Local Clustering in Provenance Graphs (Extended Version)

Peter Macko
Daniel Margo
and
Margo Seltzer

TR-03-13

Computer Science Group
Harvard University
Cambridge, Massachusetts

Local Clustering in Provenance Graphs (Extended Version)

Peter Macko
Harvard University
33 Oxford Street
Cambridge, MA, USA
pmacko@eecs.harvard.edu

Daniel Margo
Harvard University
33 Oxford Street
Cambridge, MA, USA
dmargo@eecs.harvard.edu

Margo Seltzer
Harvard University
33 Oxford Street
Cambridge, MA, USA
margo@eecs.harvard.edu

ABSTRACT

Systems that capture and store data provenance, the record of how an object has arrived at its current state, accumulate historical metadata over time, forming a large graph. Local clustering in these graphs, in which we start with a seed vertex and grow a cluster around it, is of paramount importance because it supports critical provenance applications such as identifying semantically meaningful tasks in an object's history and selecting appropriate truncation points for returning an object's ancestry or lineage. Generic graph clustering algorithms are not effective at producing semantically meaningful clusters in provenance graphs. We identify three key properties of provenance graphs and exploit them to justify two new centrality metrics we developed, specifically for use in performing local clustering on provenance graphs.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering; D.2.8 [**Software Engineering**]: Metrics—*Complexity Measures*

General Terms

Algorithms, Experimentation

Keywords

Provenance, Local Clustering, Centrality, Lineage Query

1. INTRODUCTION

Provenance is metadata that describes the history of digital objects: where they came from, how they came to be in their present state, who or what acted upon them, etc. The study of provenance is an emerging research field with applications in diverse areas ranging from computational science to trustworthiness. Provenance increases the value of the objects it describes; for example, the results of an experiment are more valuable if their provenance shows how they were obtained. In computational science, there is strong interest in reproducibility, which is ultimately about capturing sufficient provenance to permit future reproduction.

Despite this interest there is still much work that needs to be done to make provenance practical. For example, it is surprisingly difficult to obtain a meaningful answer to the query, "How was this object produced?" Since provenance accumulates over time, after a few weeks or months of repeated experimentation, the ancestry of an object (which is the answer to this query) can become long and complicated. Existing solutions return either the immediate ancestor, which frequently lacks complete context, or the entire history, which can be so long as to be overwhelming. We address this issue using *local clustering*, identifying cluster boundaries to produce results that map to conceptual tasks.

Provenance forms a directed acyclic graph in which the nodes are entities and processes that act on these entities, and edges are historical relationships. The edges are usually directed from outputs to inputs [16]. For example, if a user read `data.in` to produce `graph.eps` using `/usr/bin/gnuplot`, the provenance would contain an edge from `graph.eps` to `/usr/bin/gnuplot` and another edge from `/usr/bin/gnuplot` to `data.in`, signifying that the resulting file was written by the `gnuplot` process, which in turn read `data.in`.

We call the query that asks how an object was produced a *lineage query*. Such queries are evaluated by starting at a *query node* (e.g., `graph.eps`) and returning the subgraph containing all the ancestors of the *query node*. Lineage queries are the most important practical use of provenance; they explain how a given object was produced and what steps a user can take to reproduce it. Existing provenance systems usually return either the immediate ancestors or the entire ancestry, but returning a semantically meaningful cluster of nodes around the query node is vastly preferable.

Clustering is one of the most well-studied problems in data mining, but there exists little work directed at clustering in provenance graphs, and many existing techniques do not work well with provenance. For example, we initially tried Markov Chain Clustering [23], but it produced clusters that were not semantically meaningful to our users. Likewise, we had little luck with algorithms that optimize conductance [12] or that primarily take advantage of a node degree, such as when trying to optimize the Cheeger ratio [5]. A study carried out by one of our collaborators [4] revealed the necessity of having a deterministic, semantically meaningful clusters.

We studied many large provenance graphs and identified three important properties that are related to our ability to produce meaningful clusters:

- The ubiquity of a node depends on the node's descendants, not its ancestors.
- A provenance graph frequently has many distinct regions, produced by different collections of tools and at different granularities. This variability makes the concept of distance between two nodes complex.
- Provenance exhibits strong temporal relationships.

We leverage these properties to the task of local clustering

in provenance graphs. We use local clustering to answer lineage queries by starting with a *query node* and growing a cluster around it until a "stopping" condition is met. We focus on local clustering, rather than on general clustering, due to its relationship to lineage queries and the fact that provenance graphs can be large – even orders of magnitude larger than the data being described. For example, the MiMI protein structure data set [11] contains 270 MB of data and 6 GB of provenance.

We explore two kinds of local clustering methods: offline metrics that require precomputation and real-time metrics that do not. The former is feasible for static graphs that fit in main memory and/or are easily parallelizable, while the latter is useful for large or dynamic graphs. We acknowledge that our set of clustering algorithms is not exhaustive, but instead representative of methods that show promise and current limitations that might inform future work.

The contributions of this paper are:

- Identification of properties of provenance graphs that are useful for clustering.
- Introduction of two centrality metrics particularly appropriate for provenance graphs.
- Adaptation of two existing centrality metrics for provenance and their novel application to provenance graph clustering.

The rest of this paper is organized as follows. After discussing applications of local clustering in provenance, we introduce provenance graphs and identify their key properties in Section 2. We describe our approach to clustering in Section 3, our metrics for the use with clustering in Section 4, and an algorithm for detecting good cluster boundaries in Section 5. We evaluate our approach in Section 6, discuss related work in Section 7, and conclude in Section 8.

1.1 Applications

Local clustering in provenance graphs has two important applications, both of which are related to lineage queries: *task identification* and *constraining underspecified lineage queries*.

Task identification.

Users frequently want to identify the (semantically meaningful) tasks that occur over time. For example, executing the First Provenance Challenge [15] workflow involved several subtasks: the installation of some tools, compilation of others, and running a few instances of an fMRI image processing analysis. While each subtask has many steps, it is useful to provide users an intuitive overview of the ancestry in terms of these high-level tasks, presenting the workflow in terms of clusters of steps.

