value $\mathbf{R^{i-1}}$, a *decay factor* d, a column vector of ones $\bar{1}$, and a matrix $\mathbf{M}$ where $\mathbf{M}_{ij}$ is $1/L(p_j)$ if page j links to page i, and 0 otherwise. Then:

$$\mathbf{R^i} = d\mathbf{M}\mathbf{R^{i-1}} + \frac{1-d}{n}\bar{1}$$

The PageRank algorithm, with minor variations, has been extended to other domains, such as ObjectRank [2] for interconnected objects, XRank [9] for XML, and TrustRank [10] for computing trustworthiness.

We summarize a number of popular enhancements to the basic problem formulation.

Biased starting points. PageRank assumes that the random surfer has an equal probability of starting at (or randomly jumping to) any node. Variations allow for a nonuniform probability distribution across nodes [16].

Topic-sensitive or personalized PageRank. Work such as that of [11] computes PageRank starting the random walk from a subset of the nodes.

Nonuniform transfer of score to connected nodes. ObjectRank [2] allows for edges of different types to propagate different amounts of weight.

Normalization or threshold. Robust PageRank [1] limits the amount of PageRank any single node can accumulate, by thresholding it at a particular level after every iteration. Algorithms such as label propagation (described below) re-normalize the total accumulated value at the end of each iteration.

Label propagation. More complex algorithms based on label propagation, like adsorption [3, 17], generalize PageRank. Instead of assuming a stationary distribution, they assume a set of *start nodes* and create a *label* for each. Each random walk step "propagates" some weight for each label at the previous node. Adsorption computes, for every node n in the graph, a distribution across labels specifying the proportion of the time a visit to n originated at each given start node s. If a large ratio of the paths originating from s pass through n, n will have a high score associated with s's label. This forms a means of *clustering nodes*, and thus making recommendations.

Our goal is to develop a high-level framework for performing all of the above computations. We wish to perform this over graphs that represent provenance as opposed to Web graphs. In some cases we may wish to *project* out portions of the provenance graph and only operate over those. This suggests building random walk capabilities over a provenance query language such as ProQL [12], supplementing its capabilities for extracting provenance subgraphs with new constructs for controlling weight assignment and propagation.

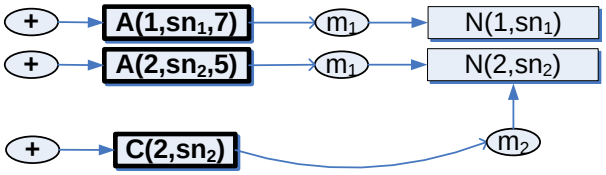

Figure 1: Example provenance graph.

3 ProQL Language & Model

We briefly summarize the existing ProQL model and language, of [12], to set the context for our extensions. ProQL operates over *provenance graph* representations of semiring provenance. A provenance graph includes *tuple nodes* representing the tuples in the database instance, and *derivation nodes* representing direct derivations (*immediate consequents* in Datalog parlance) of tuples from other tuples. Tuple nodes can only be connected via edges with derivation nodes: an edge from a tuple node to a derivation node represents the *use* of the tuple in a derivation, and an edge from a derivation node to a tuple node represents the result of the derivation.

Example 3.1. *Suppose we have two base relations A and C (representing animals and their canonical names) and a view N relating the three sources of data A, C, and N (representing animals, common names, and scientific names) that are related as follows:*

$m_1 : N(i,n)$:- $A(i,n,x)$
$m_2 : N(i,n)$:- $C(i,n)$

An example provenance graph shown in Figure 1. Tuple nodes are rectangles and derivations nodes are ellipses (labeled with the names of the queries, or "+" if the data was directly inserted).

The graph shows the derivations of tuples directly in terms of one another. For a tuple node in the provenance graph, its *alternate direct derivations* are captured by the set of derivation nodes that have directed edges pointing to it. (These represent unions.) In turn, each derivation connects a set of source tuples that are joined — the set of tuple nodes with edges going to the derivation node — and a set of consequents — the set of tuple nodes that have edges pointing to them from the derivation node.

