Discovering Similar Workflows via Provenance Clustering: a Case Study

Abdussalam Alawini, Leshang Chen, Susan Davidson, Stephen Fisher*, and Junhyong Kim*

University of Pennsylvania,
Philadelphia, USA
{alawini,leshangc,susan}@cis.upenn.edu,{safisher,junhyong}@sas.upenn.edu
http://www.upenn.edu

Abstract. Several workflow management systems and scripting languages have adopted provenance tracking, yet many researchers choose to manually capture or instrument their processing scripts to write provenance information to files. The Next Generation Sequencing (NGS) project we are associated with is tracking provenance in such manner. The NGS project is a collaboration between multiple groups at different sites, where each group is collecting and processing samples using an agreed-upon workflow. The workflow contains many stages with varying degrees of complexity. Over time workflow stages are modified, but data samples are only comparable when processed with identical versions of the workflow. However, for various reasons (including the distributed nature of the collaboration) it is not always clear which samples have been processed with which version of the workflow. In this paper, we introduce new techniques for clustering provenance datasets and attempt to discover the ones that are likely to be generated by same workflow. Based on the clustering result, users can identify similar provenance and would be able to categorize them into different clusters for debugging and zoom-in/zoom-out viewing.

Keywords: Workflow Provenance · Clustering · Document Classification · Structural Features · K-Means

1 Introduction

Workflow management systems and scripting languages are increasingly being instrumented to capture provenance, however many scientists choose not to use these tools. Instead, they manually capture or instrument their processing scripts to write a certain amount of provenance information to files (e.g. spreadsheets) and use this to enable verifiability and reproducibility. This is the case in the Next Generation Sequencing (NGS) project we are associated with, which is a collaboration between multiple groups at different sites. As sequencing samples are collected within a group, they are processed using an agreed-upon workflow.

* Evolutionary and Molecular Biology (Kim) Lab, University of Pennsylvania

The workflow contains many stages with varying degrees of complexity. Over time workflow stages are modified, e.g. by updating software packages, updating reference libraries or simply changing parameter values. Data samples are only comparable when processed with identical versions of the workflow, however for various reasons (including the distributed nature of the collaboration) it is not always clear which samples have been processed with which version of the workflow. The goal of this work is to determine which samples were processed by which version of the workflow by clustering the collected provenance information.

In this paper, we introduce new techniques for clustering provenance datasets and attempt to discover the ones that are likely to be generated by same workflow. Based on the clustering result, users are able to identify similar provenance, and would be able to categorize them into different subsections for debugging and zoom-in/zoom-out viewing. Our approach takes as input a set of provenance datasets (modeled as PROV-DM graphs), computes abstracted provenance graphs, extracts textual and structural features from each graph, and uses these features to cluster provenance graphs into groups; each group indicates a similar workflow template. We have tested our approach using two datasets, a realistic gene-sequencing dataset and a synthetic dataset, as discussed in Section 4. To enable testing the clustering accuracy, our real dataset was manually labeled by human experts, and for our synthetic dataset we attached a label to each provenance graph that indicates the workflow that was used to generate them. In practice, we assume that no labels are attached.

The remainder of this paper is organized as follows. We introduce the motivation for this problem within next generation sequencing and discuss related work in Section 2. In Section 3, we introduce our provenance clustering framework and discuss our data model. In Section 4 we present our preliminary experimental results, and we conclude and discuss future work in Section 5.

2 Background

We start by introducing our running example and then discuss related work, in particular, techniques related to clustering workflow templates and their provenance.

2.1 Next Generation Gene Sequencing (NGS)

Sequencing is a technique used to decipher the nucleotide code in a strand of DNA or RNA. Next generation sequencing (NGS) is a high-throughput method of sequencing that has revolutionized genomic research over the past ten years. NGS experiments contain complex experimental procedures with extensive processing of the data after the actual sequencing event. Post-sequencing analysis workflows typically have many stages involving different programs, scripts, and reference libraries.