Constraining underspecified queries.

A related challenge occurs when users issue lineage queries. Continuing the example from above, one of the queries in the First Provenance Challenge asked for the ancestry of a particular image file produced by the fMRI analysis. This is an example of an *underspecified query* – does the user want to know the immediate ancestor, the last task, or the history from the beginning of time? Expert users who are familiar with their workflow and provenance system can explicitly adjust their queries and expectations to produce results they understand. However, users that expert probably don't need to issue the query in the first place; other users need a mechanism to produce a meaningful answer to the most fundamental of provenance queries. A good solution to task identification is likely to be useful in addressing this challenge as well.

2. PROVENANCE GRAPHS

Provenance forms a directed acyclic graph, in which the vertices represent *provenance objects* and the edges represent *dependency relationships*. What constitutes an "object" or a "relationship" is largely defined by the semantics of the system that captured the provenance. Usually the objects can be categorized as passive and active, where active objects act upon passive objects and/or communicate directly with other active objects. Dependencies can be categorized as data or control dependencies. For example, scientific workflow engines frequently represent provenance in terms of input data, output data, and processing objects with edges representing information flow between them. A relational database represents provenance in terms of tuple objects with edges representing relational operations.

We are agnostic with regards to the particular semantics associated with objects or edges, but assume provenance that adheres to the following principles. Objects are always versioned, either explicitly or implicitly. That is, if process A writes to file B and subsequently reads file B, there is no "chicken and the egg" paradox between A and B, because A read a version of B and created a newer version of B using its write. In most provenance systems, each write creates a new provenance object, related to the original object only by its lineage. Provenance systems that allow modifications to existing objects check each such modification to see if the resulting graph would contain a cycle; when it does, the system automatically breaks the cycle by creating a new version of the necessary object(s). Thus, relationships form an acyclic graph, as they would otherwise express a dependency paradox.

Provenance relationships are always directed. Broadly, this means that the child *depends on* its parent. Although the literature is inconsistent in the directionality of edges in provenance graphs, we consistently express relationships as being directed *from child to parent*, i.e., from outputs to inputs. Thus, our graphs express the *depends on* relationship rather than a *produced* relationship. The *depends on* representation permits immutability of nodes in the graph and is simpler to implement. The implication of this representation is that the *lineage* of an object is naturally expressed as the entirety of the graph reachable from that object.

2.1 Properties

There are three properties of provenance graphs that distinguish them from other natural graphs for which generic graph clustering techniques were developed.

The ubiquity of a node is a function of a node's descendants, not its ancestors.

The history of a node can be arbitrarily long, depending on when it was created relative to when provenance collection began – which is itself quite arbitrary. In contrast, the ubiquity or importance of a node is defined by its influence on other nodes. A node that has many descendants is thus important, regardless of when it was created, while a *leaf* node (with no descendants) has minimal influence, even if it

was created when the system was first initialized.

We distinguish between *global* and *local* importance. Globally important nodes influence a significant fraction of other nodes, while locally important nodes have many immediate descendants, regardless of their total number of descendants. For example, consider citation networks as describing the provenance of knowledge: A paper that was cited many times is locally important. A paper that was cited only once is globally important if it inspired another paper that in turn revolutionized the entire field.

We found global importance to be generally more useful than local importance when studying provenance graphs. In either case, the ubiquity, or importance, of a node is a function of descendants, not ancestors.

Variations in granularity.

Different components of a system may capture provenance. Each component selects the semantically meaningful entities for which it records provenance, producing a provenance store with objects of many different *granularities*. For example, the operating system might record provenance on files. In contrast, a database system operates on tuples or records and should record provenance for tuples. Alternately, a build system might represent the compilation of a simple program as a relationship between a source file (`myprog.c`), a compiler (`gcc`), and an executable (`myprog.out`). However, the operating system views this compilation as involving many source files including header files and libraries, multiple programs such as a compiler, assembler, and linker, and a single output file.

There is no agreed-upon granularity in which provenance should be captured, and the granularity depends on the creator of the provenance, the intended applications of the collected provenance, and the overhead that the user is willing to tolerate. The same provenance trace can contain multiple tasks captured at different granularities, so trying to compare, for example, the "distance" between an executable and its source, and the "distance" between the result of a brain imaging workflow and its source data files does not make any sense. While tasks of the same type tend to have comparable granularities, that is not always the case, as we saw in the multiple views of the same compilation process.

This has a profound implication on the notion of distance between two nodes that are related by ancestry in a provenance graph: the number of edges, or any measure that is a function of the number of edges, is not a good notion of distance – and ideally should be ignored or discounted.

The Temporal Nature of Provenance.

Provenance also has a temporal component, and this temporal component is related to the relationships expressed – an object cannot depend on an object that has yet to be created. In related work, we found that users find temporal clustering intuitively appealing [4], but temporal approaches alone frequently fail to separate semantically meaningful tasks that occur at approximately the same time. Such phenomena occur frequently in the presence of certain long-running processes or if the user (or more commonly, a job scheduler) runs a new task immediately after the previous task finishes.

Perhaps not surprisingly, we found that clustering techniques that depend on the graph structure perform better than those that rely only on temporal structure. Although we were able to get good temporal clusters, we found that choosing the right thresholds to do so was quite challenging. While we present results on temporal clustering in Section 6, combining temporal methods with other kinds of clustering techniques is left for future work.

2.2 Implications for Clustering

Nodes that are at the start and/or the end of a semantically meaningful task tend to be "relatively more influential" than the intermediate nodes within the task. For example, consider the central workflow from a brain imaging workload [15] (see Figure 3 in Section 6 for the immediate lineage of one of its output files as captured by PASSv2 [17]). The first stage of the workflow takes four scans of a brain and aligns them to a canonical reference image. The reference image is relatively important, because every result from every such workflow depends on it. In comparison, an individual brain scan influences only its own workflow.

