Towards Specificationless Monitoring of Provenance-Emitting Systems

Martin Stoffers[0000−0003−2987−4345] and Alexander Weinert[0000−0001−8143−246X]

German Aerospace Center (DLR)
Institute for Software Technology
Cologne, Germany
`firstname.lastname@dlr.de`

Abstract. Monitoring often requires insight into the monitored system as well as concrete specifications of expected behavior. More and more systems, however, provide information about their inner procedures by emitting provenance information in a W3C-standardized graph format. In this work, we present an approach to monitor such provenance data for anomalous behavior by performing spectral graph analysis on slices of the constructed provenance graph and by comparing the characteristics of each slice with those of a sliding window over recently seen slices. We argue that this approach not only simplifies the monitoring of heterogeneous distributed systems, but also enables applying a host of well-studied techniques to monitor such systems.

Keywords: W3C Provenance · Runtime Monitoring · Spectral Graph Analysis

1 Introduction

In current research on monitoring complex systems, the system is often abstracted to a set of streams of values [12]. Even when monitoring distributed systems, the problem of transporting the distributed data streams to the monitor is assumed to be solved by the monitored system [35]. In modern real-world systems, in contrast, it is a non-trivial effort to engineer a central component that efficiently consolidates data for monitoring in a heavily distributed system.

Moreover, these streams may be annotated with metadata, e.g., information about their creation times [38], whether an agent is a natural person or a software agent, or whether some input data was required or optional for the execution of a process. However, these metadata typically do not include information on their relation. Consider, e.g., a scenario in which a system provides a value x as an output to the user and also uses x as input for further computations. This monitored information typically does not indicate whether the two values coincide by design or by accident. Although this relation might be recovered from the logging stream, doing so is typically incomplete and error-prone. The metadata and relations of data produced by a system are known as provenance data [51].

Monitoring the provenance data of a system addresses both issues identified above: Provenance data describes, among other information, the relation between individual data points emitted by the system. Thus, it is typically consolidated by the system itself and made available for inspection or monitoring. Moreover, it contains more information on the inner workings of the system than the functional data. Hence, monitoring both provenance data and the functional data may lead to earlier detection of undesired system states.

In practice, monitoring approaches may take metadata of functional data into account. These metadata, however, are usually domain-specific. In contrast, provenance data are non-domain-specific, yet provide information about the structure of the monitored data as well as meta-data.

One major challenge when monitoring provenance data, however, is that users typically lack intuition about the relation between data they expect from the system. Thus, formulating specifications for monitoring provenance data is harder for users than formulating specifications for classical monitoring.

To alleviate this shortcoming, in this work we instead focus on anomaly detection, thus using previously seen data as specification. We believe monitoring of provenance data to be an interesting problem. The explicit graphs of provenance data allow for the application of well-known graph analyses to monitoring.

We present an approach for monitoring provenance data for anomalies without requiring explicit specifications. To this end, we proceed as follows: After discussing related work in Section 2, we we formally define provenance data as provenance graphs in Section 3 before subsequently describing the monitoring of provenance-emitting systems in Section 4. Afterwards we outline how to use spectral graph theory to detect anomalous provenance data in Section 5. Finally, in Section 6 we summarize our work and give an outlook on future work.

2 Related Work

Provenance Early works highlight the importance of provenance to enable audits of automated workflow systems [17, 29]. Moreau identified the building blocks for standardized provenance recording and proposes the Open Provenance Model (OPM) [36], which was later superseded by the W3C PROV standard [37, 51]. Provenance data is either extracted from software systems after [43] or during their operation [24, 27, 47]. There is active work towards recording provenance information without instrumenting the system or process [3, 5, 18, 39].

System Monitoring Runtime verification (or system monitoring) is an established building block for ensuring system correctness [4, 28]. Existing approaches to monitoring often take a specification of “good” or “bad” patterns and efficiently detect them in the output data of the system. This specification is typically given in temporal logics [6, 13, 14, 26, 32, 40] or in higher-level languages [7, 11, 15].

