# Addressing Underspecified Lineage Queries on Provenance

Daniel Margo, Peter Macko, and Margo Seltzer

TR-01-12


Computer Science Group
Harvard University
Cambridge, Massachusetts

# Addressing Underspecified Lineage Queries on Provenance

Daniel Margo
Harvard University
Cambridge, Massachusetts
dmargo@eecs.harvard.edu

Peter Macko
Harvard University
Cambridge, Massachusetts
pmacko@eecs

Margo Seltzer
Harvard University
Cambridge, Massachusetts
margo@eecs

## ABSTRACT

State-of-the-art provenance systems accumulate data over time, creating deep lineage trees. When queried for the lineage of an object, these systems can return excessive results due to the longevity and depth of their provenance. Such a query is underspecified: it does not constrain its result to a finite span of history. Unfortunately, specifying queries correctly often requires in-depth knowledge of the data set.

We address the problem of underspecified lineage queries on provenance with techniques inspired by Web search. We present two metrics, SubRank and ProvRank, that measure the frequency of a particular result across the space of all possible lineage queries. We then use these metrics to define a subset of the lineage with which to respond to a query. These metric-defined result sets closely approximate a user's conceptual view of relevant history. We evaluate our techniques on diverse workflows ranging from Wikipedia revision data to fMRI processing.

## 1. INTRODUCTION

Provenance is metadata that describes the history of a digital object: where it came from, how it came to be in its present state, who or what acted upon it, etc. The study of provenance is an emerging research field with applications in diverse areas such as computational science and trust. Provenance increases the value of its corresponding objects; for example, the results of an experiment are more valuable if their provenance shows how they were obtained. Provenance is often represented as a directed acyclic graph in which the nodes are objects and the edges are historical relationships. This metadata can be orders of magnitude larger than its corresponding data; the MiMI protein structure data set [5] contains 270MB of data and 6GB of provenance.

An important class of queries on provenance are *lineage queries* of the form, "What is the lineage of this object?" Such a query can be represented as a graph traversal starting from the object in question and passing through the objects that participated in its creation. Unfortunately, a simple lineage query can potentially return the entire depth of the provenance graph. For example, the first query of the first and second provenance challenges [9, 15] was, "Find everything that caused Atlas X Graphic [an object node] to be as it is." A pedantic system might respond with the complete history of the operating system: "Three years ago, you installed Linux..." Though this result is *literally* correct, the majority of it is unlikely to be useful or otherwise relevant. Conversely, a curt response such as "Atlas X Graphic came from the `convert` program" is relevant, but incomplete.

The fundamental problem is that this lineage query is *underspecified*: it does not constrain its result to a finite span of relevant history. A "good" result should constrain itself to exactly that part of the lineage in which the user is implicitly interested, but the query does not convey this constraint. Instead, it is only the limitations of the provenance system that constrain the result at all. An omniscient system might respond, "Thirteen-point-seven billion years ago, there was the Big Bang..." Provenance systems that collect constrained data necessarily return constrained query results, but as the data grows the system must inevitably decide how to respond meaningfully to underspecified queries. While it is tempting to simply require more precise queries from users, one should recall that provenance data is diverse, large, and accumulates with time. Correctly specifying such queries demands unsustainable knowledge about the details, history, and structure of large and often technical data sets.

We address this problem with techniques inspired by Web search, a field in which underspecified graph queries are of fundamental concern. We hypothesize that the relevance and importance of a given result depends on several properties; in particular, its frequency within the space of all possible lineage query results. For example, the provenance of the operating system is not often relevant because it is present in most objects' lineage; its presence is obvious and trivial. Conversely, a result that is unique to one object's lineage is both highly descriptive of that object and most likely unknown to the user. Frequency within query results is a function of graph and query structure, and Web search offers us tools to assess and quantify this property.

In this paper, we introduce the problem of underspecified lineage queries and explore a solution. We present two metrics, SubRank and ProvRank, which measure the frequency of an object across the space of all possible lineage queries. We incorporate these metrics into an algorithm that constrains lineage queries to an appropriate span of history. Our approach relies entirely on graph topology and is widely applicable; we demonstrate its effectiveness on diverse work-

*Harvard School of engineering and Applied Sciences* January 24, 2012
Harvard TR-01-12.

flows ranging from Wikipedia revisions to fMRI processing.

This paper is structured as follows. In Section 2 we identify related work. Section 3 models the underspecified lineage query problem. Section 4 introduces our metrics and their usage. We discuss our evaluation in Section 5, future work in Section 6, and conclusions in Section 7.

## 2. RELATED WORK

The underspecified lineage query problem is not yet fully recognized. One system that experiences this problem is the Provenance-Aware Storage System [10], which collects provenance at the Linux system call level. PASS also provides an API for applications to report provenance to the PASS database; thus, PASS can aggregate provenance from many sources. At this scale, underspecification becomes a serious concern. Other provenance systems such as ES3 [3], Panda [4], and VisTrails [2] are substantially different from PASS and from each other, but all of them encounter this problem at sufficient time scales.

The most common provenance access and query methods are path-oriented query languages such as SPARQL, XQuery, the Provenance Query Protocol [7], PQL [14], or programmatic traversals such as neo4j [11]. In such languages it is the responsibility of the user to form a precise query that will return the desired result. However, on a large, diverse data set the identification and specification of appropriate constraints requires in-depth knowledge of the data set and query locale. Ontologies such as the Open Provenance Model [8] and Provenir [17] define a standard representation that can make query writing easier for users, but only if the data is already in that representation and the query can be easily expressed in terms of its elements. The purpose of our research is to develop widely-applicable methods to appropriately constrain results without relying on user input, a formal ontology, or other external knowledge of the data set.