Now consider the provenance of the individual scans: They had to come from somewhere. We observed that data files frequently arrive in bulk, so the download, copy, or extract-from-archive process is also important, because many data files depend on it. This transfer process is the last active node in the "data copy" task.

This implies that if we start at a seed (query) node S and follow its outgoing edges (i.e., towards the ancestors), we expect a large jump in node importance on either side of a task boundary.

3. LOCAL CLUSTERING

In the context of lineage queries, we wish to construct ancestry clusters starting from some seed vertex, S. We call such clustering *local clustering* and define is as follows.

1. Create a cluster that contains only S.
2. Compute "importance" of S using a centrality metric, function $f(\cdot)$.
3. Expand the cluster by adding all immediate ancestors of S where $f(v) \leq \delta$, where δ is a threshold (discussed below).
4. Repeat step 3 recursively for each added node.

The result is a cluster that contains the lineage of S truncated just *before* reaching the important nodes. Alternately, if we want to truncate immediately *after* important nodes, we can add one final step.

5. Add all immediate ancestors of each node in the cluster to the cluster.

Step 5 might introduce a few extraneous nodes. The decision of whether to add step 5 is an explicit trade-off between precision and recall. We will perform this extra step unless stated otherwise, favoring recall over precision.

Intuitively, this approach (including the optional step at the end) corresponds to the scenario in which a user explores the lineage of a query node one neighborhood at the time for each node with which she is not familiar – assuming that she is more likely to be familiar with nodes that are more influential.

We experimented with multiple methods for computing importance; rather than listing them all, we introduce two that are noteworthy: Let:

- S = the seed vertex
- $f(\cdot)$ = a function that returns the value of a metric

Algorithm 1: Local Clustering

Data: S = the seed vertex, $f(\cdot)$ = the metric, δ = the threshold
Result: $\mathcal{C}(S)$ = the local cluster with seed S

```
C(S) ← {S};
Q ← new Queue;
Q.enqueue(S);
while Q is not empty do
    v ← Q.dequeue();
    foreach e in outgoing edges from v do
        u ← the other endpoint of e;
        if f(u) − f(S) ≤ δ then
            Q.enqueue(u);
            C(S) ← C(S) ∪ {u};
if configured to include the influential nodes then
    N ← {};
    foreach v in C(S) do
        foreach e in outgoing edges from v do
            u ← the other endpoint of e;
            N ← N ∪ {u};
    C(S) ← C(S) ∪ N;
```

- δ_1, δ_2 = thresholds

Empirically, we found that the two most suitable (and simplest) criteria for determining when vertex v should be included in a cluster are:

- *Difference in local importance*:
  Traverse edge $u \to v$ if $|f(u) - f(v)| \leq \delta_1$
- *Difference in global importance*:
  Include node v if $f(v) - f(S) \leq \delta_2$

Despite their obvious differences, both produce similar clusters in practice. This is not entirely surprising, because we are looking for relatively large jumps in node importance – and both criteria capture that. When the methods produce different clusters, we find that global importance generally works better. The method that considers only local importance sometimes adds narrow, long strips of extraneous nodes to our clusters. We therefore use global importance in the rest of the paper, unless stated otherwise. Algorithm 1 summarizes our clustering algorithm.

Next, we examine a range of centrality metrics and a simple but effective method for threshold selection.

4. MODELS OF NODE INFLUENCE

Our ability to identify influential nodes is crucial for both applications of clustering described in Section 1.1. We categorize metrics in two ways, first whether they can be computed efficiently in real-time or require offline precomputation and the feature on which the metric is based (e.g., node-degree, closeness, etc.). For each feature, we select representatives, either a commonly-used metric or a custom metric developed specifically for provenance graphs.

Category	Presented Best Metric and Type
Degree	In-Degree Centrality (Real-Time)
Betweenness	Ancestor Centrality (Offline)
Closeness	Opsahl's Closeness Centrality (Offline)
Eigenvector	Provenance Eigenvector Centrality (Offline)

To this collection of structure-based metrics, we add the single semantic metric that captures the temporal relationship in provenance graphs, age.

4.1 Real-Time Metrics

Although our results will show that real-time metrics do not perform as well as offline ones, they are nonetheless important when precomputation is either too time consuming or impractical (e.g., the data changes frequently).

In-Degree Centrality.

Degree centrality is perhaps the simplest centrality metric. However, recall that the ubiquity of a node in a provenance graph depends on its descendants, not its ancestors (see Section 2.1), so we should use *in-degree* rather than the total node degree. This is a measure of node's local importance, and it corresponds to the number of times a node was directly used by other nodes. For example, in the context of file system provenance, in-degree corresponds to the number of times a file was used as input to process. In a citation network, it represents the number of times a paper is cited.

Although in most cases, we apply a threshold to the difference of centrality metrics, for in-degree, we use the value itself as a global metric (0 thus takes the place of $f(S)$):

$$\text{Include node } v \text{ in cluster } \mathcal{C}(S) \text{ if } \text{InDegree}(v) \leq \delta$$

Age.

In prior work [4], we observed that users found temporal clustering useful in understanding large provenance data sets. Although, timestamps do not directly correlate to influence, a node's age is more likely to, because it is a proxy for the length of time during which descendants could be produced. Since age can be expressed as the difference between a node's timestamp and the timestamp of the youngest node in the graph, it satisfies the properties identified in Section 2.1 that we want for a metric. Using timestamps is equivalent to using age: Recall that if we want to use it as a real-time metric, we include node v in a cluster $\mathcal{C}(S)$ built around seed S if:

$$(T_{max} - \mathrm{T}(v)) - (T_{max} - \mathrm{T}(S)) = \mathrm{T}(v) - \mathrm{T}(S) \leq \delta$$

where $\mathrm{T}(\cdot)$ returns a node's timestamp and

$$T_{max} = \max_{x \in V} \mathrm{T}(x)$$

Some provenance graphs place timestamps on edges instead of nodes. In these graphs, we define the timestamp of a node to be the minimum of the timestamps of the outgoing edges (which are the edges that "produced" this node), which is the time when the node was created.

