BIG DATA ANALYTICS IN STATIC AND STREAMING PROVENANCE

Peng Chen

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements for the degree

Doctor of Philosophy

in the School of Informatics and Computing,

Indiana University

April 2016

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Beth Plale, Ph.D.
(Chairperson)

David Crandall, Ph.D.

Ryan Newton, Ph.D.

Tom Evans, Ph.D.

December 4th, 2015

To my wife Shuya, my parents, and my grandparents.

Acknowledgements

First, I would like to express my sincere gratitude to my advisor Professor Beth Plale for her continuous guidance and support throughout my PhD study. I feel truly fortunate to be able to work with her, and what I have learned from her is far beyond conducting research. I have been constantly surprised by her insightful thoughts and visionary suggestions. Her dedication, thrust, and energy will continue to inspire me.

I would also like to individually thank the rest of my thesis committee. Professor Tom Evans is my minor advisor in Geography, and I was enlightened by his crystal clear illustration of geographic concepts. His consideration and special sense of humor ensured my experience collaborating with him was truly enjoyable. Professor David Crandall has always being encouraging as well. He gave me lots of insightful comments from the very beginning and spent valuable time on proofreading and commenting extensively on my thesis. Professor Ryan Newton has a very sharp mind. He showed me different perspectives to tackle my research question and helped me frame the solution.

My sincere thanks also goes to Dr. Mehmet Aktas, who was a mentor and co-author with me during my first year at Indiana University; and Dr Yi Miao, who provided an opportunity to work at Facebook on a very interesting project that was beneficial to my PhD research.

I thank my fellow labmates for the stimulating discussions, for the fun and learning opportunities we have enjoyed together. In particular, I want to offer appreciation to these individuals: Dr. Miao Chen, Dr. Youwei Cheah, Dr. Devarshi Ghoshal, Dr. Scott Jensen, Dr. Yuan Luo, Yu Luo, Milinda Pathirage, Zong Peng, Guangchen Ruan, Isuru Suriarachchi, Gabriel Zhou, Jiaan Zeng.

I also want to thank the administrative staff, Jenny Olmes-Stevens and Jodi Stern for making the lab an enjoyable and productive place to work.

Last but not the least, I would like to thank my family: my wife, Shuya Xu, and my parents, Yuliang Chen and Meizhen Fang, who have been supportive and encouraging through the journey.

Peng Chen

BIG DATA ANALYTICS IN STATIC AND STREAMING PROVENANCE

With recent technological and computational advances, scientists increasingly integrate sensors and model simulations to understand spatial, temporal, social, and ecological relationships at unprecedented scale. Data provenance traces relationships of entities over time, thus providing a unique view on over-time behavior under study. However, provenance can be overwhelming in both volume and complexity; the now forecasting potential of provenance creates additional demands.

This dissertation focuses on Big Data analytics of static and streaming provenance. It develops filters and a non-preprocessing slicing technique for in-situ querying of static provenance. It presents a stream processing framework for online processing of provenance data at high receiving rate. While the former is sufficient for answering queries that are given prior to the application start (forward queries), the latter deals with queries whose targets are unknown beforehand (backward queries). Finally, it explores data mining on large collections of provenance and proposes a temporal representation of provenance that can reduce the high dimensionality while effectively supporting mining tasks like clustering, classification and association rules mining; and the temporal representation can be further applied to streaming provenance as well. The proposed techniques are verified through software prototypes applied to Big Data provenance captured from computer network data, weather models, ocean models, remote (satellite) imagery data, and agent-based simulations of agricultural decision making.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

E-Science (or eScience) is data intensive, computationally-based science where computer science meets applications, and brings advances in both fields [75]. With recent technological and computational advances, eScience research increasingly integrates sensors and model simulations to understand spatial, temporal, social, and ecological relationships at unprecedented scale. This data intensive aspect of eScience reflects the increasing value of observational, experimental and computer-generated data in virtually all domains, from physics to social and ecological sciences. However, the explosion in volume and dimensionality of data and the extraordinarily broad field of eScience introduce severe challenges for data management. These include reliably archiving large volumes (many terabytes) of data, sharing such data with users within access control policies, and maintaining sufficient context and provenance of sensor data using correct metadata [53].

Provenance, sometimes referred to as the lineage or pedigree of data derivation, is a key piece of metadata that facilitates data sharing and re-use in eScience data management. It has also been widely recognized that provenance is critical to sharing, trust, authentication and reproducibility of eScience process [100]. However, given the sheer volume of datasets and the complex nature of the interactions in an eScience infrastructure, provenance can be overwhelming in both volume and complexity, requiring new techniques to handle it and make it useful.

The development of provenance research has its roots in databases and workflow management systems [145], where it has been shown to facilitate the reproducibility of computational experiments [116]. In eScience, it has been shown that provenance can identify event causality; enable broader forms of sharing, reuse, and long-term preservation of scientific data; can be used to attribute ownership and determine the quality of a particular data set [143]. Earlier research in provenance has studied the capture, modeling, and storage of provenance. Collaborative efforts have yielded the Open Provenance Model (OPM) [117] and its successor, the W3C recommendation named PROV [23], for modeling and exchange of provenance. Both OPM and PROV model provenance as a Direct Acyclic Graph (DAG), where nodes are artifacts (data), processes and agents (person), and edges represent the rich relationships between nodes.

Despite advances in capturing and modeling provenance, its size remains a significant issue, as does its complexity. There has been research on visualizing [107] and mining provenance data [110], but the analysis of provenance in general has been underexplored. Gobe [73] and Simmhan et al. [145] summarize five major reasons to use provenance: data quality, audit trail, replication recipes, attribution, and informational. We aim to further expand the use of data provenance that is captured from computer network, workflow, and agent-based model. Efficient and scalable provenance retrieval is the key to obtaining critical information from large scale provenance datasets. Being able to process streaming provenance generated from running applications further enables online monitoring, debugging, and calibration. Data mining is useful in discovering hidden information from provenance datasets, but in order to support that, the representation of provenance has to address the challenge of high-dimensional graph data. Provenance brings a unique view on over-time behavior under study and thus its forecasting potential also suggests new solutions to data mining.

This dissertation aims to help data scientists to better understand, debug, and use big data, by providing techniques and software solutions to analyzing Big Data provenance of high complexity, large volume, and high velocity. The research is mainly driven by two factors:

- To exploit the intrinsic value in provenance graphs' attribute and structure. Furthermore, to discover the hidden information from large collections of provenance graphs that are often generated by scientific workflows/simulations ran with different parameters and input data;
- To develop general analytical techniques that can deal with the large volume, high complexity and velocity of provenance data.

1.2 Challenges

This research investigates Big Data challenges at different stages of provenance analysis, namely, capture, retrieval, and mining.

1.2.1 Selective Capture and In-Situ Retrieval of Provenance

We have demonstrated earlier [43] that data provenance is capable of capturing the cause-effect relations in agent based models, aiding in the understanding of the complex processes and to enable the analysis of internal dynamics that were previously hidden as black box.

However, for large scale agent-based models that involve thousands of interacting agents, the sheer volume of provenance can be overwhelming. Traditional retrieval techniques fall short of processing this Big Data provenance since they often require persistence of the entire provenance dataset, despite the fact that only a small portion is desired. The existing method that captures provenance from program logs needs to recover all the entities and dependencies from the logs before being able to answer queries [9]. Instead, if we know the provenance query prior to the application start (for example, a forward provenance query), we can reduce the amount of provenance stored by preserving only the relevant provenance information; we can also perform the provenance queries in-situ and on-demand to avoid recovering and maintaining provenance entities and dependencies that are unnecessary.

1.2.2 Online Processing of Live Provenance

Scientific experiments that run at large scale and for a long time can produce a huge amount of provenance data that is too big to be stored persistently. One approach to handling provenance data in-motion is to process it continuously in the data stream model where the input is defined as a stream of data. Algorithms in this model must process the input stream in the order it arrives while using only a limited amount of memory [18]. Earlier work on processing real time streams of data [3, 17, 39, 126] focuses on high velocity data that arrives continuously (as streams of indefinite duration). However, provenance is modeled as a graph, and most of the work on graph streams has occurred in the last decade and focuses on the semi-streaming model [65, 121]. In this model the data stream algorithm is permitted $O(n\text{polylog}n)$ space where n is the number of nodes in the graph thus it does not suit for continuous processing. This is because most problems are provably intractable if the available space is sub-linear in n, whereas many problems become feasible once there is memory roughly proportional to the number of nodes in the graph. Thus the challenge here is to develop a streaming solution that can process the live provenance using limited memory.

1.2.3 Representation for Data Mining on Large Scale Provenance

Provenance modeled in either the OPM or the PROV standard can be represented as a graph with node/edge attributes. Both the graph structure and node/edge attributes contain valuable information that is useful for data mining operations. Workflow graph, sometimes referred to as the workflow plan, is in some cases a coarse approximation of a provenance graph, but existing work on mining workflow graphs uses data representations that either lose the structure information or the attribute information. Besides, as the size of provenance graph grows, the high dimensionality in the attribute information also poses a problem for data mining. Finally, existing research on mining provenance data requires that provenance be stored persistently, which could be intractable for the Big Data provenance with high velocity.

How to treat data with temporal dependencies is another problem in the discovery process of hidden information. The ultimate goal of data mining is to discover hidden relations between sequences and subsequences of events [16]. Provenance information has an implicit temporal ordering that can be exploited for data clustering and relationship discovery.

1.3 Contributions

This dissertation develops analytical techniques for efficient retrieval and representation of Big Data provenance. It further expands the use of data provenance in computer network, workflow, and agent-based simulation to help data scientists better understand, debug, and use their data. In computer network, we demonstrate network provenance can be used to explore, present, and debug various network experiments. In workflow system, we use provenance to detect and diagnose failure executions, and to predict the workflow type of a provenance graph. In agent-based simulation, provenance captures the internal dynamics that can be used to debug the simulation and analyze the model and the simulation results. By enabling the real-time processing of live provenance, we are even able to support the online debugging and calibration of simulations.

On the technical level, the dissertation contributes the following:

1. Selective capture of provenance is explored through filters that aggregate or filter out irrelevant provenance information for pregiven queries. A slicing technique is proposed for provenance called "non-preprocessing provenance slicing" that avoids recovering provenance entities and dependencies unnecessary for queries. The combination of the two is shown to be effective for the retrieval of large scale static provenance generated from agent-based simulations.
2. A stream model of data provenance and a streaming algorithm that enable the backward provenance query on live provenance data from agent-based model. This avoids the persistent storage of Big Data provenance and requires few computational resources. A general provenance stream processing framework is prototyped using Apache Kafka and Spark Streaming. The proposed streaming algorithm is shown to have high throughput, low latency and good scalability.
3. A reduced temporal representation for provenance graphs that preserves temporal order between node subsets and retains structural information together with attribute information. Experiments on semi-synthetic and real life scientific provenance datasets show that the temporal representation can detect failures in the workflow execution

or the provenance capture; can be used to predict the type of new workflow instance; and can describe/distinguish clusters from one another. The scalability study indicates that data mining on proposed temporal representations is scalable because the size and dimensionality of the dataset is greatly decreased in our reduction (representation) process, and the representation process can be parallelized using MapReduce. Furthermore, a window-based streaming algorithm is proposed to apply the temporal representation to the provenance stream model. To measure the similarity between window-based temporal representations that vary in time or speed, we propose a new similarity measurement based on Dynamic Time Warping (DTW).

1.4 Thesis outline

The remainder of this dissertation is organized as follows. We present related research in Chapter 2. Next, we discuss the background information, our earlier work on provenance capture and visualization, and the experimental datasets in Chapter 3. In Chapter 4, we present our solutions to efficient static provenance retrieval. In Chapter 5, we propose a stream model for provenance and a streaming solution to online retrieval of live provenance. Chapter 6 introduces the temporal representation for mining large collections of provenance data. Finally, we conclude and discuss future directions in Chapter 7.

Chapter 2

Related Work

In this chapter, we discuss existing research in areas that are related to our study. The first section examines provenance visualization techniques that have been developed. The subsequent section discusses efforts to efficiently store and retrieve provenance data. While there has been little research on the stream processing of provenance data, provenance data of streaming systems has been extensively studied. The third section examines related research on mining provenance data, and since workflow graphs can be considered as coarse approximations of provenance graphs, the existing work on mining workflow graphs is also included. Since one of the goals in our research is to expand the use of data provenance, we also discuss the existing work on the use of provenance in the final section.

2.1 Visualization of Provenance

Iliinsky and Steele (2011) [87] identify two categories of data visualization: exploration and explanation. *Exploratory visualization* is designed to support a researcher who has large volumes of data and not certain what is in it. *Explanatory visualization*, on the other hand, takes place when a researcher knows what the data has to say, and is trying to tell that story to someone else. The two serve different purposes, and there are tools and approaches that may be appropriate for one and not the other.

Prior provenance visualization tends to deal with small graphs, and seldom addresses the issue of graph layout. Specifically, Taverna [123] uses visualization to help answer questions that establish how the experiment results were obtained; VisTrails [69] allows users to navigate workflow versions in an intuitive way, to visually compare different workflows and their results, and to examine the actions that led to a result; Probe It! [56] enables scientists to move the visualization focus from intermediate and final results to provenance back and forth; the Prototype Lineage Server [29] allows users to browse lineage information by navigating through the sets of metadata that provide useful details about the data products and transformations in a workflow invocation; Pedigree Graph [122], one of tools in Multi-Scale Chemistry (MSC) portal from the Collaboratory for Multi-Scale Chemical Science (CMCS), is designed to enable users to view multi-scale data provenance; the My-Grid project renders graph-based views of RDF-coded provenances using Haystack [170]; Provenance Explorer [47], a secure provenance visualization tool, dynamically generates customized views of scientific data provenance that depend on the viewer requirements and/or access privileges.

Provenance Map Orbiter [107] uses graph summarization and semantic zoom to navigate large provenance graphs. It gives a high-level abstracted view of a graph and the ability to incrementally drill down to the details. However, node summarization depends on having available sufficient semantic information. Besides, a summarized view may not work well for visual comparison of multiple graph components at a detailed level.

While there is no prior research on graph matching in provenance visualization, there has been some work on comparing pairs of graphs visually, including the visual "diff" in VisTrails for comparing two workflows by Freire et al. [69]. Bao et al. [19] visualize pairs of provenance graphs to compare workflow executions of the same specification and understand the difference between them.

There has been substantial work in the area of graph visualization, ranging from layout algorithms, to methods for visualizing large graphs, to techniques for interacting with graphs [83]. The existing graph visualization tools could also be used for provenance graphs. Some of the tools are implemented in JavaScript to provide interactive data visualizations for the Web [67, 102], but handling large scale graphs requires more computational resources, and thus we focus on the desktop tools instead. Cytoscape [141] is an open source software platform for visualizing complex networks. Cytoscape allows diverse types of attribute data to be visualized on the nodes and edges of a graph, and thus is suitable for visualizing provenance graphs with many attributes. Gephi [22] is another interactive visualization and exploration platform for networks and complex systems, dynamic and hierarchical graphs. The Science of Science (Sci2) Tool [150] is a modular toolset specifically designed for the study of science. It supports the temporal, geospatial, topical, and network analysis and visualization of scholarly datasets at the micro (individual), meso (local), and macro (global) levels. While Gephi and Sci2 also have good support for large graphs, we choose Cytoscape as the base framework to implement our proposed techniques due to Cytoscape's abundant features for rendering the attribute data in provenance graphs.

2.2 Storage and Retrieval

2.2.1 Storage and Retrieval of Large Scale Provenance Data

The existing solutions to provenance management adopt a variety of formats and databases. Kepler [105] uses a provenance framework, called Collection-Oriented Modeling and Design (COMAD) [30], which stores provenance data in an XML file and in a relational database [14]. Taverna [114] features a provenance ontology and employs semantic web technologies for provenance collection and representation. VisTrails [69] and Karma [147] use XML and relational database technologies to represent, store, and query workflow provenance. Since provenance can be represented as a DAG, graph databases are also used for storing and querying provenance [115, 153]. Trident [144] and PreServ/PASOA [119] are not tied to any particular provenance storage model, but rather define interfaces and adaptors to enable different storage systems, such as a file system, relational database, XML database and so forth.

Provenance query processing has been studied in some existing work. Miles [112] defines a provenance query and describes techniques for scoped execution of provenance queries. Zhao et al. [172] propose a logic programming approach to scientific workflow provenance querying and management. Holland et al. [85] present a provenance query model based on the Lorel query language. Based on a general model of provenance supporting scientific workflows that process XML data and employ update semantics [14], Anand et al. [13] define a formal semantics of a provenance query language, QLP, and discuss efficient query processing approaches.

Kementsietsidis and Wang [92] first define the provenance backward/forward query. The provenance backward query is defined as a query addressing what input was used to generate the output, and the provenance forward query is defined as a query identifying what output was derived from the input.

More recently, there has been research on efficient and scalable storage and querying of large scale provenance graphs. Zhao et al. [167] explore the feasibility of a general metadata storage and management layer for parallel file systems, in which metadata includes both file operations and provenance metadata. They consider two systems that have applied similar distributed concepts to metadata management, but focusing singularly on provenance data: (1) FusionFS [168], which implements a distributed file metadata management based on distributed hash tables, and (2) SPADE [71], which uses a graph database to store audited provenance data and provides distributed module for querying provenance. Chebotko et al. [38] address the challenge of storage and querying of large collections of provenance graphs serialized as RDF graphs in an Apache HBase database. Rozsnyai et al. [128] introduce a large-scale distributed provenance solution built around HBase/Hadoop to track a consistent and accurate history, the relationships and derivations between artifacts in order to be able to monitor and analyze business process.

Provenance query processing in distributed environment has been studied in several papers. In [109], Malik et al. propose a decentralized architecture in which each host maintains an authoritative local repository of the provenance metadata gathered on that host, and the use of provenance sketches (based on Bloom filters) to optimize subsequent data provenance queries. Groth and Moreau [77] also discuss how to represent provenance information in distributed environments by using the Open Provenance Model [117]. Gadelha et al. [70] present and evaluate the Swift parallel scripting language in recording and analyzing provenance information collected from large-scale distributed systems. Kim et al. [93] discuss the generation and refinement of application-level provenance in the context of very large-scale workflows that often involve thousands of computations over distributed, shared resources.

It is also possible to process large provenance graphs with general distributed graph processing frameworks such as the Parallel Boost Graph Library (BGL), GraphX, Distributed GraphLab, Pregel, Giraph, GPS and PowerGraph. Parallel BGL [76] is a generic

C++ library for distributed graph computation. Its primary goals are to provide a flexible, efficient library of parallel and distributed graph algorithms and data structures, and to understand how generic programming interacts with parallel programming. GraphX [162] combines the advantages of both data-parallel and graph-parallel systems by efficiently expressing graph computation within the Spark data-parallel framework. It leverages ideas in distributed graph representation to efficiently distribute graphs as tabular data-structures, and advances in data-flow systems to exploit in-memory computation and fault-tolerance. The Distributed GraphLab [103] adopts graph based extensions to pipelined locking and data versioning to reduce network congestion and mitigate the effect of network latency. It adds fault tolerance to the GraphLab abstraction using the classic Chandy-Lamport snapshot algorithm. Pregel [108] is a bulk synchronous message passing abstraction in which all vertex-programs run simultaneously in a sequence of super-steps. Within a super-step each program instance receives all messages from the previous super-step and sends messages to its neighbors in the next super-step. The program terminates when there are no messages remaining and every program has voted to halt. Giraph [48] originated as the open-source counterpart to Pregel. Giraph adds several features beyond the basic Pregel model, including master computation, sharded aggregators, edge-oriented input, and out-of-core computation. GPS [132] is another open source implementation of the Pregel system, with three new features: (1) an extended API to make global computations more easily expressed and more efficient; (2) a dynamic repartitioning scheme that reassigns vertices to different workers during the computation, based on messaging patterns; and (3) an optimization that distributes adjacency lists of high-degree vertices across all compute nodes to improve performance. The PowerGraph [74] abstraction exploits the Gather-Apply-Scatter model of computation to factor vertex-programs over edges, splitting high-degree vertices and exposing greater parallelism in natural graphs. It is the first to introduce vertex-cuts and a collection of fast greedy heuristics to substantially reduce the storage and communication costs of large distributed power-law graphs.

Although a lot of effort has been put on improving the performance of provenance queries, most existing work, even if discussed within a distributed environment, only provides provenance retrieval functionality based on complete provenance. However, the Big Data provenance that we captured from simulations is at exascale and cannot be stored persistently. Furthermore, the provenance from a continuously running application such as an agent based simulation is also generated continuously. We cannot wait until all the provenance data is generated to start the analysis. To address these problems, we believe that stream processing of provenance queries that does not persist the data and provides real-time results is a natural solution.

2.2.2 Provenance Filtering and Non-preprocessing Slicing

Systems that gather fine-grained provenance metadata must process large amounts of information, and there is some existing research on provenance filtering. SPADE [71] is an open source software platform that supports collecting, filtering, storing, and querying provenance metadata. SPADE provides a framework for implementing filters (that can be stacked in arbitrary order). A filter receives a stream of provenance graph vertices and edges, and can rewrite their annotations (in which domain-specific semantics are embedded).

Filtering has also been used in provenance query processing. Zhao [169] gives a formal definition of provenance filter query, and provides a general and efficient approach for distributed query processing. Sahoo et al. [130] propose an example filter query when discussing semantic provenance management for e-Science. Groth [78] describes an algorithm, D-PQuery, for determining the provenance of data from distributed sources of provenance information in a parallel fashion. The D-PQuery algorithm is made up of six steps: translate, filter, traverse, consolidate, pare and merge.

Our research focuses on filtering during provenance capture and we propose several use-inspired filters to keep provenance traces relevant to the queries that are known prior to the application start. In addition, since the raw provenance data is usually in the form of semi-structured log files [72], recovering the complete provenance information from the raw data is costly and unnecessary, so instead we perform on-demand recovering and processing, which are inspired by the dynamic program slicing technique that is reviewed below.

Program slicing is a well-explored technique in software engineering. Intuitively, program slicing attempts to provide a concise explanation of a bug or anomalous program behavior, in the form of a fragment of the program that shows only those parts "relevant" to the bug or anomaly. Cheney [45] argues that there is a compelling analogy between program slicing and data provenance and defines the dependency provenance of an output as the set of all input fields on which it depends (a data slice). However, the data slice alone does not explain why there are dependences, thus we propose the provenance slice as a combination of data slice and its related program fragment (program slice).

The concept of program slicing is first introduced by Mark Weiser [157]. He introduced program slicing as a debugging aid and gave the first static slicing algorithm. Since then a great deal of research has been conducted on static slicing and a survey of many of the proposed techniques and tools can be found in [151]. It is widely recognized that for programs that make extensive use of pointers, the highly conservative nature of data dependency analysis leads to highly imprecise and considerably larger slices. Recognizing the need for accurate slicing, Korel and Laski propose the idea of dynamic slicing [7], where the data dependences that are exercised during a program execution are captured precisely and saved. It has been shown that precise dynamic slices can be considerably smaller than static slices [84, 152].

While precise dynamic slices can be very useful, it is also well known that computing them is expensive, as it needs to build the dependence graph from the program's execution trace before slicing – so called preprocessing. To address this challenge, Zhang et al. [166] present the design and evaluation of three precise dynamic slicing algorithms called the full preprocessing (FP), no preprocessing (NP) and limited preprocessing (LP) algorithms. The no preprocessing (NP) algorithm does not perform any preprocessing but rather during slicing it uses demand driven analysis that recovers dynamic dependencies and caches the recovered dependencies for potential future reuse. We derive our non-preprocessing provenance slicing technique from this no preprocessing (NP) algorithm.

2.2.3 Stream Processing of Provenance Data

Stream processing is a new computing paradigm that enables continuous and real-time analysis of massive streaming data. In contrast to traditional data analysis approaches, stream computing does not require data to be persisted before processing. Earlier work on processing real time streams of data [3, 17, 39, 126] focuses on high velocity data that arrives continuously (as streams of indefinite duration).

The research on stream provenance has been focused on the provenance for data streams. The existing work in that field can be categorized as coarse-grained provenance method that identifies dependencies between streams or sets of streams, and fine-grained provenance method that identifies dependencies among individual stream elements.

Towards coarse-grained provenance, Vijayakumar and Plale [154, 155] define provenance of data streams as information that helps to determine the derivation history of a data product, where the data product is the derived time bounded stream. A similar coarse-grained technique for recording provenance has been used in sensor archive systems [51, 95]. In this technique, processing modules of archive systems have to employ standardized logging methods to capture only important events during their processing.

Towards fine-grained stream provenance, Sansrimahachai et al. [136] propose a fine-grained provenance solution called Stream Ancestor Function – a reverse mapping function used to express precise dependencies between input and output stream elements. De Pauw et al. [54] provide a visual and analytic environment based on fine-grained provenance to support debugging, performance analysis, and troubleshooting for stream processing applications. Wang et al. [156] propose a hybrid provenance model called Time-Value Centric (TVC) provenance for systems processing high-volume, continuous medical streams.

However, our study focuses on the continuous processing of provenance data streams, which has been largely unexplored. The most relevant work is Sansrimahachai et al. [135] on tracking fine-grained provenance in stream processing systems. They develop an on-the-fly provenance tracking service that performs provenance queries dynamically over streams of provenance assertions without requiring the assertions to be stored persistently. However, their focus is on provenance tracking by essentially pre-computing the query results at each stream operation and storing results into provenance assertions as the provenance-related property.

There is existing work on provenance collection that treats provenance data as continuously generating events. For example, the ingest API of Komado [148] receives provenance relationships and attributes as XML messages; SPADEv2 [71] includes a number of reporters that transparently transform computational activity into provenance events. However, they do not process provenance events as in a stream and the stream model of provenance has not been defined. In this work, we present our early result of defining a stream model for provenance events. Our preliminary model only covers the dependencies between data products (analogous to the "Derivation and Revision" in W3C PROV [23]) and their temporal ordering. We demonstrate that our model is sufficient for the continuous backward provenance query.

Provenance can be represented as a DAG, and there is work on querying provenance graph databases [115, 153]. Our study focuses on the continuous querying of massive provenance data streams. McGregor [111] survey the state-of-art algorithms for processing massive graphs as streams, most of which focus on the semi-streaming model that have $O(n\text{polylog}n)$ space. There has been little research on the stream processing of graph queries, and the closest related work is the stream processing of XPath queries [79] and SPARQL queries [15, 21]. XPath queries need to consider the relationships between XML messages that are similar to graph edges, and SPARQL queries are performed on RDF graphs. However, these extended SPARQL languages are developed for specific goals such as to support semantic-based event processing and reasoning on abstractions, not to support typical graph analysis based on the patterns in nodes and paths. The same holds true for XPath queries.

There is also a little work on analyzing graph streams [20, 66, 138]. In this case, the edges of the graph are received continuously over time, and the objective is to perform the computation using small amount of memory (preferably sub-linear in the number of nodes n) and a smaller number of passes. A typical approach is to construct a synopsis from the graph stream, and leverage it for the purpose of structural analysis [50]. We take the same approach to extend our temporal representation of persistent provenance to be used as the synopsis of the stream elements within the current sidling window to support data mining.

To the best of our knowledge, we are the first to define the stream model of provenance graph, and to develop stream processing techniques to support the query and analysis.

2.3 Data Mining on Provenance

Margo and Smogor [110] use data mining and machine learning techniques to extract semantic information from I/O provenance gathered through the file system interface of a computer. The mining step reduces the large, singular provenance graph to a small number of per-file features. Our research is complementary in that we examine a collection of provenance graphs and treat a whole provenance graph as an entity. Like Margo's work, we also reduce the size and dimensionality of provenance, but we achieve this by partitioning the graph and applying statistical post-processing. Phala [99] uses provenance information as a new experience-based knowledge source, and utilizes the information to suggest possible completion scenarios to workflow graphs. It does not, however, provide descriptive knowledge for a large provenance dataset.

Clustering techniques have been applied to workflow graphs. A workflow script or graph is either an abstract or implementation plan of execution. A provenance graph, on the other hand, is a record of execution. A provenance record may or may not have the benefit of an accompanying workflow script, so a workflow graph is in some cases a coarse approximation of provenance graph. Santos et al. [137] apply clustering techniques to organize large collections of workflow graphs. They propose two different representations: the labeled workflow graph and the multidimensional vector. However, their representation using labeled workflow graphs becomes too large if the workflow is big, and the structural information is completely lost if using a multidimensional vector. Jung and Bae [90] propose the cluster process model represented as a weighted complete dependency graph. Similarities among graph vectors are measured based on relative frequency of each activity and transition. It has the same scalability issue as Santos et al. Our work addresses the problem of mining and discovering knowledge from provenance graphs, while overcoming the scalability issue by reducing the large provenance graph to a small temporal representation sequence, and retaining structural information together with attribute information.

There is existing work that studies the workflow execution data, in particular to collect, discover and predicate workflow errors. The temporal representation that we propose is for all kinds of provenance, but we evaluate its usefulness by representing and mining workflow provenance that leads to the discovery of failed workflow executions. Thus, we share many commonalities with these works in terms of motivations and techniques. For example, Benabdelkader et al. [24] develop a software tool that collects execution information from various sources, and the information includes the error occurrence that can be used to visually explore and trace the source of errors. We use the *k-means* clustering algorithm to find centroid provenance graphs that can be further visualized to help understand the experiment and explore failures; Samak et al. [134] use clustering-based classification for early detection of failing workflows and a regression tree analysis to identify problematic resources and application job types. We also use the *k-means* clustering algorithm to separate the failed workflow executions from normal executions, but we adopt a graph matching algorithm to locate the root-causes; Silva et al. [68] present a practical method for autonomous detection and handling of operational incidents in workflow activities. They model workflow activities as Fuzzy Finite State Machines (FuSM) where degrees of membership are computed from metrics measuring long-tail effect, application efficiency, data transfer issues, and site-specific problems. While they use association rules for a predictive purpose, we use association rules for discovering variation patterns offline. However, the most important difference between our work and all others is probably that we make little assumption on the dataset: we do not have the status information of workflows and their processes to tell whether they completed or failed; while clustering the provenance graphs, we do not know which workflow type the provenance graph belongs to (though we use the information of workflow type for performance evaluation). We experiment with datasets that come from the Karma [147] system that gathers structured and unstructured provenance data without the assumption of a single and coherent system.

How to treat data with temporal dependencies is another problem in the discovery process of hidden information. The ultimate goal of data mining is to discover hidden relations between sequences and subsequences of events [16]. Provenance information stored in a form amenable to representation as a graph has an implicit temporal ordering, which can be exploited for data clustering and relationship discovery. To our best knowledge, there is no previous study on discovering the hidden relations in provenance.

Though there is not much work on mining provenance data, the data mining on graph in general has been well studied. The existing algorithms on clustering graph data include both classical graph clustering algorithms as well as algorithms for clustering XML data – XML data is quite similar to graph data in terms of how the data is organized structurally. Two main techniques [6] used for clustering XML and graph data are: structural distance-based approach that computes structural distances between graphs or XML documents [36], and structural summary based approach that creates summaries from the underlying graphs and documents for clustering [5]. It has been shown in [5] that a structural summary based approach is significantly superior to a similarity function based approach as presented in [36], and our temporal representation can be considered as a structural summary based approach. Finally, for graph classification where the task is to classify unlabeled graphs based on labeled graphs, the main methods [6] include kernel-based graph classification and boosting-based graph classification [131].

We also extend our temporal representation to create synopses in the provenance stream model, which can be used for clustering, classification and association rules mining of provenance streams. There is related work on mining graph streams in areas such as social networks, communication networks, and web log analysis. Graph streams are very challenging to mine, because the structure of the graph needs to be mined in real time. Therefore, a typical approach is to construct a synopsis from the graph stream, and leverage it for the purpose of structural analysis [6]. It has been shown in [66] how to summarize the graph in such a way that the underlying distances are preserved. Therefore, this summarization can be used for distance-based applications such as the shortest path problem. Furthermore, the graph synopses are used to estimate the aggregate structural properties of the underlying graphs. Cormode and Muthukrishnan [50] propose a set of techniques for estimating the statistics of the degrees in the underlying graph stream, including sketches, sampling, hashing and distinct counting. Methods have been proposed for determining the moments of the degrees, determining heavy hitter degrees, and determining range sums of degrees. In addition, techniques have been proposed in [20] to perform space-efficient reductions in data streams. This reduction has been used in order to count triangles in the data stream. A particularly useful application in graph streams is that of the problem of PageRank. Atish Das et al. [138] use a sampling technique to estimate page rank for graph streams. The idea is to sample the nodes in the graph independently and perform random walks starting from these nodes.