In prior work, Ré and Suciu [16] identified the problem of unconstrained lineage in probabilistic databases (a subset of provenance). They addressed this problem by omitting objects that did not significantly contribute to results. However, their concept of a "contribution" is tightly linked to their domain, in which lineage is used for forensic inference and lineage relationships are probabilistic. In this context one can make assertions about an object's probabilistic contribution to its resulting inferences; however, this method does not obviously apply to general provenance.

Underspecified graph queries are an important subject of research in the field of Web search. Algorithms such as PageRank [13] and Kleinberg's HITS [6] use graph structure to make probabilistic assertions about the relevance of pages with respect to searches. In particular, PageRank simulates an agent searching the Web graph via random link traversal; the PageRank of a page is the probability that it is visited at a random step in this traversal process. Traversal processes are typical in graph query and are the basis of many provenance access methods. We use a similar approach to simulate the behavior of the class of lineage queries; to the best of our knowledge, this is a novel application of such simulations. PageRank and HITS can both be calculated from the eigenvectors of transition matrices and belong to a field of mathematics known as spectral graph theory.

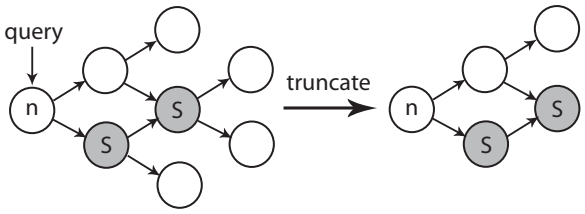


**Figure 1: Lineage traversal is truncated at nodes in $S(n)$ (gray nodes).**

## 3. PROBLEM STATEMENT

We begin by formally stating the underspecified lineage query problem and outlining our solution. Intuitively, whenever the user asks a lineage query there is a set of provenance objects within its result that the user considers irrelevant, obvious, or otherwise "unworthy" of further provenance description. In response to such a query we would like to determine those objects and return a query result that terminates at them. Thus, our result will be constrained within the region of worthwhile objects.

Provenance graphs can be represented in many ways. We assume the representation is a directed acyclic graph in which the nodes are provenance objects and edges point from newer objects to their historical predecessors. This means edges point in the direction of history, but opposite to the direction of data flow. Let $G$ be such a provenance graph. Let $L(n)$ be the connected subgraph of $G$ that contains all nodes reachable from node $n$: that is, the complete lineage of $n$. Let $S(n)$ be the aforementioned set of "unworthy" nodes in $L(n)$. Our goal is to find $T(n)$, a connected subgraph of $L(n)$ such that:

$$T(n).nodes = L(n).nodes - \{L(s).nodes - s \ \forall \ s \in S(n)\}.$$

That is, $T(n)$ contains all of $L(n)$ except the provenance of objects in $S(n)$ (though it includes those objects themselves). We call $T(n)$ the *truncated result of* $L(n)$ (see Figure 1 for an example).

It should be clear that $L(n)$ is just the transitive closure of $n$, and $T(n)$ is just a traversal of $L(n)$ that halts at $s \in S(n)$. The problem of finding transitive closures in graph query is both elementary and well-studied, and while important to lineage query in general, is not the focus of our research. Rather, we are concerned with the problem of finding the set of "unworthy" objects $S(n)$, which represents the missing constraints on an underspecified lineage query.

## 4. METRICS AND USAGE

We propose that "unworthy" objects in $S(n)$ satisfy the following two properties: *high frequency* and *frequency dissimilarity*. We begin by discussing these properties in detail. We then present two metrics that measure frequency, and an algorithm to find $S(n)$ that exploits these properties.

**High Frequency.** Unworthy objects appear in the results of too many lineage queries. At one extreme, everything is a consequence of the Big Bang (or first OS install), so its inclusion in a query result provides no additional information. At the other, an object present in only one lineage uniquely identifies that lineage and is also likely to be novel to the user. This use of frequency is reminiscent of the information retrieval concept of inverse document frequency:

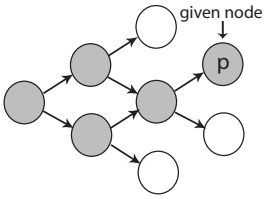


**Figure 2: SubRank. There are 5 lineages containing $p$ (including $p$ itself). If the user asks for one random $L(n)$, then the probability that query returns $p$ is 5/8.**

the significance of a provenance object is diminished if that object is frequent within the corpus of lineage query results.

**Frequency Dissimilarity.** Not only is the absolute frequency of an object important, but its frequency relative to the objects around it is also important. For example, while the Big Bang is unlikely to contribute meaningful information to most lineage results, it may be the only relevant member in a query for the lineage of the universe. In a different context, when querying for the lineage of a core system utility such as `mkdir`, the operating system is more relevant than when querying for the lineage of a user's document. Frequency dissimilarity (or similarity) is a function of a particular query locale and exploits our knowledge of the particular query that the user issued.

Our hypothesis is that high frequency and frequency dissimilarity are good measures of the importance of results to a given query. If an object is rare *with respect to the query locale*, then it is important and worthy of inclusion in lineage query results; otherwise it is "unworthy". We can measure these properties without further input from users, because they are entirely functions of the graph structure and query.