Anomaly Detection in Provenance Graphs Since provenance data give structured information about the relation between data points emitted by the system, there has been work towards identifying anomalies in provenance data. However, such work explicitly focuses on the detection of attacks on the system [8], uses bespoke projections from graphs to vector spaces [20, 21] or only analyzes individual characteristics of the computation process [41]. Other approaches aim to compare provenance graphs by "summarizing" via clustering [2, 31].

3 Provenance Graphs

The W3C defines provenance information as "information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness." [51]

The PROV standard prescribes a non-domain-specific graph-based ontology of such information. Each vertex in such a graph denotes either an *entity*, i.e., some piece of data, an *activity*, i.e., a process or action, or an *agent*, i.e., a person, machine or software responsible for a process. The edges between these vertices denote the relationships between entities, activities, and agents.

We illustrate the possible relationships in Figure 1. We draw entities as yellow rounded boxes, activities as blue rectangular boxes, and agents as red pentagons. In Figure 1, we write ATTR, DERIV, USE, GEN, and ASSOC to abbreviate "was attributed to", "was derived from", "used", "was generated by", and "was associated with", respectively. The full standard admits additional vertices and edges. We restrict ourselves to the edges shown in Figure 1 for conciseness.

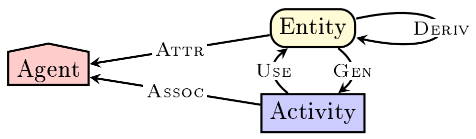


Fig. 1: The core provenance meta-model. Reproduced after and adapted from the Prov Primer [50].

Intuitively, the provenance of a software system is a record of a) the data that was generated or used, b) the process that generated and used these data, and c) the responsible entities (both human and software) for these processes.

Formally, a provenance graph $G = (V_{\text{AGT}}, V_{\text{ENT}}, V_{\text{ACT}}, E)$ consists of finite sets of agent vertices V_{AGT}, entity vertices V_{ENT}, and activity vertices V_{ACT}, all of which are pairwise disjoint, and a set of edges $E \subseteq (V_{\text{ENT}} \times (V_{\text{AGT}} \cup V_{\text{ENT}} \cup V_{\text{ACT}})) \cup (V_{\text{ACT}} \times (V_{\text{AGT}} \cup V_{\text{ENT}}))$. We call $V_{\text{AGT}} \cup V_{\text{ENT}} \cup V_{\text{ACT}}$ the *vertex set* of G.

In practice, a provenance-emitting system does not provide its complete provenance at the end of its computation. Instead, whenever an activity has terminated, the system provides a "partial" provenance graph that contains the respective activity as well as the entities that this activity used and generated. We also call these provenance graphs emitted during execution of the system *provenance updates* to differentiate them from the complete graph that the system constructs during its execution. To obtain a less local view of the provenance of the complete system execution, we construct the union over provenance via the component-wise union of the constituent elements of a provenance graph.

In this work, a *provenance-emitting system* is a software or hardware system that constructs a provenance graph that contains vertices representing the data points it generated, the processes that used and generated these data points, and makes this provenance graph accessible to external systems. In the following section, we provide greater detail and a more formal description of such systems. We moreover describe a process for monitoring provenance-emitting systems.

Having given a brief introduction to the W3C Provenance standard, we now illustrate how provenance data is collected from distributed provenance-emitting systems and made available for monitoring in the following section.

4 Monitoring Provenance-Emitting Systems

Intuitively, a provenance-emitting system emits information about its execution by making its provenance graph available for monitoring and inspection. To do so, it emits provenance updates at certain points in time, i.e., sub-graphs of the complete provenance graph of its execution. A monitor can observe these updates and monitor them for anomalies. In Figure 2 we illustrate a lightweight architecture for recording provenance, initially outlined in [47].