2.4 Usage of Data Provenance

Research on data provenance has been established in many different environments, including database [33], workflow [146], web and grid services [149], cloud [120], network [176], geospatial service [163], visualization [34], etc. Data provenance has a wide range of usage, for example, Gobe [73] summarizes five major reasons to use provenance: reliability and quality; justification and audit; re-usability, reproducibility and repeatability; change and evolution; ownership, security, credit and copyright. This thesis also expands the use of data provenance captured from computer network, workflow, and agent-based model, so we focus on the existing work in these fields below.

Davidson and Freire [52] describe a number of emerging applications for workflow provenance: to reproduce data, to simplify exploratory processes, to enable social analysis of scientific data, and to be used in education. More specifically, Kepler [11] uses provenance to enable efficient workflow rerun; Vistrails [142] uses provenance to support comparison of data products as well as their corresponding workflows; Scheidegger et al. [139] use provenance to simplify the construction of workflows. In our research, we propose a temporal representation of provenance, based on which we apply clustering algorithms to detect failure executions, and classification algorithms to predict the workflow type of a future provenance graph; and we also develop advanced visualization techniques including a provenance graph matching algorithm to better compare workflow executions.

Zhou et al. [175] survey a list of existing work in the networking literature that motivates the use of network provenance. They classify the use-cases as real-time diagnostics, forensics, accountability, and trust management. They argue that network accountability and forensic analysis can be posed as data provenance computations and queries over distributed streams. In our research, we demonstrate that visualization of network provenance can be used to explore, present, and debug various network experiments.

Bennett et al. [25] illustrates the importance of explicitly considering provenance in agent-based modeling through the development of a spatially explicit agent-based land use simulation framework. While their research is speculative, we implement the automated provenance tracing and capture for NetLogo and demonstrate some example provenance queries that can help understand and analyze the simulation. Furthermore, our research allows the in-motion processing of live provenance data so that we can analyze the simulation progress in real-time, and can enable operations like online steering [140] and automated parameter readjustment. There is a relevant but different concept introduced by Acar [4] in programming language called self-adjust computation. In self-adjusting computation, programs respond automatically and efficiently to modifications to their data by tracking the dynamic data dependences of the computation and incrementally updating their output as needed. The dynamic data dependencies are captured into a graph that is similar to provenance graph. However, a dynamic dependency graph (DDG) only contains method calls, while a provenance graph could also include intermediate data and people that is involved. In self-adjusting computation, a change propagation algorithm is developed to find the method calls that need to be recomputed when there are changes in the external input, which can be posed as a forward provenance query. In our research on querying the provenance stream, we aim to address a more difficult challenge – the backward provenance query that finds out the input parameters that influenced an output data item.

2.5 Summary

In this chapter, we first discuss the existing work in areas that are related to the analysis of provenance. While there are tools for provenance visualization, they generally focus on small graphs, and none of them incorporates graph matching for comparing multiple graphs. Our earlier work on provenance visualization results in several techniques, including layout algorithms and graph matching, for interactive navigation, manipulation, and analysis of large-scale data provenance. Although a lot of effort has been put on improving the performance of provenance queries, most existing work, even if discussed within a distributed environment, only provides provenance retrieval functionality based on complete provenance. Instead, we develop several use-inspired filters during provenance capture to keep only relevant provenance traces; we perform on-demand recovering and processing on the raw provenance traces, to avoid unnecessary construction and maintenance of provenance information; we propose a stream processing solution that does not persist the data and provides real-time analytical results. There has been little research on mining provenance data, and our research addresses the Big Data challenges by proposing a scalable and effective temporal representation of provenance for mining purposes.

Finally, we review the existing usage of provenance, and demonstrate that our study further expands the use of provenance in computer network, workflow, and agent-based simulation. In computer network, we demonstrate network provenance can be used to explore, present, and debug various network experiments. In workflow system, we use provenance to detect and diagnose failure executions, and to predict the workflow type of a provenance graph. In agent-based simulation, provenance captures the internal dynamics that can be used to debug the simulation and analyze the model and the simulation results.

By enabling the real-time processing of live provenance, we are even able to support the online debugging and calibration of simulations.

Chapter 3

Background Information and Earlier Work

In this chapter, we first present the background information on provenance modeling and our earlier work on provenance visualization and capture, and then introduce the provenance datasets that we use in our study.

3.1 Provenance Representation and Interoperability Models

We start with introducing the widely adopted Open Provenance Model (OPM) [117] and its successor W3C PROV [23], which provide a good set of vocabularies that can be used to model the provenance.

3.1.1 Open Provenance Model

The Open Provenance Model (OPM) is designed to exchange provenance information between systems and represents the different elements of provenance as directed graphs. OPM has been a prominent interoperability model amongst different provenance systems, in use by a number of systems such as SPADE [71] and Karma [147]. It is based primarily on three kinds of nodes as defined in the core specification:

- Artifact: An immutable piece of state

- Process: Action or series of actions performed on or caused by artifacts, and resulting in new artifacts.
- Agent: Contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, or affecting its execution.

The edges in the graph represent causal dependencies and refer to the past execution between a source node (the effect) and a target node (the cause). There are primarily five types of causal dependencies in OPM:

- used: A process used an artifact.
- wasGeneratedBy: An artifact was generated by a process.
- wasTriggeredBy: A process was triggered by another process.
- wasDerivedFrom: An artifact was derived from another artifact.
- wasControlledBy: A process was controlled by an agent.

Figure 3.1(a) shows the nodes and edges of OPM graphs. It is also possible to extend the basic elements of an OPM graph based on the use case.

3.1.2 W3C PROV

The W3C recommended PROV model [23] is a relatively new data model for provenance interchange on the Web. W3C PROV has many provenance concepts that are not captured in OPM, such as versioning, mechanisms for linking the different descriptions of the same thing, containment relationships and collections and so on. The key concepts in PROV consist of entities, activities and agents (see Figure 3.1(b)).

- Entity: Any physical, digital, conceptual, or other kinds of thing.
- Activity: Activities are how entities come into existence and how their attributes change to become new entities, often using previously existing entities.

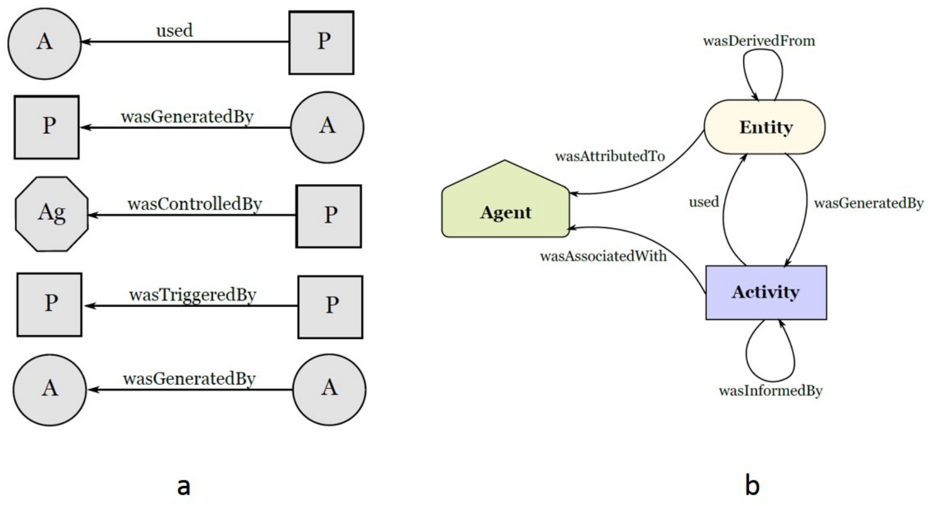


Figure 3.1: Basic nodes and relationships in OPM (Figure a, where A stands for artifact, P stands for process and Ag stands for agent) and PROV (Figure b).

- Agent: An agent takes a role in an activity such that the agent can be assigned some degree of responsibility for the activity taking place.

The key relations between the different PROV concepts are:

- Usage and Generation: Activities “generate” new entities and make “use” of new entities.
- Responsibility: An agent can be “associated with” an activity and an entity can be “attributed to” an agent. An agent can also act “on behalf of” another agent.
- Derivation: When one entity’s existence, content, or characteristics are at least partly due to another entity, then the former “was derived from” the latter.
- Communication: If two activities exchange some unspecified entity then the informed activity is said to be “informed by” the informant.

3.2 Provenance Capture in ABM

We next describe our work in capturing provenance from agent-based simulations. Researchers who use agent-based models (ABM) to model social patterns often focus on the model's aggregate phenomena. However, aggregation of individuals complicates the understanding of agent interactions and the uniqueness of individuals. We develop a method for tracing and capturing the provenance of individuals and their interactions in the NetLogo ABM, and from this create a "dependency provenance slice", which combines a data slice and a program slice to yield insights into the cause-effect relations among system behaviors.

3.2.1 Agent Based Model

An agent-based model (ABM) is a simulation of distributed decision-makers (agents) who interact through prescribed rules. ABM has been effective in studies such as complex adaptive spatial system (CASS) [25], ecological modeling [12], and neuromechanical modeling [88], because of its ability to represent heterogeneous individuals and the interactions between them. Agent-based models (ABM) often focus on the emergent outcomes of aggregated behaviors. The fundamental philosophy in ABMs of methodological individualism warns that aggregation may yield misleading results, and advocates a focus on the uniqueness of individuals and interactions [12]. In addition, the interactions between agents are inherently associated with cause-effect relations in the transition of system states through time. These cause-effect relations are important and necessary for an understanding of a complex process, but elucidating these relations poses a significant challenge to the research community because of the complex dynamics in ABM [25]. We have demonstrated earlier [43] that data provenance is capable of capturing the cause-effect relations in agent based models, aiding in understanding the complex processes and to enable the analysis of internal dynamics that were previously hidden as black box.

NetLogo [160] is an agent-based modeling platform in wide use in research and education worldwide. NetLogo has its own Logo programming language, containing high level primitives for performing batch operations over a group of agents. To understand the scale of provenance generated by ABM, we perform an empirical study using a classic NetLogo model for ecologists called wolf-sheep-predation. The model, designed by Uri Wilensky [159], explores the stability of predator-prey ecosystems. A system is stable if it trends to maintaining itself over time, despite fluctuations in population sizes. A system is similarly unstable if it trends to extinction of one or more species involved. The "wolf-sheep-predator" model has two variations. In the first variation, wolves and sheep wander the landscape randomly while wolves are busy looking for sheep to prey on. Each model step is an action by a wolf that costs them energy units, and they must eat sheep in order to replenish their energy. When a wolf runs out of energy, it dies. Each of wolf and sheep has a fixed probability of reproducing at each time step. This variation produces interesting population dynamics, but is ultimately unstable. The second variation includes grass in addition to wolves and sheep. The behavior of the wolves is identical to the first variation, however this time the sheep must eat grass in order to maintain their energy – when they run out of energy they die. Once grass is eaten it will only regrow after a fixed amount of time. This variation is more complex than the first, but it is generally stable. We choose the first variation to capture the Big Data provenance when the model is unstable so that it grows exponentially.

We also experiment with a distributed agricultural decision making model that we are developing at Indiana University to study the impact of climate change on food security [35]. The current model has 53 thousand household agents in Monze District, Zambia, carrying out farming activities biweekly over a long period of time (years), on approximately 89 thousand hectares of cropland. Each of the household agents makes decisions on farming and labor sharing based on its current status and the context. While NetLogo itself cannot run simulations at this scale, we create a framework that can split the large district level model into small ward level models and run the simulations in parallel in the Big Red II supercomputer at Indiana University.

3.2.2 Provenance in ABM

The first step in provenance capture is to decide what information to capture as provenance. There are various types of information at different levels of granularities that can be collected as provenance during the data generation process. For example, there are "why provenance," "where provenance," "how provenance," and dependency provenance, and each can be captured at system, middle-ware or application levels [174]. Choosing the scope and granularity of provenance to capture is often determined based on the characteristics of the application and system that generates the data, and the use cases on the resulting provenance dataset.

Dependency provenance is the information relating each part of the output of a query to a set of parts of the input on which the output depends [46]. Dependency provenance can be used to compute data slices, or summaries of the parts of the input relevant to a given part of the output. We want to define dependency provenance in ABM similar to this, but we also want to compute the program slice (relevant processes) as a complement to the data slice. While the data slice tells what the relevant input data is, the program slice tells how the output data depends on the input data. Specifically, the dependency provenance in ABM that we propose contains the information of:

- All data products and their dependencies;
- Procedures associated with these dependencies.

The next step is to select a standard provenance model to represent the captured information. We choose W3C PROV [23] to record the dependency provenance in ABM, because PROV allows us to express the provenance of agents and the evolution of a variable. Unlike its predecessor OPM, an agent in PROV is also a particular type of entity and the PROV has the concept of versions. The mapping of ABM provenance to PROV is accomplished by first identifying the entities and dependencies in a NetLogo program, then mapping concepts in PROV to them (see Table 3.2.2).

Table 3.1: Mappings from PROV ontology to NetLogo concepts

Concept in PROV	**Concept in NetLogo**	**Code example**
Agent	Agent	breed [wolves wolf] *;; wolf is an agent*
Activity	Execution of a procedure	ask sheep [move *;; an activity* ...]
Entity	State of a global/agent/local variable	set color white *;; the current state of color (an agent variable) is an entity*
Relationship: used	1) Procedure reads the current value of a variable 2) Procedure depends on the current value of a variable	1) to reproduce-sheep ... set energy (energy / 2) *;; the activity "reproduce-sheep" used the entity "energy"* ... end 2) if grass? [... eat-grass *;; the activity "eat-grass" used the entity "grass?"*
Relationship: wasGenerateBy	Procedure writes the value of a variable	to reproduce-sheep ... set energy (energy / 2) *;; the entity"energy" was generated by activity "reproduce-sheep"* ... end

Relationship: wasDerivedFrom	1) A statement reads var2 before writing var1 2) A statement writing var1 depends on var2	1) set energy random (2 * wolf-gain-from-food) *;; the entity "energy" was derived from entity "wolf-gain-from-food"* 2) if grass? [set energy energy 1 *;; the entity "energy" was derived from entity "grass?"*

Relationship: wasRevisionOf	If an variable was derived from itself	to reproduce-sheep
		  ...
		  set energy (energy / 2) *;; the entity "energy" was a revision of itself*
		  ...
		end

Relationship: wasInformedBy	A procedure is invoked inside another procedure	to go
		  ...
		  ask sheep [
		    move *;; activity "move" was informed by activity "go"*
		  ...
		end

Relationship: wasAssociatedWith	A procedure is invoked by an agent	ask sheep [move *;; the activity "move" was associated with a sheep agent*
Relationship: wasAttributedTo	Variable belongs to an agent	sheep-own [energy] *;; the entity "energy" was attributed to a sheep agent*

Relationship: alternateOf	A variable is a reference to an agent	let prey one-of sheep-here *;; the entity "prey" is an alternate of a sheep agent (an agent is also an entity in PROV)*

3.2.3 Provenance Tracing

A primary source of provenance is the log file, and writing parsers to extract provenance from logs is a common practice in provenance capture [72]. In the case that the original logs do not contain enough provenance information, instrumentation is widely used to output the provenance information into the logs, such as in capturing the NASA AMSR-E provenance archive dataset. We follow the same approach to collect the fine grained provenance information at statement level by instrumenting the agent-based model's source code.

We capture the provenance of a NetLogo model by adding probes to the model source code. These probes generate provenance traces. We propose three types of probes for this purpose, probes that log:

- Procedure invocations (Type 1),
- Write and read operations (Type 2),
- Conditional statements (Type 3).

Type 1 probes generate information used to compute the program slice. Type 2 and Type 3 probes produce provenance about data dependencies that is used to compute the data slice.

The open source tool that we built to implement our abstractions is called PIN (acronym for "Provenance in NetLogo") [43]. It can trace, capture, query and visualize the dependency provenance in NetLogo. It consists of four main components: a source code analyzer used to automatically add probes to the model's source code, a NetLogo extension for capturing the provenance traces generated from probes, a non-preprocessing (NP) provenance slicing technique for computing provenance slices using provenance traces, and a visualization component for visualizing the provenance slices. Figure 3.2 shows how the NetLogo provenance tool works. The tool is compatible with NetLogo version 5.0.3.

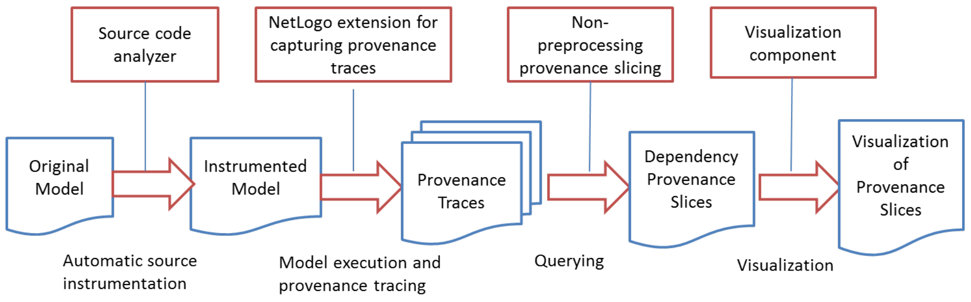

Source code analyzer
NetLogo extension for capturing provenance traces
Non-preprocessing provenance slicing
Visualization component
Original Model
Instrumented Model
Provenance Traces
Dependency Provenance Slices
Visualization of Provenance Slices
Automatic source instrumentation
Model execution and provenance tracing
Querying
Visualization

Figure 3.2: PIN overview: Red rectangles represent major components, and blue document charts represent input and output of each component

To understand the time overhead introduced by provenance tracing and capture, we identify four performance metrics that are measured through timing information gathered during the "Model execution and provenance tracing" step (in Figure 3.2). These metrics are:

- *Execution time*: model execution;
- *Message passing time*: the probes pass the trace messages to the NetLogo extension by calling the primitive "provenance:write";
- *Collecting time*: the NetLogo extension collects trace messages and their context information from the model;
- *Writing time*: the NetLogo extension writes traces into a provenance log file.

Execution time is calculated by running the model without instrumentation, and the others are computed indirectly by running simulations with different versions of the NetLogo extension and calculating the average time differences. The model is running in NetLogo 5.0.3 on a single machine with Win8 64bit OS, 8GB memory, and a Core i5 2.53GHz dual core CPU. Figure 3.3 summarizes the results.

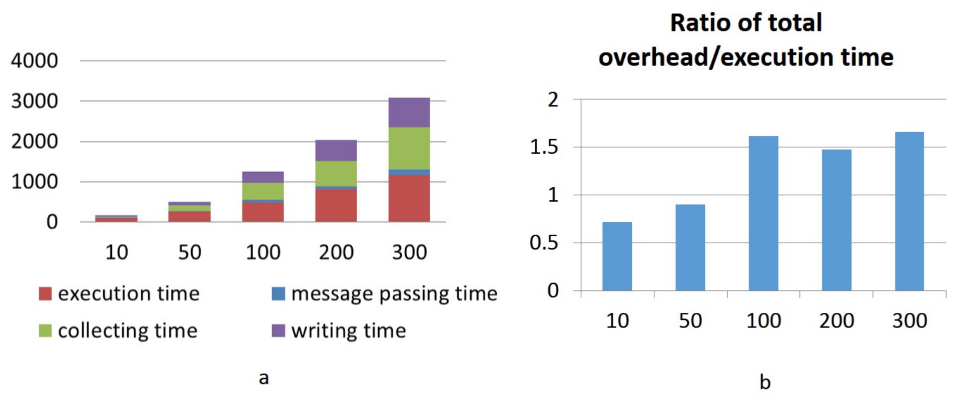


Figure 3.3: Provenance tracing and capture: X-axis is number of iterations run; Y-axis has time cost (ms). (a) shows where time costs lie for up to 300 iterations. (b) shows the ratio of total overhead to the execution time.

From Figure 3.3(a) we see that the “message passing time” is small compared with the “execution time” and “collecting time”, which means that we can turn off tracing once the capture finishes by not processing the trace messages received in the extension. We can also use filters to reduce the “writing time”. Figure 3.3(b) tells that the overhead has the same order of magnitude as the original simulation time. This means that simulation analysis over its provenance is more efficient than traditional statistical analysis over many repeated runs.

3.3 Provenance Visualization

Iliinsky and Steele (2011) [87] identify two categories of data visualization: exploration and explanation. *Exploratory visualization* is designed to support a researcher who has large volumes of data and not certain what is in it, whereas *Explanatory visualization* takes place when a researcher knows what the data has to say, and is trying to tell that story to someone else. In our research, we use the exploratory visualization to identify the important information within provenance data, and use the explanatory visualization to present the results of provenance queries in a meaningful way.

The visualization of provenance, although supported in existing workflow management systems, generally focuses on small (medium) sized provenance data. Big Data provenance visualization differs from prior provenance visualizations in its large volume and high complexity, necessitating techniques such as navigation, abstraction, and layout algorithms to make the large graph meaningful.

In this section, we introduce a number of provenance visualization techniques that we developed for interactive navigation, manipulation, and analysis of large-scale data provenance. We demonstrated in [42] that our visualization tool can support both exploratory and explanatory types of visualization. The visualization techniques are implemented into a Cytoscape [141] plugin.

3.3.1 Requirements

Kunde et. al [96] derive abstract types of user requirements for provenance visualization, including 1) process: the sequence of the process steps is in the center of inspection; 2) results: the intermediate or end results of interactions are in the center of the users view; 3) relationship: the relationship of interactions or actors is important; 4) timeline: the time is important to observe; 5) participation: the correctness of the participants is important; 6) compare: the comparison of two subjects shows the difference between them; 7) interpretation: an individual visualization view depending on the special question of the end-user.

The goal of our visualization research is to serve both broad and narrowly focused audiences, so it addresses each of the above requirements as follows: 1-3) our visualization tool is based on an accepted model for provenance representation, namely, Open Provenance Model (OPM) [117], which denotes entities (processes and artifacts) as nodes, and relationship as edges in a graph. It is able to show a complete graph with both process steps and intermediate (final) results, or abstract graphs focusing on either one of them; 4) OPM is capable of representing time information of nodes and edges, and our visualization tool has a special function called "Play movie by time" that can help a user understand the dynamics through time; 5) participation is represented by agents through "was controlled by" relationships in OPM, so our tool helps a user visually evaluate the correctness of participations; 6) users can compare attributes of nodes using our tool, and can compare two provenance graphs with the graph matching algorithm we improved from DePiero's work [59]; 7) for the last type of user requirement (interpretation), we will show how we satisfy it with customized layout algorithms and visual styles in our use cases.

Prior provenance visualization tends to deal with small graphs, and seldom addresses the issue of graph layout and graph matching. The visualization tool we developed can satisfy different types of user requirements, and can handle large-scale visualization with customized layout algorithm and visual styles.

3.3.2 Provenance Visualization Techniques

The goal of provenance visualization is to help a user navigate provenance. A researcher brings a mental map of what is going on in an experiment, and uses this model to interact with and explore the provenance. We develop a number of visualization techniques that we discuss here. The techniques are implemented into a plugin to Cytoscape, an open source software platform for complex network analysis and visualization. Cytoscape is appropriate for provenance visualization because of its support for detail and overlaying visualizations with additional annotations. The Cytoscape plugin can generate provenance graph visualization by interacting with the Karma provenance server [147] to extract provenance in the form of a graph as XML.

Incremental Loading

Provenance can be very large, not only because a lineage record can be long but because OPM v1.1 [117] supports key value annotations to provenance graphs, the latter of which opens the opportunity for capture of extended information about execution or object creation. To better support visualizations over large graphs, we support reads of provenance graph in XML format with and without annotations. Annotations can be separately queried through the Karma query API. That is, the Karma system generates an OPM compliant XML file that does not have annotations to process or artifacts, and if the visualization tool loads this XML file, it will also establish a background connection to the Karma server, to retrieve annotations when some process or artifact is selected during navigation. This incremental loading allows loading of annotations on demand.

During the interactive visualization, a user may change the graph and the node (edge) attribute, so the visualization tool also supports saving provenance data from Cytoscape to an OPM compliant XML file.

Customized Layout

Layout is a key element to increasing researchers' understanding of large-scale network data provenance. Layout depends on both the nature of provenance data and the user requirement. In fact, it often takes seeing multiple layouts for a researcher to judge which is most meaningful. Cytoscape offers several default layouts that are meaningful for provenance visualization. For example, its hierarchical layout organizes a provenance graph into layers based on relationship such that first causes appear at the top of a visualization and final effects appear at the bottom.

In addition, we designed several customized layouts including:

1. an extended hierarchical layout algorithm that sorts sibling nodes by their time order;
2. a group of layout algorithms specifically for computer network provenance;
3. a string-embed based layout algorithm for provenance data like a history chain.

Visual Style

One of Cytoscape's strengths is its ability to allow a user to encode any attribute of data (e.g., name, type, degree, weight, and expression data) as a visual property (such as color, size, transparency, or font type). A set of these encoded or mapped attributes is called a Visual Style. We create a default visual style for provenance graphs, using green to color processes, magenta for artifacts, red for agents, and different colors and arrow styles for different types of edges.

Figure 3.4 shows that a special layout algorithm and visual style for computer network provenance can enable side-by-side performance comparison of different experiment configurations.

Abstract Views

The complexity of the provenance relationships can result in graphs that overwhelm the researcher. We identify two approaches to abstracting out complexity, including:

1. clustering neighbor nodes;
2. process/artifact elimination.

Neighbor nodes can be clustered and reduced to a single node by an action in our Cytoscape plugin. This is useful when exploring a provenance graph and could be applied to deal with graphs having a large number of artifact nodes. Figure 3.5 shows the abstract view of the provenance generated from breadth first search experiment under the hierarchical layout. The experiment uses Twister MapReduce [63] and runs on PlanetLab [49], and it has one master node and ten slave nodes. However, the provenance visualization of this experiment has far too many artifact nodes connected to an extremely small number of process nodes. By clustering data artifacts, we are able to see where and how each process ran, and what output (subgraph) was generated. A deeper investigation of the results shown in Figure 3.5 reveals that the last (rightmost) BFS process ("run_BFS") was not triggered successfully, and the application as a whole failed without output.

OPM v1.1 introduces two causal dependencies "was triggered by" and "was derived from" as summary edges for a process view (where an intermediary artifact was unknown) and a data view (where an intermediary process was unknown), respectively. It also provides completion rules that can be used to transform a graph between a complete view, a

Run_id	frame_duration	Number of attacker/user	attack_backoff_start	attack_request_retry	bw_backoff_start	bw_request_retry
1	0.004	20/80	1	2	1	2
244	0.01	20/80	1	2	1	2
487	0.02	20/80	1	2	1	2

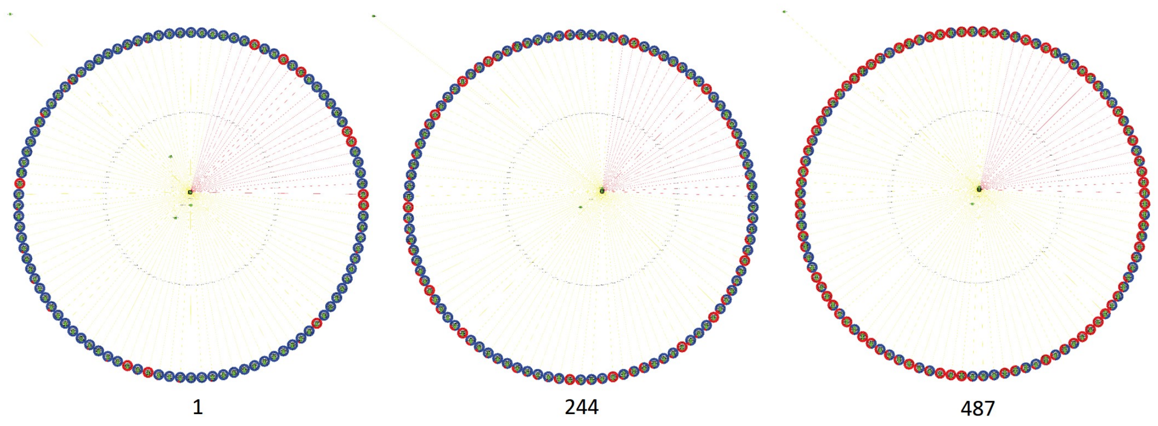


Figure 3.4: Capture provenance of DDoS Attacks Exploiting WiMAX System Parameters by researchers at Clemson [58]. Experiment uses 100 subscribers with varied configurations of 6 parameters running in a simulator. The Base Station (BS) is displayed at the center, and all Subscriber Stations (SS) are displayed in the large circle, surrounded by a small circle of their dropped packets (red) and/or received packets (blue). Normal user SS is connected with yellow edge, while attacker SS is linked via red edge. The layout algorithm is based on the geographical locations of WiMAX stations and it organizes the massive amount of nodes in a meaningful way. The visual style (different colors of nodes and edges) allows the audience to capture the information in an intuitive and visual way. By comparing the visualizations side by side, we can easily tell that the number of packets dropped increases as frame_duration increases from 0.004s to 0.02s.

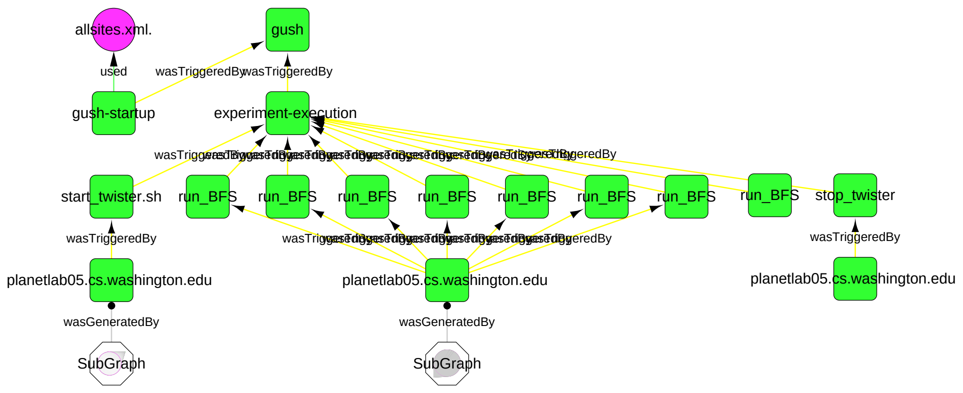


Figure 3.5: Abstract view of a breadth first search application. Subgraph nodes (shown as two octagons at bottom) result from clustering neighbor nodes; a user can navigate into subgraphs by clicking on them. In this way, the number of nodes and the complexity of graph are both reduced so that the most important information is revealed.

process view, and a data view. We developed two techniques of transformation based on completion rules, namely, process elimination and artifact elimination. Process elimination eliminates all process nodes that link two artifact nodes with an outgoing edge "used" and an incoming edge "was generated by". The process node is replaced by a new edge "was derived from" (see Figure 3.7). Artifact elimination eliminates an artifact node in the case where the outgoing edge of an artifact represents the relationship "was generated by" and its incoming edge represents the relationship "used". The artifact will be replaced by a new edge between these two process nodes that represents the relationship "was triggered by" (see Figure 3.8).

The same process elimination technology can be applied to the W3C PROV model: we can replace the process nodes connecting two artifact nodes that have an outgoing edge "used" and an incoming edge "was generated by" with an edge "was derived from." For ex-

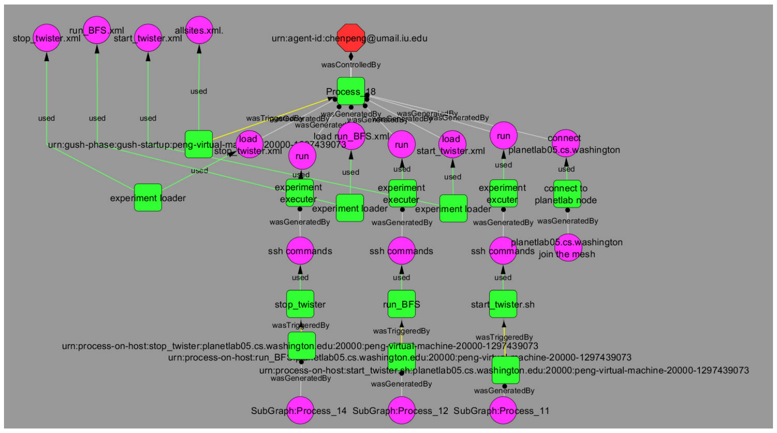


Figure 3.6: Complete provenance for a breadth-first search application, before the process elimination in Figure 3.7 and the artifact elimination in Figure 3.8.

ample, we can provide a data-dependency-only view when visualizing forward provenance by utilizing the process elimination technique (see Figure 3.10).