We now turn the problem of measuring object frequency in the space of lineage query results.

## 4.1 SubRank

One way to measure frequency within the space of lineage query results is directly: that is, for each provenance object, simply count the number of lineages in which it appears. Let $D(p)$ be the set containing $p$ and those objects for which $p$ is in their lineage, defined recursively maas:

$$D(p) = p \cup \{D(n) \mid (n,p) \in G.edges\}$$

Then, the *SubRank of* $p$ is defined as:

$$SubRank(p) = \frac{|D(p)|}{|G.nodes|}$$

That is, $SubRank(p)$ is equal to the "reverse" transitive closure of $p$ (i.e., the closure formed by following edges in their reverse direction), normalized by the size of the graph. This is equal to the probability that, if one were to query for one random $L(n)$, $p$ would be in that $L(n)$ (see Figure 2).

The problem with this method is that $|D(p)|$ can be inconvenient to measure. Depending on the provenance graph's size and means of storage, the "reverse" transitive closure of $p$ may take a long time or otherwise be inconvenient to compute. This is especially true if one wishes to precompute $|D(p)|$ for the entire graph. While computing transitive closures on graph databases is a well-researched problem, it is also recognized as a nontrivial one; we would like to find a faster way to measure result frequency.

## 4.2 ProvRank

An alternative to measuring frequency is via theoretical simulation; that is, to construct and solve a model of query behavior. One classic example of this approach is PageRank [13], in which a theoretical agent queries the Web by clicking random links. This process is represented by a transition matrix that can be solved to find the probability that the agent is at a given Web page at a random instant. This probability reflects the frequency with which each page is visited by the agent and is thus a measure of the page's frequency within the space of random browsing.

Similarly, ProvRank is a measure of an object's frequency in the space of random lineage queries. It reflects the frequency with which an object appears in the result of a random lineage query, in which the user begins at some node and computes its transitive closure, thereby traversing recursively through all objects that participated in its creation. These queries occur on acyclic provenance graphs and therefore terminate; consequently, query restarts (called "teleportation" in PageRank) are needed only when encountering a node with no outgoing edges. This process can be modeled as a transition matrix, which we construct as follows:

Let M be a transition matrix such that:

$$M[n][p] = \begin{cases} 1, & \text{if } (n,p) \in G.edges; \\ 1/|G.nodes|, & \text{if } \nexists (n,v) \in G.edges; \\ 0 & \text{otherwise.} \end{cases}$$

That is, there is a 100% probability of transitioning from a node $n$ to the objects that participated in its creation and an equal probability of transitioning from a node with no such objects to any other node, thereby "restarting" the query. The matrix's principle left eigenvector represents the probability that each node is considered at a random instant in a random lineage query process of infinite duration. We call the entry in this eigenvector corresponding to node $n$ the *ProvRank of* $n$.

In practice, ProvRank, like PageRank, is efficiently approximated by simulating many transitions, rather than by explicitly solving for its eigenvectors. With each step of the simulation, each node adds its current value to that of its neighbors. Contrast with SubRank, whose cost is proportional to the size of each node's reverse transitive closure.

## 4.3 Usage

After computing either metric on a graph, we determine the contents of a truncated lineage $T(n)$ as follows: we traverse the untruncated lineage $L(n)$ starting from node $n$, following only edges with a small difference in the ranks of the two nodes (see Figure 3). That is, we follow an edge $(u,v)$ only if $Rank(v) - Rank(u) \leq threshold$ for some predetermined threshold, the choice of which we will describe in the rest of this section. $T(n)$ is the set of visited nodes.

According to the second property of "unworthy objects", *frequency dissimilarity* (see Section 4), objects are most relevant within a locale of similar objects. We should be less concerned with the absolute frequency of nodes and more with their frequency relative to their surroundings. Thus, rather than placing a threshold on the ranks of nodes, we threshold the relative change in rank when crossing edges.

A well-chosen threshold should partition $L(n)$ into distinct sets of relatively low and high-frequency objects, the former of which is $T(n)$. One general technique is to sam-

ple a range of thresholds and search for an "exceptional" one. Starting from zero, we increase the threshold and count how many nodes are assigned to $T(n)$. In practice the size of $T(n)$ does not increase at a steady rate, but rather in jumps. This is because the traversal will often encounter an especially frequent object (such as the operating system) that represents an unprecedented increase in frequency over previously seen objects. To incorporate such an object into $T(n)$, the threshold must increase proportionally; if it does so, it may not only incorporate the object in question but many others as well, resulting in a jump in the size of $T(n)$. In practice this behavior limits our selection of meaningful thresholds to a small number of discrete choices, each corresponding to such a jump. See Figure 4 for a visual example, and Figure 10 in our evaluation for a real-world example.

We can test all possible thresholds in one pass over $L(n)$ as follows. Starting with a threshold of zero, we traverse the lineage of $n$ until we encounter an edge from $u$ to $v$ such that $Rank(v) - Rank(u) > threshold$. At that point we record the current threshold and the size of $T(n)$, increase the threshold to $Rank(v) - Rank(u)$, and continue. In order to detect all possible thresholds, we traverse deterministically in the order of ascending differences in node-to-node rank. We continue until we have traversed all of $L(n)$, at which point we have a complete list of thresholds and their corresponding $|T(n)|$'s.