While the post-sequencing workflow is largely automated, due to the multitude of diverse programs and files used in the workflow there are many ways

things can go wrong due to both human and computer errors. The programs and reference libraries used in the workflow are also under active development, and incorporating updates can be problematic. It is therefore critically important to be able to identify when either erroneous programs or reference libraries are used as these errors can have direct effects on downstream, high-level analyses. It is also important to be able to recover from unexpected or unidentified loss of provenance data. Hence it is necessary to be able to identify which pipeline was used to process a given sample which might be missing provenance data. Our provenance clustering approach can help with identifying workflow executions with erroneous or missing provenance as such provenance graphs would not be placed in the same cluster as those with correct and complete provenance information.

2.2 Related Work

There has been work on clustering workflow and provenance graphs using machine learning techniques such as K-Means and hierarchical clustering [4, 11]. Because the accuracy of clustering relies heavily on the effectiveness of extracted features, it is crucial to identify an indicative set of features for workflows and provenance graphs. For workflow templates, clustering is more straightforward as module identifiers can be used to determine the identity of workflow modules and group workflows with similar modules. Santos et al. [11] developed two techniques for clustering workflow templates. In the first technique, they represent workflow graphs using one-hot encoding,[1] whereas in the second they use Maximum Common Induced Subgraphs. [2] Jung et al. [4] use both of these ideas in a two-phase clustering scheme. Since these papers treat workflow nodes as the smallest identifiable and labeled unit, their approaches can only be applied to workflows.

Another related line of work is process mining from log files. Lu et al. [5] studied detecting varying behaviors among executions of processes in the field of business analysis. There has been also work on subgraph mining. Garijo et al. [3] developed an approach that combines exact and inexact graph mining methods to identify the most common sub-graphs in a corpus of workflows.

Unlike workflow clustering, clustering provenance graphs imposes several challenges. Provenance graphs only capture workflow execution information, which makes it very difficult to infer information about the original workflow modules. Additionally, each workflow run may produce somewhat different provenance graphs due to different agents, timing or process id information. In [5], the minimum unit in each execution is labeled atomic event, while the label is not obtainable in our setting. Chen et al. [1] represented provenance graphs as temporal data, which are built from the execution order of the original workflow. They also introduce clustering methods that use temporal and connectivity information, such as the number of incoming edges of a particular type. However,

[1] One-hot encoding is a process in which categorical data is converted into a bit vector.
[2] An induced subgraph is a subset of nodes along with the edges connecting them in original graph.

most of their features were manually selected. Further, the features contain connectivity information, which can only be used for a specific application domain. Clustering provenance graphs requires a set of indicative and domain-agnostic features that allow for clustering provenances regardless of their application domain.

Graph edit distance (GED) has been studied in previous graph matching work [2, 13]. GED is a metric used to quantify the distance between a pair of graphs by counting the minimum number of edge/node operations (insert, delete or update) required to transform one graph into the other. Computing the GED between two graphs is know to be NP-Hard [13]; it has also been shown to be hard to approximate (APX-Hard) [12]. Thus, in this paper, we use other similarity metrics as we discuss in the next section.

3 Clustering Workflow Provenance

In this section, we introduce our provenance clustering framework and data model.

3.1 Provenance Clustering Framework

As shown in Figure 1, our provenance clustering framework consists of four processes: *Compute APG Graphs, Extract Textual Features, Extract Structural Features,* and *Cluster APG Graphs*. First, Compute APG Graphs builds Abstract Provenance Graphs (APGs) from each provenance graph. Then, Extract Textual and Structural Feature processes extract text and structural vectors, which will be used by the clustering algorithm. Next, Cluster APG Graphs uses these feature vectors to approximate similarity between APG graphs and cluster them accordingly. We discuss the details of our framework processes below.