Graph Comparison

Comparing two provenance graphs is a key piece of provenance analysis. We improve and implement the Direct Classification of node Attendance (DCA) algorithm to compare two provenance graphs by finding matched subgraphs and unmatched nodes.

Direct classification of node Attendance (DCA) finds isomorphisms between graphs and subgraphs [59]. The method first evaluates evidence describing the likelihood of a node's predicted attendance in another graph. The evidence is based on measures that are local to each node, including connectivity and the attributes of adjacent nodes and edges. It then finds a node-to-node mapping and reorders nodes to permit a direct comparison to be made between the resultant graphs. The algorithm is of polynomial order. It yields approximate results, maintaining a performance level for subgraph isomorphisms at or above 95%

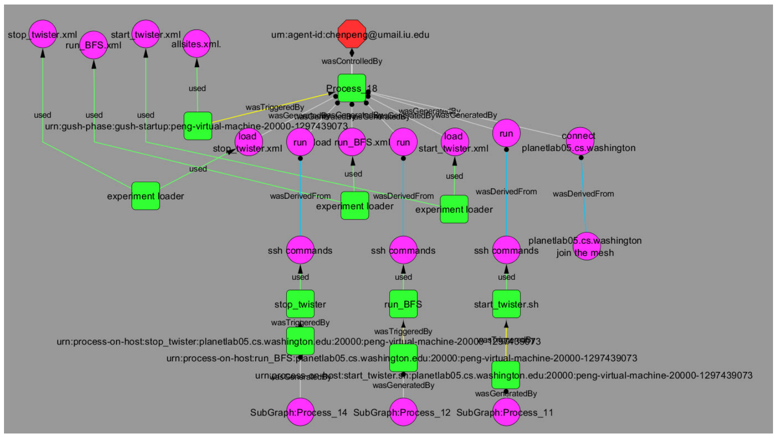

stop_twister.xml
run_BFS.xml
start_twister.xml
allsites.xml.
urn:agent-id:chenpeng@umail.iu.edu
wasControlledBy
Process_18
used
wasTriggeredBy
wasGeneratedBy
urn:gush-phase:gush-startup:peng-virtual-machine-20000-1297439073
load stop_twister.xml
run
load run_BFS.xml
run
load start_twister.xml
run
connect planetlab05.cs.washington
experiment loader
wasDerivedFrom
ssh commands
planetlab05.cs.washington join the mesh
stop_twister
run_BFS
start_twister.sh
urn:process-on-host:stop_twister:planetlab05.cs.washington.edu:20000:peng-virtual-machine-20000-1297439073
urn:process-on-host:run_BFS:planetlab05.cs.washington.edu:20000:peng-virtual-machine-20000-1297439073
urn:process-on-host:start_twister.sh:planetlab05.cs.washington.edu:20000:peng-virtual-machine-20000-1297439073
SubGraph:Process_14
SubGraph:Process_12
SubGraph:Process_11

Figure 3.7: Breadth-first search application after process elimination. Process nodes between artifact nodes in Figure 3.6 are now replaced by new edges "was derived from." This can eliminate the unwanted process nodes without misrepresenting the provenance information.

under a wide variety of conditions and with varying levels of noise. The performance on the full size comparisons associated with graph isomorphisms has been found to be 100/100, also under a variety of conditions [59]. However, the result of the original DCA algorithm depends on the input order. That is, the result of matching graph A to graph B is different from matching B to A. We improve the DCA algorithm by making the matching process consistent regardless of the input order. Besides, the original algorithm forms only one pair of matched subgraphs and is purely based on values of node attendance. We extend that to form multiple pairs of matched subgraphs and to consider the nodes' topology during the formation.

Algorithm 1 is the improved version of the DCA algorithm. Steps 4 to 8 are the changes we made from the original DCA algorithm to make the result of step 19 independent of input order and more reasonable. Considering an extreme case in the original DCA algorithm,

Algorithm 1 Algorithm that finds multiple pairs of matched subgraphs between two provenance graphs

```
function MATCH($G_1$, $G_2$)
    for nodes $n_i$ in $G_1$ do                    ▷ Compare each pair of nodes in each graph
        for nodes $n_j$ in $G_2$ do
            if number of incident edges: $n_i < n_j$ then    ▷ Always find best matched edges and adjacent nodes for node $n_i$ or $n_j$ with larger degree, but the number of matches equals to the smaller degree of $n_i$, $n_j$
                $n_1$= $n_i$ and $n_2$ = $n_j$
            else
                $n_1$= $n_j$ and $n_2$ = $n_i$
            end if
            for edge $e_k^1$ incident to $n_1$ do          ▷ Compare edges incident to current nodes
                for edge $e_l^2$ incident to $n_2$ do
                    Compare connectivity along $e_k^1$ with $e_l^2$
                    Compare edge properties of $e_k^1$ with $e_l^2$
                    Compare adjacent nodes' properties of $n_1$ with $n_2$
                    Combine results of step 11 to 13
                end for
                Save comparison of best matched edges and adjacent nodes
            end for
            Compare node properties of $n_i$ with $n_j$        ▷ Compare current nodes
            Combine result of step 16 and 18 to form attendance rating of $n_i$ to $n_j$, namely $attendance(n_i, n_j)$    ▷ Find similarity of current nodes
        end for
    end for
```

Algorithm 1 Algorithm that finds multiple pairs of matched subgraphs between two provenance graphs

```
22:     Let list of matched node-pair: $S \leftarrow empty$
23:     loop                                                          ▷ Find all matchings
24:         Let list of current matched node-pair: $L \leftarrow empty$
25:         Let peak attendance $A_1 \leftarrow 0$
26:         Let best matched node-pair $P \leftarrow null$   ▷ Find the node-pair with peak attendance $A_1$,
   based on which to grow a new matched subgraph
27:         for nodes $n_i$ in $G_1$ do
28:             for nodes $n_j$ in $G_2$ do
29:                 if $n_i, n_j$ not exist in $S$ and $A_1 < attendance(n_i, n_j)$ then
30:                     $A_1 \leftarrow attendance(n_i, n_j)$
31:                     Let $P \leftarrow < n_i, n_j >$
32:                 end if
33:             end for
34:         end for
35:         if $A_1 > THRESHOLD$ then                  ▷ Add the best matched node-pair into $L$
36:             add node-pair $P$ into $L$
37:             add node-pair $P$ into $S$
38:         else
39:             break
40:         end if
```

Algorithm 1 Algorithm that finds multiple pairs of matched subgraphs between two provenance graphs

```
41:         loop                                                    ▷ Grow the subgraph
42:             for node-pair $< n_i, n_j >$ in $L$ do
43:                 for node $n_k^i$ connected to $n_i$ do
44:                     Let peak attendance $A_2 \leftarrow 0$
45:                     for node $n_l^j$ connected to $n_j$ do
46:                         if $< n_k^i, n_l^j >$ not exist in $S$ and $A_2 < attendance(n_k^i, n_l^j)$ then
47:                             $A_2 \leftarrow attendance(n_k^i, n_l^j)$
48:                         end if
49:                     end for
50:                 end for
51:                 if $A_2 > THRESHOLD$ then
52:                     add $< n_k^i , n_l^j >$ into $L$
53:                     add $< n_k^i , n_l^j >$ into $S$
54:                 else
55:                     break
56:                 end if
57:             end for
58:         end loop
59:     end loop
60:     return $S$
61: end function
```

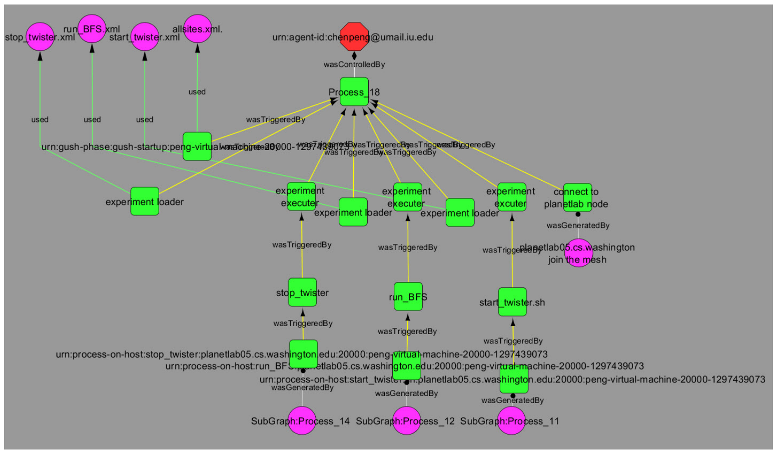

stop_twister.xml
run_BFS.xml
start_twister.xml
allsites.xml.
urn:agent-id:chenpeng@umail.iu.edu
wasControlledBy
Process_18
used
used
used
used
wasTriggeredBy
urn:gush-phase:gush-startup:peng-virtual-machine-20000-1297439073
experiment loader
experiment executer
experiment loader
experiment executer
experiment loader
experiment excuter
connect to planetlab node
wasGeneratedBy
planetlab05.cs.washington join the mesh
wasTriggeredBy
stop_twister
run_BFS
start_twister.sh
urn:process-on-host:stop_twister:planetlab05.cs.washington.edu:20000:peng-virtual-machine-20000-1297439073
urn:process-on-host:run_BFS:planetlab05.cs.washington.edu:20000:peng-virtual-machine-20000-1297439073
urn:process-on-host:start_twister:planetlab05.cs.washington.edu:20000:peng-virtual-machine-20000-1297439073
wasGeneratedBy
SubGraph:Process_14
SubGraph:Process_12
SubGraph:Process_11

Figure 3.8: Breadth-first search application after artifact elimination. Artifact nodes between process nodes in Figure 3.6 are now replaced by new edges "was triggered by." This can eliminate the unwanted artifact nodes without misrepresenting the provenance information.

where node n_1 in step 9 could have many edges while node n_2 has only one, then step 16 will be executed many times in which the only edge of n_2 is always matched. In contrast, our improved version guarantees that n_1 has less or equal number of edges than n_2, and the number of matched edges in step 16 always equals to the smaller degree of n_1 and n_2; what's more, the unmatched edges of n_1 or n_2 are also considered when combining the results in step 19. Step 22 to 60 in our implementation is a greedy algorithm that takes the node attendance as well as its topology into consideration while forming subgraphs.

3.4 Provenance Datasets

Finally, we introduce the datasets to be used in this thesis, which include a NASA AMSR-E provenance archive dataset [106], a large 10GB semi-synthetic provenance database [37], and the provenance we captured from ABM. The Open Provenance Model (OPM) is used in the NASA AMSR-E provenance archive dataset and the 10GB semi-synthetic provenance database, while the W3C PROV is used in our agent-based modeling provenance dataset.

These provenance datasets cover two major categories of provenance in eScience: workflow and simulation. In addition, we select the AMSR-E dataset since it has large forward/backward provenance graphs generated from daily and monthly workflows, so that it can demonstrate the capability of our visualization tool; we choose the 10GB provenance database because of its large volumes of workflow provenance with failure patterns that can be used to test the performance of data mining tasks on our proposed temporal representations; the provenance data we captured from agent-based simulation is at an unprecedented scale that is too big to be stored and processed in the traditional persistent approach.

3.4.1 Provenance Captured from Agent-based Simulation

As we described earlier, we capture the provenance from the NetLogo mode “wolf-sheep-predation” and a large scale agricultural decision making model. Both of the two agent-based models can run continuously for a very long time (hundreds of iterations). To give a sense of the scale of provenance traces generated from the “wolf-sheep-predation” model, we run it for up to 400 iterations and collect the raw provenance traces from its log file. Table 3.2 shows the size of the provenance log and the agent population after 10, 50, 100, etc. iterations. The size of provenance captured depends on both the number of iterations and the size of agent population. The provenance size increases dramatically at 400 iterations, which is because the wolves almost die out and the number of sheep has increased exponentially at 400 iterations. For the agricultural decision making model on Monze District Zambia, we have about 53 thousand household agents in total, and running that for 12

Table 3.2: Size of provenance log after certain number of iterations

Number of iterations	**10**	**50**	**100**	**200**	**300**	**400**
Number of sheep	129	338	170	38	2,413	112,402
Number of wolves	63	82	349	1	1	116
Log size	523KB	3.63MB	12.1MB	20.4MB	35.3MB	773MB
Number of traces	20,793	144,590	475,061	799,677	1,351,128	27,860,316

iterations (1 growing season/year) generates 66.1MB of provenance log.

This means that, for simulations that have a large population of agents or need to run over a long period of time, the size of provenance traces captured using our method can be overwhelming. Besides, the sizes mentioned above are for the raw provenance traces. When we extract the provenance graph (8,855 nodes and 18,330 edges) from the 10-iteration “wolf-sheep-predation” log file that is only 523KB and ingest it into a Neo4J [1] graph database, it takes about 17mins in a machine with 8GB memory and a Core i5 2.53GHz dual core CPU. This also suggests that the full recovery of provenance data from logs is intractable for Big Data provenance.

3.4.2 NASA AMSR-E Provenance Archive Dataset

A real life dataset that we experiment with is the NASA AMSR-E provenance archive dataset [106]. The University of Alabama in Huntsville processes data from the NASA AMSR-E instrument, and the Karma project at Indiana University instrumented the ingest processing system and captured provenance for 3,890 runs for period Sept 2 – Oct 4, 2011. There are six types of daily workflows, one five-day workflow, one weekly workflow, and three monthly workflows in that dataset. The smallest provenance graph is the Daily Rain graph that when represented as a XML is 11KB, and the largest is the Monthly Rain graph

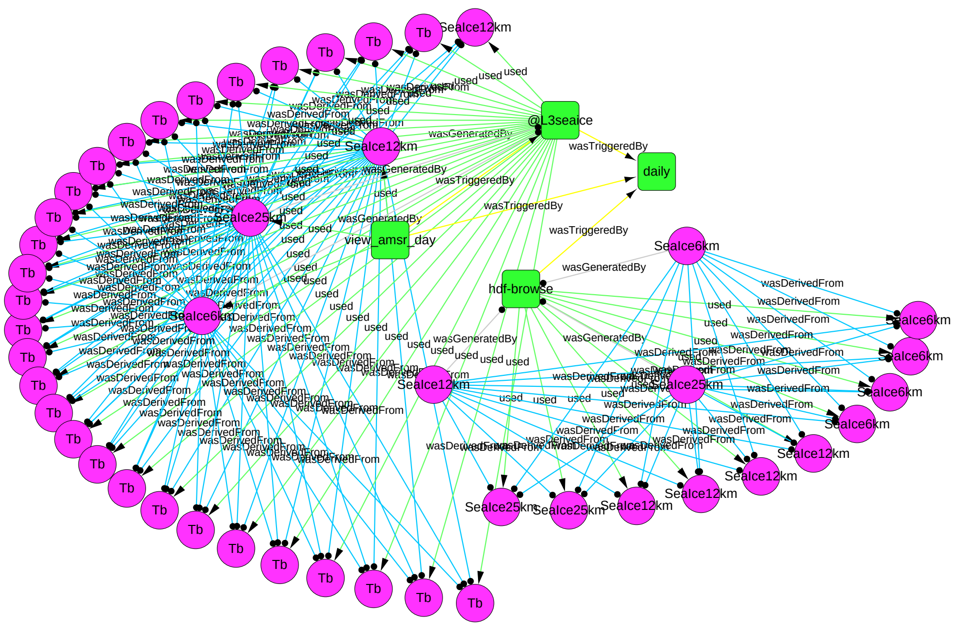


Figure 3.9: Visualization of the provenance of a sea ice daily workflow in the imagery ingest pipeline. The magenta nodes are data products and green nodes are process of the workflow. The magenta nodes on the outside circle are data artifacts consumed by the sea ice daily workflow ("Tb" stands for "temperature brightness," and "SeaIce6km," "SeaIce12km" and "SeaIc25km" are the satellite images of different resolutions). The magenta nodes on the inside circle are output data generated by the services that are running different algorithms in the workflow (i.e. the green node "view_amsr_day" and "hdf-browse"). The workflow controller is captured as the process "daily" (green node). An audience can look at this visualization to understand and examine the execution of the workflow, and the data products. If an audience finds one data product particularly interesting, he/she can perform a backward of forward provenance query to start an investigation.

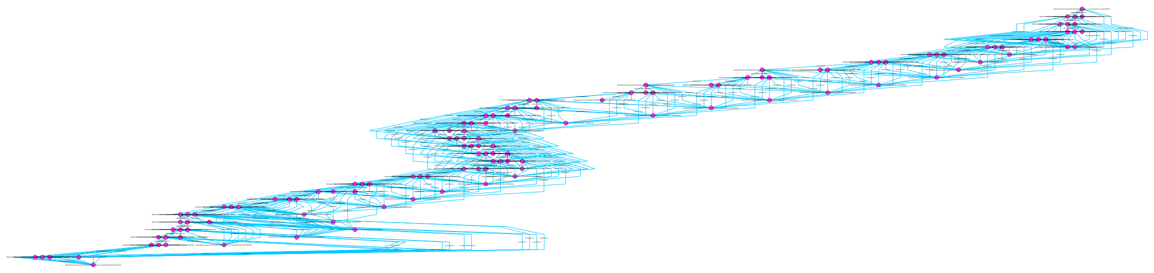


Figure 3.10: Visualization of the forward provenance of an input data to a sea ice daily workflow. The input data is a magenta node in the top right corner, and the rest magenta nodes are all the data artifacts that depend on it. The forward provenance contains all the data artifacts that were influenced by that input data, thus it can be used to study the error propagation. While the complete forward provenance includes the processes as well, we apply the "process elimination" technique to eliminate the processes but still display the correct dependencies between data artifacts. This filters out the unnecessary information to make the visualization more concise. This visualization is particularly useful when the input data is erroneous and we want to find out its impact.

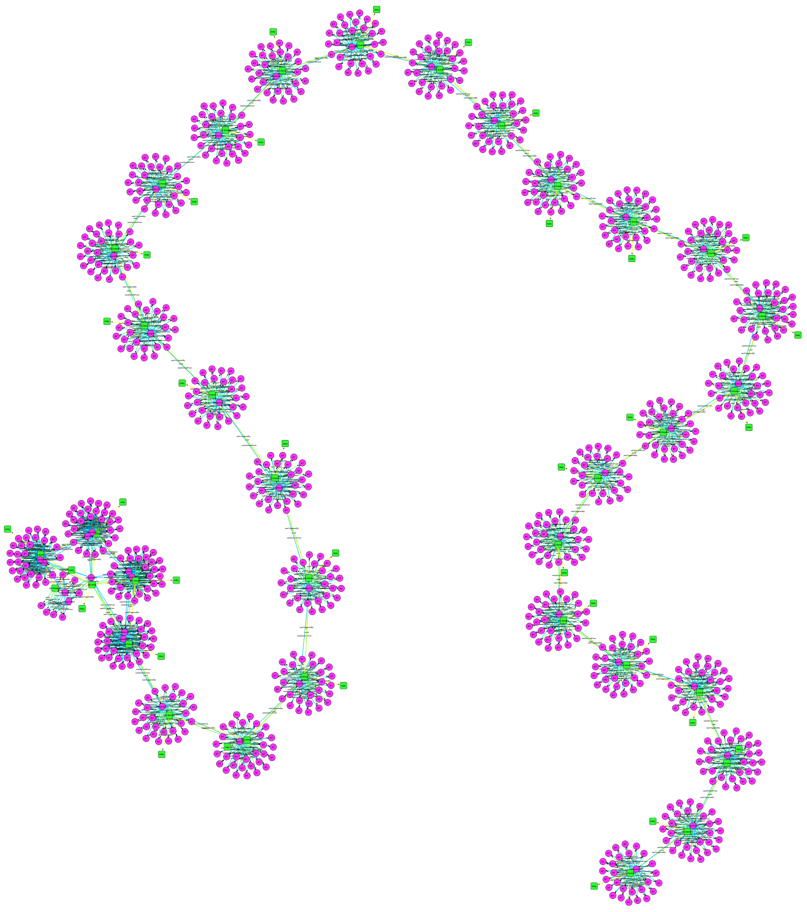


Figure 3.11: Backward provenance visualized in a history chain shows the relationship between daily products (each clover flower in the clover leaf chain) and final monthly products at the left-end. The connected daily provenance graphs (clover flowers) reflect the fact that in creating a daily sea ice product, a moving window mask of prior days is used to locate the boundary between land ice and sea ice. If we have a problem in the monthly product, we could use this chain to examine where the root cause is.

that is 9MB. The total size of AMSR-E dataset as XML files is 83.0MB.

The NASA AMSR-E provenance archive dataset is stored in a Karma server. We can use our visualization tool to retrieve and visualize the provenance of a daily workflow (Figure 3.9), the forward provenance of an input data (Figure 3.10), and the backward provenance of a monthly output (Figure 3.10).

3.4.3 10GB Noisy Provenance Database

The 10GB provenance database created at Indiana University is generated from real world workflows using the WORKEM emulator [127], with known failure patterns [37]. The database is populated with the provenance of approximately 48,000 workflow execution instances, modeled on six real workflows: LEAD North American Mesocale (LEAD NAM) forecast (weather), SCOOP ADCIRC (coastline), NCFS (ocean), Gene2Life (bio), Animation (CS) and MotifNetwork (bio). Some of the workflows are small, having a few nodes and edges, while others like the Motif workflow have a few hundred nodes and edges. Each workflow type has approximately 2000 instances per failure mode, with failure modes including random dropped messages (a task completes but the notification is not successfully transmitted) and workflows that fail. We use the Karma provenance tool [147] to export the 10GB provenance dataset into OPM compliant XML strings. This results in a 2.1GB file with 47912 lines, each an OPM graph.

Since we want to apply the association rules onto the 10GB provenance dataset, there is an issue that must be dealt with beforehand. Despite failed workflows and dropped messages modeled into the semi-synthetic database, there are few significant structural variations amongst the workflow instances. In order to provide association rules that reflect meaningful variations, we manually introduce two variants of NAM weather forecast workflow (see Figure 3.12).

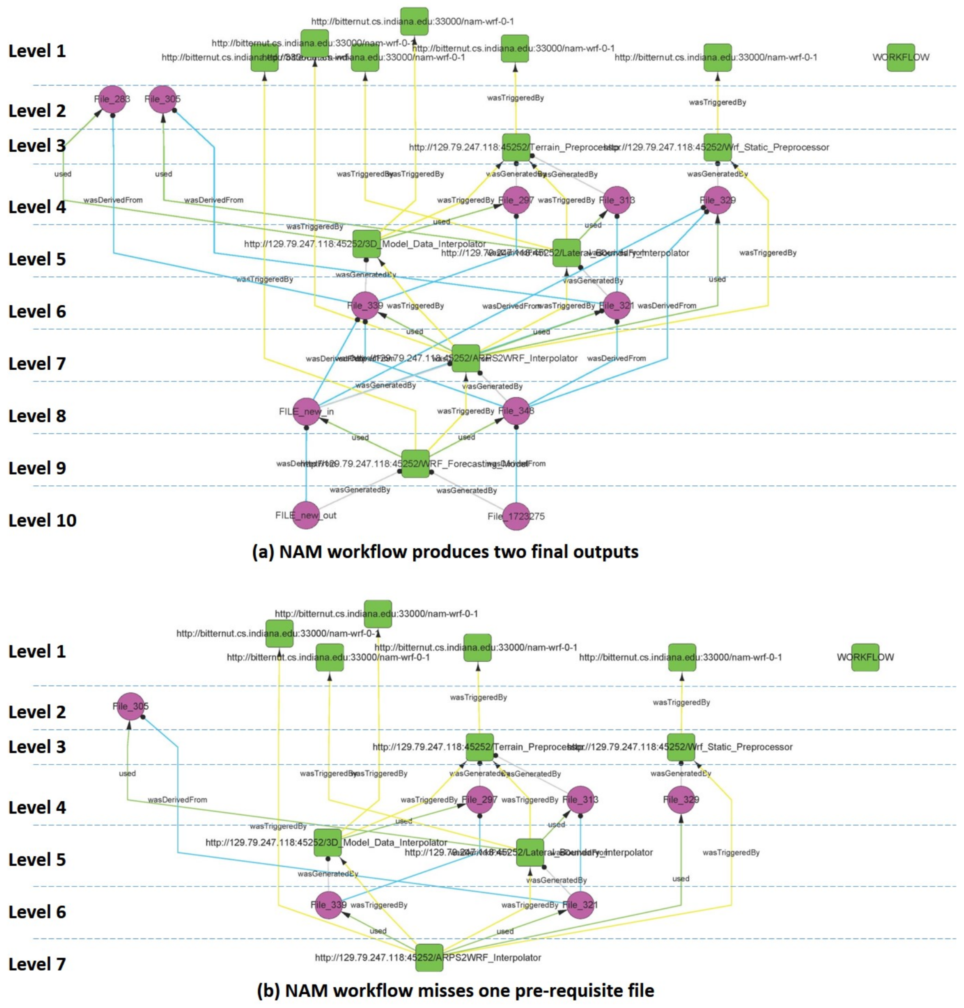

Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Level 8
Level 9
Level 10
http://bitternut.cs.indiana.edu:33000/nam-wrf-0-1
WORKFLOW
File_283
File_305
http://129.79.247.118:45252/Terrain_Preprocessor
http://129.79.247.118:45252/Wrf_Static_Preprocessor
File_297
File_313
File_329
http://129.79.247.118:45252/3D_Model_Data_Interpolator
http://129.79.247.118:45252/Lateral_Boundary_Interpolator
File_339
File_321
http://129.79.247.118:45252/ARPS2WRF_Interpolator
FILE_new_in
File_348
http://129.79.247.118:45252/WRF_Forecasting_Model
FILE_new_out
File_1723275
used
wasDerivedFrom
wasTriggeredBy
wasGeneratedBy
(a) NAM workflow produces two final outputs
(b) NAM workflow misses one pre-requisite file

Figure 3.12: Two variants introduced to association rules mining: (a) introduces two intermediate data products generated for the last processing step, which leads to two final outputs; (b) has one of the two prerequisite files missing, which leads to the intermediate processes unable to continue, resulting in a failed execution.

Chapter 4

Provenance Filtering and Efficient Offline Retrieval

Capturing the provenance of complex system behaviors in agent-based simulation requires collecting information about the execution of every statement in the model, which can generate huge amounts of fine-grained provenance that poses a big challenge for storage and subsequent querying. Chapter 3 gives an example of how quickly the size of provenance captured from NetLogo can accrue as the number of simulations and agents increases. In addition, Pignotti et al. [124] discuss typical queries over a provenance record in ABM, and they also find that as the number of simulation runs and agents in the simulation increases, these queries become exponentially complex.

In this chapter, we take a different approach to dealing with the Big Data provenance, that is to assume the query is known beforehand, and then we selectively preserve the provenance traces during capture and process the traces on demand during query execution.

4.1 Provenance Filter

To reduce the overhead and simplify subsequent querying, we develop filters that apply intelligence to reduce provenance before it is written to log tiles. Some of the filters abstract statement level traces and others drop unrelated traces and turn off the tracing once the capture finishes. The filters can be categorized into two types: aggregation and filtering.

4.1.1 Aggregation

Keeping track of fine-grained provenance provides a detailed view of the dependencies between entities in a simulation, but at the expense of additional storage and processing overhead. We can extract the fine-grained provenance from provenance traces and aggregate them into high level provenance records.

For example, one of the interactions between sheep agents and wolf agents in the model “wolf-sheep-predator” is that a wolf agent acquires reference to a sheep agent and then kills it. We can aggregate these two steps into a single activity “kill”, or to a generic relationship “interact with.” We propose an aggregation filter that uses the generic relationship “interact with” to represent all the complex interactions between agents, since identifying less generic activities like “kill” is more complicated and may need user effort. This aggregation filter is inspired by the interest of domain scientist in studying the interactions between agents in simulation, and we can visualize its output as a social network (see Figure 4.1 for an example). We implement this filter by buffering all provenance traces within an iteration at runtime and then extracting and writing the abstract provenance to the log file.

4.1.2 Filtering

To answer the forward and backward provenance query, we propose filtering that computes a forward provenance slice or backward provenance slice for a given variable. Our NetLogo extension (Section 3.2.3) captures provenance traces of all iterations by default, but we also develop one filter that collects the provenance traces from only a single iteration. In this section, we introduce and describe the use cases and implementations of the forward,

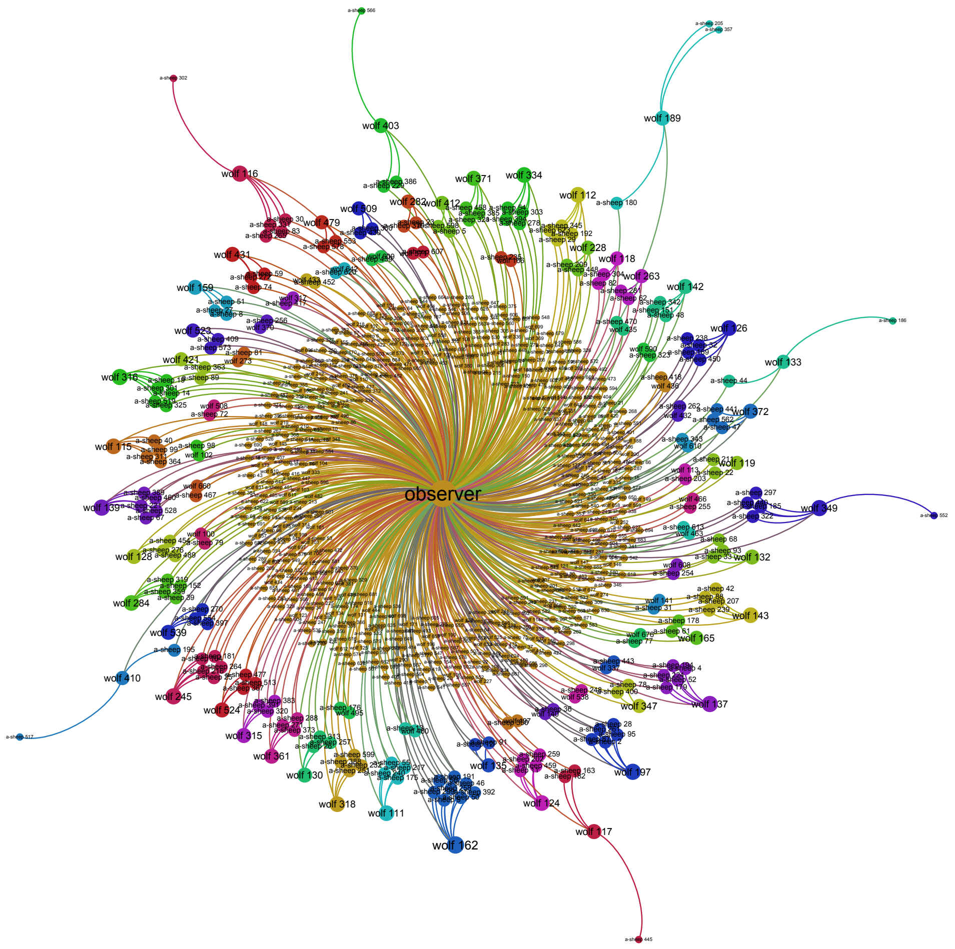


Figure 4.1: An example visualization of interactions between agents in “wolf-sheep-predation”. Edges represent interactions between vertices (agents). Edges and vertices are partitioned into strongly (or weakly) connected communities and are dyed accordingly. The biggest community in the center has the observer (a manager agent) and the agents that have no other interactions. Each of the smaller communities far from the center has one wolf agent and its preys

backward, and single-iteration filters.

The backward provenance slice includes the processes, input data, intermediate data and agents that are involved in the generation of an output data. This can help user to better understand why a particular result is achieved and can be used for debugging. Figure 4.2 is the visualization of an example backward provenance of the NetLogo mode "wolf-sheep-predation," from which we can see how a wolf agent's energy level changes during the simulation. It is usually difficult to decide the backward provenance slice for a variable before the simulation terminates, this is because the target variable can be directly or indirectly affected by any current data/process in the future. However, by observing the backward provenance we can get from the provenance traces of a finished simulation, we find that it is more like a linear structure rather than an upside down tree – it consists of information about how a given variable evolves as processes running on the same agent to which the variable belongs. So we implement an imprecise filter that drops all provenance traces that are not associated with the agent to which the target variable belongs.

The forward provenance slice of one input data tells information about the future data products that are derived from the input data. It can be used to understand the impact of an input parameter, or to control the error propagation if an input data is corrupted. Figure 4.3 is the visualization of an example forward provenance of a parameter named "wolf-reproduce". Note that each "was derived from" relationship means that the target entity was used in the generation of the source entity, and we do not consider any other indirect dependency in this way. We implement the forward provenance filter by keeping track of all data products that are derived from the input data or the previously derived data.