We then choose the most "exceptional" threshold from this list. Currently, we default to the threshold that corresponds to the largest relative jump in $|T(n)|$, but the user is free to choose any other from the list. Our experience suggests that different thresholds correspond to different "resolutions" of provenance and are valid for different purposes. For example, we might observe that one threshold corresponds to the start of a single experiment, a second might correspond to a collection of similar experiments, and a third to the compilation of tools used by those experiments. Consequently, the threshold selection scheme has an impact on the practical uses of the resulting truncation. Investigating this behavior and the properties of different thresholding schemes is an important part of our future work.

## 5. EVALUATION

Evaluating underspecified query results is inherently subjective because one must make qualitative claims. Therefore, we have identified several provenance data sets in which a "good" truncated result is relatively well-defined. These include a Wikipedia revision history of the "Barack Obama" page and two traces collected by PASSv2 [10]: the first provenance challenge [9] and the compilation of the BSD automounter utilities (`amutils`) v6.1.5 [1].

For each data set, we construct a hand-tuned "oracle" that defines a good truncated result for a subset of lineage queries. We then evaluate our methods on those queries with respect to each oracle. We define the "goodness" of a particular result $T(n)$ to be the absolute difference between the cardinality of that result and the oracle's result for object $n$. We use this criterion because it allows us to compare the relative sizes of $T(n)$, the oracle, and their difference with respect to the total sizes of objects' lineages and the provenance graphs.

We implement both metrics and our thresholding algorithm in C++ and Python. We compute SubRank using the Floyd-Warshall transitive closure algorithm [18]. To compute ProvRank, we iteratively approximate the principle left eigenvector of the graph's $|G.nodes| \times |G.nodes|$ transition matrix with a typical PageRank-like score-pushing loop. We evaluate on an Intel Core2 Duo 2.8GHz system with 4GB of RAM, running 64-bit Ubuntu 10.04 Linux v2.6.32-24.

```
queue ← [(0, n)]
T ← empty_set()
threshold ← best_threshold(n)
while queue is not empty do
  (δ, u) ← queue.pop()
  T.add(u)
  if δ ≤ threshold then
    for v in u.parents do
      queue.insert((v.rank − u.rank, v))
    end for
  end if
end while
return T
```

**Figure 3: The algorithm for computing $T(n)$, the truncated lineage.**

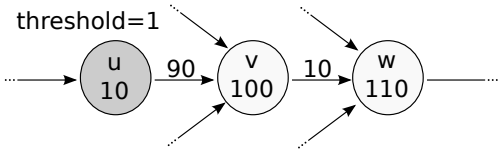


**Figure 4: Choosing a threshold. Suppose we are at $u$ with a threshold of 1. Consider the transition $u$ to $v$ where $Rank(v) - Rank(u) = 90$. If we raise our threshold to 90, we will not only add $v$, but also nodes beyond $v$ such as $w$. Thus, there are a finite selection of thresholds, each associated with a "jump" in result size. We choose the threshold just prior to the largest relative jump in size.**

## 5.1 Wikipedia Revisions

We created a provenance graph of Wikipedia's "Barack Obama" article [12] from its revision history as follows. Using the first 19,560 revisions of the article, we determined the inter-revision text dependencies of each revision by building a word-by-word `diff` patch for each revision. The nodes in our graph are thus revisions, and the edges are direct text

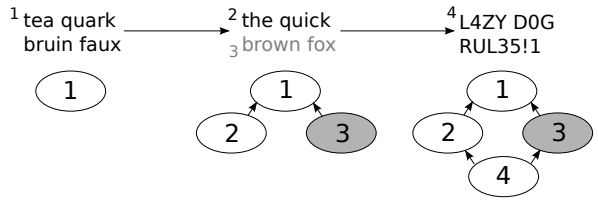


**Figure 5: Wikipedia revision provenance begins with a node corresponding to the original document (1). Later revisions (2,3) depend on the text they modify. If vandalism destroys the text (4), then the vandalism depends on all previous revisions and all subsequent revisions must depend on the vandalism.**

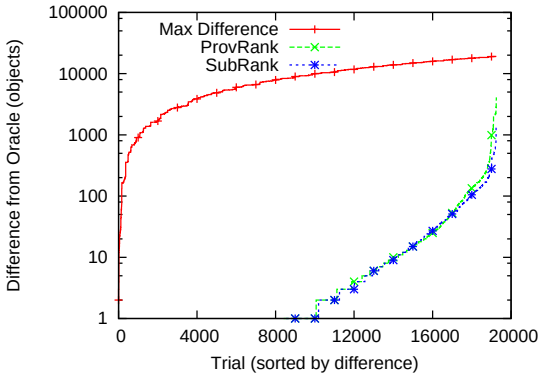


**Figure 6: Difference from the oracle for 19,251 queries on the "Barack Obama" Wikipedia revision history in sorted order, log scale.**

dependencies (see Figure 5). The graph consists of 19,560 nodes and 49,713 edges.

Analysis of the article's edit history reveals that text is initially created in large segments (such as sections and paragraphs), then edited iteratively in increasingly finer detail. This proceeds until a large edit rewrites most (if not all) of the page. Afterwards, all future edits proceed from the most recent large edit; relationships across large edits are trivially implied. Consequently, large edits separate regions of normal editing activity, forming natural truncation points. In the case of the "Barack Obama" page, the overwhelming majority of large edits are due to vandalism or reversions; we can approximately identify both by searching for the phrase "revert" in editors' comments. Thus, our oracle defines a good result as truncating at all revisions prior to the most recent "revert". "Reverts" are sufficiently common such that this oracle is defined for all but 309 nodes.