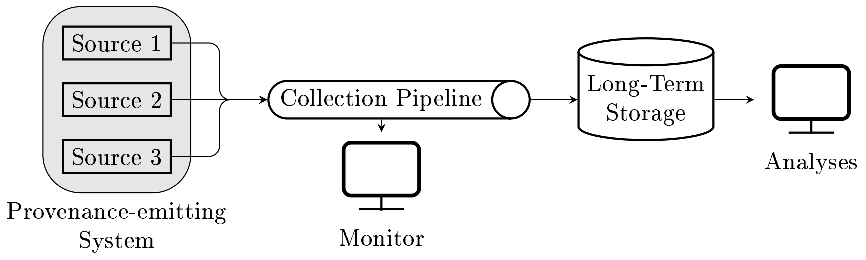


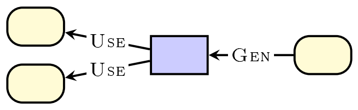
Fig. 2: A lightweight architecture for capturing provenance information from a complex distributed system. (Taken from [47] and simplified.)

Formally, we say that an *execution* of a provenance-emitting system is an infinite sequence $G_1G_2G_3\cdots$. The graphs are not necessarily temporally ordered. As an example, G_3 may only contain activities that started at time t, while G_4 only contains activities that ended at time $t' < t$.

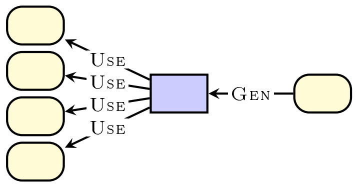
In this section we present a method to monitor an execution of provenance-emitting systems for anomalies. The complete provenance graph constructed by the system is unbounded and may be infinite for non-terminating systems. It is the task of the long-term Storage (cf. Figure 2) to provide sufficient storage capacity for this complete graph as required for post-hoc analyses. The monitor, in contrast, should not require unbounded memory, but instead work with restricted resources to function as lightweight as possible.

> Let $\varphi = G_1 G_2 G_3 \cdots$ be the execution of a provenance-emitting system. In each time step the monitor obtains the earliest G from from φ that it has not yet obtained. The monitor shall produce a sequence $b_1 b_2 b_3 \cdots$ where b_i is one of ✓, ✗, ?. The value ✓ (✗) denotes that the monitor has not (has) detected an anomaly in the last time step, respectively, while ? denotes that the monitor does not have sufficient information to make a decision. Moreover, the monitor shall not require unbounded memory.

In this problem formulation, we explicitly omit a definition of "anomalies". Recall that the PROV meta model only imposes very limited structure on provenance graphs. Thus, whether a provenance graph describes expected or unexpected behavior is strongly application-dependent. Analogously, it is not expedient to formally define anomalies independent of the monitored application.



(a) The provenance of an activity that used two entities and generated one entity.



(b) The provenance of an activity that used four entities and generated one entity.

Fig. 3: Examples of provenance graphs

Consider, e.g., a provenance-emitting system in which all activities so far take two entities as inputs and produce one entity as output. Assume there arrives a provenance update in which an activity takes four inputs and produces one output. We illustrate the former and latter case in Figure 3a and in Figure 3b, respectively. Whether or not the latter update is anomalous depends on the purpose of the system. If the system processes temperature readings since the last step, then the occurrence of fewer temperature readings than previously indicates, e.g., faulty sensors. In contrast, if a system processes observations made by an optical telescope [16, 48], then the system may process more information due to improved weather conditions. This would not be considered anomalous.

This example illustrates that there can be no "turnkey" solution for monitoring provenance-emitting systems: Either the user provides an explicit specification of expected or unexpected provenance patterns, or the monitor requires knowledge about the purpose of the system to infer anomalous graphs itself.

Thus, we aim to construct a parametrized monitor that measures the "anomaly" of a provenance update against those updates witnessed previously. Due to space constraints, this monitor cannot store all previously witnessed provenance graphs explicitly. Instead, it retains only a window of previously witnessed partial provenance graphs. We illustrate our monitoring architecture in Figure 4.