The last filter we propose keeps the execution traces only for a single iteration, which is inspired by the user's interest in studying the various agent behaviors and the interactions

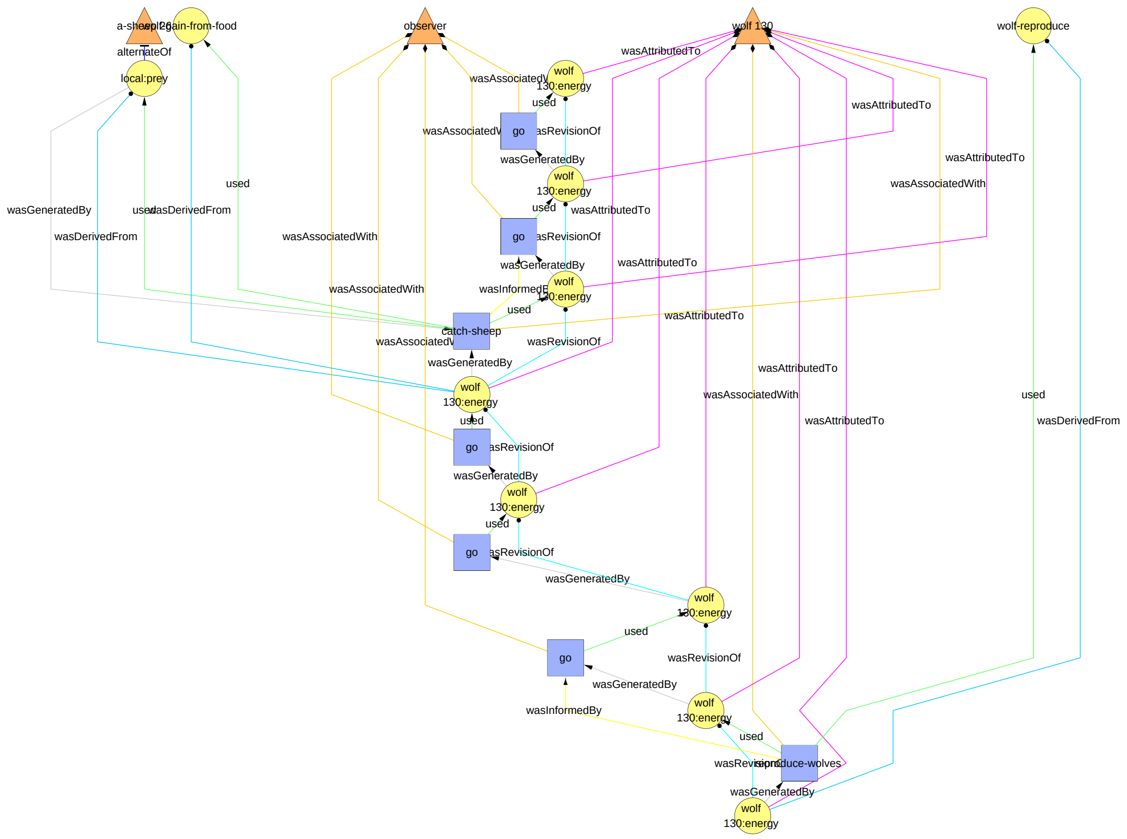


Figure 4.2: The backward provenance of a data product “energy” of agent “wolf 130”. We create a different visual styles for W3C PROV – the orange triangles represent agents, the blue rectangles represent processes and the yellow circles represent data products. This visualization could be used for debugging. It is clear to see how the different behaviors of a wolf agent affect its level of energy: there are three types of processes involved in the generation –“go”, “catch-sheep” and “wolf-reproduce”; the agent variable “energy” is reduced by 1 each time the “observer” agent (global manager) invokes the “go” procedure (which means one iteration); the agent variable “energy” increases by the value of “wolf-gain-from-food” after catching sheep agent “sheep 26”; the “energy” is reduced from 46 to 23 after the process “wolf -reproduce” took place.

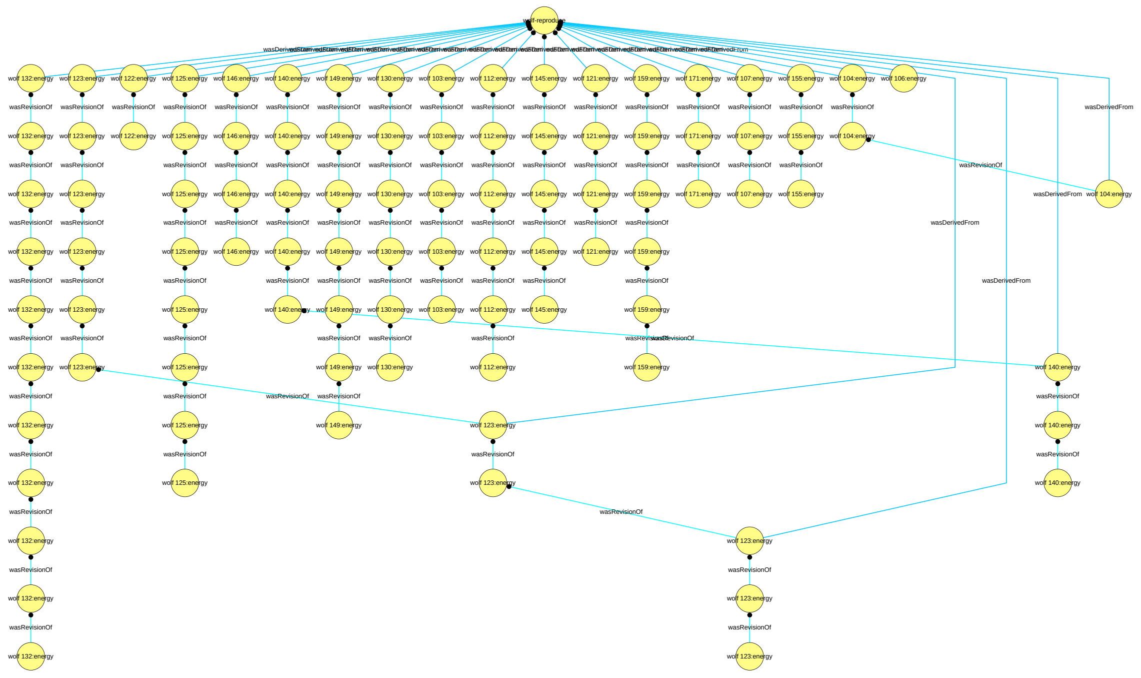


Figure 4.3: The forward provenance of global variable “wolf-reproduce.”

between them. By recovering and visualizing the provenance from provenance traces of a single iteration, we can discover the different agent behavior patterns. Figure 4.4 is an example visualization that shows six different agent behaviors within an iteration. The implementation of the single-iteration provenance filter is simple. We ask the user to start the filtering by entering the name of the entry procedure for the next iteration (like the procedure “go” in the model “wolf-sheep-predator”). After the simulation exits the entry procedure, the filter shuts down the tracing to minimize the overhead.

In sum, we have proposed two types of filters and included visualizations of the provenance computed from the output provenance traces of these filters. We examine their performance in Section 4.3.

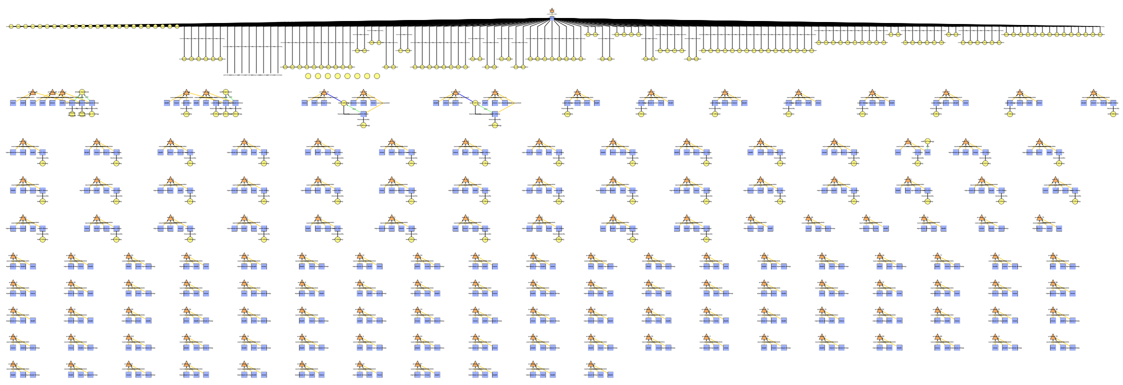


Figure 4.4: An example single-iteration provenance of the "wolf-sheep-predation" model. It shows six different agent behaviors within an iteration: 1) the global agent "observer" invokes "display-labels" on each wolf and sheep agent; 2) the wolf agents invoke the procedure "go", "move" and "death"; 3) the wolf agents invoke the procedure "wolf-reproduce" in addition to the procedure "go", "move" and "death"; 4) the sheep agents invoke the procedure "go", "move" and "death"; 5) the sheep agents invoke the procedure "sheep-reproduce" in addition to the procedure "go", "move" and "death"; 6) the wolf agents catch sheep agents via the procedure "catch-sheep".

4.2 Non-preprocessing Provenance Slicing

The existing method that captures provenance from program logs needs to recover all the entities and dependencies from the logs before being able to answer queries [9]. However, our empirical study indicates that this can be infeasible for large provenance graph. For example, the log file that has the 20,793 provenance traces from a 10-iteration simulation of the model "wolf-sheep-predation" is only 523KB (see Chapter 3 for more information). However, recovering the full provenance (that has 8,855 nodes and 18,330 edges) from the 10-iteration log file and storing it into the Neo4J [1] graph database takes 1,035,845ms (about 17 mins) in a machine with 8GB memory and Core i5 2.53GHz dual core CPU.

However, we can actually avoid recovering and maintaining provenance entities and dependencies that are unnecessary for answering a specific query. To achieve this, we propose a query technique we call non-preprocessing (NP) provenance slicing that is derived from Zhang's no preprocessing (NP) program slicing algorithm [166]. The non-preprocessing (NP) provenance slicing technique employs demand driven analysis of the provenance traces to recover dependency provenance. When a provenance query begins we traverse the provenance traces forward (or backward) to recover the dynamic dependencies required for the provenance slice computation. For example, if we need the provenance slice for the final value of some variable v (backward provenance), we traverse the execution traces backward till the first access of the variable was found. If that value of v depends on another variable w (see possible dependencies in Table 3.2.2 in Chapter 3), we resume the traversal to also calculate the provenance slice of w.

In essence this algorithm performs partial preprocessing for extracting entities, activities and dependencies relevant to a provenance query. It is possible that two different querying requests involve common information. In such a situation, the non-preprocessing (NP) provenance slicing algorithm will recover the common information from the execution trace during both provenance slice computations. To avoid this repetitive work we can cache the recovered entities, activities and dependencies. Therefore at any given point in time, all entities, activities and dependencies that have been computed so far can be found in the cache. Similar to Zhang's definition, we also define two versions of this demand driven algorithm, that is, without caching and with caching, as non-preprocessing without caching (NPwoC) provenance slicing and non-preprocessing with caching (NPwC) provenance slicing. We evaluate the performance of NPwoC provenance slicing in Section 4.3

Table 4.1: Evaluation of storage cost

Number of Iterations	**10**	**50**	**100**	**200**	**300**	**400**
full provenance capture	523KB	3.63MB	12.1MB	20.4MB	35.3MB	773MB
aggregation filter	8.22KB	30.9KB	100KB	140KB	230KB	4.53MB
backward provenance filter	5.36KB	18.1KB	18.1KB	18.1KB	18.1KB	18.1KB
forward provenance filter	13.9KB	163KB	925KB	2.08MB	2.10MB	2.75MB
single-iteration filter	46.8KB	45.2KB	45.2KB	45.2KB	45.2KB	45.2KB

and find that it is faster than traversing the pre-recovered full provenance graph.

4.3 Performance Evaluation

In this section, we first evaluate the proposed filters by comparing the performance of provenance capture with these filters against the performance of a full provenance capture without any filter.

Table 4.1 compares the size of provenance logs generated using different filters with the size of a full provenance log. We can see that all filters can dramatically reduce the storage cost. While the provenance log generated from the single-iteration filter remains the same (after the capture finishes), the logs generated from other provenance filters get larger as simulation goes on: the log size from aggregation filter keeps increasing since there are more and more agents; the log size from backward provenance filter keeps increasing until the target agent dies; the log size from forward provenance filter does not increase much after 200 iterations since we were tracing the usage of the global variable "wolf-reproduce" by wolf agents and there are few wolf agents after 200 iterations.

Figure 4.5 shows the average time of provenance capture with different filters and without any filter (the full provenance capture). It shows that while the single-iteration filter can significantly reduce the time overhead, the backward (history) provenance filter doesn't reduce the time overhead much, and the forward provenance filter and aggregation filter have even higher overhead than the full provenance capture. However, we argue that this can be optimized by detaching the filtering from the tracing and the model execution using a messaging bus. In this way, the "writing time" and the additional computation time are replaced with less communication time.

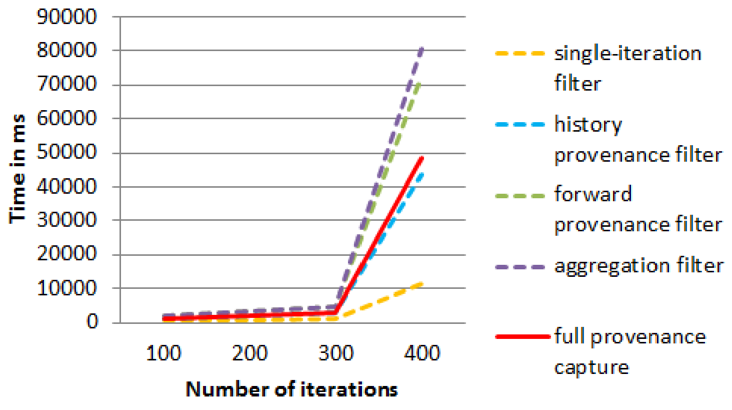


Figure 4.5: Evaluation of capture time cost.

We also evaluate the performance of proposed non-preprocessing (NP) provenance slicing technique (the NPwoC version) with various query requests (Table 4.2). The non-preprocessing (NP) provenance slicing technique can query either on the full provenance traces or on the filtered provenance traces, and we measure both performance to demonstrate that the filtered provenance traces of reduced size can further improve the querying performance. Finally, we compare the performance of the non-preprocessing (NP) provenance slicing technique with the performance of traversing pre-recovered full provenance in a Neo4j graph database (using Neo4j Java API).

Table 4.2: Evaluation of query time cost

	Same query issued to the full provenance in Neo4j (ms)[a]	**NP Provenance Slicing on full provenance traces (ms)**	**NP Provenance Slicing on filtered provenance traces (ms)**
Aggregation filter	6,046	2,565	2,025
Backward provenance filter	3,043	1,964	1,823
Forward provenance filter	3,127	2093	1,785
Single-iteration filter	6,059	3,819	3,750

[a]**Note that the construction of the full provenance database costs 1,035,845ms (about 17 min) beforehand**

4.4 Summary

In this chapter, we assume that the query requests are known beforehand and use filtering techniques accordingly to reduce the amount of provenance to capture from agent-based simulations. We also propose the non-preprocessing (NP) provenance slicing techniques that can directly retrieve provenance information from raw probe traces to avoid recovering and maintaining provenance entities and dependencies that are irrelevant to the query request. The experimental evaluation shows that the proposed filters and non-preprocessing (NP) provenance slicing technique can work together to dramatically reduce the demands on persistent storage and simplify the subsequent querying.

Chapter 5

Online Processing of Streaming Provenance

Agent Based Modeling (ABM) is powerful because of its ability of representing heterogeneous agents and their interactions so that it can capture emergent phenomena. In Chapter 3 we showed that data provenance in ABM can capture the cause-effect relations to help understanding the complex process and to enable the analysis of internal dynamics that were previously hidden as a black box. However, it is also shown in Chapter 3 that the amount of provenance captured from simulations where there are many interacting components running over a long period of time could be huge. The traditional approach that stores all the provenance data and processes it offline requires lots of resources that are unnecessary. While we can apply filters to provenance capture (Chapter 4), this requires knowing the query beforehand to determine which information can be discarded. This is inapplicable for backward provenance queries that are usually on demand. For example, scientists may find an abnormal agent state (or behavior) when monitoring the simulation run, for which they can issue a backward provenance query to find out the root causes.

Scientific experiments like agent-based simulation usually run over the course of hours or days. Operations like debugging and model calibration are usually performed when the experiment fails/finishes, often need to repeat the experiment multiple times, and thus are very time consuming. We have demonstrated that provenance data can be used for debugging [41, 44] and model analysis [43], and since provenance captures the dependencies between input parameters and intermediate status that does not match real data, it is also suitable for model calibration. The challenge, however, is to overcome storing and wading through vast volumes of information by processing the provenance data in-motion.

One approach to handling provenance data in-motion is to process it continuously in the data stream model where the input is defined as a stream of data. Algorithms in this model must process the input stream in the order it arrives while using only a limited amount of memory [18]. Work exists on capturing provenance data from streaming systems [54, 113, 136, 154–156] and [135] even tracks provenance assertions on-the-fly. However, provenance is modeled as a graph, and most of the work on graph streams has occurred in the last decade and focuses on the semi-streaming model [65, 121]. In this model the data stream algorithm is permitted $O(n\text{polylog}n)$ space where n is the number of nodes in the graph thus it does not suit for continuous processing. This is because most problems are provably intractable if the available space is sub-linear in n, whereas many problems become feasible once there is memory roughly proportional to the number of nodes in the graph.

Since a provenance graph is a directed property graph with temporal relationships, while a general graph stream is often considered to consist of undirected edges arriving in a random-order [111], a provenance stream could consist of edges and nodes with properties following a partial order. In this section, we distinguish a provenance stream from the general graph stream by emphasizing that partial order. Based on that we developed an algorithm for backward query over the provenance stream that has a space complexity limited by the maximum number of data items that the program can access at any given time during its execution, which is limited by the total number of variables declared in an agent-based model.

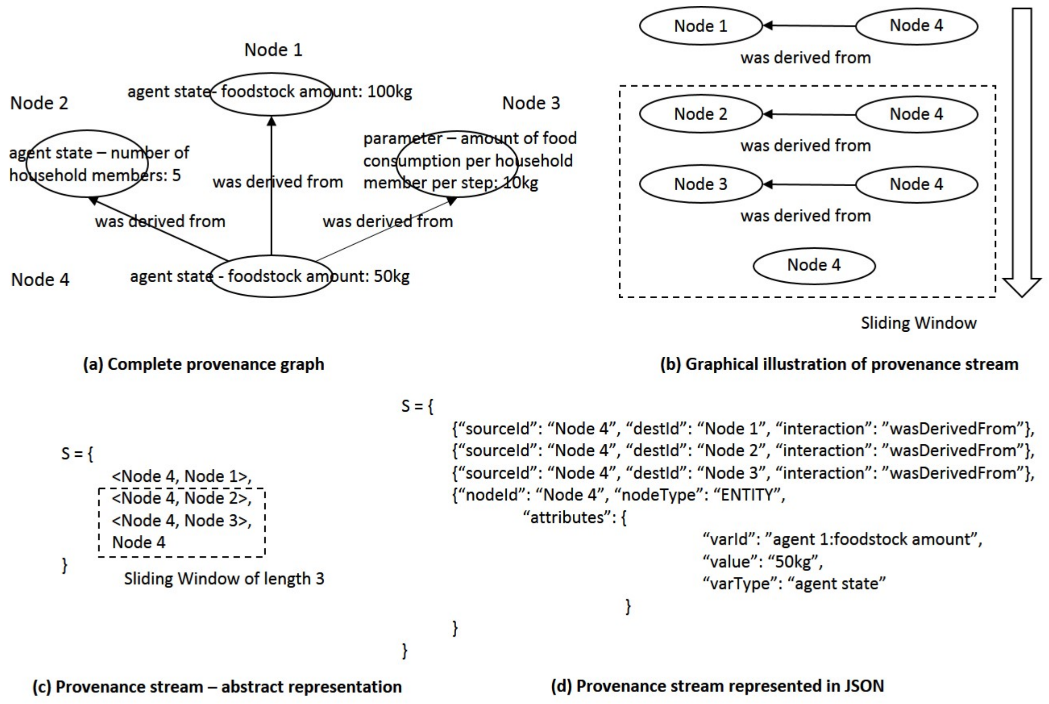
To demonstrate the feasibility of our proposed approach, we extended our automatic provenance capture framework mentioned in Chapter 3 to capture provenance streams from multiple running NetLogo [160] simulations and ingest them into the messaging system Apache Kafka [94]. We then implemented the proposed algorithm using the stream processing platform Apache Spark Streaming [165] to enable the parameter readjustment and mining analysis for agent-based simulations. The performance evaluation shows high throughput, low latency and reasonable scalability.

Finally, we discuss various applications on an environmental agent-based model that can be developed within our stream processing framework. That includes parameter calibration through online monitoring and steering [140], and simulation analysis through time series clustering and association rules mining.

The remainder of the chapter is organized as follows: Section 5.1 defines the stream model of dependency provenance. Section 5.2 gives an overview of the framework, while Section 5.3 describes its stream provenance capture component and Section 5.4 describes its stream processing component. Section 5.5 describes the application of this framework on the Zambia food security agent-based model and its performance evaluation. Section 5.6 discusses the various applications that can be developed within our framework.

5.1 Stream Model of Provenance Graph

An agent-based model (ABM) is a simulation of distributed decision-makers (agents) who interact through prescribed rules. We demonstrated in Chapter 3 that the dependency provenance in ABM can explain how and why the output data depends on the input data, and can yield insights into cause-effect relations among system behaviors. The concept of dependency provenance [45, 46] is based on the dependency analysis techniques used in program slicing, which is different from “where-provenance” and “data lineage”, but similar to “how-provenance” or “why-provenance” [32] in that it identifies a data slice showing the input data relevant to the output data. In this chapter, we focus on the dependency provenance that consists of all the data products and their dependencies within an ABM.


Node 1
agent state - foodstock amount: 100kg
Node 2
agent state – number of household members: 5
Node 3
parameter – amount of food consumption per household member per step: 10kg
was derived from
was derived from
was derived from
Node 4
agent state - foodstock amount: 50kg
(a) Complete provenance graph
Node 1
Node 4
was derived from
Node 2
Node 4
was derived from
Node 3
Node 4
was derived from
Node 4
Sliding Window
(b) Graphical illustration of provenance stream
Sliding Window of length 3
(c) Provenance stream – abstract representation
(d) Provenance stream represented in JSON

```
S = {
      <Node 4, Node 1>,
      <Node 4, Node 2>,
      <Node 4, Node 3>,
      Node 4
}
```

```
S = {
      {“sourceId”: “Node 4”, “destId”: “Node 1”, “interaction”: ”wasDerivedFrom”},
      {“sourceId”: “Node 4”, “destId”: “Node 2”, “interaction”: ”wasDerivedFrom”},
      {“sourceId”: “Node 4”, “destId”: “Node 3”, “interaction”: ”wasDerivedFrom”},
      {“nodeId”: “Node 4”, “nodeType”: “ENTITY”,
            “attributes”: {
                        “varId”: ”agent 1:foodstock amount”,
                        “value”: “50kg”,
                        “varType”: “agent state”
            }
      }
}
```

Figure 5.1: Illustration of the provenance stream model.

We use the same mapping to W3C PROV as in [43] to express the dependency provenance in ABM. PROV models provenance as a static graph, but the provenance capture can be viewed as a process of appending node/edge to the graph in their generation order. While a general graph can be streamed into a sequence of undirected edges in random-order, a provenance graph could be represented as a sequence of directed edges and nodes following the order of node/edge generation (see Figure 5.1 for an example).

Below, we propose a stream model of provenance that only captures the data products and their dependencies. We denote a dependency provenance graph as $G = (V, E, A)$, where $V = \{v_1, v_2, ..., v_n\}$ is the set of data products (nodes); $E = \{e_1, e_2, ..., e_m\}$ is the set of dependency relationships (edges) in which an edge $e = \langle v_i, v_j \rangle$ specifies that v_i depends on v_j; $A(v) = \{a_1, a_2, ...\}$ represents an arbitrary number of attributes of v.

Definition A stream of dependency provenance consists of a sequence of time ordered elements:

$$S = \langle p_1, p_2, ..., p_{m+n} \rangle \tag{5.1}$$

where $p ::= v|e$ is either a provenance edge or a provenance node, and $timestamp(p_n)$ $< timestamp(p_{n+1})$.

The provenance stream is append only and is potentially unbounded in size. Once an element of the stream has been processed it is discarded, and a query can only be evaluated over the sliding window (with length w) of recently processed elements (at time t):

$$W = \{e_{t-w+1}, ..., e_t\} \tag{5.2}$$

However, unlike the general data stream, a provenance stream has unique properties that allow for the stream processing with limited resources:

1. There is temporal order between nodes that are connected by edges in the stream. For each edge in the stream, the sink node should be generated before the source node. We assume that this temporal order can be preserved during the provenance capture and transfer (in our framework this is guaranteed by the Kafka per-partition ordering), and we describe the first property of provenance stream as below:

Property *For any edge* $e = \langle v_i, v_j \rangle$, $timestamp(v_j) < timestamp(v_i)$

2. All of the dependencies (outgoing edges) contributing to the generation of a data item (source node) must be formed by the time that the data is generated. Due to different implementations in provenance capture and different definitions of provenance edge/node in provenance modeling, the outgoing edges of a node can come before or after the node in the provenance stream. However, no out-going (upstream) edges of an existing node should be seen in the downstream when we see an in-coming edge of that node.

   Property For any two edges $e_l = \langle v_i, v_j \rangle$ and $e_m = \langle v_k, v_i \rangle$ that share a common node v_i, $timestamp(e_l) < timestamp(e_m)$

Based on these two properties, we are able to process the provenance stream using a standard stream processing approach.

5.2 System Architecture

We develop a scalable framework to support the capture and processing of live provenance streams generated from simulations running in NetLogo. Figure 5.2 is an overview of its two major components. The *provenance stream capture* component captures live provenance streams from agent-based simulations and stores them into a Kafka messaging cluster. The *provenance stream processing* component is built in a Spark Streaming cluster to support query and mining operations. The details of them are illustrated in the following sections.

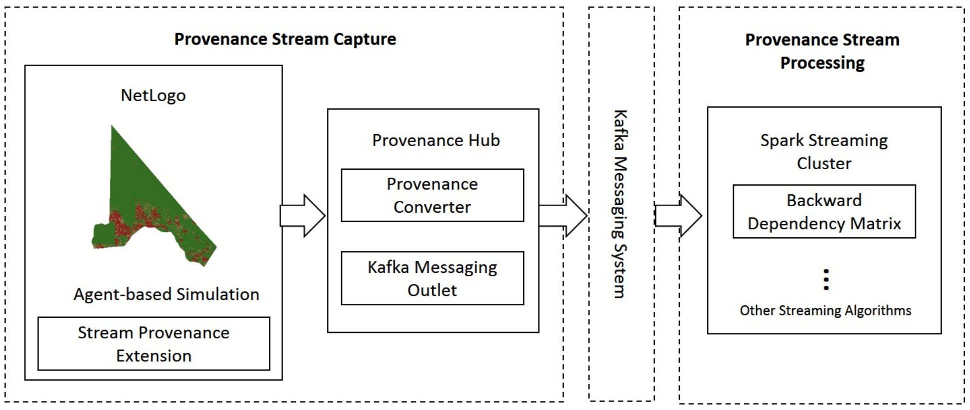

Provenance Stream Capture
NetLogo
Agent-based Simulation
Stream Provenance Extension
Provenance Hub
Provenance Converter
Kafka Messaging Outlet
Kafka Messaging System
Provenance Stream Processing
Spark Streaming Cluster
Backward Dependency Matrix
Other Streaming Algorithms

Figure 5.2: Architecture of the streaming provenance capture and processing framework.

5.3 Provenance Stream Capture

In Chapter 3 we captured the provenance traces of a NetLogo simulation through the process of adding probes into the model's source code. We developed a NetLogo extension that collects the provenance traces from probes and the context information from the model and then combines them into provenance records to be saved and processed offline. In this chapter, we extend the NetLogo extension to send the provenance records directly to a converter that converts provenance records into a live stream of provenance nodes/edges (in JSON format), which are then forwarded to the messaging system and processed in real-time. The new provenance capture mechanism is illustrated in Figure 5.3.

Note that each provenance hub uses multi-threading to receive probe traces from multiple simulations and to send them to Apache Kafka [94], which is a distributed publish-subscribe messaging system that is designed to be fast, scalable, and durable. The provenance streams from different simulations can be separated by keys (unique keys can be created by combining the hub id and the stream number within the hub). Each provenance hub can be configured either to send its streams into different partitions of one Kafka topic

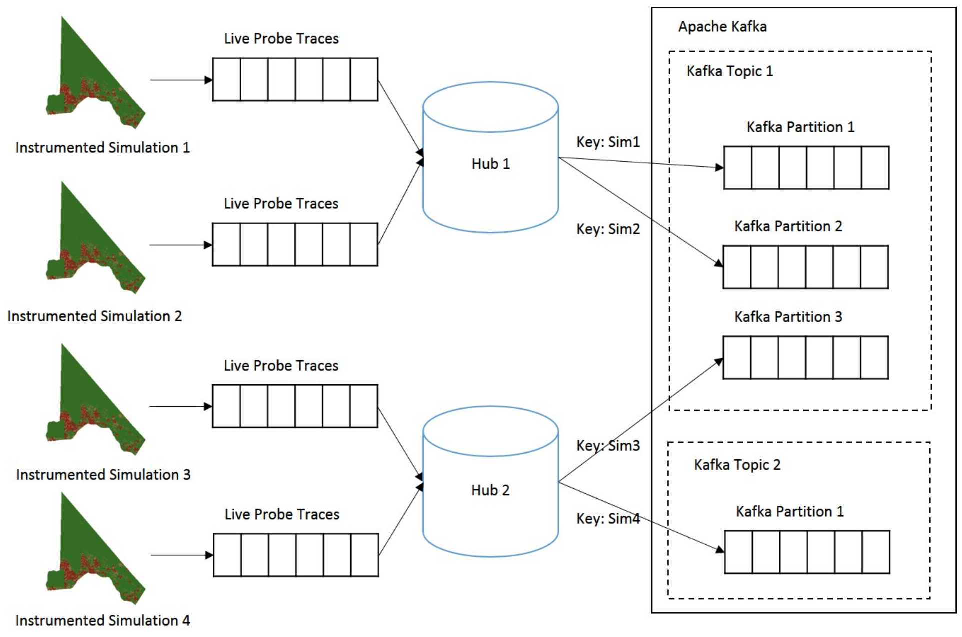


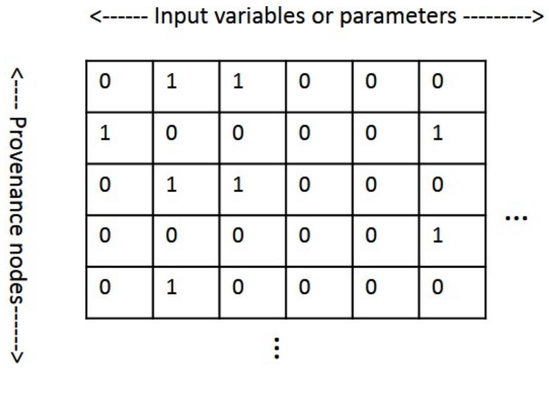
Figure 5.3: Provenance capture from multiple running NetLogo simulations.

or into different Kafka topics. This flexibility in organizing streams by topics and partitions is used to improve the throughput and the level of parallelism of stream processing in Spark Streaming (see Section 5.5). For agent-based simulations distributed across multiple machines, we can deploy one or more provenance hubs on each machine.

5.4 Provenance Stream Processing

5.4.1 Stream Processing Algorithm to Support Backward Provenance Query

Now we present our Backward Dependency Matrix (BDM) algorithm, which maintains a dependency matrix to answer the backward provenance query for the most recent provenance nodes (i.e., data products) in the stream. Given the temporal order we defined in Section 5.1, we can use a dynamic matrix to store and calculate the dependencies between all provenance nodes and the input/global parameters. So that for a newly arriving provenance node, we can use the matrix to find the input/global parameters on which it depends. Figure 5.4 illustrates the dynamic matrix, whose rows and columns can be added and removed on demand. The rows in the matrix correspond to provenance nodes (data products), and the columns corresponds to input/global parameters. A cell of value 1 in the matrix means a backward dependency from its row to its column. Each time a new provenance edge $e = \langle v_i, v_j \rangle$ arrives, we extract the backward dependencies of v_j (value 1s in its row), and add them into the backward dependencies of v_i. The temporal order guarantees that all the backward dependencies of v_j exist already. In this way, we can calculate the backward dependencies for all provenance nodes, with the matrix size being potentially unbounded. However, under the constraints of our stream model, we can only use an internal state of limited size.