Figure 6 plots the difference in cardinality from the oracle for the following three cases: truncation using SubRank, truncation using ProvRank, and the maximum error scenario (which corresponds to either no truncation or immediate truncation, depending on the size of the oracle's result). The maximum possible error is large for all queries, because revision lineages are deep, but reversions are common. In constrast, our algorithm behaves identically to the oracle on about half the queries, and closely on many more.

Across all queries, the average difference between the oracle and our thresholding is 23.99 and 45.06 nodes using SubRank and ProvRank, respectively. However, mean averages are of course misleading: 81.62% and 82.39% of queries are below the mean, and mode averages are both just one node. If we were to discard the top 5% of high-error queries from SubRank and ProvRank, then the average difference would become 10.66 and 11.94 nodes, respectively. In short, much of our error is caused by a small number of pathological cases; overall our methods are quite comparable to the oracle, especially with respect to the size of objects' lineages.

Our methods work well on this data set because of the depth of the graph and its iterative structure. When our threshold selection algorithm encounters the first transition from a small edit to a large edit, there is a jump in relative frequency because large edits, unsurprisingly, have more resulting sub-edits. If the algorithm were to accept this jump as the new threshold, then it would also have to accept every similar jump in history: consequently, it would be forced to add many "large edit cycles" to $T(n)$, exploding its size. As intended, the algorithm rejects this scenario and truncates at the first small-to-large edit transition. In summary, the structure of the Wikipedia data set maps very well to the scenario for which our algorithm was designed.

## 5.2 The First Provenance Challenge

The first provenance challenge [9] is a simple fMRI processing workflow proposed by the members of the community at the International Provenance and Annotation Workshop in 2006. For our purposes, the challenge is noteworthy because it is associated with a conceptual graph of the workflow proposed by its authors (available online at the citation, and reproduced in our appendix for convenience), which we can directly use as our "oracle". Any node not featured in this drawing is outside of the authors' conception of their workflow and therefore a good truncation point for lineages of the workflow's contents.

We ran from 1 to 30 repetitions of the challenge workflow on a PASSv2 [10] kernel, which captures provenance automatically. The resulting provenance contains much of the history of the system in addition to the workflow, including the lineages of the executables used in the workflow – such as the compilation of the Automated Image Registration (AIR) suite. The graph of one run consists of 8,038 nodes and 32,337 edges, including the compile. For comparison, the conceptual graph has only 45 nodes.

If we collect provenance on only one run of the workflow, our algorithm can do very little to truncate the query results. For one run, the truncated query results using SubRank and ProvRank differ from the oracle by an average of 100.2 and 42.98 objects, respectively. The reasons for this high error are demonstrative of the meaning, limitations, and strengths of our methods. We observed that the differences between our methods and the oracle were not erratic, but very consistent, because our methods truncate almost every query at the same place: a set of header files used in the toolchain compile. Excluding that compilation, the difference between the truncated result and the oracle would only be about 15 nodes for both metrics.

The problem is that when both the compile and experiment are run only once, they form competing thresholds. The input files of the experiment produce many more files and therefore are highly frequent relative to their locale; however, the header files of the compile are also input files that produce many more files and have similarly high frequency. The inputs of the experiment are thus not as prolific as the inputs of the compile; consequently, when searching for the best threshold, our method chooses the inputs of the compile. There is nothing in our single-run setup that privileges the experiment over its compile; in fact, one could reasonably argue that if the tools are used in only one experiment, then their construction *is* "part" of the experiment.

In a more realistic setup, the compile would be run only once (or infrequently), but experiments involving the tools might be run many times. We investigated how this would affect our results by running a series of trials in which we compiled the toolchain only once, but executed the experiment repeatedly. Figure 8 shows the difference from the oracle for SubRank and ProvRank on an increasing number of experiment repetitions; we also plot the scenario exclud-

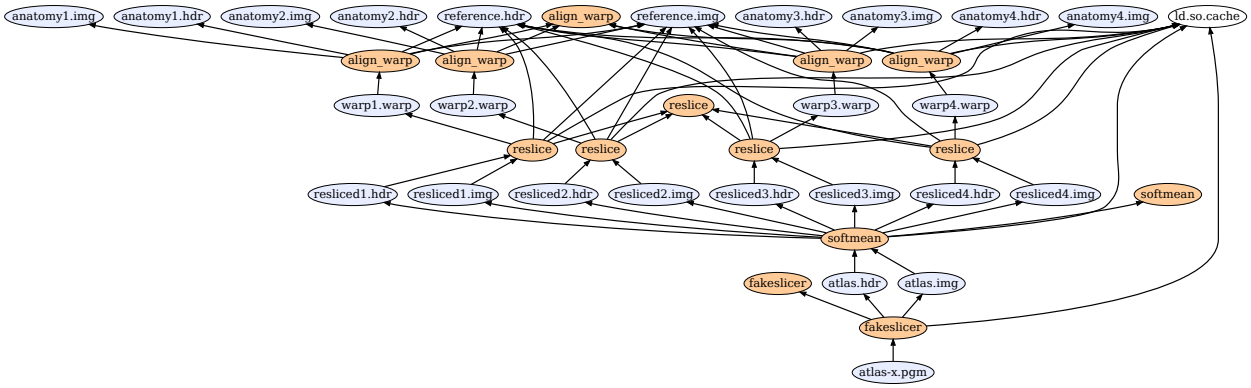


**Figure 7: Truncated provenance of atlas-x.pgm, the final output of the challenge, in a 2-repetition workflow using SubRank with a threshold of 0.0015. Colored nodes correspond to those in the authors' conceptual representation, which is reproduced in the appendix for convenience. Although this threshold emerges after 2 repetitions, it is not favored as the "best" threshold until 10 repetitions.**