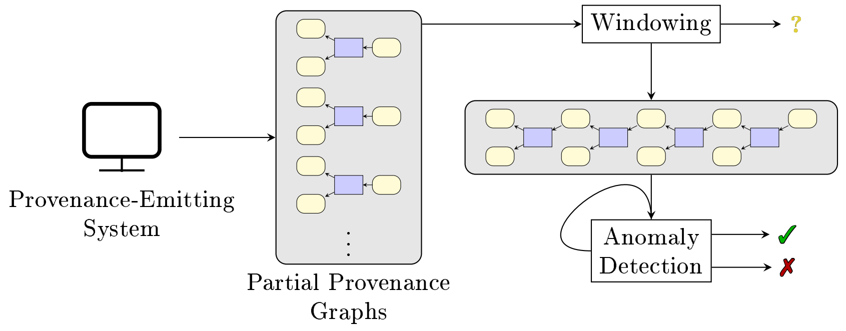


Fig. 4: An overview over our monitoring architecture.

The main purpose of the windowing is to construct a sequence of classical graphs that it then passes to anomaly detection. By retiring vertices representing entities that have not been used by the system, windowing ensures that the graphs passed to anomaly detection do not exceed a given size. We call this size the *window size*. This allows the anomaly detection to focus on comparing an incoming graph to the one previously obtained and to raise an alarm to the user if these two graphs differ significantly. To give the anomaly detection enough data to reliably identify structural anomalies, the windowing step reports ? until it has collected sufficient provenance updates to fill a predetermined window size.

When converting provenance graphs into classical directed graphs, structural information about the "kinds" of vertices is lost, as a classical graph does not differentiate between, e.g., vertices representing processes and vertices representing entities. Moreover, similar information about the kinds of edges is lost as well. This information can be reconstructed in the fragment of provenance graphs used in this work. Such a reconstruction is, however, not necessarily possible when using the full expressive power of W3C Provenance.

To retain this information, it may be encoded as vertex- or edge-weights. How weights should be assigned is again strongly application-specific and strongly influences the subsequent anomaly detection. One could, e.g., assign a high weight to all edges adjacent to activity-vertices. This would lead to the anomaly detection being highly sensitive to anomalous patterns in the vicinity of activity-vertices and less sensitive to the vicinity of other vertices. In the next section we describe possible approaches to detecting anomalies via spectral graph theory.

5 Anomaly Detection with Spectral Graph Theory

To identify structural anomalies of graphs, we need to quantify their topological properties, e.g., patterns of connectivity. We propose using spectral graph theory [45] to this end. This theory relies on studying the Eigenvalues and Eigenvectors of matrices associated with graphs, e.g., the adjacency matrix, the degree matrix, or the Laplace matrix. Intuitively, these values capture the topological properties of the investigated graphs [45]. Spectral graph theory has been successfully applied in some fields [44, 46]. In particular, Gera, Alonso, Crawford, et al. have used spectral graph theory to determine whether incoming observations of a graph significantly deviate from previous observations [19].

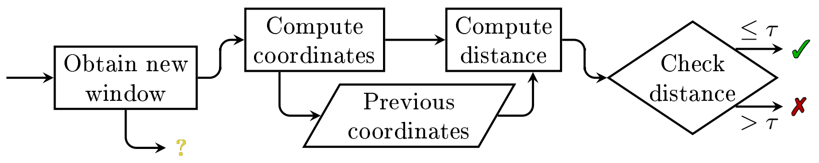


Fig. 5: Our framework for anomaly detection.