<------ Input variables or parameters --------->
<---- Provenance nodes------>
...
⋮

0	1	1	0	0	0
1	0	0	0	0	1
0	1	1	0	0	0
0	0	0	0	0	1
0	1	0	0	0	0

Figure 5.4: Dynamic dependency matrix (0: dependent; 1: independent).

One observation on agent based model in NetLogo, and in many other applications too, is that there exists only one instance (or value) of any variable at any moment – a universal value of a global variable, one copy of an agent variable within each agent, and

Algorithm 2 The BDM algorithm that maintains a dependency matrix to support the backward query on provenance stream.

```
1: function UPDATESTATE($element$, $state$)    ▷ $element$: a provenance stream element; $state$: the internal state with two dynamic matrices $current$ and $purge$, and one HashMap $varIdToNodeId$
2:     if $element$ is a provenance edge then
3:         $sourceNode \leftarrow element.source$
4:         $destNode \leftarrow element.dest$
5:         if $state.varIdToNodeId$.containsKey($sourceNode.varId$) and $state.varIdToNodeId$.get($sourceNode.varId$) != $sourceNode.nodeId$ then
6:             remove all $dependencies$ from $state.current$ whose sources match $sourceNode.varId$
7:             cache $dependencies$ in $state.purge$    ▷ older dependencies in $state.purge$ whose sources match $sourceNode.varId$ are purged
8:         end if
9:         $state.varIdToNodeId$.put($sourceNode.varId$, $sourceNode.nodeId$)
10:        if $destNode.varId$ is an input/global variable then
11:            add new dependency $sourceNode.varId \Rightarrow destNode.varId$ into $state.current$
12:        end if
13:        if $destNode.varId$ == $sourceNode.varId$ then
14:            $inputVars \leftarrow$ getBackwardProvenance($destNode.varId$, $state.purge$)
15:        else
16:            $inputVars \leftarrow$ getBackwardProvenance($destNode.varId$, $state.current$)
17:        end if
```

Algorithm 2 Algorithm that maintains a dependency matrix to support the backward query on provenance stream.

```
18:         for $var$ in $inputVars$ do
19:             add new dependency $sourceNode.varId \Rightarrow var$ into $state.current$
20:         end for
21:     end if
22: end function

23: function GETBACKWARDPROVENANCE($varId$, $matrix$)  ▷ $varId$: the variable that we want
    to find its related input/global parameters; $matrix$: the dependency matrix.
24:     $dependencies \leftarrow$ all dependencies in $matrix$ whose sources match $varId$
25:     return destinations of $dependencies$
26: end function
```

one value of a local variable inside a function invocation – and we only need to query the backward dependencies for the current value of a variable. Thus the matrix only needs to keep the dependencies of the current variable instances, and those that could be used in future calculations.

In our stream model of provenance graph, each node is assigned with a node ID (unique within the stream) and a variable ID during the provenance capture (see Figure 5.1). The variable ID is formed by concatenating the context information and the declared name of that variable. For example, "global:variable 1", "agent 1:variable 2", and "procedure 1, level 1:variable 3" ("level" specifies the depth of recursion). Two provenance nodes with different node ID but same variable ID represent different values of the same variable. We can keep dependencies of the most recent provenance node for each variable ID, except in the case that the most recent value of a variable depends on its earlier value – we use a cache matrix to temporarily store the dependencies of its earlier value. The algorithm is shown in Figure 2. It has a space complexity of $O(N)$, where N is the number of variables declared in the model that is independent of the unbounded stream length. The matrix $state.current$ stores the dependencies of current nodes to input data which can be queried using the function **getBackwardProvenance**.

5.4.2 Stream Processing Implementation

We implement the proposed algorithms inside a stream processing platform called Apache Spark Streaming [165]. Apache Spark [164] is a batch processing framework that has an extension to support continuous stream processing (Spark Streaming). The idea behind Spark Streaming is to treat streaming computations as a series of deterministic micro-batch computations on small time intervals, executed using Spark's distributed data processing framework. We choose Spark Streaming because the provenance stream from agent-based simulation has very high rate (thousands of events per second) and Spark Streaming supports higher throughput [104] compared with other streaming platforms like Storm [2].

Spark Streaming uses a resilient distributed dataset (RDD) as the basic processing unit, which is a distributed collection of elements that can be operated on in parallel. There are two approaches fetching messages from Kafka: one is the traditional approach, which uses Receivers and Kafka's high-level API that communicate with ZooKeeper; the other is direct mode, introduced since Spark 1.3, which directly links and fetches data from Kafka brokers. We integrate Spark Streaming with Kafka using the direct approach that has better efficiency and simplified parallelism – it creates one RDD partition for each Kafka partition (i.e., a provenance stream from a different simulation). Since Kafka implements the per-partition ordering and each RDD partition is processed by one task (thread) in Spark Streaming, the temporal order we defined in the provenance stream model is preserved in both the provenance capture and processing. Finally, the Kryo serialization is enabled for the BDM algorithm to reduce both the CPU and memory overhead caused by its internal state (i.e., two dynamic matrices and one HashMap).

5.5 Experimental Evaluation

To evaluate the performance of our framework, we apply it to a food security agent-based model that we built for Monze District in Zambia, Africa [35]. In that ABM, 53 thousand household agents make decisions biweekly based on a utility maximization approach within the context of local institutional regimes (i.e., ward). The goal of that model is to identify how climate change impacts adaptive capacity. We use the source code analyzer [43] to add probes into the NetLogo code and the extended provenance extension to capture the live provenance stream while the simulation is running. The amount of raw provenance traces generated by running that model on one ward in the Monze District for one year is around 66MB, which is 357MB of provenance nodes/edges in JSON format. In our experiments, we run the model continuously for five years that generates about 1.7GB of provenance stream data to be processed in real-time. The $throughput$ of our streaming framework is measured as below:

$$throughput = pSize * nSim/(nBatch * bInterval) \tag{5.3}$$

where $pSize$ is the total amount of provenance data generated by one simulation (1.7GB in our evaluation), $nSim$ is the number of simulations, $nBatch$ is the number of batches taken to finish processing all the data, and $bInterval$ is the batch interval. The $latency$ is measured as the average total time to handle a batch (i.e., the sum of scheduling delay and processing time).

We run the experiments using the CPU-only nodes from "Big Red II" supercomputer at Indiana University (each CPU-only compute node contains two 2.5GHz AMD Opteron 16-core CPUs and 64 GB of RAM, and is connected to a 40-Gb Infiniband network). In each experiment run, we use one node to run NetLogo (v5.2.0) simulations and our provenance hub, one to run the Kafka server and broker (v0.8.2), and up to nine nodes of Spark Streaming (v1.5.1) standalone clusters – one master and eight slaves. The Kafka log directory and the Spark Streaming checkpoint director are both placed in Big Red II's shared Data Capacitor II (DC2) file system, which is connected via a 56-Gb FDR InfiniBand network. By default, the Spark standalone cluster (v1.5.1) only supports a simple FIFO scheduler across applications. To allow multiple concurrent applications, we divide the resources by setting the maximum number of resources each application can use (i.e., parameter "spark.cores.max").

Spark Streaming receives input data streams and divide the data into batches. For a Spark Streaming application to be stable, it is important to set the batch interval so that the system can process data as fast as it is being received. In our implementation, the maximum processing speed for one provenance stream is limited by the power of a single CPU core, since each provenance stream is stored into one RDD partition that is processed sequentially by one Spark task. If the provenance generation rate is constantly higher than the maximum processing speed, we can throttle the generation by introducing delays in the provenance hub to slow down the simulation speed. However, in our experiments, we choose to not throttle the generation rate, but instead we measure the maximum receiving

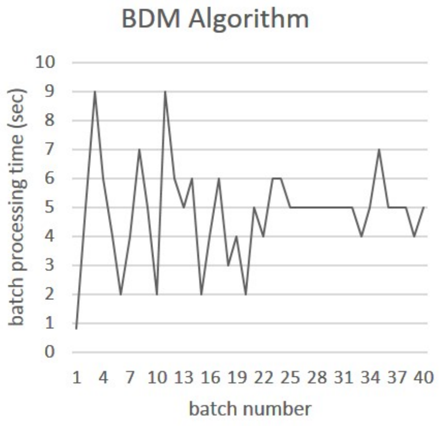


Figure 5.5: The data receiving rate is automatically controlled by the “backpressure” feature in Spark Streaming for the BDM algorithm with a batch interval of 5s.

rate by enabling the “backpressure” feature in Spark Streaming – it automatically figures out the receiving rate and dynamically adjusts it if the processing conditions change. Figure 5.5 shows that how the receiving rate is controlled to keep the batch processing time lower than the batch interval (or sliding interval).

We first measure the throughput and latency of the BDM algorithm running in a single-node Spark Streaming cluster, and the size of its internal state serialized in memory. To determine the maximum throughput under the condition of simply receiving stream elements, we also measure the Spark “collect” operation running alone. As can be seen from Figure 5.6, our proposed BDM algorithm can achieve throughput as high as 10.8MB/s per stream (77% of the maximum throughput of 14MB/s), and latency as low as 1.5 seconds; when increasing the batch interval, the BDM algorithm will have higher throughput but also longer latency. In all scenarios, the maximum size of the internal state (an RDD cached in memory) is the same – 10.2MB.

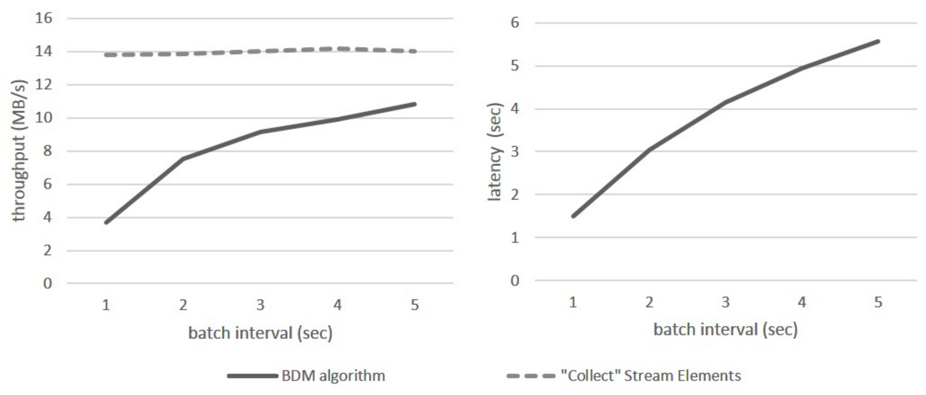


Figure 5.6: Increasing the batch interval increases the throughput as well as the latency of the BDM algorithm.

Next we demonstrate the scalability of our framework to handle increasing number of provenance streams. Since our algorithms do not parallelize the processing within one provenance stream, we can only evaluate its scalability by measuring the scaleup – the ability to keep the same performance levels (response time) when both workload and resources (CPU, memory) increase proportionally. That is, we increase the number of provenance streams together with the number of nodes in the Spark Streaming cluster. While essentially its performance is determined by the underlying Kafka and Spark Streaming system, we can choose different approaches and tune their parameters.

There are two different approaches to send provenance streams into processing: we can either create a separate streaming application to process each provenance stream, or process all provenance streams within one streaming application. The provenance hub can organize the provenance streams accordingly: one provenance stream per Kafka topic (the 1st approach), or one provenance stream per Kafka partition (the 2nd approach). We found during our experiments that the 2nd approach has restricted scalability when using the stateful operation “updateStateByKey()” – it maintains global states for all provenance streams. While the 1st approach is apparently more scalable, it restricts our ability to process multiple provenance streams (e.g., clustering and classification).

The number of parallel tasks in a Spark Streaming job is controlled by the parameter “spark.default.parallelism” by default. Since the actual number of non-idle tasks is determined by the number of RDD partitions (a.k.a. the number of provenance streams), we decide to set “spark.default.parallelism” equal to the number of provenance streams. However, when using the direct mode in Spark-Kafka integration, each Kafka partition occupies one CPU core per node for data receiving, thus there exists a cap on how many streams one node can handle.

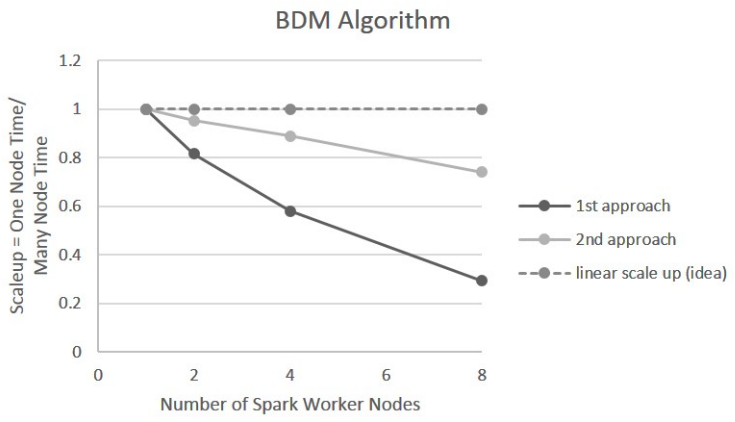


Figure 5.7: Scalability test on the BDM algorithm (5s batch interval).

Results in Figure 5.7 demonstrate that the 1st approach on the BDM algorithm has better scalability than the 2nd approach.

5.6 Applications on Agent-Based Modeling

While the main contributions of this paper are the stream model of provenance and the provenance capture and processing solution, there are various applications that can be developed based on them. We discuss a few of them below:

1. Automatic model calibration generally requires repeated simulation runs with different combinations of parameters [62], which is not suitable for large scale ABMs that run for a long period of time. Our BDM algorithm can solve this problem by providing the key information for parameter readjustment – the dependency between input parameters and mismatching results. When we observe a mismatching variable during the simulation, we can get the relevant input parameters using backward provenance query and then readjust them on-the-fly.
2. It is important to verify that the implementation of the ABM matches its design – model verification. While code walk-through can be useful, it is hard to make sure that each of the thousands of agents performed as expected. We can solve this problem by setting important logical consistency checks over the live provenance stream.
3. Finally, it is typical to run the agent-based simulations with various input parameters to get all possible results – Monte Carlo simulation, and we are interested in terminating/discarding useless simulation runs to save time and resource. By applying k-means clustering algorithm on the window-based representations generated by our WTR algorithm, we are able to group similar simulation runs that can be discarded.

5.7 Summary

In this chapter, we propose a streaming solution to backward provenance query whose target is unknown a priori and the full provenance data is too big to be persisted and processed offline. We first characterize the provenance stream with unique properties that makes it different from a general data stream. Then we develop a framework that can automatically capture the live provenance stream from agent-based simulations running in NetLogo, and then process it continuously in real-time. We propose a streaming algorithm to support backward provenance query using limited space. The framework has been tested with a real-world agent-based model that has thousands of household agents and runs in a supercomputer. The performance results show good throughput, latency and scalability. In the end, we have discussed various real-world applications that we can develop using our framework.

Chapter 6

Temporal Representation for Mining Large Scale Provenance

When provenance data becomes big, the graph representations of provenance (OPM and W3C PROV) are no longer suited to data mining for two reasons: the challenges for analyzing high-dimensional data and the difficulty to place both structural and non-structural information in a single uniform attribute space.

There are two common challenges for analyzing high-dimensional data. The first one is the curse of dimensionality. The complexity of many existing data mining algorithms is exponential with respect to the number of dimensions. With increasing dimensionality, these algorithms soon become computationally intractable and therefore inapplicable for many real applications. Secondly, the specificity of similarities between points in a high dimensional space diminishes. Clustering in a high dimensional space presents tremendous difficulty [26]. It was proven in [28] that, for any point in a high dimensional space, the expected gap between the Euclidean distance to the closest neighbor and that to the farthest point shrinks as the dimensionality grows. This could make many data mining tasks including clustering ineffective and fragile because the model becomes vulnerable to the presence of noise.

The structural and non-structural information of a provenance graph are both very important when we try to discriminate similar provenance graphs and to learn relations between the characteristic of data products and processes that generate them. While structural information can be easily represented as a connectivity matrix and the non-structural information can be represented using multidimensional vectors, the unification of structural and non-structural information from the provenance graph remains to be a challenge.

In this chapter, we propose a new approach to deal with the large volumes of provenance, and that is to selectively reduce the feature space while simultaneously preserving interesting features so that data mining on the reduced space yields provenance-useful information. More specifically, we propose a representation of provenance using logical time that reduces the feature space of provenance. We posit that the temporal presentation is an efficient and useful statistical feature representation of provenance. The mining tasks to be performed on the proposed representation include: generating patterns that describe and distinguish the general properties of the datasets in provenance repositories (by training classifier and mining association rule set), finding variants to detect faulty provenance data (by checking cluster centroids in the case where correct and faulty provenance are naturally separated into different clusters) and discovering more descriptive knowledge of provenance clusters (by mining association rules that reflects workflow variants). We also apply the temporal representation to the provenance stream model defined in Chapter 5 so that mining live provenance streams becomes possible.

6.1 Provenance Graph Partitioning

We use Lamport's logical clocks [98] as the basis for an abstract representation of provenance. We propose a graph partitioning algorithm based on the logical clocks to split a provenance graph into leveled subsets[1] with temporal order among different levels.

The graph partitioning algorithm we propose in this section is the basis for our abstract representation. Our approach has the assumption that the provenance graphs to which the representation is applied are compliant with the Open Provenance Model [117]. A discussion on how to extend it to the W3C PROV model is in Section 7.2.

6.1.1 Partial Ordering

Lamport determines a total ordering of events in a distributed computer system based on logical time order. Since the OPM reference specification [117] defines edges as causal relationships, we define the "happened before" relation in a provenance graph based on its causal relationships.

Definition The "happened before" relation, denoted by "$\rightarrow$," on the set of nodes in a provenance graph is the smallest relation satisfying the following two conditions:

1. If a and b are nodes that have an edge between them, and a is the cause, then $a \rightarrow b$
2. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$

We assume that $a \not\rightarrow a$ for any node, which implies that $\rightarrow$ is an irreflexible partial ordering on the set of all nodes in the provenance graph. We define node a and b to be concurrent nodes if $a \not\rightarrow b$ and $b \not\rightarrow a$. For example, in Figure 6.1(c): Node "6" $\rightarrow$ Node "multiplier," and Node "multiplier' $\rightarrow$ Node "54" so that Node "6" $\rightarrow$ Node "54"; Node "6" and Node "9" are concurrent nodes.

While the current OPM reference document forbids cycles, a new definition [97] allows the presence of derived-from cycles (simple cycles composed of derived-from edges) after

[1]We will call it "subset" or "level" for short in the rest of the chapter

a merge operation. However, an OPM graph resulting from a typical experimental provenance collection procedure, which is the target of this study, does not contain such cycles. In addition, new definitions (e.g., [97]) avoid using the term *causal relationship*, but the constraints in their temporal theory are more similar to Lamport's ordering, and they are still using "cause" to represent an edge source and "effect" to represent an edge destination. Thus our definition above also works in this case.

6.1.2 Logical Clock-P

We propose the Logical Clock-P, a function C that takes a node as input and produces an integer as output. This function maps an integer to each node of a given provenance graph. The correct logical clocks must satisfy the condition that if a node a occurs before another node b, then a should happen at an earlier time than b. We state this condition more formally as follows.

Definition *Clock Condition:* The Clock condition satisfies the following condition: For any node a and b, if $a \to b$ then $C(a) < C(b)$.

6.1.3 Strict Totally Ordered Partition

With Logical Clock-P defined, we define a strict totally ordered partition that divides a provenance graph into a list of non-empty leveled subsets. A typical provenance graph has three kinds of nodes: artifacts, processes, and agents. A partitioning of a provenance graph is a set of non-overlapping and non-empty subsets of nodes based on the logical clocks. More precisely, a partition of provenance graph $G = (V, E)$, where V denotes the set of all nodes and E denotes the set of all edges, is defined as follows:

Remark For a provenance graph $G = (V, E)$, partition V into k subsets $V_1, V_2, \ldots, V_k$ such that:

1. $V_1, V_2, \ldots, V_k \in V$ and $\bigcup_{1}^{k} V_i = U$
2. $\forall\, i \neq j$ and $1 \leq i\,, j \leq k$, $V_i \cap V_j = \phi$
3. $\forall\, a\,,\, b \in V_i$, we must have $C(a) = C(b)$, and the node type of a is the same as the node type of b

In addition, we define an "appears before" relation to give the level order on the set of all these subsets. Naturally, a node with a smaller Logical Clock-P comes before a node with a larger Logical Clock-P. Furthermore, for nodes with the same Logical Clock-P, we put agents before processes and processes before artifacts. This is because of the implicit order in node definition [117]: an agent is defined as an entity enabling process execution, a process is defined as an action resulting in an artifact, and an artifact is defined as a state in a physical object. Though this implicit order can be different from the real time order, it is still meaningful for us when putting concurrent nodes into a sequential representation.

Definition The "appears before" relation "$\Rightarrow$" on the set $\{V_1, V_2, \ldots, V_k\}$ in a provenance graph needs to satisfy the following conditions:

1. $\forall a \in V_i$, $\forall b \in V_j$, if $C\,(a) < C\,(b)$, then $V_i \Rightarrow V_j$

2. $\forall a \in V_i$, $\forall b \in V_j$, if $C\,(a) = C\,(b)$, and node type of a is agent, and node type of b is process, then $V_i \Rightarrow V_j$

3. $\forall a \in V_i$, $\forall b \in V_j$, if $C\,(a) = C\,(b)$, and node type of a is agent, and node type of b is artifact, then $V_i \Rightarrow V_j$

4. $\forall a \in V_i$, $\forall b \in V_j$, if $C\,(a) = C\,(b)$, and node type of a is process, and node type of b is artifact, then $V_i \Rightarrow V_j$

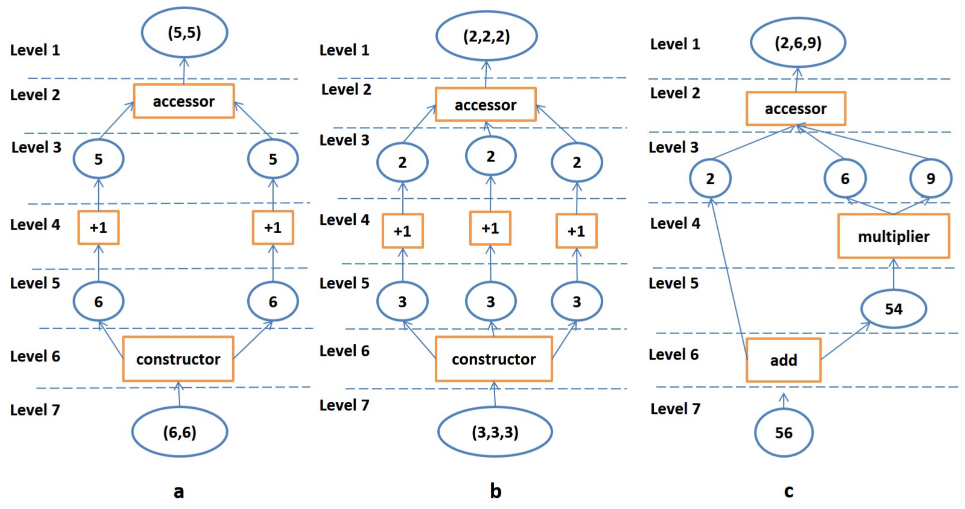


Figure 6.1: Temporal partition: (a) is an example provenance graph from [117]; (b) is from the same experiment as (a) with different input data; (c) has a similar graph structure to (a) and (b) but with different nodes.

A partition of a provenance graph with the “appears before” relation on the set $\{V_1$, V_2, . . ., $V_k\}$ is asymmetric, transitive and also totally ordered, but not unique. We show an example partitioning generated by our Logical-P algorithm in Figure 6.1, where the level with the smaller number (e.g., Level 1) “appears before” the level with larger number (e.g., Level 2, Level 3).

6.1.4 Provenance Graph Partitioning Algorithm (Logical-P Algorithm)

Given any provenance graph (we are using the XML representation [118]), we generate a unique strict totally ordered partition with the Logical-P algorithm (see algorithm 2). Steps 1–16 are derived from the topological sorting algorithms of Kahn [91], which has linear time in the number of nodes plus the number of edges $O(|V| + |E|)$. The time complexity of step 18 depends on the sorting algorithm that is used. For heapsort, the complexity is

$O(klogk)$, where k is the number of leveled subsets in the partition. Note that steps 9–11 give the maximum possible Logical Clock-P value to nodes that have multiple causes.

Algorithm 2 Logical-P algorithm

```
Require: Provenance graph $G$
 1: $S \leftarrow$ Set of all nodes in G with no outgoing edges
 2: for nodes k in S do
 3:     assign 0 to C(k)
 4: end for
 5: while S is non-empty do
 6:     remove node n from S
 7:     for node m with edge e from m to n do
 8:         remove edge e from graph
 9:         if C(n) + 1 $>$ C(m) then
10:             assign C(n) + 1 to C(m)
11:         end if
12:         if m has no other outgoing edges then
13:             insert m into S
14:         end if
15:     end for
16: end while
17: Group nodes with same Logical Clock-P value and node type into one subset
18: Sort subsets according to "appears before"
```

Table 6.1: Workflow type and number of leveled subsets for a successful execution instance

Workflow Type	**Number of leveled subsets in the provenance partition of a complete run**
LEAD North American Mesoscale (LEAD NAM) forecast (weather)	10
SCOOP ADCIRC (coastline)	5
NCFS (ocean)	10
Gene2Life (bio)	10
Animation (CS)	8
MotifNetwork (bio)	10

6.1.5 Partitioning the Provenance Graphs in the Datasets

We use the Logical Clock-P algorithm to partition the provenance graphs (in the form of OPM complaint XML strings or files) in the experimental datasets. Table 6.1 shows the six types of workflow in the 10GB dataset and the number of leveled subsets in the provenance partition for a successful run, to be referred back to in Section 6.4.2.

Once a provenance graph is partitioned into an ordered list of leveled subsets, we can create the representations of each leveled subset and organize them into a sequence representation of the graph.

6.2 Feature Selection

6.2.1 Statistical Feature Space

A typical provenance graph is a fully-labeled graph with annotations (both nodes and edges have labels and annotations), so direct representations such as feature vector spaces will result in high dimensional datasets that are not suitable for large scale mining tasks. We address this issue by using attribute transformation [26] to define a new statistical feature space. That is, instead of making each attribute of each node in the provenance graph into a feature, we can summarize the same attribute of all nodes into one (or several) features. In this way, we can select fewer features from the much smaller statistical feature space to represent each subset (subgraph).

We first give the definition of a feature space of node subset, and then extend this definition to a statistical feature space by introducing a statistical feature function.

Definition For a *feature vector subset* $N = (V, F, D)$, where $V = \{v_1, ..., v_n\}$ denotes the node subset, the function $F : V \rightarrow D_1 \times D_2 \times ... \times D_d$ is a *feature function* that assigns a feature vector to any node $v \in V$, and the set $D = \{D_1, D_2, D_3, \ldots, D_d\}$ is called the *feature space* of N.

Definition For a *statistical feature vector subset* $N' = (V, F, G, D, S)$, a *statistical function* $G : D_i \times D_i \times ... \times D_i \rightarrow S_i$ applies statistical operators such as max, min, avg, std.dev, std.err, sum, variance, and mode to feature $D_i \in D$ of all nodes in V, and the set $S = \{S_1, S_2, S_3, \ldots, S_d\}$ is called the *statistical feature space* of N. Note that feature D_i can be either numerical data or not, and choosing the right statistical operator is data dependent.

Here is an example of how to create a statistical feature space from the original feature space of a subset. The features of a provenance graph node include its attribute feature such as its labels and annotations, and its structural feature such as the attributes of its incoming/outgoing edges. A simple node attribute feature can be the number of characters in the node label, and a simple node structural feature can be its in-degree or out-degree. So the feature space for level 2 in Figure 6.1(a) can be $D = \{$ number of characters in node label, number of in-degree, number of out-degree $\} = \{(1,1),(1,1),(1,1)\}$. After applying the statistical operation "avg" to D, we get its statistical feature space $S = \{$ average number of characters in node label, average number of in-degree, average number of out-degree $\} = \{1,1,1\}$.

6.2.2 Feature Selection from Statistical Feature Space

Having more features should—in theory—result in greater discriminating power, however, in practice, adding irrelevant or distracting attributes to a dataset often confuses machine learning systems. Because of the negative effect of irrelevant attributes on most machine learning schemes, it is common to precede learning with an attribute selection stage that strives to eliminate all but the most relevant attributes. The best way to select relevant attributes is manually, based on a deep understanding of the learning problem and what the attributes actually mean. However, automatic methods can also be useful [161].

Manual Feature selection, Structured and Attribute

The selection of a good feature set from a statistical feature space depends upon both the mining targets and the nature of the provenance. Samak et al. [134] demonstrated that a clustering algorithm like *k-means* can group similar workflows to separate and identify failure workflows from normal workflows. They assume that the application domain (which we call the type) of workflows are already known, and clustering is carried out on workflows of different application domains separately. However, we make few assumptions about our dataset, and do not know the workflow type of each provenance instance, so our first target in unsupervised clustering is to group provenance instances based on their original experiment (the workflow type or the application domain for workflows). We need to select a feature set that minimizes the distance between two provenance representations derived from the same experiment; this distance should be smaller than the distance between two representations derived from different experiments. So for unsupervised clustering, we should first study the differences between provenance graphs and the efficiency of using attribute feature set and structural feature set in discriminating these differences.

We assume provenance graphs from related experiments have similar structure and similar attribute information (Figure 6.1(a) and Figure 6.1(b)); while provenance graphs from different experiments are either different in attribute information (Figure 6.1(a) and Figure 6.1(c) have different node labels), or different in structural information (Figure 6.1(a) and Figure 6.9(b) have different topology). While using any feature set, Figure 6.1(a) and Figure 6.1(b) should be clustered together.

Based on this assumption, we create a simple *attribute feature set* that includes "average number of characters in label" to discriminate between Figure 6.1(a) and Figure 6.1(c), and a simple *structural feature set* that includes "average number of in-degree/out-degree" to discriminate between Figure 6.1(a) and Figure 6.9(b). The nodes' labels in the provenance graph are agent names for "Agent" nodes, process names for "Process" nodes and data values for "Artifact" nodes. Santos et al. [137] use the module names to construct the vector-space based representation of workflow graphs for clustering, which produce good clustering results comparable to the results obtained using the more costly structural representation. Here we choose the numerical value "number of characters in label" and statistical operator "average" for simplicity, but we could also choose "node label" as nominal value with statistical operator "mode." Each leveled subset in the resulting partition consists nodes of the same type, and we map the type of nodes from their textual values "Agent", "Process", "Artifact" into numerical values 0, 1, 2. Note that while we use this

Table 6.2: Euclidean distance for attribute and structural feature sets

Figure	Distance in Attribute Feature Set
6.1(a) - 6.1(b)	0.9110
6.1(a) - 6.1(c)	1.3807
6.1(b) - 6.1(c)	1.6787
	Distance in Structural Feature Set
6.1(a) - 6.1(b)	0.7454
6.1(a) - 6.9(b)	2.5878
6.1(b) - 6.9(b)	2.5805

mapping to simply subsequent data mining operations, it might result in wrong implications of distance (for example, the Euclidean distance from "Artifact" to "Agent" is twice the distance to "Process" after mapping). A better solution might be to maintain the nominal values and to specify in the distance calculation that these nominal values are equally far from each other.

Specifically, for the *attribute feature set* we capture: <*Type of nodes in subset, Num nodes in subset, Avg num characters in node name*> which for Figure 6.1(c), gives:

$(< 2, 1, 7 >, < 1, 1, 8 >, < 2, 3, 1 >, < 1, 1, 10 >, < 2, 1, 2 >, < 1, 1, 3 >, < 2, 1, 2 >)$

For *structural feature* set we capture the following features: <*Type of nodes in subset, Num nodes in subset, Avg num in-degree of nodes in subset, Avg num out-degree of nodes in subset* > from each subset V_i. The resulting provenance partition of Figure 6.1(c) is represented as:

$(< 2, 1, 1, 0 >, < 1, 1, 3, 1 >, < 2, 3, 1, 1 >, < 1, 1, 1, 2 >, < 2, 1, 1, 1 >, < 1, 1, 1, 2 >, < 2, 1, 0, 1 >)$

Table 6.2 gives the Euclidean distance calculated from the attribute feature set and the structural feature set. We can use these distances to tell whether two graphs are relatively closer to or farther from each other than the others. As discussed earlier, graph 1(a) from Figure 6.1 is very similar to 1(b). Their distance is close for both attribute and structural feature sets. Graph 1(a) – 1(c) are different from an attribute perspective but similar structurally. The attribute difference is illustrated in top four rows of Table 6.2. Finally, the graph in Figure 6.9(b) is distinct structurally and this is evident in its Euclidean distance from graph 1(a) and 1(b). Note that attributes are normalized before calculating Euclidean distance, so that all attributes contribute equally to the result. This normalization scales each data variable into a range of 0 and 1 using the following equation:

$$x^{normalization} = \frac{x - x^{min}}{x^{max} - x^{min}}$$

where $x^{normalization}$ represents the normalized value, x represents the value of interest, x^{min} represents the minimum value and x^{min} represents the maximum value.