When we consider timestamps intuitively, large "jumps" in timestamps or ages correspond to breaks between tasks. As we discussed in Section 2.1, this metric produces many false positives.

4.2 Offline Metrics

Ancestor Centrality.

Ancestor centrality of a node v is the number of nodes from which v is reachable by following only outgoing edges;

Algorithm 2: Ancestor Centrality

Data: $G = (V, E)$ = a directed acyclic graph
Result: $\mathrm{AC}(v)$ = ancestor centrality for each $v \in V$

```
Q ← new Queue;
foreach v ∈ V do
    // S(v) will be set of all descendants of v
    // L(v) is # of unprocessed children of v
    S(v) ← {v};
    L(v) ← InDegree(v);
    if L(v) = 0 then Q.enqueue(v);
while Q is not empty do
    v ← Q.dequeue();
    foreach e in outgoing edges from v do
        u ← the other endpoint of e;
        S(u) ← S(u) ∪ S(v);
        L(u) ← L(u) − 1;
        if L(u) = 0 then Q.enqueue(u);
foreach v ∈ V do
    AC(v) ← |S(v)|;
    if normalize then AC_norm(v) ← AC(v)/|V|;
```

more intuitively, ancestor centrality is the total number of v's descendants, or the size of a subgraph "below" v. In provenance terms, ancestor centrality is equal to the number of objects on whose lineage v appears.

Let I_{ij} be the boolean value indicating the existence of a (directed) path from i to j (including the case $i = j$). We assign I_{ij} 1 if the path exists and 0 if it does not. Then,

$$\mathrm{AC}(v) = \sum_{x \in V} I_{xv}$$

We can normalize this quantity by the number of nodes $|V|$, so that the normalized ancestor centrality corresponds to the fraction of nodes from which v is reachable.

Ancestor centrality is related to betweenness centrality [8], which is a measure of the number of all shortest paths that pass through the given vertex. While betweenness centrality considers all shortest paths through a given vertex, ancestor centrality considers only directed paths from leaf towards root.

Ancestor centrality can be quickly computed using Algorithm 2. It runs in $O(|V||E|)$, but at the worst-case memory cost of $O(|V|^2)$, which can be alleviated in practice by proactively freeing memory occupied by sets $S(\cdot)$ as soon as it is no longer needed. Developing a memory-efficient algorithm for approximating the metric using Bloom Filters [3] is a part of future work.

Opsahl's Closeness Centrality.

The standard definition of closeness centrality [7] is undefined for graphs with disconnected components. Opsahl [18] proposed the following variant:

$$\mathrm{CC}(v) = \sum_{x \in V \setminus v} d_{xv}^{-1}$$

where d_{ij} is the distance between nodes i and j, which is ∞ when i and j are not connected. To apply it to provenance graphs, we modify its distance metric d_{ij} to follow only outgoing edges. We refer to this modification as d'_{ij}. Hence,

$$\mathrm{CC}'(v) = \sum_{x \in V \setminus v} (d'_{xv})^{-1}$$

This metric is also related to ancestor centrality, which uses I_{xv} instead of $(d'_{xv})^{-1}$. Therefore, it still accounts for the number of edges between v and its descendants, but it is not strongly weighted, as in, for example, Dangalchev's closeness centrality [6], which we have found to work poorly on provenance graphs.

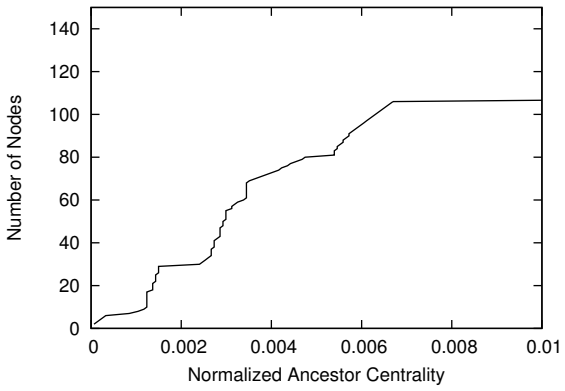

Figure 1: Threshold vs. number of nodes. The number of nodes in the cluster of `atlas-x.gif` from the `chall-mar29-2` dataset for the given threshold using normalized ancestor centrality.

Provenance Eigenvector Centrality.

Another approach to modeling a node's influence is by simulating the process of a lineage query.

Define the following transition matrix:

$$M_{ij} = \begin{cases} 1 & \text{if there is an edge } i \to j \\ 1/|V| & \text{if there is no outgoing edge from } i \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, this is a model of a linage query: When you are at node i, follow all of its outgoing edges; if there are no more edges to follow, start another query from a random node.

Provenance eigenvector centrality (PEC) is the dominant left eigenvector of this matrix. We approximate it using power iteration, which converges even though M is not a stochastic matrix.

5. CHOOSING THE THRESHOLDS

Given a set of centrality metrics our final task is to select appropriate thresholds. If we plot the number of nodes in a cluster as a function of a centrality metric, we find that a number of plateaus emerge. For example, Figure 1 demonstrates this phenomenon using ancestor centrality on a provenance challenge provenance graph. The plateaus indicate "jumps" in node importance that likely correspond to

Algorithm 3: Computing $m_{f,S}(v)$

Data: S = the seed vertex, $f(\cdot)$ = the metric
Result: $m_{f,S}(v)$ = the minimum value of the metric for vertex v to appear in the cluster $\mathcal{C}(S)$

```
Q ← new PriorityQueue;
Q.enqueue(S, priority=f(S));
m_{f,S}(v) ← f(S);
while Q is not empty do
    // Get element with the smallest priority
    v ← Q.dequeue();
    foreach e in outgoing edges from v do
        u ← the other endpoint of e;
        m ← max{m_{f,S}(v), f(u)};
        if m_{f,S}(u) is undefined then
            m_{f,S}(u) ← m;
            Q.enqueue(u, priority=m_{f,S}(u));
```

moving from one semantically meaningful task to another, but they can also include false positives.