We illustrate our general approach in Figure 5. Let G be an incoming graph obtained by anomaly detection and let n be the window size determined when constructing or configuring the windowing. By computing Eigenvalues and Eigenvectors of matrices associated with G we can obtain vectors $\nu_1, \ldots, \nu_n$, where $\nu_i \in \mathbb{C}^m$ for all ν_i and some $m \leq n$. These vectors may, e.g., comprise the Eigenvalues of the used matrix or its Eigenvectors. In the former case, we have $m = 1$, in the latter $m = n$. Intuitively, if the investigated matrix is well-chosen, this set of vectors quantifies the structure of the graph. We call this set of vectors the *coordinates* of the graph. Having obtained the coordinates of both the current and the previous provenance graph, we can then compute the distance between these two coordinates and use this distance as a measure of the structural differences between the two graphs. We discuss possibilities and challenges for both steps, obtaining coordinates and computing their distance in the following sections. Moreover, we report on the results of a preliminary evaluation in Section 5.3.

5.1 Compute Coordinates

To obtain coordinates, we compute Eigenvalues and Eigenvectors of matrices associated with the graph. Typically, one uses the adjacency matrix or the Laplacian matrix of the graph [45]. Multiple works have shown that some graph properties can be determined based on the multiplicity, size or position of the Eigenvalues and corresponding Eigenvectors [9, 22, 30, 33, 34]. In graph drawing, Eigenvectors are selected based on their Eigenvalues and used as source for coordinates to visually reveal structural properties of graphs [25].

Most applications of spectral graph theory, however, assume the graph to be undirected. In that case, the adjacency matrix and the Laplacian matrix are real symmetric matrices, thus their Eigenvalues are integers. Provenance graphs, however, are directed. Thus, to apply standard methods of spectral graph analysis to them, we have to transform them into undirected graphs [42], which loses structural information. Another approach would be to use bespoke spectral graph analysis methods that handle directed graphs [10,49]. These methods are, however, not as well-investigated as those for undirected graphs.

5.2 Compute Distance

To identify anomalous updates we need to compare the current and the previous coordinate vectors. To this end, we aim to compute a normalized distance measure. Recall that coordinates are sets of vectors. Thus, a common method is to first calculate the pairwise distances for sets of vectors separately using different metrics, such as the euclidean distances, cosine similarity or correlation. By taking the average over the resulting vector of distance measurements we can obtain a distance between coordinates.

Directly computing the difference between two sets of vertices is rather sensitive to “noisy” coordinates: Minor differences between individual vectors may lead to large differences. We can counteract this via clustering the vectors comprising the coordinates prior to distance calculation. In this case, the complete monitoring pipeline up to the computation of coordinates is tantamount to spectral clustering [30] of the provenance graph. Via clustering we obtain cluster centroids, the coordinates of which can be used to calculate a distance measurement as outlined before. Having obtained a distance measure, we can check the distance against a provided threshold. If the distance exceeds this threshold, anomaly detection alerts the user to anomalous system behavior.

5.3 Proof of Concept

We implemented our method using the Eigenvalues of the Laplacian matrix as coordinates and the distance between the centroids of the Eigenvalues as the distance metric. We evaluated this implementation on a synthetic example as well as a realistic one. In the synthetic example, in each time step the system adds two numbers during normal operation. We have injected anomalies into the provenance data representing the addition of ten numbers in one time step. In the realistic example, in each time step a robot executes some actions based on some plan [23]. There are anomalies where no such plan is present.

Our prototypical implementation was able to successfully differentiate between the nominal and anomalous updates. This illustrates that our proposed method can indeed determine anomalies at least in these two use cases.

6 Conclusion and Future Work

In contrast to streams solely comprising the output data of a system, provenance data allows far greater insight into the inner workings of such a system. Thus, we believe that monitoring provenance data in addition to output data allows for earlier detection of system failures. We have outlined an approach to monitor these provenance data that reduces the problem of monitoring provenance data to that of determining anomalies in graphs. This greatly reduces the parameter space that has to be explored when constructing a real-world monitor to determining useful well-studied approaches for detecting anomalies in graphs [1]. Moreover, we illustrated that this approach can serve as a framework for detecting anomalies in provenance data via a prototypical proof of concept.