After manually choosing a feature set, the next step is to select features from all leveled subsets. We can further remove features that have zero standard deviations, since they are not contributing to the distance calculation. However, we are not considering applying any automatic feature selection algorithms. This is because we do not have class labels in unsupervised clustering to tell if a feature is relevant or not, and the dimensionality of the proposed structural feature set is very high for clustering purposes.

The simple structural feature set discussed above has been tested in [40] on the 10GB dataset (Section 3.4.3), where we choose structural feature set over the attribute feature set or other more complicated feature sets in unsupervised clustering for the purpose of a strong evaluation – we do not want to take advantage of obvious difference in node attributes. The disadvantage of that feature set is that if we have two provenance graphs with the same structure but with different node/edge information, then it would be impossible to distinguish the two through graph structure alone. In this thesis, we propose an extension to it by further splitting edges into different types in OPM – *used*, *wasGeneratedBy*, *wasControlledBy*, *wasTriggeredBy*, and *wasDerivedFrom* – and then calculate the average, so that we can discriminate graphs that have similar structure but are semantically different. The size of the OPM graphs extracted from the original database is 2.1GB, while the size of the temporal representation is 16.9MB, a decrease by several orders of magnitude.

However, the different workflows in the AMSR-E dataset (Section 3.4.2) can have identical graph structure, thus it is no longer sufficient to use only structural information. Instead we use a feature set that contains both structural and attribute information, however, still avoiding the use of node labels for the purpose of a strong evaluation. That is, in addition to the simple structural feature set, we apply the statistical operator "average" on all numeric node attributes, including *ESDTVersion*, *PGEversion*, *artifact-size*, *qapercentmissingdata* and *qapercentoutofboundsdata*. Specifically, the feature set is $<$*Type of nodes in subset, Number nodes in subset, Avg number of in-degree, Avg number of out-degree, Avg of ESDTVersion, Avg of PGEversion, Avg of artifact-size, Avg of qapercentmissingdata, Avg of qapercentoutofboundsdata* $>$. The size of AMSR-E dataset as XML files is 83.0MB, and its initial representation using this feature set is 479KB. After cleaning attributes with zero standard deviation, the final representation for unsupervised clustering is 375KB in size (44 attributes per instance).

Manual Feature Selection, Small feature set

Association rule mining is less efficient when dealing with long sequences. If we were to select 4 features for each subset, there would be 40 attributes in a provenance representation for a workflow instance having 10 subsets (levels). Instead we select one target attribute, *Number of nodes in the subset*, for each subset (level). This new short sequence is sufficient to expose the structural variants with different number of intermediate data products/processes.

Automatic Feature Selection

Supervised learning, unlike unsupervised learning, requires class labels (such as the workflow type in our case), so it is natural to keep only the features that are relevant to or lead to these given labels [61]. There are many potential benefits of feature selection in supervised learning: facilitating data visualization and data understanding, reducing the measurement and storage requirements, reducing training and utilization times, and defying the curse of dimensionality to improve learning performance [80]. When selecting a good attribute subset, there are two fundamentally different approaches. One is to make an independent assessment based on general characteristics of the data; the other is to evaluate the subset using the machine learning algorithm that will ultimately be employed for learning. The first is called the filter method because the attribute set is filtered to produce the most promising subset before learning commences. The second is called the wrapper method because the learning algorithm is wrapped into the selection procedure [161].

For the 10GB dataset, the structural feature set that we proposed in the previous section leads to 120 features for a provenance graph that has 10 subsets (levels) in its partition. We take both the filter method and the wrapper method to further selecting features from these 120 features. Specifically, we use: ClassifierSubsetEval, a wrapper method implemented in Weka that evaluates attribute subsets on training data or testing set and uses a classifier to estimate the "merit" of a set of attributes; CfsSubsetEval [81], a filter method that assesses the predictive ability of each attribute individually and the degree of redundancy among them, preferring sets of attributes that are highly correlated with the class but with low intercorrelation.

We use NaiveBayes [89] as the evaluating classifier for ClassifierSubsetEval, and it selects 7 out of 120 features, which means it can significantly shorten training times and enhance generalization by reducing overfitting. The seven selected features are:

<Avg num of incoming "was triggered by" edges in Level 1, Avg num of incoming "was generated by" edges in Level 3, Avg num of outgoing "was generated by" edges in Level 3, Num nodes in Level 4, Avg num of incoming "was generated by" edges in Level 4, Avg num of outgoing "was derived from" edges in Level 6, Num nodes in Level 9>

CfsSubsetEval also selects 9 out of 120 features, namely:

<Avg num of incoming "was triggered by" edges in Level 1, Num nodes in Level 2, Avg num of incoming "was derived from" edges in Level 2, Num nodes in Level 3, Avg num of incoming "was generated by" edges in Level 3, Num nodes in Level 4, Avg num of outgoing "was derived from" edges in Level 4, Avg num of outgoing "was derived from" edges in Level 6, Avg num of outgoing "was triggered by" edges in Level 7>

Note that the nine features share three common features with the seven features.

For the AMSR-E dataset, we also use CfsSubsetEval, which selects two attributes from the initial feature set: *average number of in-degree* in level 2 and *average value of PGEversion* in level 4. This reduces the representation's size from 80.3MB to 21.8KB.

6.3 Methodology

We assess the efficacy of the temporal representation in revealing the kinds of information in which we are interested, for instance, “Given a new provenance graph that is either complete or incomplete, can we determine the type of workflow that generated it?” “Can we detect interesting variants including failed workflows?” and “What are the differences between variants, and whether the differences come from workflow execution or provenance capture.” This study extends the earlier study, replaces frequency domain representation with ensemble representations, and includes a more methodical evaluation of mining techniques applied to the representation.

The mining tasks that we explore include generating patterns that describe and distinguish the general properties of the datasets in provenance repositories (by training classifier and mining association rule set), finding variants to detect faulty provenance data (by checking cluster centroids in the case where correct and faulty provenance are naturally separated into different clusters) and discovering more descriptive knowledge of provenance clusters (by mining association rules that reflects workflow variants). We use the data mining software Weka [82] in our experimental evaluation, which has the implementations for all the mining algorithms that we refer to.

On choosing the best unsupervised clustering algorithm for our use, we investigate the performance of three different types of popular clustering algorithm: centroid-based, distribution based and density based, using k-means [101], DBScan [64] and EM algorithms [57] respectively. As shown in Section 6.4.2, k-means gives the best results, in that it produces clusters with the best quality and centroids that can be further used in failure detection.

Hence, we choose the k-means algorithm (SimpleKMeans in Weka) to show the usefulness and applicability of our proposed temporal representation. Using the similarity measure between sequences from Section 6.2.2, we cluster the temporal sequences to discover a number of clusters, say K, to represent the different sequences. The centroids we discovered from k-means algorithm are further compared with each other using our provenance graph matching algorithm (see Algorithm 1) to answer the problem of what the differences are and where they come from.

However, using Euclidean distance as the similarity measurement limits the application of the k-means to representation sequences of same length. We identified several solutions to this problem: 1) pre-grouping representations by length, which has the advantage of pre-separating representations using the a priori knowledge (length); 2) transforming representations into frequency domain, which can further reduce the dimensionality, but has the disadvantage of being meaningless for association rule mining; 3) filling missing features with special values of 0 or -1, which is the simplest but has problematic implications of distances. In [40] we already tested approach 1) and 2), and in this thesis we evaluate 1) in Section 6.4.2 and 3) in Section 6.4.3 instead.

Another important use of temporal representation is to train classifiers (supervised learning), for example to categorize the workflow type for a new provenance record. We assume a use case in which given a new provenance record, a researcher wants to categorize its workflow type based on the existing provenance records in the 10GB database. To achieve this, we train a classifier for workflow type from the Logical Clock-P representation. As discussed in Section 6.2, we can utilize some automatic feature selection algorithms for supervised learning. The feature selection algorithms we use in Weka are ClassifierSubsetEval – a wrapper method implemented in Weka, which evaluates attribute subsets on training data or testing set and uses a classifier to estimate the “merit” of a set of attributes, and CfsSubsetEval [81] – a filter method that assesses the predictive ability of each attribute individually and the degree of redundancy among them, preferring sets of attributes that are highly correlated with the class but with low intercorrelation.

The discovery of relevant association rules is one of the most important methods used to perform data mining on transactional databases [16]. An effective algorithm to discover association rules is the Apriori algorithm [8]. Adapting this method to better deal with temporal information is beyond our current research; instead we apply the Apriori method (Weka) on the clusters to get more descriptive knowledge of that cluster.

6.4 Data Mining Evaluation

6.4.1 Evaluation Metrics

Purity [171] and the Normalized Mutual Information (NMI) [173] are used to compare the performance of different clustering techniques. Purity assumes that all samples of a cluster are predicted to be members of the actual dominant class for that cluster. However, high purity can be achieved when the number of clusters is large. For example, purity would be one when each graph belongs to its own cluster. We cannot use purity to trade off the quality of the clustering against the number of clusters, so we also use NMI clustering evaluation metric. NMI [86] is defined as the mutual information between the cluster assignments and a pre-existing labeling of the dataset normalized by the arithmetic mean of the maximum possible entropies of the empirical marginals. k-means performance is evaluated using Within-Cluster Sum of Squares (WCSS), purity, and Normalized Mutual Information (NMI) metrics.

As for the classification, we use NaiveBayes [89] as the evaluating classifier for ClassifierSubsetEval, and we test different classification methods on the features selected by the CfsSubsetEval, including the NaiveBayes and the Random forests [31]. The accuracy is evaluated using the 10-fold cross-validation method.

We use association rules to expose variants amongst workflows, and apply Weka's Apriori algorithm to discovering the association rules on the resulting clusters generated from pre-grouped unsupervised clustering, to see if there are rules that can expose the variants that we are looking for.

6.4.2 Unsupervised Clustering, Pre-grouped

As discussed in Section 6.3, we can group together the provenance representations by their lengths and then apply the k-means algorithm within each group.

This first order breakdown by temporal subset length is shown in Figure 6.2. 46% of the provenance representations have the largest number (10) of subsets, while only a small portion (2%) have very small number of subsets (2, 3); the latter subject to early failures in the workflow execution and dropped provenance notifications.

For clustering within a grouping, we apply the k-means clustering algorithm with Euclidean distance to the representation sequences inside each group. Note that we removed the attributes with zero standard deviation, and Weka's Euclidean distance function normalizes attributes by default, just like what we showed in Section 6.2.2.

To choose the number of clusters, k, we plot the within-cluster sum of squares (WCSS) for each subset (level) and look for the "elbow point". Figure 6.3 plots WCSS for workflow instances having 3 subsets (a) and 4 subsets (b). For the former, k is chosen as $k = 2$, for latter, we choose $k = 3$ because WCSS decreases slowly after k reaches 3. We use the same procedure to choose k for the remaining groups. Finally k-means is applied for each

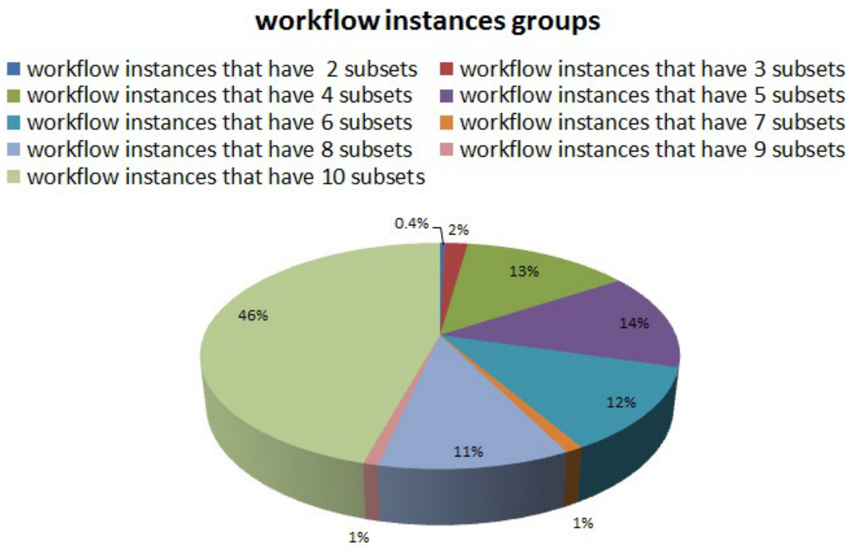


Figure 6.2: Grouping results based on temporal subset length (level depth)

group creating an overview shown in Figure 6.4.

Clustering graphs of the same temporal representation length requires different values of k, as shown in Figure 6.4. We find that the number k determined this way is slightly smaller than the number of actual classes within each group. However, it still generates major clusters and has good clustering quality (to be evaluated below). In fact, there is a trade-off between the number k and the value WCSS, since larger k always results in smaller WCSS but also has the potential to split the natural cluster into smaller clusters.

As discussed before, we evaluate the quality of resulting clusters by computing the purity and Normalized Mutual Information (NMI). The purity and NMI are not very high when the workflow representation contains a small number of subsets (levels), as shown in Figure 6.5. This is because most workflow instances that have smaller graph sizes are incomplete because of incomplete execution or dropped provenance notifications (as shown in Figure 6.9), so they are difficult to accurately cluster using only their structural informa-

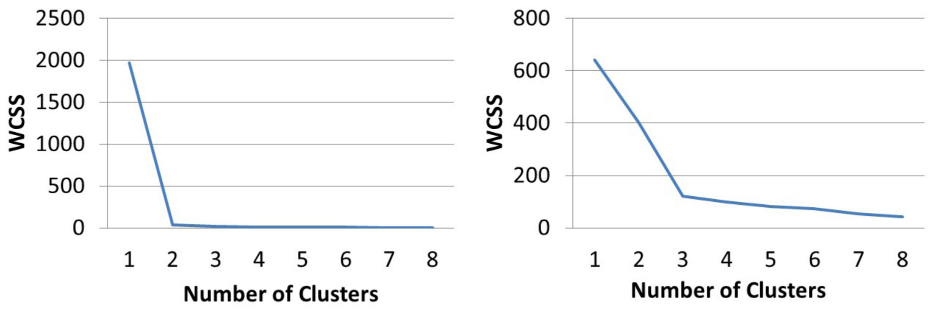


Figure 6.3: WCSS as function of number of clusters for different groups of representation sequences: (a) WCCS for workflow instances having 3 subsets, and (b) WCCS for workflow instances having 3 subsets.

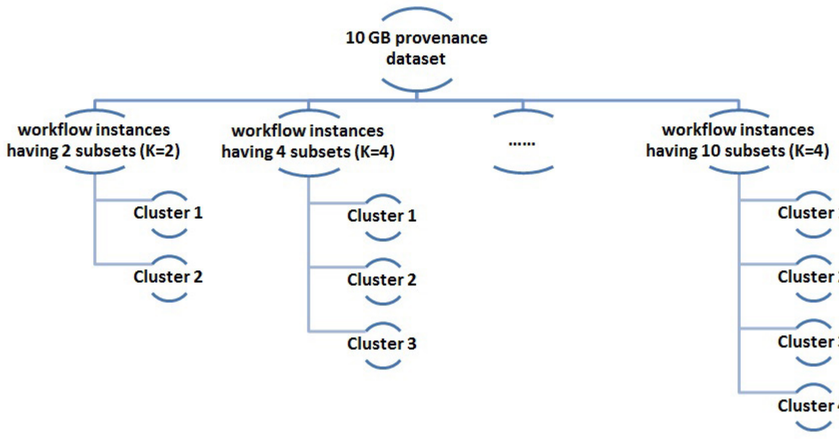

10 GB provenance dataset
workflow instances having 2 subsets (K=2)
workflow instances having 4 subsets (K=4)
......
workflow instances having 10 subsets (K=4)
Cluster 1
Cluster 2
Cluster 1
Cluster 2
Cluster 3
Cluster 1
Cluster 2
Cluster 3
Cluster 4

Figure 6.4: High level view of 10GB provenance dataset created from its structural information only

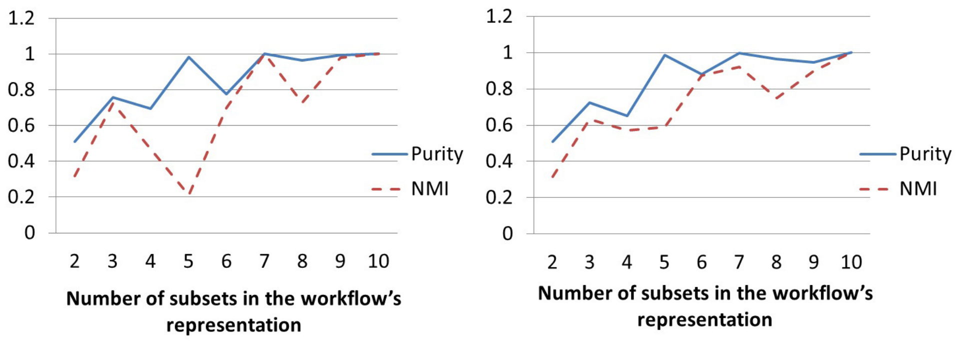


Figure 6.5: Purity and NMI results as external evaluation criteria for k-means cluster quality by workflow instance group. The graph on the left shows the clustering results from an initial version of temporal representation, while the graph on the right shows the results from the extended temporal representation.

tion. But the purity and NMI increases as the number of subsets (levels) in the provenance representation increases, and the workflow provenance that has most provenance information (with number of subsets > 4) can still support clustering well. This indicates that our representation of workflow provenance provides high level of clustering efficiency and is also robust in dealing with incomplete provenance.

Since k-means uses a random number seed to determine the starting centroids of the clusters, it might be helpful to run it with different random number seeds to see the impact. We run SimpleKMeans (Weka) on the 10-subset temporal representations with random number seed ranging from 1 to 10, and the result in Figure 6.6 shows good stability.

As we discussed earlier, k-means is representative of centroid based clustering algorithms, and we also want to evaluate the clustering performance of DBScan, a representative of distribution based algorithms, and EM, a representative of density based algorithms. The results in Figure 6.7 show that the performance of EM is worse than k-means (Fig-

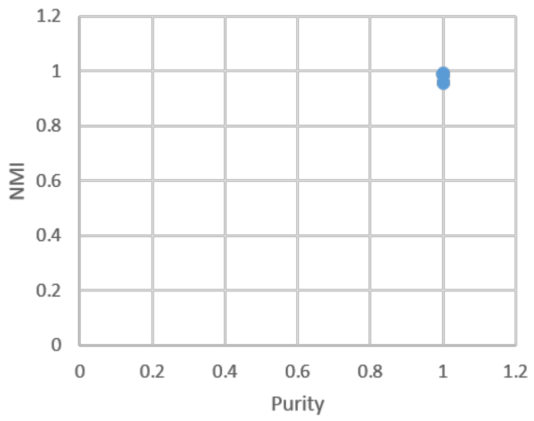


Figure 6.6: Repeating the SimpleKMeans on the 10-subset temporal representations with random number seed ranging from 1 to 10.

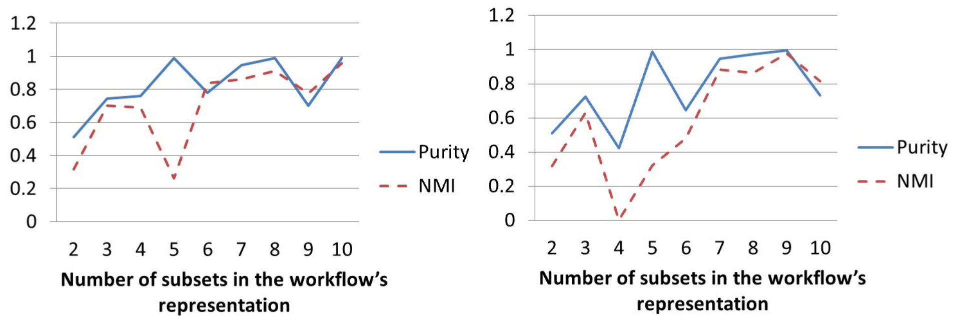


Figure 6.7: Purity and NMI results as external evaluation criteria for EM and DBScan clustering quality. The graph on the left shows the evaluation of clustering results of EM clustering, while the graph on the right shows the results for DBScan clustering. Both results come from clustering extended temporal representation.

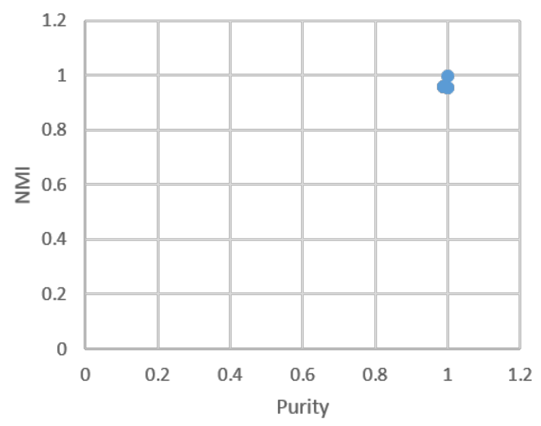


Figure 6.8: Repeating the EM on the 10-subset temporal representations with random number seed ranging from 1 to 10.

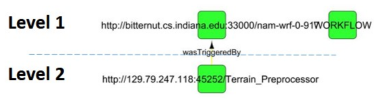
ure 6.5), and so is DBScan. It is because of 1) the characteristic of 10GB database: EM is based on a gaussian distribution, however, the workflows in 10GB database do not follow this distribution, and 2) the purpose of our clustering: we want to separate faulty provenance from correct provenance, and the structural feature set we proposed can support this if using Euclidean distance and k-means. However, density based algorithms like DBScan focus on finding major components rather than classifying outliers/minorities. Other advantages of k-means include that it is highly scalable algorithm, and that we can further use the centroid graphs to detect clusters of incorrect workflow instances and discover problems inside incorrect workflow instances.

Similar to k-means, EM also relies on a random number seed to generate the initial clusters. Figure 6.8 shows that running EM clustering on our temporal representations with different random seed values produces stable results.

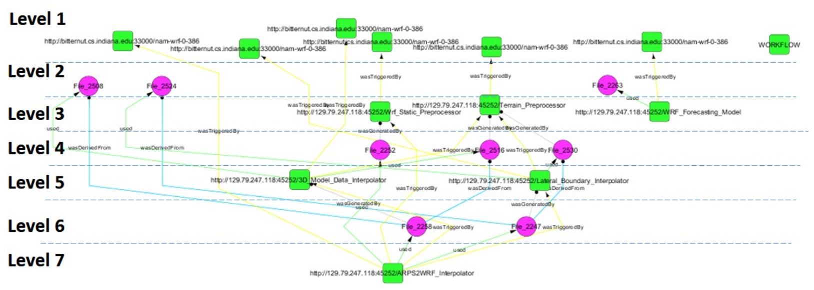
To help understand how to identify clusters of incorrect workflow instances, Figure 6.9 shows the provenance graphs of several centroids. We deliberately choose provenance graphs from the LEAD North American Mesoscale (NAM) forecast workflow [125] because it best illustrates failures in provenance capture. It turns out that the NAM provenance graph with 10 subsets (levels) is a complete graph, while difficult to discern, this is evidenced by an artifact (circle) at bottom of graph. The NAM provenance graphs with less than 10 subsets (levels) partition the graph, all versions of which are incomplete and caused by failures or dropped notifications. The NAM provenance graph with 2 subsets (levels) consists of some units of a complete provenance graph, which is very likely the result of failures.

The way to discover problems inside incorrect workflow instances is to apply the provenance matching algorithm (see Algorithm 1) to centroid graphs. Basically, the matching algorithm matches sub-graph based on both node/edge attribute and local topology. Matching a problematic provenance graph against a known correct provenance graph reveals the missing nodes and missing edges that lead to disorganized parts. So that we can know where the problem exactly is, and whether the problem comes from experiment itself (missing process, Figure 6.10) or provenance capture (missing notification, Figure 6.11).

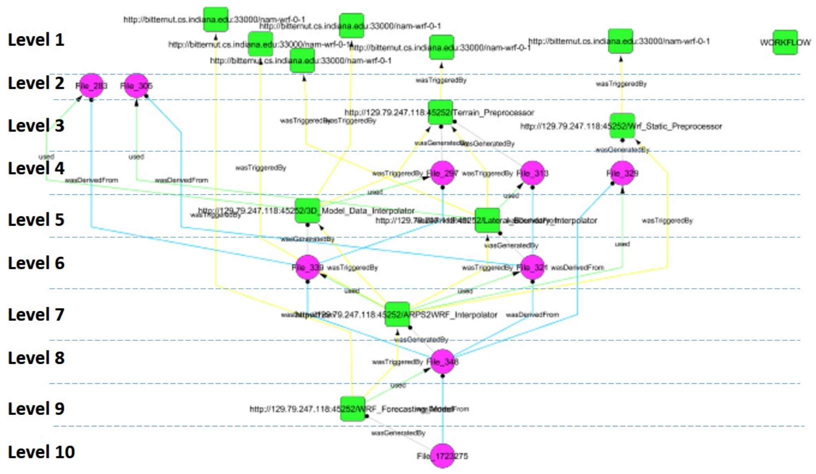
On the AMSR-E dataset, we also use the k-means (SimpleKMeans in Weka) and repeat the same clustering experiments. The clustering performance on pre-grouped representations (Figure 6.12) is also good. We separate representations into a 4-subset group, a 5-subset group and a 6-subset group. It is no longer the case that 4-subset group is the production of failures, and in fact the 4-subset group has the majority of all instances (91%). That means the AMSR-E dataset has less failures than the 10GB semi-synthetic dataset. However, we are still able to discover variants – many AMSR-E workflow instances belong to the same workflow type but are separated into different clusters. There are three types of variants discovered: 1) the same type of workflows have completely different structures,


Level 1
http://bitternut.cs.indiana.edu:33000/nam-wrf-0-917
WORKFLOW
wasTriggeredBy
Level 2
http://129.79.247.118:45252/Terrain_Preprocessor

(a) NAM workflow graph has 2 levels


Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
http://bitternut.cs.indiana.edu:33000/nam-wrf-0-386
WORKFLOW
File_2508
File_2524
File_2283
http://129.79.247.118:45252/Wrf_Static_Preprocessor
http://129.79.247.118:45252/Terrain_Preprocessor
http://129.79.247.118:45252/WRF_Forecasting_Model
File_2252
File_2516
File_2530
http://129.79.247.118:45252/3D_Model_Data_Interpolator
http://129.79.247.118:45252/Lateral_Boundary_Interpolator
File_2238
File_2247
http://129.79.247.118:45252/ARPS2WRF_Interpolator
wasTriggeredBy
wasGeneratedBy
wasDerivedFrom
used

(b) NAM workflow graph that has 7 levels


Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Level 8
Level 9
Level 10
http://bitternut.cs.indiana.edu:33000/nam-wrf-0-1
WORKFLOW
File_283
File_305
http://129.79.247.118:45252/Terrain_Preprocessor
http://129.79.247.118:45252/Wrf_Static_Preprocessor
File_297
File_313
File_329
http://129.79.247.118:45252/3D_Model_Data_Interpolator
http://129.79.247.118:45252/Lateral_Boundary_Interpolator
File_339
File_321
http://129.79.247.118:45252/ARPS2WRF_Interpolator
File_348
http://129.79.247.118:45252/WRF_Forecasting_Model
File_1723275
wasTriggeredBy
wasGeneratedBy
wasDerivedFrom
used

(c) NAM workflow graph that has 10 levels

Figure 6.9: Provenance graphs of several centroids. Square nodes represent processes, and circles represent artifacts. The graph is read top to bottom, with earlier activity at the top.

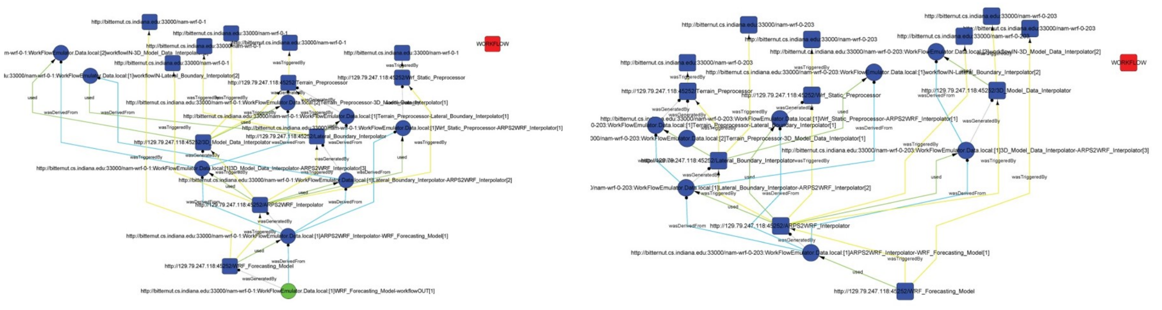

(a) Centroid NAM workflow with 10 levels (b) Centroid NAM workflow with 9 levels

Figure 6.10: Provenance graph on left is the complete provenance of a successful execution. Matching it with the provenance graph on right shows that the right one is a failure, because of that the final data product (green) in left graph cannot be matched.

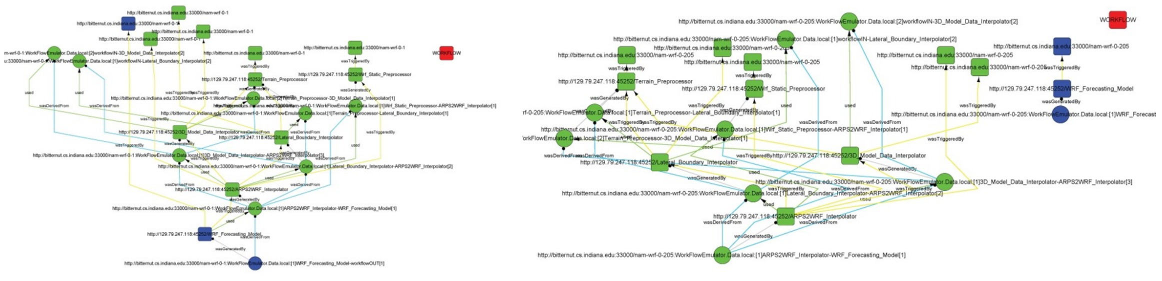

(a) Centroid NAM workflow with 10 levels (b) Centroid NAM workflow with 8 levels

Figure 6.11: Left graph is provenance of a successful execution. Graph on right shows that although the right graph is a successful execution, it has dropped notifications in provenance capture, because all nodes except some edges in left graph cannot be matched.

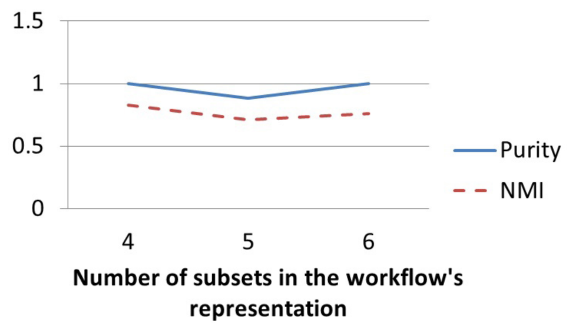


Figure 6.12: Purity and NMI results as external evaluation criteria for k-means cluster quality by workflow instance group.

such as Figure 6.13(a) and Figure 6.13(c); 2) the same type of workflows are separated only because the number of their input data artifacts are different, such as Figure 6.13(c) and Figure 6.13(d); 3) workflows have the same structure, but one cluster has larger input data (large artifact-size) and produces larger output data, such as Figure 6.13(a) and Figure 6.13(b). Note that even though we do not know what a correct AMSR-E workflow is, the identified variants may have potential problems and are worth checking.

6.4.3 Unsupervised Clustering using an Ensemble Provenance Representation

Alternatively, we can compile representations of varied-length subsets (levels) into one representation, an ensemble provenance representation. This way, we extend all representations to be the same length by filling in the missing values with the value of -1. Note that this is not a new representation, but instead just a compilation of variable length representations.

