Local clustering in provenance graphs

Local Clustering in Provenance Graphs

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. However, generic graph clustering algorithms are not effective at these tasks. We identify three key properties of provenance graphs and exploit them to justify two new centrality metrics we developed 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*

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.

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 only the immediate ancestors, 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 [14]. For example, if a user read `data.in` to produce `graph.eps` using `gnuplot`, the provenance would contain an edge from `graph.eps` to `gnuplot` and another edge from `gnuplot` to `data.in` (see Figure 1), 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*. Existing provenance systems usually return either only the immediate ancestors or the entire ancestry, but returning a semantically meaningful cluster of nodes is vastly preferable.

Clustering is one of the most well-studied problems in data mining, but there is little work directed at clustering in provenance graphs, and many existing techniques do not work well. For example, we had little luck with Markov Chain Clustering [21], algorithms that optimize conductance [9], and algorithms that primarily take advantage of a node degree, such as optimizing the Cheeger ratio [4].

We addressed this problem by studying large provenance graphs, identifying their properties, and developing new centrality metrics specifically for provenance. Leveraging these metrics, 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.

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

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2505515.2505624.

Figure 1: **Example of a Provenance Graph**. The nodes represent objects and processes; edges represent dependency relationships (opposite of the data flow).

instead representative of methods that show promise and current limitations that might inform future work.

The rest of this paper is organized as follows. After discussing applications of local clustering in provenance, we identify key properties of provenance graphs 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.

Users frequently want to identify the (semantically meaningful) tasks that occur over time. For example, executing the First Provenance Challenge [13] workflow involves 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 with 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? A good solution to task identification is likely to be useful in addressing this challenge as well.

2. PROVENANCE GRAPH 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.

Variations in granularity.

There is no agreed-upon granularity in which provenance should be captured, and different components of a system may capture provenance at 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. The number of edges between two nodes that are related by ancestry, or any such measure, is thus quite arbitrary.

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 [3], but temporal approaches alone frequently fail to separate semantically meaningful tasks that occur at approximately the same time.

2.1 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 [13]. 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 suggests 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 it is as follows.

1. Create a cluster that contains only S.
2. Compute the "importance" of S using a centrality metric function $f(\cdot)$.
3. Expand the cluster by adding all immediate ancestors of S where $f(v) - f(S) \leq \delta$, where δ is a threshold.
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 a 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.

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 automatically 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 second, 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 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), so we should use *in-degree* rather than 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 a process. In a citation network, it represents the number of times a paper is cited.

Although we usually apply a threshold to the difference between centrality metrics of two neighbors, we use the in-degree value itself; thus treaingt it as a global metric rather than a local one:

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

Age.

In prior work [3], 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 that we want for a metric.

When we consider timestamps intuitively, large "jumps" in timestamps or ages correspond to breaks between tasks. This metric produces many false positives when multiple diverse tasks are run in a short succession, especially when executed by a script.

4.2 Offline Metrics

Ancestor Centrality.

Ancestor centrality of a node v 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,

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

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

Ancestor centrality is related to betweenness centrality [7], which is a measure of the number of all shortest paths that pass through a given vertex. While betweenness centrality considers all shortest paths through a given vertex, ancestor centrality considers only directed paths from leaf towards root.

Opsahl's Closeness Centrality.

The standard definition of closeness centrality [6] is undefined for graphs with disconnected components. Opsahl [16] proposed the following variant:

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

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,

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

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 [5], which we have found to work poorly on provenance graphs.

Provenance Eigenvector Centrality.

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

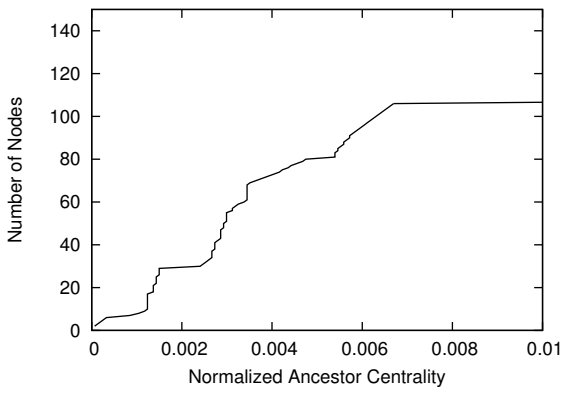

Figure 2: **Threshold vs. number of nodes**. The number of nodes in the cluster of `atlas-x.gif` from the `challmar29-2` dataset for the given threshold using normalized ancestor centrality.