As a next step, we aim to identify real-life use cases in which we can apply and evaluate our approach. This use case will allow us to compare different definitions of coordinates and distances between coordinates. Moreover, we will be able to evaluate our proposed approach against other approaches to anomaly detection in graphs [1]. In addition, we aim to quantify the structure of graphs by additional properties, e.g., their diameter or depth. Finally, we are looking to compare our approach based on spectral graph analysis against existing machine learning approaches to anomaly detection.

Acknowledgements We gratefully acknowledge suggestions by anonymous reviewers, which have significantly improved this work.

References

1. Akoglu, L., Tong, H., Koutra, D.: Graph based anomaly detection and description: a survey. Data Mining and Knowledge Discovery **29**(3), 626–688 (jul 2014). https://doi.org/10.1007/s10618-014-0365-y
2. Alawini, A., Chen, L., Davidson, S., Fisher, S., Kim, J.: Discovering Similar Workflows via Provenance Clustering: A Case Study. In: Belhajjame, K., Gehani, A., Alper, P. (eds.) IPAW 2018. pp. 115–127. Springer. https://doi.org/10.1007/978-3-319-98379-0_9
3. Alter, G.C., Gager, J., Heus, P., Hunter, C., Ionescu, S., Iverson, J., Jagadish, H., Lyle, J., Mueller, A., Nordgaard, S., Risnes, O., Smith, D., Song, J.: Capturing data provenance from statistical software. Int. J. Dig. Curat. **16**(1), 14 (2022). https://doi.org/10.2218/ijdc.v16i1.763
4. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to Runtime Verification. In: Lectures on Runtime Verification, pp. 1–33. Springer (2018). https://doi.org/10.1007/978-3-319-75632-5_1
5. Bates, A., Tian, D., Butler, K.R.B., Moyer, T.: Trustworthy Whole-System Provenance for the Linux Kernel. In: Jung, J., Holz, T. (eds.) USENIX Security Symposium 2015. pp. 319–334. USENIX (2015), https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/bates
6. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 1–64 (2011). https://doi.org/10.1145/2000799.2000800