3.1 Basic ProQL Language

A ProQL query takes as its input a provenance graph G, like the one of Figure 1. We adopt a path expression syntax where the individual "steps" consist of traversals from a node representing a tuple in a relation, through a node representing a derivation through a query, to another node representing a tuple. Within the path expression, we may restrict the tuple nodes to belong to a certain relation, or the derivation nodes to belong to a certain mapping. We may also bind variables to either type of node. Within a path expression, tuple nodes are specified using the form `[`*relation-name variable*`]`, where both *relation-name* and *variable* are optional. Derivation nodes are specified using one of the three forms:

`<-` | `<` *query-name* | `<` *variable*

A derivation node (e.g., representing the results of a join in a view) may connect multiple tuple nodes. ProQL also supports Kleene-closure operators "+" or "*" on edges.

Directly borrowing the conventions of XQuery, a ProQL query may have a **for** clause indicating which variables (prefixed with the `$` character) to bind to edges or nodes in a list of paths. A **where** clause allows for filtering conditions over variables.

Construction of returned results in ProQL is divided into two parts. A *graph construction* clause, **include path**, specifies which nodes, edges, and paths to copy to an output graph. However, the user may wish to iterate over multiple portions of this output graph — for this we also support a **return** clause that returns tuples of bindings to nodes and/or edges in the graph. To help make this concrete, we show a brief example.

Example 3.2. *Given the setting of Figure 1, return the subgraph containing all derivations of tuples in* `N` *whose ID has value 1:*

```
for [N $n]
where $n.id = 1
include path [$n] <-+ []
return $n
```

Note the use of the path wildcard (`<-+`) specifying all paths from all nodes that derive any `$x` *node.*

This core language is sufficient to project out portions of a provenance graph. We now need extensions for random walk algorithms. Our first step was to increase the set of variable types allowed in the language, to include collection-types such as node sets, paths (lists of node sequences), and edge sets; and correspondingly to allow for set-membership and aggregation functions over collections. These are necessary for reasoning about, e.g., how to divide a PageRank node weight evenly across the set of all outgoing edges.

3.2 Random Walks and Query Support

Our initial approach to supporting graph random walk algorithms over provenance graphs was to extend ProQL to full Turing-completeness with arbitrary recursion and creation of annotations. However, this makes the query harder to optimize and gives the programmer a fairly low-level abstraction. Hence, we instead propose a set of language extensions for random walks that closely align with the matrix-based specifications of the computation.

Node distribution table. We start by allowing the programmer to specify one or more tables giving a probability for visiting each node, optionally for each label. This captures the notion of the bias table.

Transfer table. Similarly, the programmer can specify a table that resembles the weight transfer matrix in PageRank, in mapping how much weight to transfer from one node to another in each traversal. This can be extended to support different transfers for different labels.

Iterative computation. We develop a **repeat..until** construct in which the main random walk computation is repeatedly applied. Optionally this may include two steps, one for propagation and a second for adjustment of weights (e.g., normalization or thresholding).

Propagation and adjustment. The **propagate** clause uses the node distribution and transfer tables to propagate weight from one node to another, with a specifiable decay factor. The **adjust** clause allows the weights on each node to be scaled (even across labels) or thresholded.

Termination condition. We can specify that the computation should run until the amount of weight change is under a threshold.

Figure 2 in Appendix A shows an example of the Robust PageRank algorithm (which thresholds the rank that any node may accumulate) in ProQL and Figure 3 shows the adsorption label propagation algorithm. A major difference between the two is the node distribution table: for PageRank we assign a uniform weight, and for adsorption we assign a value of 1.0 to two different labels at two start nodes, leaving everything else blank. Weight transfer for PageRank has a decay factor, whereas for adsorption labels and weights are simply propagated.

3.3 Optimization Opportunities

Our high-level formulation of the random walk computation simplifies optimization in a distributed setting. The node distribution and transfer tables can be easily partitioned alongside the nodes to enable efficient local computation. Moreover, given that the random walk algorithms have convergence guarantees, we can actually compute over different parts of the graph at different rates, e.g., if some node weights stabilize early. As described in our recent work on cluster-based query processing in the REX system [14], the system can "focus" its communication and computation on the parts of the data that actually change.