We evaluate k-means clustering on this representation by plotting WCSS and computing the purity and NMI evaluation metrics (Figure 6.14). WCSS decreases substantially

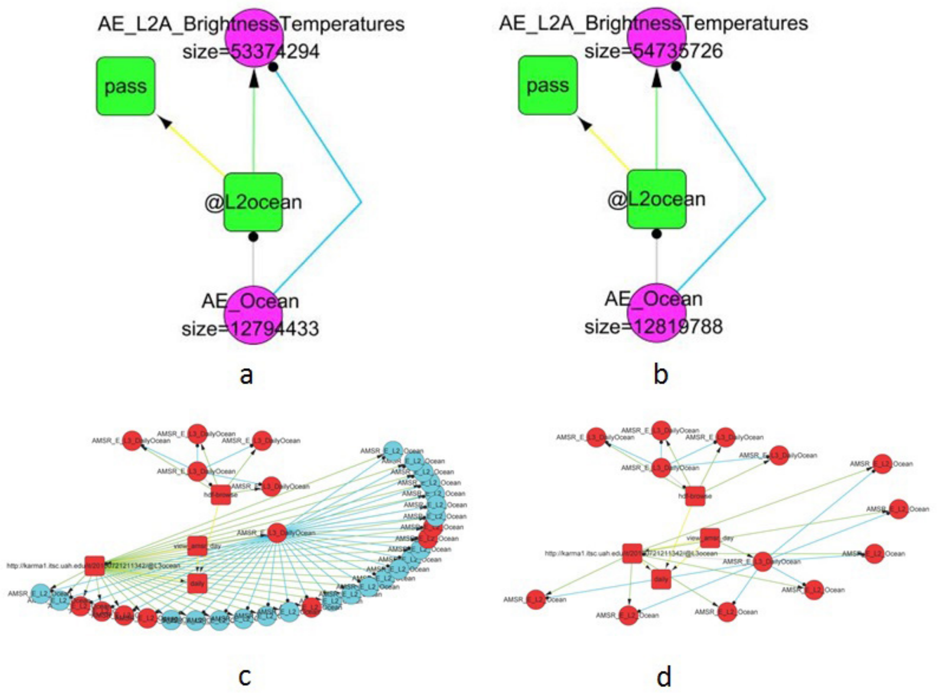


Figure 6.13: (a), (b), (c) and (d) are all AMSR-E daily ocean workflows. (a) and (b) have the same graph structure, but the size of input artifact and output artifact is larger in (a); (a) and (b) have completely different graph structure with (c) and (d); (c) has more input artifacts than (d) – the unmatched artifacts are colored differently after application of the provenance graph matching algorithm.

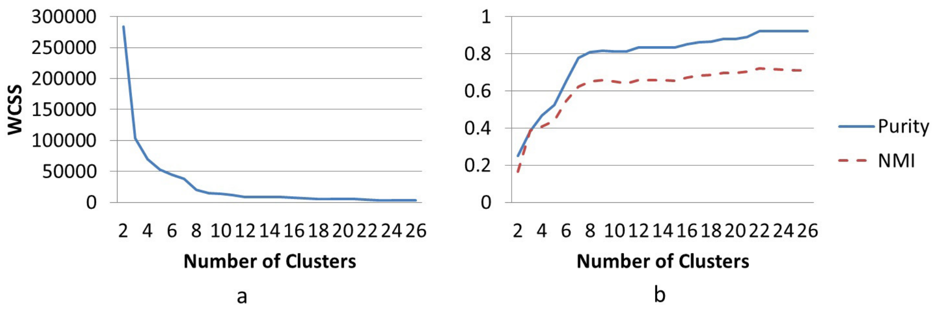


Figure 6.14: WCSS (a) and Purity & NMI (b) as function of number of clusters in k-means.

with the increase in number of clusters in k-means algorithm. After the number K reaches 20, the WCSS becomes small enough and very stable as K increases. Purity increases as K increases. After the number K reaches 22, the purity is high enough (0.92) and it also becomes stable afterwards. Compared with the 42 clusters we created from pre-grouped original provenance representations, we generate only 22 clusters from the ensemble provenance representation, with a slightly lower purity.

Our results in Figure 6.14 show that, similar to purity, NMI increases as K increases. After K reaches 22, the NMI value is 0.72 and becomes stable. These results indicate that our proposed ensemble provenance representation supports efficient unsupervised clustering.

On the AMSR-E dataset, we also use the k-means (SimpleKMeans in Weka) and take the compiling approach to dealing with representations with variable length. The performance shown in Figure 6.15 is also good.

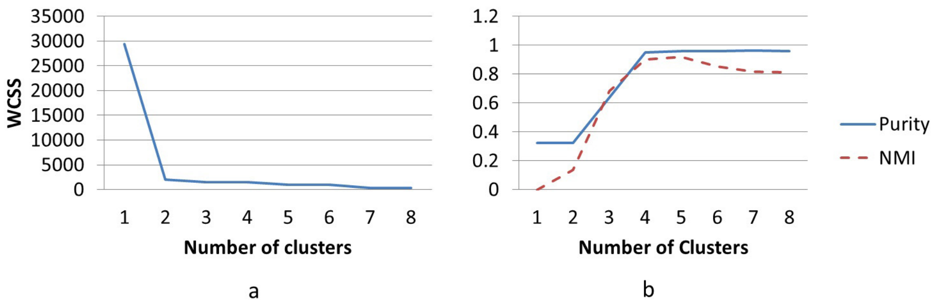


Figure 6.15: WCSS (a) and Purity & NMI (b) as function of number of clusters in k-means.

6.4.4 Workflow Type Classification

Recall that we select seven features out of 120 features by using the feature selection algorithm ClassifierSubsetEval, and select 9 features by using CfsSubsetEval. The NaveBayes model trained from the seven features has a high correctness of 95.79% in its 10-fold cross validation (Table 6.3). On the nine features, we test different classification methods and find many of them, such as NaiveBayes and Random forests [31] can achieve a correct ratio more than 95% (Table 6.3)[2]. This indicates the feature selection in our temporal representations is independent of learning algorithm, and the resulting classifier also has good performance.

For classification on the AMSR-E dataset, we also use the feature selection method CfsSubsetEval, which selects 2 attributes from the initial 40 features that further reduces the representation's size to 21.8KB – *average number of in-degree* in subset 2 and *average value of PGEversion* in subset 4. Using these two attributes to train a Naive Bayes classifier can achieve a high correctness of 96.78% (Table 6.4), which indicates that they are

[2]We also vary the random seed number for the Random forests algorithm from 1 to 10, the resulting ratios of correctly classified instances have a mean value of 96.86% and a standard deviation of 0.0002.

Table 6.3: Classification model trained on features selected by ClassifierSubsetEval and CfsSubsetEval

Feature selection algorithm	**Weka Scheme**	**Result of 10-fold cross-validation**
ClassifierSubsetEval	weka.classifiers. bayes.NaiveBayes	Correctly Classified Instances: 45895 – 95.7902%
CfsSubsetEval	weka.classifiers. bayes.NaiveBayes	Correctly Classified Instances: 45776 – 95.5418%
	weka.classifiers. trees.RandomForest -I 10 -K 0 -S 1	Correctly Classified Instances: 46408 – 96.8609%

Table 6.4: AMSR-E classification model trained on features selected by CfsSubsetEval

Weka Scheme	**Result of 10-fold cross-validation**
weka.classifiers. bayes.NaiveBayes	Correctly Classified Instances: 2798 – 96.7831%

particular important in determining different types of workflow.

6.4.5 Association Rule Mining

We utilize Weka's Apriori algorithm to discover the association rules on the resulting clusters generated from pre-grouped unsupervised clustering on the 10GB dataset. Recall that we have manually introduced two variants of NAM weather forecast workflow (Figure 3.12), and we will look to the resulting association rules for rules related to these two variants.

Table 6.5: Sampling of association rules mined by Apriori method

Weka Scheme	**Sample of association rules found**
weka. associations. Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 0.4 -M 0.1 -S -1.0 -c -1	$1.numberOfNodes\_8 =' (0.8 - 1]' ==>$ $numberOfNodes\_10 =' (0.8 - 1]'$ $2.numberOfNodes\_8 =' (1.8 - inf)' ==>$ $numberOfNodes\_10 =' (1.8 - inf)'$ $3.numberOfNodes\_2 =' (-inf - 1.1]' ==>$ $numberOfNodes\_8 =' (-inf - 0.2]'$

Table 6.5 shows the Scheme of the Weka method we applied and the resulting association rules that can reflect the variants we introduced. Rule 1 says that if the number of nodes in level 8 (which are the data inputs for the last processing step) is between 0.8 and 1 (including 1), then number of nodes in level 10 (which are the final data outputs) will be between 0.8 and 1 (including 1). Rule 2 says that if the number of nodes in level 8 is larger than 1.8, then number of nodes in level 10 will be larger than 1.8. Because the number of nodes can only be integer, rule 1 and rule 2 mean one intermediate data input for the final processing step will lead to one final data output, while more data inputs lead to more final data outputs, which reveals exactly the first variant we introduced. For the same reason, rule 3 reveals the second variant of failure execution. This single example shows that temporal provenance representation with selected number of features supports the Apriori algorithm well: the association rules can show variants during execution.

6.4.6 Summary of Mining Evaluation

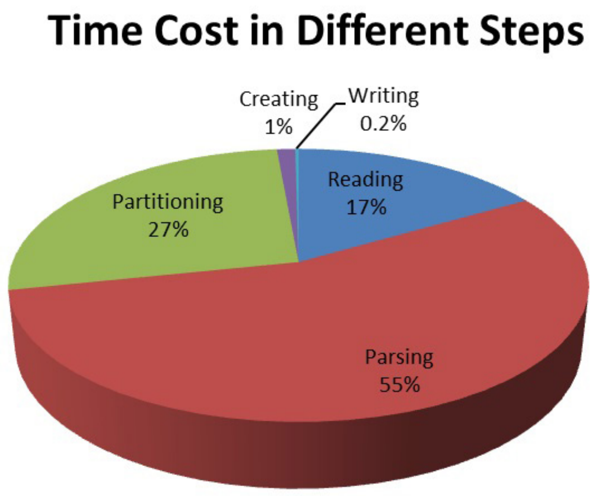
The results of the evaluation are summarized in Table 6.6.

Table 6.6: Summary of temporal mining

Approach	**Evaluation**
Unsupervised clustering, time-domain	**Demonstrates that:** representation can detect failed workflow instances and workflow variants; comparing centroid provenance graphs helps with problem analysis. **Disadvantage:** need representations grouped based on length; creates small clusters inside representation group.
Unsupervised clustering, ensemble	**Demonstrates that:** representation lead to good clustering performance.
Classification	**Demonstrates that:** can predict workflow type of new workflow instances.
Association rule mining	**Demonstrates that:** causal relationships captured between leveled subsets; reveals variants in workflow execution; some association rule sets can be used to distinguish different clusters. **Disadvantage:** Apriori algorithm favors small representation length (less number of features).

6.5 Scalability

We evaluate the performance of the mining tasks over the Logical Clock-P representation primarily by measuring NMI and purity, and the correctness of classification. As time intensive as mining can be, performance scalability is an important aspect as well. The scalability of data mining is a function of the size and dimensionality of the dataset and the complexity of the mining algorithm. Since the user can choose the mining algorithm, we are only interested in the size and dimensionality of proposed temporal representation. We demonstrated in [40] that the size and dimensionality of temporal representation is substantially decreased from the original provenance representation. For instance, the size of the original 10GB database is 10GB; whereas the size of its temporal representation in time domain is 10.01 MB, a decrease by several orders of magnitude. The process of creating the temporal representation consists of a number of different steps: 1.) reading and parsing XML files, 2.) executing the Logical Clock-P partitioning algorithm, 3.) creating representation from partition, and 4.) writing results. We measure the time cost in each step. We instrument and run the sequential version on a desktop machine (Red Hat Enterprise Linux Server release 6.3 (Santiago), Intel Core2 E8400, 4GB of RAM). Total execution time is 102 seconds for a 2.1GB file of OPM graphs. Figure 6.16 shows the detailed breakdown. Parsing XML files into XMLBeans document takes more than half of the total time. The second time consuming step is the partitioning algorithm, which takes about 27%, but creating representations from partitions is not (1%). Reading XML files takes considerable time (17%), but writing representations into a file is negligible (0.2%), which is because of the small size of output representation (16.9MB). In sum, our observation is that excluding the time spent in reading, writing and parsing XML files, the actual time



	Reading	Parsing	Partition-ing	Creating	Writing
Average execution time (ms)	16891	56215.8	27349	1229	223.8

Figure 6.16: Breakdown of representation generating time.

used in executing partitioning (Logical Clock-P) and creating representations is only 28% of the total time. In other words, this is a data intensive application. Thus, it is natural to use parallel programming paradigm such as MapReduce [55] to improve its scalability.

To examine how well Hadoop [158] MapReduce can accelerate the representing process, we measure the absolute run time as well as relative speedup and scaleup [60]. We conduct experiments on a cluster of 10 nodes, each node having a dual-socket, 2-core (4 total cores/node) AMD Opteron system with 4GB of memory. The cluster is connected internally by Gigabit Ethernet and Infiniband networks; we use the former. We use an extra node for running the master daemons to manage the Hadoop jobs and the Hadoop distributed file system. On each node, 64-bit Red Hat Enterprise Linux Server release 5.8 (Tikanga), JDK 1.6, and Hadoop 0.21.0 exist. The maximum number of map and reduce

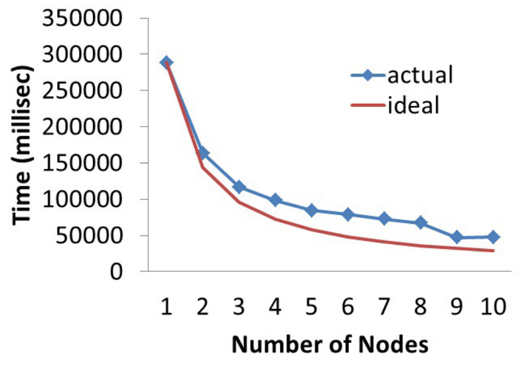


Figure 6.17: Run time on different cluster sizes.

tasks is set to be the same as number of cores, four, on each node. The MapReduce implementation follows the same logic/algorithm as previous sequential version, but it has only mappers (map only) that directly read from and write back to HDFS.

To evaluate the speedup of this approach, we fix the dataset size and vary cluster size. Figure 6.17 shows run time for 2.1GB OPM graphs on clusters ranging from 1 to 10 nodes. Also shown is ideal speedup curve (with a smooth line). For instance, if the cluster has twice as many nodes and the data size does not change, the approach should be twice as fast. In Figure 6.18 we show the same numbers, but plotted on a "relative scale". That is, for each cluster size, we plot the ratio between the run time for 1-node cluster and run time of the current cluster size. For example, for the 10-node cluster, plotted is the ratio between the run time on the 1-node cluster and the run time on the 10-node cluster. The result shows that this approach shows good speedup.

To further evaluate the scaleup of the proposed approach we increase the dataset size and the cluster size together by the number of nodes. That is, we start with 214M OPM graphs on 1-node cluster, and then multiply the size of OPM graphs by the number of nodes, so we will end up with 2.1GB OPM graphs on 10-node cluster. A perfect scaleup

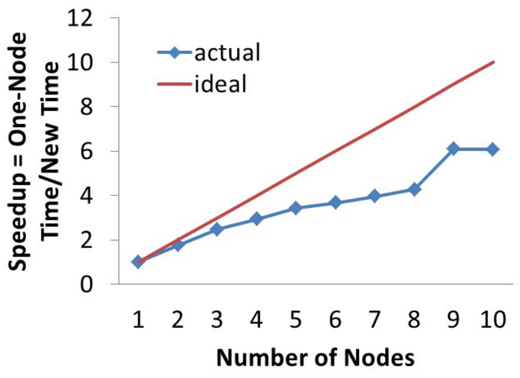


Figure 6.18: Relative run time on different cluster size.

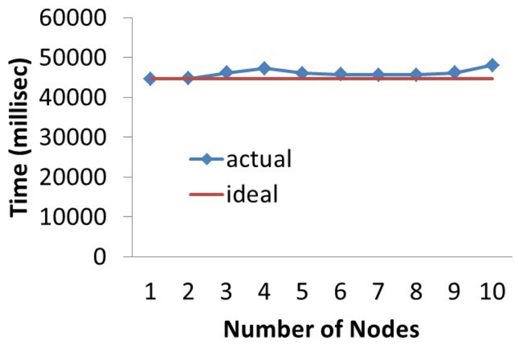


Figure 6.19: Run time when the dataset size and the cluster size (the number of nodes) scale together.

could be achieved if run time remained constant (no more overhead). Figure 6.19 shows the run time when data size is increased from 1 to 10 times, on a cluster with 1 to 10 nodes, respectively. We can see that this approach has a scaleup curve close ideal.

In summary, data mining on proposed temporal representation is scalable because of the size and dimensionality of dataset is greatly decreased in our reduction (representing) process, and the scalability of representing process can be improved using MapReduce.

6.6 Applying Temporal Representation to Provenance Stream Model

So far we have shown that our proposed temporal representation can be applied onto a large collection of provenance data to support data mining tasks with good performance and scalability. Next we extend the same methodology to the stream provenance model so that it could be used to mine live provenance that is too big to be stored persistently. The idea is to use a sliding window to take a snapshot of the current status of the provenance stream and apply the same temporal representation methodology to this snapshot to generate its temporal representation. However, these window-based temporal representations may vary in time or speed, thus the previous similarity measurement using Euclidean distance no longer works. In this section, we first introduce an algorithm to generate the window-based temporal representation, and then propose a new similarity measurement based on Dynamic Time Warping (DTW).

6.6.1 Streaming Algorithm that Generates Temporal Representation

The Window-based Temporal Representation (WTR) algorithm (Algorithm 3) generates a synopsis of the stream elements within the current sliding window. This is a direct application of our temporal representation algorithm onto the stream elements within the current sliding window.

We have shown that the temporal representation can significantly reduce the feature space while still preserving valuable information to support mining techniques like clustering, classification and association rule mining. While we can still perform the same association rule mining analysis on the window-based temporal representations, the clustering and classification analysis no longer work as the representations may vary in time

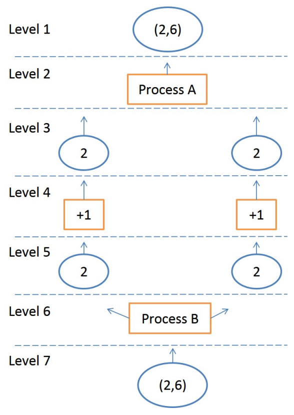
Algorithm 3 The WTR algorithm that generates the synopsis of provenance stream for the current sliding window.

```
1: function GENERATESYNOPSIS($elements$)    ▷ $element$: all the provenance stream elements within current sliding window
2:     $graph$ ← provenance graph built from $elements$
3:     $nodeSubsets$ ← $graph$ partitioned using the Logical-P algorithm
4:     $synopsis$ ← empty list
5:     for $subset$ in $nodeSubsets$ do
6:         $representation$ ← statistical features selected from $subset$
7:         append $representation$ to $synopsis$
8:     end for
9:     return $synopsis$
10: end function
```

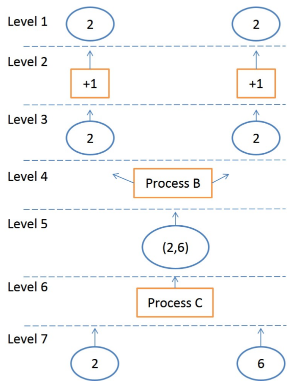
or speed (see Figure 6.20). We solve this problem with a technique called Dynamic Time Warping (DTW), which is a method to calculate the optimal match between two given sequences that is widely used in mining time series data [27]. With DTW distance, we can apply the same k-means algorithm to clustering and the k-Nearest Neighbors algorithm (k-NN) to classification. However, the original DTW algorithm aligns sequences element by element, thus we have to adapt it to use the leveled subset in temporal representation as the basic unit of alignment. The modified version of DTW algorithm (see Algorithm 4) has a time complexity of $O(n^2)$. While the time complexity can be improved with fast DTW computation algorithms such as SparseDTW [10] and FastDTW [133], it is beyond our current research.

Algorithm 4 Algorithm that calculates the Dynamic Time Warping (DTW) distance for two window-based temporal representation sequences.

```
1: function DTWDistance($synopsis1 : array[1...n]$, $synopsis2 : array[1...m]$)    ▷ element $i$ in $array[1...n]$ is feature representation of level $i$
2:     $DTW \leftarrow array[0...n][0...m]$
3:     for $i \leftarrow 1$ to $n$ do
4:         $DTW[i][0] \leftarrow \infty$
5:     end for
6:     for $i \leftarrow 1$ to $m$ do
7:         $DTW[0][i] \leftarrow \infty$
8:     end for
9:     for $i \leftarrow 1$ to $n$ do
10:        for $j \leftarrow 1$ to $m$ do
11:            $cost \leftarrow EuclideanDistance(synopsis1[i], synopsis2[j])$
12:            $DTW[i][j] \leftarrow cost + minimum(DTW[i-1][j], DTW[i][j-1], DTW[i-1][j-1])$    ▷ insertion, deletion, and match, in order
13:        end for
14:    end for
15:    return $DTW[n][m]$
16: end function
```



(a) Sequence I



(b) Sequence II

Figure 6.20: Time lag between two representation sequences, caused by the time difference between ABM simulations. When measuring the similarity, Level 3-7 in Sequence I should be aligned to Level 1-5 in Sequence II.

6.6.2 Performance of WTR Algorithm

We implement the WTR algorithm in the same Kafka & Spark Streaming framework we proposed in Chapter 5, and we test its throughput, latency and scalability with the provenance streams from the Zambia food security ABM. Note that though the provenance streams are represented in W3C PROV, they consist of only data products and the relationship "was derived from", which can be directly mapped to OPM concept "artifact" and "was derived from."

Again, we choose to not throttle the provenance generating rate, but to control the receiving rate using the "backpressure" feature in Spark Streaming. Figure 6.21 shows how the "backpressure" works on the window-based temporal representation algorithm.

Spark Streaming receives input data streams and divide the data into batches. We first measure the throughput and latency of both algorithms running in a single-node Spark

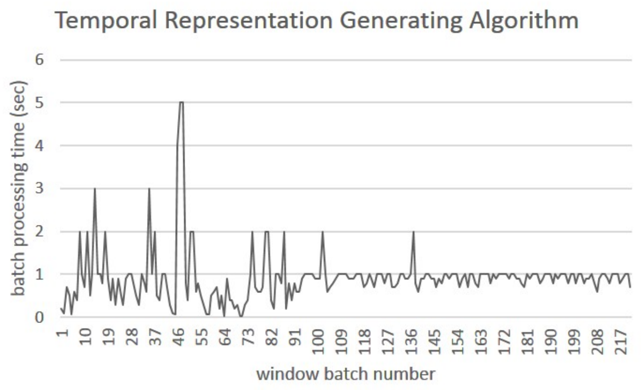


Figure 6.21: The data receiving rate is automatically controlled by the "backpressure" feature in Spark Streaming when running the WTR algorithm, with the window length and the sliding interval both equal to 1s.

Streaming cluster. To reveal the maximum throughput when just receiving stream elements from Kafka, we also run the operation "collect" in the same Spark Streaming cluster and measure its throughput. Figure 5.6 shows the result, where we vary the batch interval to show its impact on throughput and latency for both algorithms. Note that, in addition to the batch interval, our WTR algorithm has two more parameters: window length – the duration of the window, and sliding interval – the interval at which the window operation is performed, both of which must be multiples of the batch interval. To make the WTR algorithm process each element in the provenance stream exactly once, we set the window length equal to the sliding interval. We first vary the window length (and the sliding interval) together with the batch interval in Figure 6.22, and then fix the batch interval and vary the window length (and the sliding interval) in Figure 6.23.

Figure 6.22 shows that increasing the batch interval has little impact on the throughput but can increase the latency significantly. The results in Figure 6.23 shows that increasing

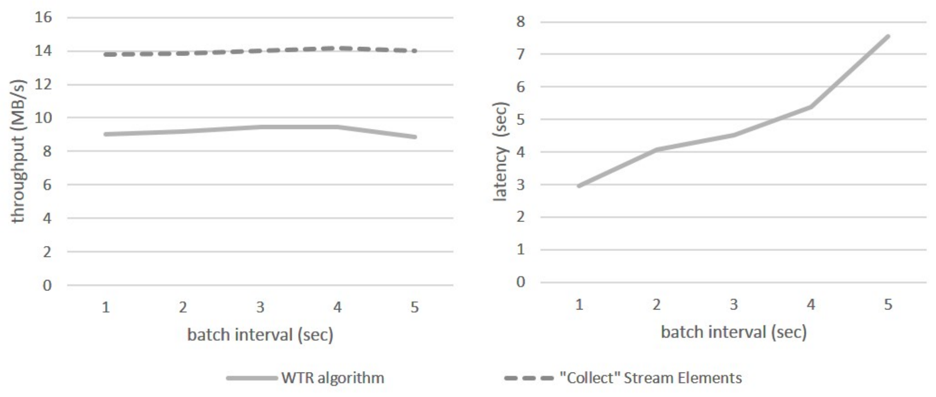


Figure 6.22: The influence of batch interval on throughput and latency (when the window length and the sliding interval are set equal to the value of batch interval).

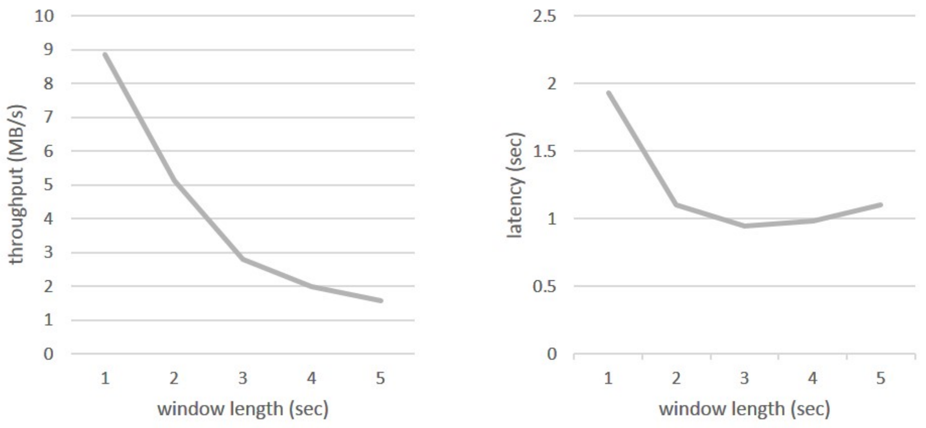


Figure 6.23: The influence of window length on throughput and latency (when the sliding interval is set equal to the value of window length).

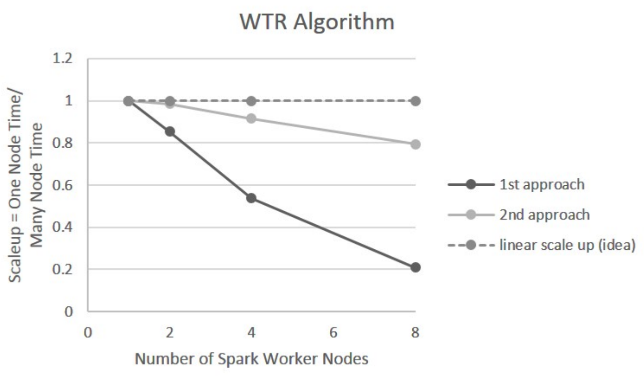


Figure 6.24: Scalability test on the WTR algorithm (with the batch interval, the window length and the sliding interval all equal to 1s).

sliding interval reduces both the throughput and the latency. By adjusting the value of window length and sliding interval, our proposed algorithm can have throughput as high as 8.8MB/s per stream and latency as low as less than 1s.

To perform data mining tasks like clustering and classification that require more than one provenance stream, a straightforward way is to create the temporal representations of multiple provenance streams and join them within one streaming application. However, similar to the results of scalability test in Section 5.5, Figure 6.24 demonstrates that the 1st approach (creating a new streaming application for each provenance stream) is more scalable than the 2nd approach (processing all provenance streams within one streaming application). This suggests that if we want to further scale up the temporal representation generation, we can first generate the representations using the 2nd approach and output them into a shared file system like HDFS, and then read and process them all together using another streaming application.

6.7 Summary

In this chapter, we develop the Logical Clock-P algorithm to produce partitions that preserve temporal orders between node subsets. Based on that, we propose a temporal representation for provenance graphs, which includes both structure and attribute information into a single uniform attribute space where statistical features can be extracted for data mining. Experiments on semi-synthetic and real life provenance datasets show that the temporal representation can detect failed workflow instances; can be used to predict the type of new workflow instance; and can reveal workflow variants. The performance evaluation shows good purity and NMI in unsupervised clustering, and high correctness ratio in supervised classification. The scalability study indicates that data mining on proposed temporal representations are scalable because of the size and dimensionality of dataset are greatly decreased in our reduction (representing) process, and the scalability of representing process can be improved using MapReduce.

We also propose a window-based temporal representation (WTR) algorithm for the provenance stream model to support data mining on streaming provenance data with few computational resources. The performance evaluation of the representation algorithm using the same streaming framework we presented in Chapter 5 shows high throughput, low latency and suggests the right way to scale it up. To address the time and speed differences between window-based temporal representations, we propose a similarity measurement method based on Dynamic Time Warping (DTW).

Chapter 7

Conclusion and Future Work

Our earlier work captures and visualizes the Big Data provenance generated from data intensive eScience: we create a plugin to Cytoscape for visualizing large scale provenance graphs; we develop an automatic provenance capture extension to NetLogo, and demonstrate the usefulness of provenance analysis in a social-ecological agent-based model. In this dissertation, we continue to address the Big Data challenges in provenance analysis: the volume, velocity, and complexity of provenance for offline and online retrieval and data mining.

7.1 Contributions

Our research aims to help data scientists to better understand, debug, and use big data. It is widely recognized the importance of provenance (or data lineage) in understanding, reproducing, and reusing data, and our efforts further expand the use of provenance in several domains of eScience, including workflow, computer network, and agent-based simulation. To that end, the major outcomes are summarized below:

1. **Efficient Offline and Online Retrieval of Big Data Provenance**: Forward provenance query tells which output data depends on a given input data, thus it is helpful

for data scientists to understand the impact of the input data, and the propagation of errors from the input to the output data. Backward provenance query answers on which input data the given output data relies, hence it is useful for debugging the output, updating data sources, and tuning application parameters. However, the traditional approach to provenance retrieval, which processes the queries on complete provenance data, is no longer applicable when the size of provenance is too big to be stored persistently. Our research enables the efficient retrieval of Big Data provenance, with online and offline solutions depending on when the query is given: if we know the query before capturing the provenance (often in forward provenance query), we can preserve and process in situ the provenance traces that are relevant to the query; if the query is given on-the-fly (often in backward provenance query), we can use stream processing to calculate the query results in real-time.

2. **Temporal Representation to Support Data Mining on Large Collections of Provenance**: Variants and exceptions in data processing can be discovered by mining provenance. However, provenance represented as property graph can have large size and high dimensionality that makes the data mining ineffective or even intractable. In the thesis, we propose an efficient and scalable representation of provenance that can be mined to reveal the variants and exceptions in experiment runs. We also apply the temporal representation to the provenance stream model to support data mining on streaming provenance. The evaluation on a 10GB semi-synthetic provenance database and a real life social-ecological dataset shows that the temporal representation can detect failed workflow instances; can be used to predict the type of new workflow instance; and can describe/distinguish clusters from one another.

To achieve the above outcomes, we develop technical solutions that address the challenges that we mentioned in Section 1.2. The technical contributions of this dissertation can be summarized as follows:

1. For scientific simulations that generate Big Data provenance but the queries are pre-given, we propose filtering techniques accordingly to reduce the amount of provenance stored. We also propose a non-preprocessing (NP) provenance slicing technique that processes the raw provenance traces on demand to answer queries. The experimental evaluation shows that their combination can dramatically reduce the demands on persistent storage and the processing time of subsequent querying.
2. For scientific simulations that generate provenance traces at high speed but the query subject can be any of the future data product, we propose a streaming algorithm that can dynamically track the dependencies for all future data products. The streaming algorithm has been implemented in our stream processing framework and tested to have high throughput, low latency and good scalability.
3. For mining a large collection of provenance graphs, we propose a temporal representation that can effectively reduce the volume and dimensionality thus is scalable for data mining operations including clustering, classification and association rules mining. To use the temporal representation to support data mining on streaming provenance, we present a window-based algorithm and a similarity measurement based on Dynamic Time Warping (DTW). The performance evaluation of the window-based temporal representation algorithm shows good throughput, latency and scalability.