The existence of multiple plateaus thus often suggests the existence of multiple task boundaries. We find that choosing the first or the second plateau usually produces the best results for task identification, depending on the metric and on the data set. One could imagine using this local clustering technique as the front-end to a provenance database or visualization system. Users could use "next" and "previous" buttons to visualize larger and smaller clusters that correspond to the detected plateaus.

Automatically detecting the beginning/end of plateaus is tricky. Discontinuities in the metric can vary in size by orders of magnitude depending on their location in the graph. We developed the following threshold detection algorithm, which works well in practice. Let $f(\cdot)$ be the metric function and S the seed vertex. Define $m_{f,S}(v)$ to be the *minimum* value of the given metric that causes vertex v to be included in the cluster. Then:

1. Compute $m_{f,S}(v)$ for every node in the ancestry of S, optionally constraining it by a maximum depth of traversal. See Algorithm 3 for details.
2. Sort the computed values. When plotted against 1 + their index in this list, this produces a plot like Figure 1.
3. Set the minimum jump threshold to the average difference between two consecutive elements in the sorted list, times a parameter α.

The value α is relatively insensitive to the provenance graph and the choice of a metric; we usually use $\alpha = 1$ for everything except age. Automatically determining a good algorithm for use with age is left to the future work; we currently pick the parameter α_{age} manually.

6. EMPIRICAL EVALUATION

We evaluate the use of thresholding and centrality metrics in local clustering to support applications such as task identification and query truncation. We base our experiments on multiple provenance graphs, including the Third Provenance Challenge [21], from the Provenance Aware Storage System (PASS) [17] and the Third Provenance Challenge graph produced by the ProtoProv system [22]. We focus the core of our evaluation on the PASS traces, because they are long-running traces that accumulate multiple diverse tasks. We evaluate our clustering results on a combined task identification and lineage query truncation scenario, which starts with a seed node S and build a cluster $\mathcal{C}(S)$ that contains just the nodes from S's ancestry that correspond to the most recent semantically meaningful task that produced S.

We consider a result successful if a cluster yields high precision and recall using the first or second detected threshold. For each metric and tested scenario, we report the threshold that produced the best accuracy, the number of correct nodes in the cluster, the number of extraneous nodes that should be excluded, and the precision. We do not report recall, because it was 100% in all cases. (As we mentioned in Section 3, we configured the clustering algorithm to prefer recall over precision.)

6.1 Overview of the Provenance Datasets

Dataset	Nodes	Edges	Comp. Time (s)		
			AC	CC'	PEC
`am-utils`	51,358	167,359	20.5	14.6	19.4
`chall-mar29-2`	12,595	40,227	0.7	1.1	3.7
`chall3-failed`	4,286	7,691	0.5	1.4	0.5
`chall3-twc`	63	94	0.01	0.01	0.07

Table 1: Dataset sizes and times to precompute ancestor centrality, Opsahl's closeness centrality, and provenance eigenvector centrality.

We present results from the following four traces:

- `am-utils`: The compilation of the BSD auto-mounter utilities ver. 6.1.5 [2], collected by PASSv2 [17]. It consists of archive extraction, execution of `./configure`, and compilation of each tool.
- `chall-mar29-2`: An fMRI workflow dataset from the First Provenance Challenge [15] from the PASS group, consisting of archive extraction, compilation of four Bioinformatics tools, data copying, and execution of two instances of the fMRI workflow.
- `chall3-failed`: An intentionally failed execution of a scientific database loading workflow submitted by the PASS group to the Third Provenance Challenge [21].
- `chall3-twc`: A complete execution of the Third Provenance Challenge workflow from the Tetherless World Constellation's ProtoProv system (TWC) [22].

In all traces, we consider only those files tracked by the provenance system. Table 1 presents the dataset sizes and times to compute the three offline metrics, on an Intel Core 2 Duo 2.33 GHz, 4 GB RAM, running Ubuntu 12.04 and OpenJDK Java 6. These algorithms have not been tuned; we present the timings to provide the reader a sense of the cost of precomputation.

6.2 Results

am-utils: Compilation of wire-test.

We begin by presenting results for `wire-test`, one of the standard `am-utils` tools, because its compilation is shorter

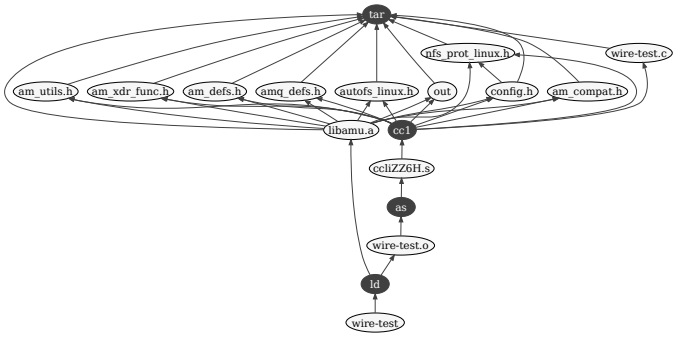

Figure 2: Task that produced `wire-test`. The cluster has perfect recall and one extraneous node – the `tar` process at the top. The result of the compilation is at the bottom. Files are white ellipses; processes are dark gray. An edge represents the existence of a path between its endpoints in the underlying graph, not necessarily that they are directly connected.

Algorithm	Threshold	Correct Nodes	Extra Nodes	Precision
Grnd. Truth	–	17	–	–
AC	1	17	1	94%
CC'	5	17	1	94%
PEC	2	17	1	94%
In-Degree	1	17	1	94%
Age	1	17	0	100%

Table 2: Summary of the clusters seeded by `wire-test` in the scenario excluding the compilation of `libamu.a`. Recall is 100% in all cases.