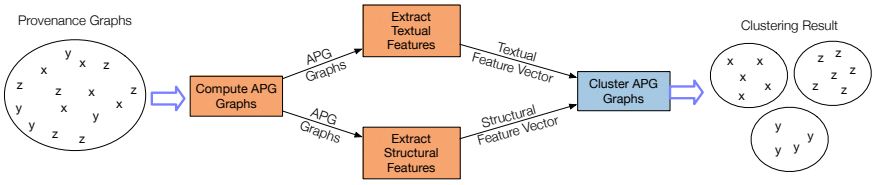

Fig. 1. Graph Clustering Framework

3.2 Data Model: Abstract Provenance Graphs

Provenance data is typically modeled as a graph $G = (V, E)$, where V denotes the set of workflow modules and E denotes the direction of data flow between modules. Specifically, we use the PROV-DM [7] as our data model, but the approach we propose is general and applicable to other formats. PROV graphs

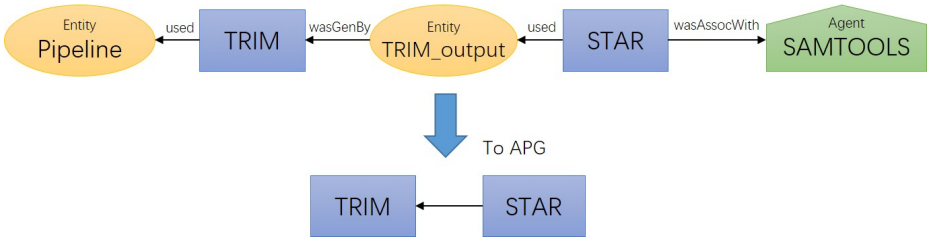

Fig. 2. Example of an Prov Graph and its Abstract Activity Graph

are very large and complex in structure. They store information about agents, entities and activities involved in a workflow execution. To improve clustering performance and accuracy, we summarize PROV graphs by removing certain types of provenance data, such as agents and entities, that are not useful for clustering provenance based on the workflow template.

In this paper, we introduce an approach for converting a PROV graph into an Abstract Provenance Graph[3]. $APG = (V_{APG}, E_{APG})$ is a graph where V_{APG} is a set of activity nodes and E_{APG} is a set of edges connecting them. We focus on analyzing activity nodes because they contain workflow modules (programs, scripts, software libraries, etc.) and execution information (e.g., function calls, execution parameters, software version, etc.) that we can use to reconstruct the modules of the original workflow template.

Because activity nodes in PROV graphs may not be directly connected, our technique also reconstructs connections between activities. For example, two activities could be connected via an entity node (i.e., one activity generated an entity, which was used by another activity). Thus, when constructing an APG graph, we connect two activities if the first activity is connected to an entity node via "GENERATED_BY" edge, and the second activity is connected to the same entity via "USED" edge. An example of a provenance graph and its APG is shown in Figure 2. APG nodes also contain key-value properties containing provenance information of each activity. Listing 1 shows the properties of the "Trim" activity shown in Figure 2.

```
{"name": "kimlab:_225f2b0c-e5bd-4a15-b606-5accc184b26f",
 "attributes":
 { "{.../provDefs/trim#}remove-N": "1",
    "{.../provDefs/trim#}contaminants-file":
        ".../provDefs/trim/contaminants.fa",
    "...":"..." }
}
```

Listing 1. Properties of the "Trim" activity shown in Figure 2

[3] Linking two activities with a common entity is now supported by PROV Constraint 33

3.3 Feature Extraction

Selecting the right feature sets is crucial to the accuracy of any clustering algorithm. Our provenance clustering approach extracts indicative textual and structural feature vectors from APG graphs, and combines these features to calculate similarity between PROV graphs.

Textual Features. To compute text features of APG graphs, we build on ideas from textual cluster analysis (document clustering) techniques [10]. *Term Frequency and Inverse Document Frequency* (TF-IDF) is a widely used technique in information retrieval and text mining for weighting the importance of terms in a document [8]. We use TF-IDF to build a a textual feature vector for each APG graph, as described below.