7. Baumeister, J., Finkbeiner, B., Schirmer, S., Schwenger, M., Torens, C.: RTLola Cleared for Take-Off: Monitoring Autonomous Aircraft. In: CAV 2020, pp. 28–39. Springer. https://doi.org/10.1007/978-3-030-53291-8_3
8. Berrada, G., Cheney, J.: Aggregating unsupervised provenance anomaly detectors. In: TaPP 2019. USENIX, https://www.usenix.org/conference/tapp2019/presentation/berrada
9. Biyikoğu, T., Leydold, J., Stadler, P.F.: Laplacian Eigenvectors of Graphs. Springer (2007). https://doi.org/10.1007/978-3-540-73510-6
10. Chung, F.: Laplacians and the cheeger inequality for directed graphs. Annals of Combinatorics **9**(1), 1–19 (apr 2005). https://doi.org/10.1007/s00026-005-0237-z
11. D'Angelo, B., Sankaranarayanan, S., Sanchez, C., Robinson, W., Finkbeiner, B., Sipma, H., Mehrotra, S., Manna, Z.: LOLA: Runtime Monitoring of Synchronous Systems. In: TIME 2005. IEEE. https://doi.org/10.1109/time.2005.26
12. Dauer, J.C., Finkbeiner, B., Schirmer, S.: Monitoring with Verified Guarantees. In: RV 2021, pp. 62–80. Springer. https://doi.org/10.1007/978-3-030-88494-9_4
13. Dawes, J.H., Bianculli, D.: Specifying Properties over Inter-procedural, Source Code Level Behaviour of Programs. In: RV 2021, pp. 23–41. Springer (2021). https://doi.org/10.1007/978-3-030-88494-9_2
14. Dawes, J.H., Reger, G.: Specification of temporal properties of functions for runtime verification. In: SAC 2019. ACM. https://doi.org/10.1145/3297280.3297497
15. Faymonville, P., Finkbeiner, B., Schledjewski, M., Schwenger, M., Stenger, M., Tentrup, L., Torfah, H.: StreamLAB: Stream-based Monitoring of Cyber-Physical Systems. In: CAV 2019, pp. 421–431. Springer. https://doi.org/10.1007/978-3-030-25540-4_24
16. Fiedler, H., Herzog, J., Prohaska, M., Schildknecht, T., Weigel, M.: SMARTnet(TM)–Status and Statistics. In: IAC 2017. https://elib.dlr.de/115884/
17. Foster, I., Vockler, J., Wilde, M., Zhao, Y.: Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In: SSDBM 2002. pp. 37–46 (2002). https://doi.org/10.1109/SSDM.2002.1029704
18. Gehani, A., Tariq, D.: SPADE: Support for Provenance Auditing in Distributed Environments. In: Middleware 2012, pp. 101–120. Springer. https://doi.org/10.1007/978-3-642-35170-9_6
19. Gera, R., Alonso, L., Crawford, B., House, J., Mendez-Bermudez, J.A., Knuth, T., Miller, R.: Identifying network structure similarity using spectral graph theory. Applied Network Science **3**(1) (jan 2018). https://doi.org/10.1007/s41109-017-0042-3
20. Han, X., Pasquier, T., Ranjan, T., Goldstein, M., Seltzer, M.: FRAPpuccino: Fault-detection through Runtime Analysis of Provenance. In: HotCloud 2017. USENIX, https://www.usenix.org/conference/hotcloud17/program/presentation/han
21. Han, X., Pasquier, T., Seltzer, M.: Provenance-based Intrusion Detection: Opportunities and Challenges. In: TaPP 2018. USENIX, London, https://www.usenix.org/conference/tapp2018/presentation/han
22. Hong, Y.: Bounds of eigenvalues of graphs. Discret. Math. **123**(1), 65–74 (1993). https://doi.org/10.1016/0012-365X(93)90007-G
23. Huynh, D.: c50b1f3a-89c2-11ec-a4ca-24418cc781d9 (2022), available at https://openprovenance.org/store/documents/5427
24. Johnson, M.A.C., Paradies, M., Dembska, M., Lackeos, K., Klöckner, H.R., Champion, D.J., Schindler, S.: Astronomical Pipeline Provenance: A Use Case Evaluation. In: TaPP 2021. USENIX, https://www.usenix.org/conference/tapp2021/presentation/johnson
25. Koren, Y.: Drawing Graphs by Eigenvectors: Theory and Practice. Comp. Math. Appl. **49**(11), 1867–1888 (2005). https://doi.org/10.1016/j.camwa.2004.08.015