Algorithm	Threshold	Correct Nodes	Extra Nodes	Precision
Grnd. Truth	–	99	–	–
AC	2	99	1	99%
CC'	7	99	1	99%
PEC	3	99	1	99%
In-Degree	2	99	1	99%
Age	2	99	0	100%

Table 3: Summary for the scenario in which the compilations of both `wire-test` and `libamu.a` are included in the cluster. Recall is 100% in all cases.

than that of most other tools and can therefore be visualized more easily than other, longer executions. We ran the same clustering algorithms on the other, larger workloads and obtained identical behavior.

The complete lineage of the `wire-test` executable consists of 9,502 nodes and 9,852 edges, most of which are due to the `./configure` process before the actual build (this is correctly included in the lineage because it produces `config.h`, an important header file used in the compilation of `wire-test`).

Ground truth can be defined in either of two ways:

1. The compilation of `wire-test` itself from `wire-test.c`, the header files, and the static library `libamu.a`.
2. The compilation of `wire-test` plus the compilation of `libamu.a`, the static library used by all tools in the `am-utils` package.

We evaluate both scenarios: Ideally, each case will correspond to a cluster built around the executable `wire-test` for one of the detected thresholds. We expect in each case that the cluster will contain the lineage up until (and including) the .c and .h files, but that it will exclude the initial archive extraction and the `./configure` process.

Figure 2 displays the cluster for the first scenario, excluding the compilation of `libamu.a`. All metrics except age produce the same cluster. The cluster exhibits perfect recall and only a single extraneous node – the `tar` archive-extraction process on the top. Table 2 summarizes the results. The first discontinuity in ancestor centrality and in-degree centrality produced this cluster, while provenance eigenvector centrality needed the second threshold, and Opsahl's closeness centrality the fifth. Age produces the ideal cluster, matching ground truth, at the first large discontinuity.

Table 3 summarizes the results for the second scenario, where the goal is discover the task including the compilation of both `wire-test` and `libamu.a`. The metrics behave identically.

chall-mar29-2: Lineage of "brain atlas" atlas-x.gif.

Algorithm	Threshold	Correct Nodes	Extra Nodes	Precision
Grnd. Truth	–	38	–	–
AC	4	38	4	90%
CC'	4	38	4	90%
PEC	4	38	14	73%
In-Degree	2	38	30	56%
Age	9	38	4	90%

Table 4: Summary of the clusters seeded by `atlas-x.gif`, one of the results of an fMRI workflow. Recall is 100%.

Having demonstrated the basic experimental methodology, we now move on to the provenance challenge workloads. `atlas-x.gif` is one of the three results of an invocation of the fMRI analysis. A good cluster should include all steps of the workflow up to and including the data files and the canonical reference brain image, but not data copying, archive extraction, or compilation. The entire lineage of `atlas-x.gif` consists of 5,190 nodes and 9,835 edges, most of which correspond to compilation of the tools.

Figure 3 illustrates that clusterings based on ancestor centrality and on Opsahl's closeness centrality both produce ideal or almost-ideal results using the fourth detected threshold. If one considers the tool executables part of the task, then the results are ideal; if not, then the results are near-ideal. The cluster produced corresponds closely to the scientists' own drawing of the workflow [15]. Table 4 summarizes these results numerically.

On this data set, provenance eigenvector centrality performs less well, because it includes several extraneous nodes that are part of the extraction and copying tasks. The extraneous nodes included using in-degree centrality come from practically every other task appearing in the workload, suggesting that in-degree is not effectively at task identification in such a workload.

We can use age to obtain the same near-perfect cluster as using ancestor centrality or Opsahl's closeness centrality, but only at the 9th detected threshold, using the minimum value of α_{age} for "jump" detection. The fundamental

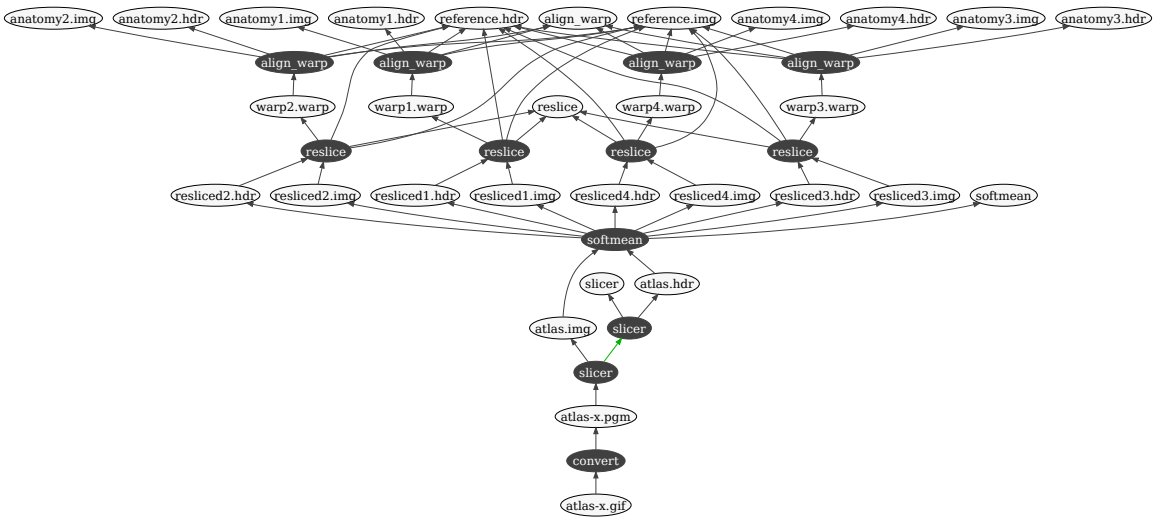

Figure 3: Task that produced `atlas-x.gif`. The cluster has perfect recall, but includes four extra nodes that correspond to the executables of the Bioinformatics tools used by the workflow.

problem here is that the entire collection of tasks were executed by a script, so there are few significant breaks between tasks. We hypothesize that age will be more effective when applied to interactive workloads than to automatically executed workloads.

chall3-failed: Examining the task that failed.