For each APG graph, we compute a feature vector as follows. First, we extract properties from each activity node. Second, we extract the key part of each key-value property (ignoring values as they change with each run). Third, for each APG graph, we generate a feature vector by computing TF-IDF over the extracted keys. TF-IDF can be computed by the following formula:

$$\text{TF-IDF}(g) = \text{TF}(g) * \text{IDF}(g) \tag{1}$$

$$\text{TF}_{t,g} = 1 + \log(c_{t,g}) \tag{2}$$

$$\text{IDF}_{t,g} = \log(\frac{|g|}{|g_t| + 1}) \tag{3}$$

Here, $c_{t,g}$ is the count of occurrence of word t in graph g. $|g_t|$ is the count of all graphs who contain word t, and $|g|$ is the total number of graphs in the dataset.

Table 1 shows an example of the feature vector extracted from the document shown in Listing 1:

	kim	**bio**	**upenn**	**provDef**	**trim**	**remove-N**	**contaminats-file**	...
Graph1	0.2	0.2	0.2	0.2	0.2	0.1	0.1	...

Table 1. Text features extracted from example graph in Fig. 2

Structural Features. In data mining, *spectral graph analysis* [2] is often used to analyze graph structure. Previous research [9] proposed clustering graphs by structural patterns, where they compute the distance of graphs by finding the sequence of edit operations, and costs are determined by the components of leading eigenvectors of adjacency matrix. The experimental results have shown that the algorithm is a good approximation of distance between graphs [6]. We use this method in extracting structural features from APG graphs.

First, we represent an APG graph $G_k = (V_k, E_k)$ by an adjacency matrix:

$$A_k(i,j) = \begin{cases} 1 & (i,j) \in E_k, \text{ or } (j,i) \in E_k \\ 0 & otherwise \end{cases} \tag{4}$$

where k refers to the k'th provenance graph. To ensure that feature values are real numbers, we convert the provenance graph to be undirected.

Then we calculate the eigenvalues of the adjacency matrix and place them in a vector by descending order.

$$B_k = \text{LeadingEigenvals}(A_k) = [\lambda_k^1, \ldots, \lambda_k^n]^T \tag{5}$$

Here n is the size of the adjacency matrix and λ_k^i is the eigenvalue. In spectral analysis, this vector stands for the major structural information (spectrum, and connectivity) of the provenance graph. Then, we compute Principle Component Analysis (PCA) to reduce the size of the feature space and get a regularized structural feature vector on a common vector space.

Here is an example of the structural feature extracted from an APG graph of 5 activity nodes linked in pipeline structure.

	Largest Eigenvalue	**2nd Largest Eig**	**3rd Largest Eig**
Graph1	1.73	1.0	0

Table 2. Structural features extracted from an example graph

Combining Features. There are cases where two provenance graphs have similar structure, but are generated from two different workflow templates, or visa versa. Thus, we need to combine text features and structural features to avoid clustering unrelated graphs together. To enable generalizability and customizability of our approach, we allow users to assign normalization weights as discussed below. The formula for combining textual and structural features is as follows:

$$\begin{aligned}
\text{Text-Feature}(g) &= \text{TFIDF}(g) = [f_{t_1,g}, \ \ldots \ , f_{t_d,g}] \\
\text{Structural-Feature}(g) &= \text{LeadEigval}(g) = [x_{t_1,g}, \ \ldots \ , x_{t_m,g}] \\
\text{Combined-Feature}(g) &= [\lambda * \text{TFIDF}, \quad \sqrt{1-\lambda^2} * \text{LeadEigval}]
\end{aligned}$$

where λ is normalization factor used to determine feature importance. It is easy to see that if we only want to rely on textual feature, we can set $\lambda = 1$, ignoring structural features as their normalization factor will be set to zero in Combined-Feature. To enable combined textual and structural feature set, we can tune the value as $\lambda = \sqrt{\sum_j {x_{t_j,g}}^2 / (\sum_i {f_{t_i,g}}^2 + \sum_j {x_{t_j,g}}^2)}$ over a small subset of feature vectors g so that both features have approximately equal contribution to the objective function (squared distance) of K-Means (introduced later). Note that we also need to normalize the scale of the two kinds of features to make their average to be at the same level, in case that they are very different.