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.

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 2 demonstrates this phenomenon using ancestor centrality on a provenance challenge provenance graph. The plateaus indicate "jumps" in node importance that likely correspond to 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 a 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:

Algorithm 1: 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));
```

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 1 for details.
2. Sort the computed values. When plotted against 1 + their index in this list, this produces a plot like Figure 2.
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. We currently select the parameter α_{age} manually; automatically selecting a good parameter is left as future work.

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 [19] data from the Provenance Aware Storage System (PASS) [15] and the ProtoProv system [20]. We evaluate our clustering results on a combined task identification and lineage query truncation scenario, which starts with a seed node S and builds 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 Provenance Datasets and Seed Nodes

We present results from the following four traces. We specify the seed node for the cluster separately for each trace together with a high-level description of a "good cluster" that

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%

(a) `am-utils`: Compilation of `wire-test`.

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%

(b) `chall-mar29-2`: Lineage of "brain atlas" `atlas-x.gif`.

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%

(c) `chall3-failed`: Examining the task that failed.

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%
Age	(There are no timestamps in the dataset.)			

(d) `chall3-twc`: Successful loading of a scientific database.

Table 1: Summary of the results. Recall is 100% in all cases.

describes the semantically meaningful task that produced the given node.

`am-utils` (51,358 nodes, 167,359 edges): the compilation of the BSD auto-mounter utilities ver. 6.1.5 [2], collected by PASSv2 [15]. The trace consists of archive extraction, execution of `./configure`, and compilation of each tool.
Seed: The executable for `wire-test`, one of the tools in the package. The complete lineage consists of 9,502 nodes and 9,852 edges, most of which are due to the `./configure` process. A good cluster shoulould include only those nodes that are directly relevant for the compilation, but not for `./configure`.

`chall-mar29-2` (12,595 nodes, 40,227 edges): an fMRI workflow dataset from the First Provenance Challenge [13] from the PASS group. The trace consists of archive extraction, compilation of four Bioinformatics tools, data copying, and execution of two instances of the fMRI workflow.
Seed: `atlas-x.gif`, one of the three main results of the workflow. The complete lineage consists of 5,190 nodes and 9,835 edges, most of which correspond to the compilation of the tools. A good cluster should contain only those nodes that are directly relevant to the workflow execution.

`chall3-failed` (4,286 nodes, 7,691 edges): an intentionally failed execution of a scientific database loading workflow submitted by the PASS group to the Third Provenance Challenge [19].
Seed: `fail.log`, which contains the detailed error information. The complete lineage has 310 nodes and 362 edges, most of which correspond to copying the workflow files. A good cluster should contain nodes relevant to the failed stage of the workflow, but no nodes related to copying the files.
`chall3-twc` (63 nodes, 94 edges): a complete execution of the Third Provenance Challenge workflow from the Tetherless World Constellation's ProtoProv system (TWC) [20], which loads three database tables.
Seed: The final processing step of the last table. The complete lineage has 61 nodes and 92 edges. A good cluster should only contain nodes that are relevant to loading this table.

6.2 Results

Table 1 summarizes the results. 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 additional 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.

Please refer to the extended version of this paper for more details [11].

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 [8] or for a novel comparative analysis of existing metrics in the new domain [10]. 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 [18], but as mentioned in the introduction, most existing approaches 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 [3, 12].

Ré and Suciu [17] 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 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, Matt Welsh, and the anonymous CIKM reviewers 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] 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.

[4] 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.

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

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

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

[8] 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.

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

[10] 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.

[11] P. Macko, D. Margo, and M. Seltzer. Local clustering in provenance graphs (extended version). Technical report, Harvard University, 2013.

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

[13] 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.

[14] 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/.

[15] 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.

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

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

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

[19] 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.

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

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