Algorithm	Threshold	Correct Nodes	Extra Nodes	Precision
Grnd. Truth	–	26	–	–
AC	1	26	6	81%
CC'	1	26	6	81%
PEC	1	26	6	81%
In-Degree	1	26	5	84%
Age	1	26	42	38%

Table 5: Summary of the clusters seeded by `fail.log`, which captures information about a failed workflow stage. Recall is 100% in all cases.

Our next experiment examines a case of a failed scientific database loading workflow from the Third Provenance Challenge [21]. This is particularly interesting as it represents one of the most frequently cited uses for provenance: debugging workflows. Access to the lineage of a failed run should provide information useful in determining what went wrong. The failing workflow produced the file `fail.log`.

The overall workflow is driven by a shell script, which invokes each individual stage as `java` or shell script processes and parses and converts files between the different stages. The script calls a `java`-based process that creates three files, `FileEntry[012].xml` which it then processes in order. Each of these files contains the metadata mapping a .csv file to the target database table. The workflow fails after just beginning to process `FileEntry2.xml` and writes out information about the crash to `fail.log`.

The complete lineage of the log file has 310 nodes and 362 edges, most of which correspond to copying the workflow files into the directory from which the workflow is running. The immediate task that produced this file is an instance of `LoadWorkflow.sh` plus all of its inputs.

Figure 4 shows the cluster produced by ancestor centrality, Opsahl's closeness centrality, and provenance eigenvector centrality, each using the first detected threshold. Table 5 summarizes the results quantitatively. The cluster exhibits perfect recall, but only 81% precision. Of the 32 nodes returned three are extraneous nodes related to the operating system's `mount` process and two correspond to copying the main shell script into place. The `java` process that created the three `xml` files corresponds to the end of the previous task; it is again unclear whether it should be included in this task or not; we treat it as extraneous to err on the side of conservatism. This node also accounts for the one node difference between the three offline metrics and in-degree centrality. In this case, regardless of threshold, ages does not produce a good cluster. The resulting graph exhibits low (38%) precision, because the many ancestors of `FileEntry2.xml` were produced in a short succession, only a few hundreds of milliseconds apart.

This is a particularly challenging scenario, because files `FileEntry0.xml` and `FileEntry1.xml` have a large number of descendants (which are correctly excluded from the lineage of our log file), while `FileEntry2.xml` does not, so the metric values that measure global importance of a node (including the time to the end of the trace) are uneven. It is thus remarkable that most of the results are nonetheless quite reasonable.

chall3-twc: Successful loading of a scientific database.

Our final experiment derives from the same database loading workflow of the Third Provenance Challenge [21], but was recorded using TWC's provenance system [22] instead of PASS. Also, this trace is from a completed execution, rather than a failed one. TWC tracks provenance at a much higher semantic level than PASS; the dataset contains only the workflow itself. Copying of the files and all OS activity is thus excluded.

The workflow loads three database tables. Before loading each table, it reads the column names from the correspond-

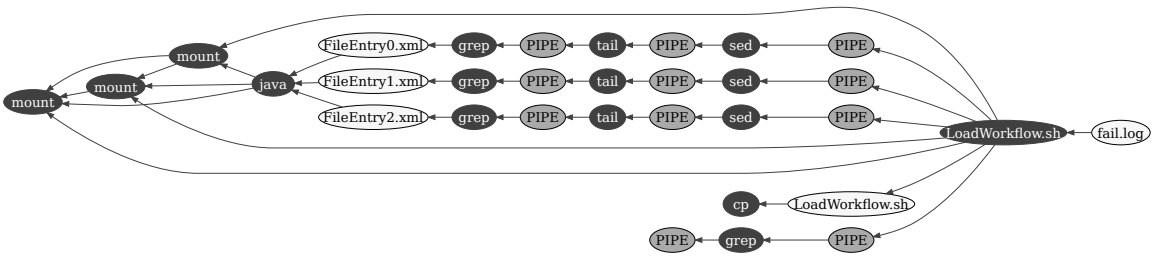

Figure 4: Task that produced `fail.log`. The medium gray ellipses correspond to the UNIX pipes instantiated by the shell script `LoadWorkflow.sh`.

Algorithm	Threshold	Correct Nodes	Extra Nodes	Precision
Grnd. Truth	–	10	–	–
AC	1	10	2	83%
CC'	2	10	2	83%
PEC	1	10	4	71%
In-Degree	1	10	2	83%

Table 6: Summary of the clusters seeded by the final database state from `chall3-twc`. Recall is 100% in all cases.

Algorithm	Threshold	Correct Nodes	Extra Nodes	Precision
Grnd. Truth	–	8	–	–
AC	1	8	2	80%
CC'	1	8	4	67%
PEC	1	8	4	67%
In-Degree	1	8	2	80%

Table 7: Summary of the clusters seeded by the intermediate database state from `chall3-twc`. Recall is also 100%.

ing .csv file and checks that they match the columns in the database schema. The workflow then actually loads the table, updates computed columns, and runs a series of checks on the loaded data to confirm that no errors occurred while loading.

We performed task identification on the final processing steps of two of the three tables; we did not analyze the third table because the provenance graph does not indicate that any tuples were actually loaded in it. The ideal cluster boundary should correspond to the process that loads the input CSV file to the table.

Table 6 summarizes the results which show that all the metrics produced decent clusters. The clusters of all but provenance eigenvector centrality are identical, including two potentially extraneous nodes that are not directly related to loading the data: creating an empty database and reading the column names from a .csv file. The cluster produced using provenance eigenvector centrality included two additional nodes that check that the column names in the .csv match those in the table, which again are objects whose presence in the result set could easily be justified. As this dataset had no timestamps, we could not run the age-based clustering.

The results in the other tested scenario, summarized in Table 7, are similar, except that using Opsahl's closeness centrality includes two extraneous nodes.

6.3 Discussion

We can make a number of high-level observations from these results. First and foremost, we achieve excellent accuracy in task identification, an important application of local clustering in provenance – and thus arguably also in the second application, because task identification can be used to produce good answers to underspecified lineage queries, as explained in Section 1.1.