26. Koymans, R.: Specifying Real-Time Properties with Metric Temporal Logic. Real-Time Syst. **2**(4), 255–299 (1990). https://doi.org/10.1007/bf01995674
27. Kühnert, J., Göddeke, D., Herschel, M.: Provenance-integrated parameter selection and optimization in numerical simulations. In: TaPP 2021. USENIX, https://www.usenix.org/conference/tapp2021/presentation/kühnert
28. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebraic Methods Program. **78**(5), 293–303 (2009). https://doi.org/10.1016/j.jlap.2008.08.004
29. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. Concurr. Comput. Pract. Exp. **18**(10), 1039–1065 (2006). https://doi.org/10.1002/cpe.994
30. von Luxburg, U.: A tutorial on spectral clustering. Stat. and Comp. **17**(4), 395–416 (2007). https://doi.org/10.1007/s11222-007-9033-z
31. Macko, P., Margo, D., Seltzer, M.: Local Clustering in Provenance Graphs. In: CIKM 2013. p. 835–840. ACM. https://doi.org/10.1145/2505515.2505624
32. Maler, O., Ničković, D.: Monitoring properties of analog and mixed-signal circuits. Int. J. Softw. Tools Technol. Transf. **15**(3), 247–268 (2012). https://doi.org/10.1007/s10009-012-0247-9
33. Merris, R.: Laplacian matrices of graphs: a survey. Lin. Alg. App. **197-198**, 143–176 (1994). https://doi.org/10.1016/0024-3795(94)90486-3
34. Mohar, B.: Laplace eigenvalues of graphs—a survey. Discret. Math. **109**(1), 171–183 (1992). https://doi.org/10.1016/0012-365X(92)90288-Q
35. Momtaz, A., Basnet, N., Abbas, H., Bonakdarpour, B.: Predicate Monitoring in Distributed Cyber-Physical Systems. In: RV 2021, pp. 3–22. Springer. https://doi.org/10.1007/978-3-030-88494-9_1
36. Moreau, L.: The Foundations for Provenance on the Web. Found. Trends Web Sci. **2**(2-3), 99–241 (2010). https://doi.org/10.1561/1800000010
37. Moreau, L., Groth, P., Cheney, J., Lebo, T., Miles, S.: The rationale of PROV. J. Web Semant. **35**, 235–257 (2015). https://doi.org/10.1016/j.websem.2015.04.001
38. Nenzi, L., Bortolussi, L., Ciancia, V., Loreti, M., Massink, M.: Qualitative and Quantitative Monitoring of Spatio-Temporal Properties. In: RV 2015, pp. 21–37. Springer. https://doi.org/10.1007/978-3-319-23820-3_2
39. Pasquier, T., Han, X., Goldstein, M., Moyer, T., Eyers, D., Seltzer, M., Bacon, J.: Practical whole-system provenance capture. In: SoCC 2017. ACM. https://doi.org/10.1145/3127479.3129249
40. Pnueli, A.: The temporal logic of programs. In: SFCS 1977. IEEE. https://doi.org/10.1109/sfcs.1977.32
41. Pouchard, L., Huck, K., Matyasfalvi, G., Tao, D., Tang, L., Dam, H.V., Yoo, S.: Prescriptive provenance for streaming analysis of workflows at scale. In: NYSDS 2018. IEEE. https://doi.org/10.1109/nysds.2018.8538951
42. Satuluri, V., Parthasarathy, S.: Symmetrizations for clustering directed graphs. In: EDBT 2011. ACM (2011). https://doi.org/10.1145/1951365.1951407
43. Schreiber, A., de Boer, C., von Kurnatowski, L.: GitLab2PROV—Provenance of Software Projects hosted on GitLab. In: TaPP 2021. USENIX, https://www.usenix.org/conference/tapp2021/presentation/schreiber
44. Shi, J., Malik, J.: Normalized Cuts and Image Segmentation. In: CVPR 1997. pp. 731–737. https://doi.org/10.1109/CVPR.1997.609407
45. Spielman, D.: Combinatorial Scientific Computing, chap. Spectral Graph Theory, p. 30. Chapman and Hall/CRC (2011)

46. Spielman, D.A.: Spectral Graph Theory and its Applications. In: FOCS 2007. pp. 29–38 (2007). https://doi.org/10.1109/FOCS.2007.56
47. Stoffers, M., Meinel, M., Hofmann, B., Schreiber, A.: Integrating Provenance-Awareness into the Space Debris Processing System BACARDI. In: IEEE Aerospace Conference 2022. To appear.
48. Stoffers, M., Meinel, M., Weigel, M., Siggel, M., Fiedler, H., Rack, K., Wasser, Y.: BACARDI: A System to track Space Debris. In: ESA NEO and Debris Detection Conference. https://elib.dlr.de/126572/
49. Van Lierde, H.: Spectral clustering algorithms for directed graphs. Master's thesis, Université catholique de Louvain (2015)
50. W3C Working Group: PROV Model Primer (2013), available at https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/, retrieved April 28, 2022.
51. W3C Working Group: PROV-Overview. An Overview of the PROV Family of Documents (2013), available at https://www.w3.org/TR/2013/NOTE-prov-overview-20130430/, retrieved April 28, 2022.