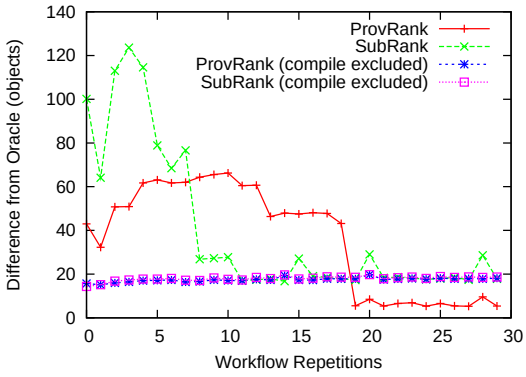


**Figure 8: Average difference from the oracle versus repetitions of the experiment for the first provenance challenge. In the "compile excluded" scenarios, the toolchain compile is excluded from the provenance data set. About about 10 repetitions, SubRank begins truncating out the compile and stopping at the same place as if the compile were excluded. After about 20 repetitions, ProvRank begins truncating close to the oracle itself.**

ing compilation (that is, only the provenance of the experiment). We observe that thresholding with SubRank converges to the scenario in which we exclude compilation after 10 iterations, while ProvRank converges to the oracle (the author's conceptual drawing) after about 20 repetitions.

In short, thresholding can truncate quite accurately on this data set, given certain assumptions. In particular, when choosing between high-frequency thresholds, there should be a structural reason to pick one over another. If both the compile and experiment are run only once, there's no reason to privilege the inputs of the experiment over those of the compile; the authors choose the experiment, and our methods choose the compile. However, in the more realistic scenario where the experiment is run more frequently than the compile, the experiment is the structurally-preferred choice and our methods converge with the authors' intent.

Note that a threshold corresponding to the authors' intent very clearly emerges with just two repetitions of the experiment (see Figure 7 for a compelling visual example). It is simply not the "best" threshold by our current definition. A different threshold selection scheme, or a user presented with a choice of several thresholds, could produce more immediate results. As previously mentioned, this is one example of how different thresholds correspond to different granularities of provenance (experiment vs. compile), and how different thresholds might be valid for different purposes. A computational scientist trying to understand why her results had unexpectedly changed might be more interested in her tools' provenance than a scientist just trying to manage her experimental data. The selection and use of truncation thresholds for different tasks is an interesting topic for future work.

## 5.3 amutils

Our final workflow is a compilation of the BSD automounter utilities (`amutils`) v6.1.5 [1] . This is a large, multi-stage compile that produces many executables; the provenance graph consists of 83,447 nodes and 183,444 edges. While this workflow is fairly heterogeneous, one recurring theme is the transformation of a collection of inputs – libraries, headers, and source files – into a single output executable. These inputs seem like obvious truncation points for a compile; therefore, our oracle truncates the lineages of the outputs of the compile (a total of $3,947$ nodes) at `.so` (library), `.h` (header), and `.c` (source) files.

Figure 9 plots the result size difference from the oracle for truncation using both SubRank and ProvRank and for the maximum error scenario. Across all $3,947$ queries, SubRank and ProvRank average a difference of 29.84 and 39.29 nodes respectively. As in the first provenance challenge, the reasons for this initial error are quite informative. Our threshold selection scheme often truncates too aggressively due to the following two complimentary effects. First, the compiler (`cc`) is a highly frequent object that is always encountered just prior to the input files, forming a competing threshold. Second, though there usually exists a threshold corresponding to the input files, the inputs are not far removed historically from the start of the compile; thus, the jump from truncating at inputs to simply not truncating at all is relatively small.

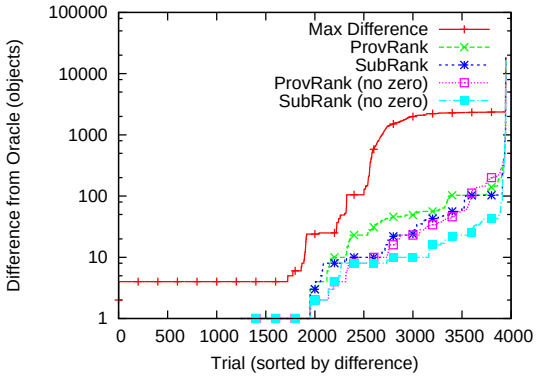


**Figure 9: Difference from the oracle for $3,947$ queries on `amutils` in sorted order, log scale. In the "no zero" case, thresholds of zero are forbidden.**

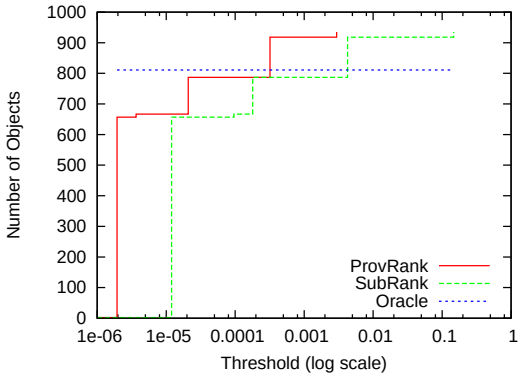


**Figure 10: Threshold versus objects returned for `amd`, an output of `amutils`. SubRank and ProvRank exhibit similar behavior, but at different threshold scales. The "correct" threshold (as defined by the oracle) is present in both, but is not selected because of the large jump between $0$ and the first threshold.**