Ancestor centrality produced the best results, generating ideal or almost-ideal clusters with perfect recall and a minimal number of extraneous nodes. In almost all cases, the extraneous nodes were executables for the process executed by the task; it can be argued that including them is technically correct, although filtering them out would also be relatively straightforward.

Opsahl's closeness centrality produces similar results with a few extra extraneous nodes in a small number of cases, but using it, requires considering more discontinuities in the metric (i.e., we have to look at several different threshold values). As a result, it is more difficult to automatically select the right threshold. Provenance eigenvector centrality produces fewer such jumps, so it is easier to pick thresholds, but its accuracy is lower.

The readily-available metrics, in-degree centrality and age, do not perform consistently; they either work well or very poorly.

Ghosh and Lerman [9] classify node influence metrics as either conservative or non-conservative in their weighting. They claim that one's choice of influence metric should correspond with the underlying processes tracked by the graph. Provenance is often argued to represent information flows or causality, which are non-conservative, this argument would imply we should favor non-conservative influence metrics. This may be one reason we find that ancestor centrality, which is non-conservative, outperforms conservative metrics such as eigenvector centrality.

7. RELATED WORK

Although centrality metrics are well-studied, it is typical to find that the standard metrics are inadequate for a specific domain or application. This can create the need for a novel metric [10] or for a novel comparative analysis of existing metrics in the new domain [13]. Provenance graphs seem relatively unknown in the graph theory community, so little work has been done studying their structure [1]. Our work extends the tradition of centrality analyses to provenance graphs.

Local clustering is likewise a well-studied problem [20], but as mentioned in the introduction, most existing ap-

proaches focus on optimizing metrics that do not work well for provenance graphs. It is also uncommon to study local clustering using metrics that need to be precomputed, which we choose to do because of the nature of our applications. Little work that has been done in clustering provenance graphs, such as by using semantic information [4, 14].

Ré and Suciu [19] addressed unconstrained lineage queries in probabilistic databases, a subfield of provenance. Their method omits objects that do not "significantly contribute" to a result; however, this concept is tightly linked to their domain, in which provenance relationships (edges) are probability-weighted and used for forensic inference. This method does not obviously extend to provenance data in general, in which relationship strength may be more difficult to define and quantify.

8. CONCLUSION

The paper introduced the problem of local clustering in provenance graphs, which has important applications in effective use of provenance. We demonstrate that our new metric for directed acyclic graphs, ancestor centrality (AC), coupled with our threshold detection algorithm works well for provenance applications. We also demonstrated that a straightforward adaptation of closeness centrality and provenance eigenvector centrality work with some degree of success. In the process, we identified the relative strengths and limitations of real-time metrics such as in-degree and age. Facilitating effective provenance query in large and/or dynamic graphs will likely require combining these real-time metrics with other algorithms to obtain adequate behavior when precomputing values is infeasible. Finally, we suggest several avenues for other areas of future research, such as developing algorithms to incrementally maintain offline metrics, incorporating temporal clustering in conjunction with other methods, and developing efficient techniques for approximating centrality metrics.

9. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0937914 and Oracle Corporation. We would also like to thank Michael Mitzenmacher and Matt Welsh for their comments and their helpful guidance.

10. REFERENCES

[1] U. Acar, P. Buneman, J. Cheney, J. Van Den Bussche, N. Kwasnikowska, and S. Vansummeren. A graph model of data and workflow provenance. In *TaPP*, 2010.

[2] 4.4BSD automounter utilities. `http://www.fsl.cs.sunysb.edu/docs/am-utils/am-utils.html`, March 2011.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[4] M. Borkin, C. Yeh, M. Boyd, P. Macko, K. Z. Gajos, M. Seltzer, and H. Pfister. Evaluation of filesystem provenance visualization tools. In *InfoVis*, 2013.

[5] J. Cheeger. A lower bound for the smallest eigenvalue of the laplacian. *Problems in Analysis: Symposium in Honor of Salomon Bochner (1969)*, page 195, 1970.

[6] C. Dangalchev. Residual closeness in networks. *Physica A: Statistical Mechanics and its Applications*, 365(2):556–564, June 2006.

[7] L. Freeman. Centrality in Social Networks: Conceptual Clarification. *Social Networks*, 1:215–239, 1979.

[8] L. C. Freeman. A set of measures of centrality based upon betweenness. *Sociometry*, 40:35–41, 1977.

[9] R. Ghosh and K. Lerman. Predicting influential users in online social networks. In *SNA-KDD: Proceedings of KDD Workshop on Social Network Analysis*, 2010.

[10] W. Hwang, T. Kim, M. Ramanathan, and A. Zhang. Bridging centrality: graph mining from element level to group level. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 336–344, New York, NY, USA, 2008. ACM.

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

[12] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.

[13] E. Le Merrer and G. Trédan. Centralities: capturing the fuzzy notion of importance in social graphs. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, SNS '09, pages 33–38, New York, NY, USA, 2009. ACM.

[14] P. Macko and M. Seltzer. Provenance map orbiter: Interactive exploration of large provenance graphs. In *TaPP*, 2011.

[15] 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.

[16] L. Moreau and P. Missier. PROV-DM: The PROV data model. Recommendation, W3C, Apr. 2013. http://www.w3.org/TR/2013/REC-prov-dm-20130430/.

[17] 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.

[18] T. Opsahl, F. Agneessens, and J. Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32(3):245–251, 2010.

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

[20] S. E. Schaeffer. Survey: Graph clustering. *Comput. Sci. Rev.*, 1(1):27–64, Aug. 2007.

[21] Y. Simmhan, P. T. Groth, and L. Moreau. Special section: The third provenance challenge on using the open provenance model for interoperability. *Future Generation Comp. Syst.*, 27(6):737–742, 2011.

[22] Tetherless world for the third provenance challenge. `http://tw.rpi.edu/wiki/TetherlessPC3`.

[23] S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.