3.4 Measuring Graph Similarity

The similarity between two graphs is measured by the distance between their corresponding feature vectors. The closer the distance measure between a pair of feature vectors, the more similar their corresponding graphs are. Several similarity metrics can be used. If we naively treat the occurrence of each word independently, then the similarity between documents can be calculated based on *Euclidean Distance* between corresponding points. Another metric is the *Cosine-Similarity* metric, which can be used to compute the angle between a pair of document vectors. A third similarity metric is to compute the correlation between a pair of feature vectors. In Section 4, we report the results of clustering provenance graphs using these three similarity metrics.

Given a pair of feature vectors:

$$\text{Vector}(G) = \text{Features}(G) = [\text{feature}_1, \ldots, \text{feature}_m] \tag{6}$$

We can compute the similarity between a pair of APG graphs (G_1, G_2) as following, meaning that they are negatively correlated:

$$\text{Similarity}(G_1, G_2) \propto 1/\text{Distance}(\text{Vector}(G_1), \text{Vector}(G_2)) \tag{7}$$

3.5 Clustering Algorithm

There are several clustering algorithms, including K-Means, Hierarchical and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). One of the most popular clustering approaches is the K-means algorithm. K-means groups an input set of data points into K clusters so it minimizes the intra cluster distance. K-Means algorithm aims at minimizing the following objective function (squared error function): $J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2$
where $||x_n - \mu_k||^2$ is the distance measure between a data point x_n and the cluster center μ_k. In our approach, we use K-Means with Euclidean, Cosine, and Correlation distance metrics, as we discuss in the next section.

4 Preliminary Experiments

In this section, we report the results of our preliminary evaluation on real and synthetic provenance datasets. We ran our experiments on a machine with a 3.2 GHz Intel i7 processor and 12GB of RAM. We use Python machine learning library, scikit-learn, which includes packages for K-Means, TF-IDF, nltk.tokenize (word tokenizer), and NumPy (scientific computing library) . We also use MATLAB to test K-Means over several distance metrics, as discussed below.

4.1 Provenance Datasets

Real Datasets We compiled a dataset[4] from approximately 1,300 NGS experiments with extensive data provenance describing pre- and post-sequencing

[4] Our dataset is available for download at https://github.com/alawinia/provClustering

events. In particular, this dataset includes experimental samples processed by nine different variants of a post-sequencing workflow used for the primary analysis of NGS data. The analysis workflow includes six possible stages (i.e. Blast, FastQC, Trim, STAR, HTSeq, and Verse) with each sample being processed by three to five of the stages.

Synthetic Datasets To test the performance of our graph clustering algorithm with varying and complex provenance graph structures, we generated a set of synthetic provenance graphs. We modified the structure of a random sample of the realistic datasets we described above by inserting or deleting activities, or adding subgraphs from other provenance graphs, generating about 866 provenance graphs with three different structures.