In particular, the jump in result size from an initial threshold of zero to the input file threshold is often greater than the jump from the input file threshold to no truncation at all (see Figure 10). If we simply declare "never choose a threshold of zero" to reduce aggressive truncation, our results improve significantly. In this case, across $3,947$ queries we average a difference from the oracle of 17.19 and 29.42 nodes for SubRank and ProvRank (see Figure 9). After discarding the top 5% of high-error queries, we average 6.256 and 15.14 respectively. However, "never aggressively truncate" is not a good general policy; for example, on the Wikipedia data set aggressive truncation with small thresholds is often correct.

## 5.4 Discussion

From our evaluation we can make a number of high-level observations. First and foremost: in several scenarios our methods can approximate data set-specific oracles with relatively high accuracy. This is especially significant because our methods are topological and independent of content, whereas our oracles are content-sensitive and abuse external knowledge. In a large, complex, and unfamiliar data set in which a well-specified lineage query is difficult to write, our methods could provide an easy to implement, general alternative. Though SubRank and ProvRank exhibit mild semantic differences on the first provenance challenge, overall they perform similarly; in practice, ProvRank would probably be preferred as it is faster to compute and approximate.

Our methods are most effective on the Wikipedia data set, in which the majority of nodes possess deep and iterative histories. They are also effective on the first provenance challenge when the experiment is repeated many times, but weaken if the experiment is run only once. They are also weak on the `amutils` compile, which is broad and shallow, but improve greatly with a slight thresholding policy change.

We observe our methods work best when there is a large volume of repetitively structured provenance; this is due to two effects. Depth and iteration, as in the Wikipedia data set, helps our threshold selection scheme by creating consequences for setting the threshold too generously; specifically, too many nodes may be added to $T(n)$. Breadth and repetition, as in the first provenance challenge, also helps our threshold selection scheme by elevating the frequency of key truncation points, such as the input files of a repetitive experiment. When provenance is shallow and each workflow is run only once, our methods perform poorly; however, in this scenario truncation is also less important.

It bears repeating that in both the first provenance challenge and the `amutils` compile there usually exists thresholds corresponding to the oracle, even if they are not chosen as the "best" threshold. Different thresholds correspond to different jumps in frequency and result set size, and if the provenance data consists of heterogenous or multigranular subprocesses (e.g., experiments or compile structure), thresholds will often correspond to the beginnings of those subprocesses. This line of reasoning leads to interesting questions about the topology and growth models for large heterogeneous provenance graphs. In the same way that the topology and growth of social networks are well-studied entities, we think that studies of the structure of provenance "in the wild" are a great direction for future research.

# 6. FUTURE WORK

Although we are encouraged by our results, underspecified queries on provenance are still a young problem, and there are many aspects of our approach that are worthy of further investigation. In particular, questions of threshold selection and how thresholds correspond to heterogenous and multigranular processes are important for future research. Different tasks might demand different thresholds, such as a scientist reviewing her experiments versus an IT specialist trying to verify the scientist's toolchain. One could imagine an interactive query system in which the user requests lineage at various discrete levels of detail.

In this paper our approach was to constrain unbounded query results by introducing artificial constraints to truncate the result set. There are other ways in which one might reduce results, such as coarse-resolution summarizations. For example, in the first provenance challenge one might express the provenance of an output as consisting of a "compile" followed by "an experiment", without fine details. Similarly, in the Wikipedia data set one might not want to truncate at recent vandalism, but merely to identify and remove it.

We believe that provenance summarization is closely related to our truncation methods and in particular to thresh-

olding. Our methods can distinguish the compile and the experiment, or the different edit cycles between vandalism, by identifying different discrete thresholds. Frequency metrics allow us to make statements about what constitutes a "fine detail" versus a frequent, general object such as an oft-used executable. In summary, future research into thresholding and the structural properties of provenance graphs may allow us to exploit our metrics in novel ways.

Topological analysis is, of course, only one of many resources one might use to truncate or summarize provenance. Object metadata, content analysis, etc. contain valuable information and are important in application-specific scenarios. PageRank and similar topological metrics are only a few of the tools in the larger toolbox of search engines. Similarly, we offer SubRank and ProvRank not as definitive solutions, but as one set of tools for what we hope will be the broad toolbox of provenance analysis.

# 7. CONCLUSION

Lineage queries are the fundamental class of queries in provenance systems. We have observed that many lineage queries are underspecified. Underspecified lineage queries fail to constrain their results to a finite span of history and are instead constrained only by the limitations of the system.

We addressed this problem by truncating results at frequent objects that are "unworthy" of further lineage description. We developed a framework for this problem and then produced two metrics, SubRank and ProvRank, that measure the relative frequency of provenance objects in lineage query results. SubRank measures the frequency of an object across the space of all possible results; ProvRank measures the frequency of an object within our query model. We then applied a simple thresholding scheme to truncate lineages at the jump between low and high frequency objects.

We evaluated our results on several workloads for which a "good result" was either structurally transparent (Wikipedia), defined by the authors (the first provenance challenge), or semantically transparent (the `amutils` compile). Our results corresponded well with these oracles under many conditions, in particular when a large volume of iterative or repetitive provenance was involved. Opportunities for future work are diverse, but should focus primarily on the relationship between thresholding and heterogenous/multigranular provenance, and growth models for provenance graphs.