4 Conclusions & Future Work

The problem of making recommendations or assigning scores based on *influence* or *common connectivity* is essential to a variety of collaborative scenarios. We propose that a natural way of performing such computations is via random-walk algorithms, which can be applied over the provenance graph. We have developed a number of extensions to the ProQL language to specify a broad array of different random-walk algorithms in a high-level way. These high-level specifications are also amenable to distribution and optimization, which we are currently studying.

Acknowledgments

This work was funded by NSF grants IIS-1050448, CNS-071541, and CNS-1065130, and DARPA grant HR-011-11-C-0096.

References

[1] R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, K. Jain, V. Mirrokni, and S. Teng. Robust pagerank and locally computable spam detection features. In *AIRWeb*, 2008.

[2] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, 2004.

[3] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. Video suggestion and discovery for YouTube: taking random walks through the view graph. In *WWW*, 2008.

[4] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[5] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[6] J. N. Foster, T. J. Green, and V. Tannen. Annotated XML: queries and provenance. In *PODS*, 2008.

[7] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007. Amended version available as Univ. of Pennsylvania report MS-CIS-07-26.

[8] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[9] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, 2003.

[10] Z. Gyöngyi, H. Garcia-Molina, and J. O. Pedersen. Combating web spam with TrustRank. In *VLDB*, 2004.

[11] T. H. Haveliwala. Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search. *TKDE*, 15(4), 2003.

[12] G. Karvounarakis and Z. G. Ives. Querying data provenance. In *SIGMOD*, 2010.

[13] B. Litt and et al. The international epilepsy electrophysiology portal. `www.ieeg.org`.

[14] S. R. Mihaylov, Z. G. Ives, and S. Guha. Rex: Recursive, delta-based data-centric computation. In *Proc. VLDB*, 2012. To appear.

[15] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[16] M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in PageRank, 2002.

[17] P. Talukdar and K. Crammer. New Regularized Algorithms for Transductive Learning. In *ECML/PKDD (2)*, 2009.

[18] P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. G. Ives, F. Pereira, and S. Guha. Learning to create data-integrating queries. In *VLDB*, 2008.

A Appendix

```
populate table bias(toNode,weight)
  let $v = []
  for [$n]
  insert ($n, 1 / count($v))
populate table transfer(source,
    destination, ratio)
  for [$n] <- [$n2]
  let $outEdges = [] <- [$n2]
  insert ($n, $n2, 1/count($outEdges))
initialize
  for [$n]
  annotate $n with pr = bias($n)
repeat
  for [$n] <- [$n2]
  propagate $n.pr to $n2.pr using decay =
       0.85 [transfer($n,$n2), bias]
 then
  for [$n]
  adjust $n.pr threshold min(bias($n),
      0.5)
until empty (
  for [$n]
  where change($n.pr) > 0.1
  return $n
)
```

Figure 2: Robust PageRank in ProQL+

```
populate table bias(toNode,label,weight)
  for [$n]
  insert ('start1', 'label1', 1)
  insert ('start2', 'label2', 1)
populate table transfer(source,
    sourcelabel, destination,
    destinationlabel, ratio)
  for [$n] <- [$n2], $l in labels($n)
  let $outEdges = [] <- [$n2]
  insert ($n, $l, $n2, $l, 1/count(
      $outEdges))
initialize
  for [$n]
  annotate $n with pr = bias($n)
repeat
  for [$n] <- [$n2], $l in labels($n)
  propagate $n.$l to $n2.$l using
      transfer($n,$l,$n2,$l)
 then
  for [$n], $l in labels($n)
  let $labels = (for $l2 in labels($n)
      return $n.$l2)
  adjust $n.$l scale 1/(sum($labels))
until empty (
  for [$n], $l in labels($n)
  where change($n.$l) > 0.1
  return $n
)
```

Figure 3: Label propagation in ProQL+