4.2 Analysis over Real Datasets

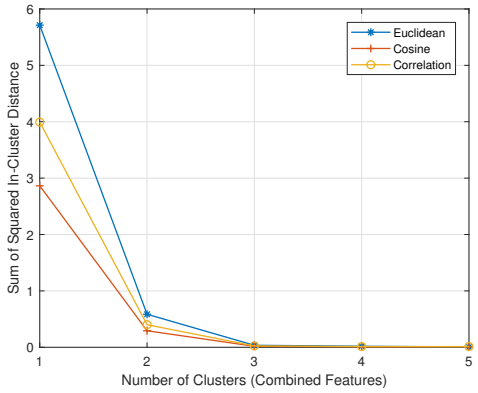

Fig. 3. Squared Error of Clustering over Real Datasets

Actual Cluster	In-Cluster Recall (%)	Data Count
1	92.8	111
2	98.5	199
3	89.7	215

Table 3. Matching of Clusters when k = 3 using Euclidean distance

We first report on experimental results over real datasets. In the first experiment, we evaluated our clustering algorithm on combined textual and structural features using three distance metrics (Euclidean, cosine and correlation). We used the *elbow* method, a technique for determining the optimal number of clusters, by analyzing the change in the sum of squared intra-cluster distance error

(squared error) as a function of the number of clusters. The optimal number of clusters should 1) minimize the squared error; and 2) reduce the number of clusters to avoid overfitting.

Figure 3 shows the result of this experiment. The Euclidean distance works best with our combined features as it assigns larger distance than the cosine or correlation metrics. We also see that the optimal number of clusters for our NGS dataset is 3, as three clusters reduced the error to almost zero. This results matches the golden standard of our real (labeled) datasets. We define the *in-cluster recall* to be the percentage of provenance graphs that are produced by the same workflow template and have been clustered together. Table 3 shows the recall of our clustering technique, which has a weighted average accuracy of 93.7%.

4.3 Analysis over Synthetic Datasets

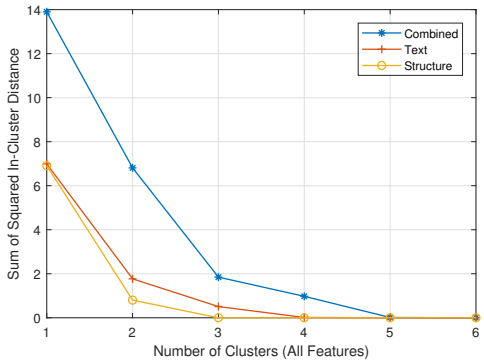

Fig. 4. In-cluster Squared Error over Synthetic Datasets

Cluster	In-Cluster Recall (%)	Data Count
1	100	95
2	100	157
3	91.0	334
4	100	95
5	100	185

Table 4. Clustering at k = 5 using combined features and Euclidean distance

We evaluated our clustering approach over the synthetic dataset using textual, structural and a combination of structural and textual features. Figure 4 plots the sum of intra-cluster squared distance of K-Means algorithm using Euclidean distance with varying number of clusters. The results shows that the optimal number of clusters are 3, 4 and 5 using structural, textual and combined features, respectively. Using combined textual and structural feature vectors, we were able to get the correct number of clusters in the synthetic dataset. Table 4 shows that the per-cluster accuracy of our approach is 96%.

4.4 Running Time Analysis

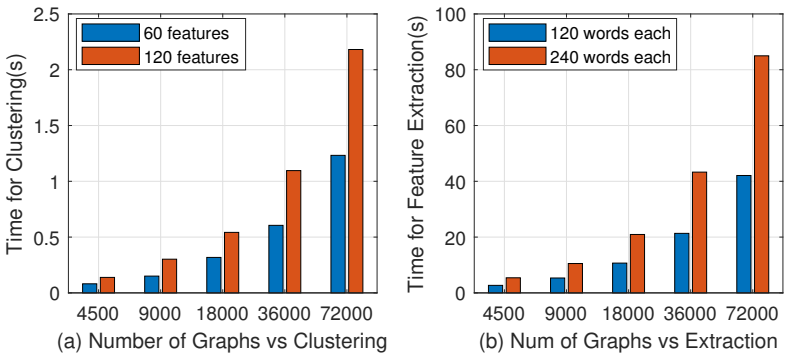

Fig. 5. (a) Time for clustering step vs. the number of graphs; (b) Time for feature extraction step vs. the number of graphs. Note that the number of input graphs increases exponentially, but the running time is linear.