# 8. REFERENCES


[1] 4.4BSD automounter utilities. `http://www.fsl.cs.sunysb.edu/docs/am-utils/am-utils.html`, March 2011.

[2] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: Visualization meets data management. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 745–747, New York, NY, USA, 2006. ACM.

[3] J. Frew and P. Slaughter. Es3: A demonstration of transparent provenance for scientific computation. *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop (IPAW 2008), Revised Selected Papers*, pages 200–207, 2008.

[4] R. Ikeda, S. Salihoglu, and J. Widom. Provenance-based refresh in data-oriented workflows. Technical report, Stanford University, October 2010.

[5] M. Jayapandian et al. Michigan Molecular Interactions (MiMI): putting the jigsaw puzzle together. *Nucleic Acids Research*, 35(Database-Issue):566–571, 2007.

[6] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[7] S. Miles, L. Moreau, P. Groth, V. Tan, S. Munroe, and S. Jiang. Provenance query protocol. Technical report, University of Southampton, 2006.

[8] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. den Bussche. The open provenance model core specification (v1.1). *Future Generation Computer Systems*, In Press, Corrected Proof:–, 2010.

[9] L. Moreau et al. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 20(5):409–418, April 2008. Published online. DOI 10.1002/cpe.1233.

[10] K.-K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. Maclean, D. Margo, M. Seltzer, and R. Smogor. Layering in provenance systems. In *Proceedings of the 2009 USENIX Annual Technical Conference*. USENIX, June 2009.

[11] Neo4j: the graph database. `http://neo4j.org`, March 2011.

[12] Barack Obama - Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Barack_Obama`, March 2011.

[13] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[14] PQL - path query language. `http://www.eecs.harvard.edu/syrah/pql/`, March 2011.

[15] The Second Provenance Challenge. `http://twiki.ipaw.info/bin/view/Challenge/SecondProvenanceChallenge`, March 2011.

[16] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *Proc. VLDB Endow.*, 1(1):797–808, 2008.

[17] S. S. Sahoo and A. Sheth. Provenir ontology: Towards a framework for escience provenance management. Microsoft eScience Workshop, October 2009.

[18] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962.

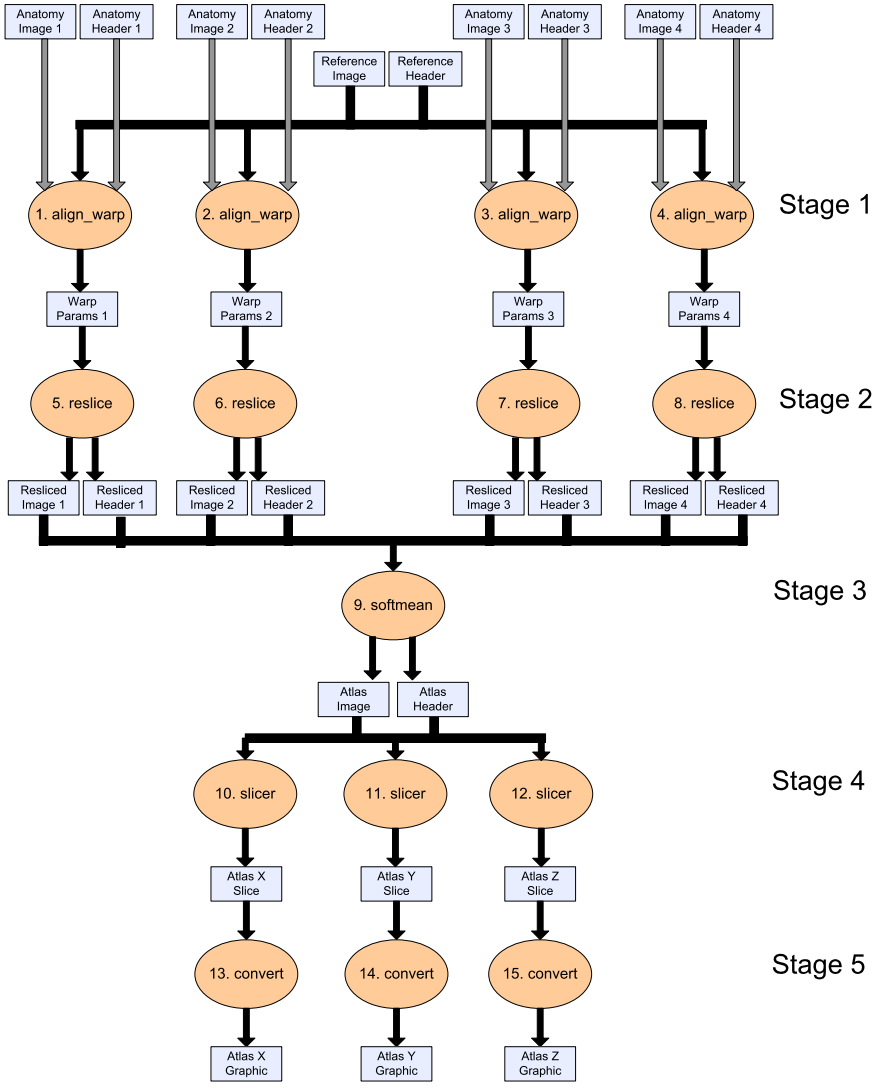


**Figure 11: Appendix. The authors' conceptual drawing of the first provenance challenge.**