We also test the running time of our provenance clustering approach under large workloads. For this experiment, we developed two synthetic provenance datasets. The first set has about 60 features and 120 words extracted from each graph in the dataset. The second set has about 120 features and 240 words. We evaluated the performance of feature extraction and clustering over different number of samples. The number of samples doubles at each test.

Figure 5 (a) shows the time analysis. We can see that clustering takes less than 2.5s, which is very reasonable. Doubling the number of features (from 60 to 120) increases the processing time by about $1.75x$. As a result, we can infer that our clustering approach is roughly linear in the size of the input.

Figure 5 (b) shows that extracting text features takes more time than clustering. When extracting a large number of text features for a large graph dataset, feature extraction takes up to 85 seconds. However, since features are extracted on a per-graph-basis, we can run text feature extraction for different graphs in parallel. Meanwhile, extracting structural features is really fast, and the time is typically less than 10s. The reason is that the structural information only consists of connectivity, and that various optimization techniques can be used to calculate eigenvalues.

5 Conclusion

This paper introduced a new approach for clustering workflow provenance, enabling effective management and utilization of large provenance datasets. Our approach uses text and structural feature sets extracted from summaries of the provenance graphs. We tested our approach on real and synthetic workloads; preliminary results show an accuracy of over 93% and a running time that is linear

to the size of the input. The textual and structural information are domain-independent, and can be therefore used to cluster any type of provenance graph.

In future work, we plan to develop a visualization technique that uses our provenance clustering approach to enable workflow visualizations that can zoom-out to higher levels of abstraction. We will also explore clustering provenance graphs based on common subgraphs. To do so, we need to implement a fine-grained method for analyzing provenance graphs at the node- and edge-level. We will also explore adding features from entities (such as input and output parameters of workflow modules) and agents.

References

1. P. Chen, B. Plale, and M. S. Aktas. Temporal representation for mining scientific data provenance. *Future Generation Computer Systems*, 36:363 – 378, 2014. Special Section: Intelligent Big Data Processing.
2. X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129, Feb 2010.
3. D. Garijo, O. Corcho, Y. Gil, B. A. Gutman, I. D. Dinov, P. Thompson, and A. W. Toga. Fragflow automated fragment detection in scientific workflows. In *2014 IEEE 10th International Conference on e-Science*, volume 1, pages 281–289, Oct 2014.
4. J.-Y. Jung and J. Bae. Workflow clustering method based on process similarity. In M. L. Gavrilova, O. Gervasi, V. Kumar, C. J. K. Tan, D. Taniar, A. Laganá, Y. Mun, and H. Choo, editors, *Computational Science and Its Applications - ICCSA 2006*, pages 379–389, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
5. X. Lu, D. Fahland, F. J. H. M. van den Biggelaar, and W. M. P. van der Aalst. Detecting deviating behaviors without models. In M. Reichert and H. A. Reijers, editors, *Business Process Management Workshops*, pages 126–139, Cham, 2016. Springer International Publishing.
6. B. Luo, R. C. Wilson, and E. R. Hancock. Spectral clustering of graphs. *Lecture notes in computer science*, pages 190–201, 2003.
7. L. Moreau and P. Missier. PROV-DM: The PROV Data Model. `http://www.w3.org/TR/2013/REC-prov-dm-20130430/`, April 2013.
8. S. Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520, 2004.
9. A. Robles-Kelly and E. R. Hancock. Graph edit distance from spectral seriation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):365–378, March 2005.
10. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.
11. E. Santos, L. Lins, J. P. Ahrens, J. Freire, and C. T. Silva. *A First Study on Clustering Collections of Workflow Graphs*, pages 160–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
12. C. K. Selcuk and M. L. Sapino. *Data Management for Multimedia Retrieval*, page 114. Cambridge University Press, 2010.
13. Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *Proc. VLDB Endow.*, 2(1):25–36, Aug. 2009.