7.2 Future Work

The provenance stream model that we defined in this dissertation assumes that temporal order can be preserved during provenance capture and processing. To satisfy this assumption, our prototype framework sends each provenance stream into one Kafka partition that is processed in one RDD partition by Spark Streaming. This limits the level of parallelism and one future direction is to relax the assumption and to parallelize the processing of each provenance stream. In addition, based on the provenance stream model, our prototype streaming framework, and our proposed streaming algorithms, various useful applications can be developed, some of which are already described in Section 5.6.

Our temporal representation is based on OPM's definition of provenance, but can also support other provenance models by mapping them to OPM. W3C Provenance Incubator Group defined provenance vocabulary mappings in [129], in which OPM is the reference model and map to it are nine provenance vocabularies and models. While some mappings are inaccurate and lose information, the effort found that the notion of processes, artifacts, and agents as defined in OPM are common or highly related, so can be mapped quite naturally between the models. However, the nine vocabularies and models do not include W3C PROV, which is a standard that emerged after OPM. While OPM is built on causality, PROV is not and it only has some edges that imply temporal ordering. W3C PROV also has provenance concepts that are not captured in OPM, such as versioning, mechanisms for linking the different descriptions of the same object, and containment relationships and collections. We think that simply mapping PROV to OPM is not the best solution to extend our representation to PROV, and instead it is worth extending our Logical-P algorithm to support the definitions of PROV. For example, we can modify our definition of "happened before” from a causal relationship to a temporal relationship.

In addition, when evaluating the performance of data mining tasks on proposed temporal representations, we handle nominal values and missing values by mapping them into numeric values, which makes problematic implications of distances. One way to address this is to develop special distance measurement methods to handle the nominal and missing values properly.

Finally, we apply the temporal representation to the provenance stream model through sliding window and a new distance measurement method. While the efficacy of the temporal representation in data mining has been verified on static datasets, its performance on streaming provenance needs to be evaluated directly.

Bibliography

[1] Neo4j. `http://www.neo4j.org/`, 2013.

[2] Apache Storm. Apache Software Foundation, `https://storm.apache.org/`, 2015.

[3] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, et al. The design of the borealis stream processing engine. In *The Conference on Innovative Data Systems Research (CIDR)*, volume 5, pages 277–289, 2005.

[4] Umut A Acar. Self-adjusting computation:(an overview). In *Proceedings of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, pages 1–6. ACM, 2009.

[5] Charu C Aggarwal, Na Ta, Jianyong Wang, Jianhua Feng, and Mohammed Zaki. Xproj: a framework for projected structural clustering of xml documents. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 46–55. ACM, 2007.

[6] Charu C Aggarwal and Haixun Wang. *Managing and mining graph data*, volume 40. Springer, 2010.

[7] Hiralal Agrawal and Joseph R Horgan. Dynamic program slicing. In *ACM SIGPLAN Notices*, volume 25, pages 246–256. ACM, 1990.

[8] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994.

[9] Mehmet S Aktas, Beth Plale, David Leake, and Nirmal K Mukhi. Unmanaged workflows: Their provenance and use. In *Data Provenance and Data Management in eScience*, pages 59–81. Springer, 2013.

[10] Ghazi Al-Naymat, Sanjay Chawla, and Javid Taheri. Sparsedtw: a novel approach to speed up dynamic time warping. In *Proceedings of the Eighth Australasian Data Mining Conference-Volume 101*, pages 117–127. Australian Computer Society, Inc., 2009.

[11] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. In *International Provenance and Annotation Workshop (IPAW)*, pages 118–132. Springer, 2006.

[12] Li An. Modeling human decisions in coupled human and natural systems: Review of agent-based models. *Ecological Modelling*, 229:25–36, 2012.

[13] Manish Kumar Anand, Shawn Bowers, and Bertram Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *International Conference on Extending Database Technology*, volume 10, pages 287–298, 2010.

[14] Manish Kumar Anand, Shawn Bowers, Timothy McPhillips, and Bertram Ludäscher. Efficient provenance storage over nested data collections. In *Proceedings*

of the 12th International Conference on Extending Database Technology: Advances in Database Technology, pages 958–969. ACM, 2009.

[15] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *20th International conference on World Wide Web*, pages 635–644, 2011.

[16] Cludia M. Antunes and Arlindo L. Oliveira. Temporal data mining: An overview. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) Workshop on Temporal Data Mining*, pages 1–13. Citeseer, 2001.

[17] Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Mayur Datar, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom. Stream: The stanford data stream management system. *Data Stream Management*, 2004.

[18] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–16. ACM, 2002.

[19] Zhuowei Bao, Sarah Cohen-Boulakia, Susan B Davidson, and Pierrick Girard. Pdiffview: viewing the difference in provenance of workflow results. *Proceedings of the Very Large Data Bases (VLDB) Endowment*, 2(2):1638–1641, 2009.

[20] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 623–632. Society for Industrial and Applied Mathematics, 2002.

[21] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *18th International conference on World Wide Web*, pages 1061–1062, 2009.

[22] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *International Conference on Weblogs and Social Media (ICWSM)*, 8:361–362, 2009.

[23] Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, Satya Sahoo, and Curt Tilmes. Prov-dm: The prov data model. *W3C Candidate Recommendation 11 December 2012*, 2012.

[24] Ammar Benabdelkader, Mark Santcroos, Souley Madougou, Antoine HC van Kampen, and Silvia D Olabarriaga. A provenance approach to trace scientific experiments on a grid infrastructure. In *2011 IEEE 7th International Conference on E-Science (e-Science)*, pages 134–141. IEEE, 2011.

[25] David A Bennett, Wenwu Tang, and Shaowen Wang. Toward an understanding of provenance in complex land use dynamics. *Journal of Land Use Science*, 6(2-3):211–230, 2011.

[26] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[27] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *AAAI-94 Workshop on Knowledge Discovery in Databases (KDD-94)*, volume 10, pages 359–370. Seattle, WA, 1994.

[28] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? In *the International Conference on Database Theory (ICDT)*, pages 217–235. Springer, 1999.

[29] R. Bose and J. Frew. Composing lineage metadata with xml for custom satellite-derived data products. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management*, pages 275–284. IEEE, 2004.

[30] Shawn Bowers, Timothy McPhillips, Sean Riddle, Manish Kumar Anand, and Bertram Ludäscher. Kepler/ppod: Scientific workflow and provenance support for assembling the tree of life. In *International Provenance and Annotation Workshop (IPAW)*, pages 70–77. Springer, 2008.

[31] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[32] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *the International Conference on Database Theory (ICDT)*, pages 316–330. Springer, 2001.

[33] Peter Buneman and Wang-Chiew Tan. Provenance in databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 1171–1173. ACM, 2007.

[34] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. Vistrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 745–747. ACM, 2006.

[35] K. Caylor, T. Evans, L. Estes, J. Sheffield, B. Plale, and S. Attari. Impacts of agricultural decision making and adaptive management on food security in africa. *AGU Fall Meeting Abstracts*, 1:06, 2014.

[36] Sudarshan S Chawathe. Comparing hierarchical data in external memory. In *Very Large Data Bases (VLDB)*, volume 99, pages 90–101. Citeseer, 1999.

[37] You-Wei Cheah, Beth Plale, Joey Kendall-Morwick, David Leake, and Lavanya Ramakrishnan. A noisy 10GB provenance database. In *2nd Int'l Workshop on Traceability and Compliance of Semi-Structured Processes (TC4SP2011), co-located with Business Process Management (BPM 2011), Clermont-Ferrand, France*, pages 370–381, Aug 2011.

[38] Artem Chebotko, Jibi Abraham, Pearl Brazier, Anthony Piazza, Andrey Kashlev, and Shiyong Lu. Storing, indexing and querying large provenance data sets as RDF graphs in apache hbase. In *IEEE Ninth World Congress on Services (SERVICES)*, pages 1–8. IEEE, 2013.

[39] Liang Chen, Kolagatla Reddy, and Gagan Agrawal. Gates: A grid-based middleware for distributed processing of data streams. In *Proceedings of IEEE Conference on High Performance Distributed Computing (HPDC)*, pages 192–201, 2004.

[40] Peng Chen, Beth Plale, and Mehmet S Aktas. Temporal representation for scientific data provenance. In *IEEE 8th International Conference on E-Science (e-Science)*, pages 1–8, 2012.

[41] Peng Chen, Beth Plale, and Mehmet S Aktas. Temporal representation for mining scientific data provenance. *Future Generation Computer Systems*, 36:363–378, 2014.

[42] Peng Chen, Beth Plale, You-Wei Cheah, Devarshi Ghoshal, Scott Jensen, and Yuan Luo. Visualization of network data provenance. In *Workshop on Massive Data Analytics on Scalable Systems (DataMASS), co-located with High Performance Computing Conference, Pune India*, Dec 2012.

[43] Peng Chen, Beth Plale, and Tom Evans. Dependency provenance in agent based modeling. In *IEEE 9th International Conference on eScience (eScience)*, pages 180–187. IEEE, 2013.

[44] Peng Chen and Beth A Plale. Proverr: System level statistical fault diagnosis using dependency model. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 525–534. IEEE, 2015.

[45] James Cheney. Program slicing and data provenance. *IEEE Data Eng. Bull*, 30(4):22–28, 2007.

[46] James Cheney, Amal Ahmed, and Umut A Acar. Provenance as dependency analysis. In *Database Programming Languages*, pages 138–152. Springer, 2007.

[47] K. Cheung and J. Hunter. Provenance explorer–customized provenance views using semantic inferencing. *the International Semantic Web Conference (ISWC)*, pages 215–227, 2006.

[48] Avery Ching. Giraph: Large-scale graph processing infrastruction on hadoop. *Proc. of the Hadoop Summit. Santa Clara, USA:[sn]*, 2011.

[49] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[50] Graham Cormode and S Muthukrishnan. Space efficient mining of multigraph streams. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 271–282. ACM, 2005.

[51] Bryce Cutt and Ramon Lawrence. Managing data quality in a terabyte-scale sensor archive. In *Proceedings of the 2008 ACM symposium on Applied Computing*, pages 982–986. ACM, 2008.

[52] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1345–1350. ACM, 2008.

[53] Nicholas Dawes, K Ashwin Kumar, Sebastian Michel, Karl Aberer, and Michael Lehning. Sensor metadata management and its application in collaborative environmental research. In *IEEE Fourth International Conference on eScience*, pages 143–150. IEEE, 2008.

[54] Wim De Pauw, Mihai Leţia, Buğra Gedik, Henrique Andrade, Andy Frenkiel, Michael Pfeifer, and Daby Sow. Visual debugging for stream processing applications. In *Runtime Verification*, pages 18–35. Springer, 2010.

[55] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[56] N. Del Rio and P.P. Da Silva. Probe-it!: visualization support for provenance. In *Proceedings of the 3rd International Conference on Advances in Visual Computing-Volume Part II*, pages 732–741. Springer-Verlag, 2007.

[57] A. P. Dempster, . N. M. Lairp, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[58] J. Deng, R.R. Brooks, and J. Martin. Assessing the effect of wimax system parameter settings on mac-level local dos vulnerability. *International Journal of Performability Engineering*, 8(2):183, 2012.

[59] F. DePiero, M. Trivedi, and S. Serbin. Graph matching using a direct classification of node attendance. *Pattern Recognition*, 29(6):1031–1048, 1996.

[60] David J. DeWitt and Jim Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98, 1992.

[61] Jennifer G. Dy and Carla E. Brodley. Feature selection for unsupervised learning. *The Journal of Machine Learning Research*, 5:845–889, 2004.

[62] K Eckhardt and JG Arnold. Automatic calibration of a distributed catchment model. *Journal of Hydrology*, 251(1):103–109, 2001.

[63] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[64] Martin Ester, Hans-Peter Kriegel, Jrg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining*, volume 1996, pages 226–231. AAAI Press, 1996.

[65] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2005.

[66] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2008.

[67] Jean-Daniel Fekete. The infovis toolkit. In *IEEE Symposium on Information Visualization, 2004. INFOVIS 2004*, pages 167–174. IEEE, 2004.

[68] Rafael Ferreira da Silva, Tristan Glatard, and Frédéric Desprez. Self-healing of operational workflow incidents on distributed computing infrastructures. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 318–325. IEEE Computer Society, 2012.

[69] J. Freire, C. Silva, S. Callahan, E. Santos, C. Scheidegger, and H. Vo. Managing rapidly-evolving scientific workflows. *International Provenance and Annotation Workshop (IPAW)*, pages 10–18, 2006.

[70] Luiz MR Gadelha Jr, Ben Clifford, Marta Mattoso, Michael Wilde, and Ian Foster. Provenance management in swift. *Future Generation Computer Systems*, 27(6):775–780, 2011.

[71] Ashish Gehani and Dawood Tariq. Spade: Support for provenance auditing in distributed environments. In *Proceedings of the 13th International Middleware Conference*, pages 101–120. Springer-Verlag New York, Inc., 2012.

[72] Devarshi Ghoshal and Beth Plale. Provenance from log files: a bigdata problem. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 290–297. ACM, 2013.

[73] Carole Goble. Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics. In *Workshop on Data Derivation and Provenance, Chicago*, 2002.

[74] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Operating Systems Design and Implementation (OSDI)*, volume 12, page 2, 2012.

[75] Jim Gray. Jim gray on escience: A transformed scientific method. *The fourth paradigm: Data-intensive scientific discovery*, pages xvii–xxxi, 2009.

[76] Douglas Gregor and Andrew Lumsdaine. The parallel bgl: A generic library for distributed graph computations. *Parallel Object-Oriented Scientific Computing (POOSC)*, 2:1–18, 2005.

[77] Paul Groth and Luc Moreau. Representing distributed systems using the open provenance model. *Future Generation Computer Systems*, 27(6):757–765, 2011.

[78] Paul T Groth. A distributed algorithm for determining the provenance of data. In *IEEE Fourth International Conference on eScience*, pages 166–173. IEEE, 2008.

[79] Ashish Kumar Gupta and Dan Suciu. Stream processing of xpath queries with predicates. In *ACM SIGMOD International conference on Management of data*, pages 419–430, 2003.

[80] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[81] M.A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[82] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[83] Ivan Herman, Guy Melançon, and M Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[84] Tommy Hoffner. *Evaluation and comparison of program slicing tools*. Citeseer, 1994.

[85] David A Holland, Uri Jacob Braun, Diana Maclean, Kiran-Kumar Muniswamy-Reddy, and Margo I Seltzer. Choosing a data model and query language for provenance. In *Proceedings of the 2nd International Provenance and Annotation Workshop (IPAW '08)*. Springer, 2008.

[86] Xiaohua Hu, Xiaodan Zhang, Caimei Lu, E. K. Park, and Xiaohua Zhou. Exploiting wikipedia as external knowledge for document clustering. In *Proceedings of the*

15th ACM SIGKDD International Conference on Knowledge Discovery and Data mining, pages 389–396. ACM, 2009.

[87] Noah Iliinsky and Julie Steele. *Designing Data Visualizations: Representing Informational Relationships*. O'Reilly Media, Inc., 2011.

[88] Eduardo J Izquierdo and Randall D Beer. An integrated neuromechanical model of steering in c. elegans. In *Proceeding of the European Conference on Artificial Life*, pages 199–206, 2015.

[89] George John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[90] Jae-Yoon Jung and Joonsoo Bae. Workflow clustering method based on process similarity. In *the International Conference on Computational Science and Applications (ICCSA)*, pages 379–389. Springer, 2006.

[91] A.B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

[92] Anastasios Kementsietsidis and Min Wang. Provenance query evaluation: what's so special about it? In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 681–690. ACM, 2009.

[93] Jihie Kim, Ewa Deelman, Yolanda Gil, Gaurang Mehta, and Varun Ratnakar. Provenance trails in the wings/pegasus system. *Concurrency and Computation: Practice and Experience*, 20(5):587–597, 2008.

[94] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the International Workshop on Networking Meets Databases (NetDB)*, pages 1–7, 2011.

[95] Anton Kruger, Ramon Lawrence, and Eduard Constantin Dragut. Building a terabyte nexrad radar database for hydrometeorology research. *Computers & Geosciences*, 32(2):247–258, 2006.

[96] M. Kunde, H. Bergmeyer, and A. Schreiber. Requirements for a provenance visualization component. *International Provenance and Annotation Workshop (IPAW)*, pages 241–252, 2008.

[97] Natalia Kwasnikowska, Luc Moreau, and Jan Van Den Bussche. A formal account of the open provenance model. *ACM Transactions on the Web (TWEB)*, 9(2):10, 2015.

[98] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[99] David Leake and Joseph Kendall-Morwick. Towards case-based support for e-science workflow generation by mining provenance. *Advances in Case-Based Reasoning*, pages 269–283, 2008.

[100] Qing Liu, Quan Bai, Stephen Giugni, Darrell Williamson, and John Taylor. *Data provenance and data management in eScience*. Springer, 2013.

[101] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[102] Christian T Lopes, Max Franz, Farzana Kazi, Sylva L Donaldson, Quaid Morris, and Gary D Bader. Cytoscape web: an interactive web-based network browser. *Bioinformatics*, 26(18):2347–2348, 2010.

[103] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the Very Large Data Bases (VLDB) Endowment (PVLDB)*, 5(8):716–727, 2012.

[104] Ruirui Lu, Gang Wu, Bin Xie, and Jingtong Hu. Stream bench: Towards benchmarking modern distributed stream computing frameworks. In *IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, pages 69–78. IEEE, 2014.

[105] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew B Jones, Edward A Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[106] Yuan Luo, Beth Plale, Scott Jensen, You-Wei Cheah, and Helen Conover. Provenance of amsr-e data from the national snow and ice data center (nsidc). opm xml ver. 1.1., september 2 - october 4, 2011. Data To Insight Center, Bloomington, Indiana, `http://dx.doi.org/10.5967/M0F47M2D`, 2012.

[107] Peter Macko and Margo Seltzer. Provenance map orbiter: Interactive exploration of large provenance graphs. In *3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2011.

[108] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *ACM SIGMOD International Conference on Management of data*, pages 135–146, 2010.

[109] Tanu Malik, Ligia Nistor, and Ashish Gehani. Tracking and sketching distributed data provenance. In *IEEE Sixth International Conference on e-Science (e-Science)*, pages 190–197. IEEE, 2010.

[110] Daniel W Margo and Robin Smogor. Using provenance to extract semantic file attributes. In *2nd USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2010.

[111] Andrew McGregor. Graph stream algorithms: A survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

[112] Simon Miles. Electronically querying for the provenance of entities. In *International Provenance and Annotation Workshop (IPAW)*, pages 184–192. Springer, 2006.

[113] Archan Misra, Marion Blount, Anastasios Kementsietsidis, Daby Sow, and Min Wang. Advances and challenges for scalable provenance in stream processing systems. In *International Provenance and Annotation Workshop (IPAW)*, pages 253–265. Springer, 2008.

[114] Paolo Missier, Khalid Belhajjame, Jun Zhao, Marco Roos, and Carole Goble. Data lineage model for taverna workflows with lightweight annotation requirements. In *International Provenance and Annotation Workshop (IPAW)*, pages 17–30. Springer, 2008.

[115] Paolo Missier, Fernando Chirigati, Yaxing Wei, David Koop, and Saumen Dey. Provenance storage, querying, and visualization in pbase. In *International Provenance and Annotation Workshop (IPAW)*, volume 8628, page 239. Springer, 2015.

[116] Luc Moreau. The foundations for provenance on the web. *Foundations and Trends in Web Science*, 2(2–3):99–241, 2010.

[117] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The open provenance model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743–756, 2011.

[118] Luc Moreau and Paul Groth. the open provenance model xml schema. `http://openprovenance.org/model/opmx-20101012.xsd`, October 2010.

[119] Luc Moreau, Paul Groth, Simon Miles, Javier Vazquez-Salceda, John Ibbotson, Sheng Jiang, Steve Munroe, Omer Rana, Andreas Schreiber, Victor Tan, et al. The provenance of electronic data. *Communications of the ACM*, 51(4):52–58, 2008.

[120] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo I Seltzer. Provenance for the cloud. In *USENIX Conference on File and Storage Technologies (FAST)*, volume 10, pages 15–14, 2010.

[121] Shanmugavelayutham Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[122] J. Myers, C. Pancerella, C. Lansing, K. Schuchardt, B. Didier, N. Ashish, and C.A. Goble. Multi-scale science, supporting emerging practice with semantically derived

provenance. In *ISWC workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*. Florida, 2003.

[123] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[124] Edoardo Pignotti, Gary Polhill, and Peter Edwards. Using provenance to analyse agent-based simulations. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 319–322. ACM, 2013.

[125] Beth Plale, Dennis Gannon, Dan Reed, Kelvin Droegemeier, Bob Wilhelmson, and Mohan Ramamurthy. Towards dynamically adaptive weather analysis and forecasting in lead. *the International Conference on Computational Science (ICCS)*, pages 87–124, 2005.

[126] Beth Plale and Karsten Schwan. Dynamic querying of streaming data with the dquob system. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):422–432, 2003.

[127] Lavanya Ramakrishnan, Dennis Gannon, and Beth Plale. Workem: Representing and emulating distributed scientific workflow execution state. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 283–292. IEEE Computer Society, 2010.

[128] Szabolcs Rozsnyai, Aleksander Slominski, and Yurdaer Doganata. Large-scale distributed storage system for business provenance. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 516–524, 2011.

[129] Satya Sahoo, Paul Groth, Olaf Hartig, Simon Miles, Sam Coppens, James Myers, Yolanda Gil, Luc Moreau, Jun Zhao, Michael Panzer, and Daniel Garijo. Provenance vocabulary mappings. Technical report, W3C, 2010.

[130] Satya S Sahoo, Amit Sheth, and Cory Henson. Semantic provenance for escience: Managing the deluge of scientific data. *IEEE Internet Computing*, 12(4):46–54, 2008.

[131] Hiroto Saigo, Sebastian Nowozin, Tadashi Kadowaki, Taku Kudo, and Koji Tsuda. gBoost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75(1):69–89, 2009.

[132] Semih Salihoglu and Jennifer Widom. GPS: A graph processing system. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, page 22. ACM, 2013.

[133] Stan Salvador and Philip Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) workshop on Mining Temporal and Sequential Data*. Citeseer, 2004.

[134] Taghrid Samak, Dan Gunter, Monte Goode, Ewa Deelman, Gaurang Mehta, Fabio Silva, and Karan Vahi. Failure prediction and localization in large scientific workflows. In *Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science*, pages 107–116. ACM, 2011.

[135] Watsawee Sansrimahachai, Luc Moreau, and Mark J Weal. An on-the-fly provenance tracking mechanism for stream processing systems. In *2013 IEEE/ACIS 12th*

International Conference on Computer and Information Science (ICIS), pages 475–481. IEEE, 2013.

[136] Watsawee Sansrimahachai, Mark J Weal, and Luc Moreau. Stream ancestor function: A mechanism for fine-grained provenance in stream processing systems. In *the sixth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12. IEEE, 2012.

[137] Emanuele Santos, Lauro Lins, James P Ahrens, Juliana Freire, and Claudio T Silva. A first study on clustering collections of workflow graphs. In *International Provenance and Annotation Workshop (IPAW)*, pages 160–173. Springer, 2008.

[138] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. *Journal of the ACM (JACM)*, 58(3):13, 2011.

[139] Carlos E Scheidegger, Huy T Vo, David Koop, Juliana Freire, and Cláudio T Silva. Querying and creating visualizations by analogy. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1560–1567, 2007.

[140] Beth Schroeder, Greg Eisenhauer, Karsten Schwan, Fred Alyea, Jeremy Heiner, Vernard Martin, Bill Ribarsky, Mary Trauner, Je rey Vetter, Ray Wang, et al. Framework for collaborative steering of scientific applications. *Science Information Systems Newsletter*, 4(40), 1997.

[141] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

[142] Claudio T. Silva, Juliana Freire, and Steven P. Callahan. Provenance for visualizations: Reproducibility and beyond. *Computing in Science & Engineering*, pages 82–89, 2007.

[143] Y.L. Simmhan, B. Plale, and D. Gannon. Towards a quality model for effective data selection in collaboratories. In *Workshop on Workflow and Data Flow for Scientific Applications (SciFlow06)*, pages 72–72. IEEE, 2006.

[144] Yogesh Simmhan, Roger Barga, and Catharine van Ingen. Automatic provenance recording for scientific data using trident. In *AGU Fall Meeting Abstracts*, volume 1, page 1048, 2008.

[145] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36, 2005.

[146] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A framework for collecting provenance in data-centric scientific workflows. In *International Conference on Web Services (ICWS)*, pages 427–436. IEEE, 2006.

[147] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. Karma2: Provenance management for data-driven workflows. *International Journal of Web Services Research (IJWSR)*, 5(2):1–22, 2008.

[148] Isuru Suriarachchi, Quan Zhou, and Beth Plale. Komadu: A capture and visualization system for scientific data provenance. *Journal of Open Research Software*, 3(1), 2015.

[149] Martin Szomszor and Luc Moreau. Recording and reasoning over data provenance in web and grid services. In *Proceedings of the International Conference on*

Ontologies, Databases and Applications of Semantics (ODBASE), pages 603–620. Springer, 2003.

[150] Sci^2 Team. Science of science (Sci^2) tool. Indiana University and SciTech Strategies, `https://sci2.cns.iu.edu`, 2009.

[151] Frank Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, 1995.

[152] GA Venkatesh. Experimental results from dynamic slicing of c programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 17(2):197–216, 1995.

[153] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, page 42. ACM, 2010.

[154] Nithya Vijayakumar and Beth Plale. Tracking stream provenance in complex event processing systems for workflow-driven computing. In *International Workshop on Event-driven Architecture, Processing and Systems (EDA-PS)*, 2007.

[155] Nithya N Vijayakumar and Beth Plale. Towards low overhead provenance tracking in near real-time stream filtering. In *International Provenance and Annotation Workshop (IPAW)*, pages 46–54. Springer, 2006.

[156] Min Wang, Marion Blount, John Davis, Archan Misra, and Daby Sow. A time-and-value centric provenance model and architecture for medical event streams. In *Proceedings of the 1st ACM SIGMOBILE international workshop on Systems and*

Networking Support for Healthcare and Assisted Living Environments, pages 95–100. ACM, 2007.

[157] Mark Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, pages 439–449. IEEE Press, 1981.

[158] Tom White. *Hadoop: The definitive guide*. O'Reilly Media, 2012.

[159] Uri Wilensky. Wolf sheep predation model. Northwestern University, `http://ccl.northwestern.edu/netlogo/`, 1997.

[160] Uri Wilensky. Netlogo. Northwestern University, `http://ccl.northwestern.edu/netlogo/`, 1999.

[161] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[162] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2, 2013.

[163] Peng Yue, Jianya Gong, and Liping Di. Augmenting geospatial data provenance through metadata tracking in geospatial service chaining. *Computers & Geosciences*, 36(3):270–281, 2010.

[164] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.

[165] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, pages 10–10. USENIX Association, 2012.

[166] Xiangyu Zhang, Rajiv Gupta, and Youtao Zhang. Precise dynamic slicing algorithms. In *Proceedings. 25th International Conference on Software Engineering*, pages 319–329. IEEE, 2003.

[167] Dongfang Zhao, Chen Shou, Tanu Malik, and Ioan Raicu. Distributed data provenance for large-scale data-intensive computing. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8, 2013.

[168] Dongfang Zhao, Zhao Zhang, Xiaobing Zhou, Tonglin Li, Ke Wang, Dries Kimpe, Philip Carns, Robert Ross, and Ioan Raicu. Fusionfs: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems. In *IEEE International Conference on Big Data (Big Data)*, pages 61–70. IEEE, 2014.

[169] Jing Zhao. *Provenance management for dynamic, distributed and dataflow environments*. PhD thesis, University of Southern California, 2012.

[170] Jun Zhao, Chris Wroe, Carole Goble, Robert Stevens, Dennis Quan, and Mark Greenwood. Using semantic web technologies for representing e-science provenance. *the International Semantic Web Conference (ISWC)*, pages 92–106, 2004.

[171] Ying Zhao and George Karypis. Criterion functions for document clustering: Experiments and analysis. *Machine Learning*, 2001.

[172] Yong Zhao and Shiyong Lu. A logic programming approach to scientific workflow provenance querying. In *International Provenance and Annotation Workshop (IPAW)*, pages 31–44. Springer, 2008.

[173] Shi Zhong and Joydeep Ghosh. Generative model-based document clustering: a comparative study. *Knowledge and Information Systems*, 8(3):374–384, 2005.

[174] Quan Zhou, Devarshi Ghoshal, and Beth Plale. Study in usefulness of middleware-only provenance. In *2014 IEEE 10th International Conference on e-Science (e-Science)*, volume 1, pages 215–222. IEEE, 2014.

[175] Wenchao Zhou, Eric Cronin, and Thau Loo. Provenance-aware secure networks. In *IEEE 24th International Conference on Data Engineering Workshop (ICDEW)*, pages 188–193. IEEE, 2008.

[176] Wenchao Zhou, Micah Sherr, Tao Tao, Xiaozhou Li, Boon Thau Loo, and Yun Mao. Efficient querying and maintenance of network provenance at internet-scale. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pages 615–626. ACM, 2010.

Peng Chen

Mar 20, 2016; chenpeng@indiana.edu

EDUCATION

Ph.D. in Computer Science 2016	Indiana University, Bloomington Advisor and Doctoral Committee Chair: Prof. Beth Plale *Dissertation: Big Data Analytics in Static and Streaming Provenance*
M.S. in Computer Science 2012	Indiana University, Bloomington *GPA:* 4.0/4.0
B.A. in Software Engineering 2010	Nanjing University, China Outstanding Graduates (top 5%)

PUBLICATIONS

P. Chen and B. Plale. "ProvErr: System Level Statistical Fault Diagnosis Using Dependency Model*", the 15th IEEE/ACM Int'l Symposium on Cluster, Cloud and Grid Computing (CCGrid),* 2015 (acceptance ratio 25.7%).

P. Chen and B. Plale. "Big Data Provenance Analysis and Visualization*", the Doctoral Symposium of the 15th IEEE/ACM Int'l Symposium on Cluster, Cloud and Grid Computing (CCGrid),* 2015.

P. Chen, B. Plale, and M. Aktas. "Temporal Representation for Mining Scientific Data Provenance", *Future Generation Computer Systems (FGCS) Special Issue on eScience Applications and Infrastructure*, 2014.

P. Chen, B. Plale, and T. Evans. "Dependency Provenance in Agent Based Modeling", *the 9th Int'l IEEE Conference on e-Science*, Beijing, 2013 (acceptance ratio 41.8%).

S. Jensen, B. Plale, M. Aktas, Y. Luo, **P. Chen**, and H. Conover. "Provenance Capture and Use in a Satellite Data Processing Pipeline". *Journal of Transactions on Geoscience and Remote Sensing*, 2013.

P. Chen, B. Plale, and M. Aktas. "Temporal Representation for Scientific Data Provenance", *the 8th Int'l IEEE Conference on e-Science*, Chicago, 2012 (acceptance ratio 31.2%).

P. Chen, B. Plale, Y.W. Cheah, D. Ghoshal, S. Jensen, and Y. Luo, "Visualization of network data provenance," *in Workshop on Massive Data Analytics on Scalable Systems (DataMASS), co-located with High Performance Computing Conference*, Pune India, 2012.

POSTERS

P. Chen and B. Plale, "Visualizing Large Scale Scientific Data Provenance", *the Int'l Conference for High Performance Computing, Networking, Storage and Analysis (SC12)*, Salt Lake City, Nov 10-16, 2012 (acceptance ratio 47%).

P. Chen, S. Jensen, and B. Plale, "NetKarma Provenance of ORBIT WiMAX Network Experiment", *the 14th GENI Engineering Conference (GEC14)*, Boston, Jul 9-11, 2012.

P. Chen, S. Jensen, and B. Plale, "Provenance Visualization of Packet Loss and Packet Throughput", *the 13th GENI Engineering Conference (GEC13)*, Los Angeles, Mar 13-15, 2012.

RESEARCH

Data to Insight Center, Indiana University Pervasive Technology Institute

Research Assistant	*Impacts of Agricultural Decision Making and Adaptive Management on Food Security in Africa*
2016 - Present	Mentoring PhD and master students.
2013 - 2016	Responsibilities included spatial modeling with NetLogo, R and ArcGIS, spatial data visualization with ArcGIS and Gephi, large scale agent-based simulation, and spatial-temporal data management with MongoDB.
Research Assistant	*Enhancing Predictive Capability of Social Ecological Systems Research*
2013 - 2014	Responsibilities included graph modeling and mining of a social ecological dataset with Neo4j.
Research Assistant	*Global Environment for Network Innovations Provenance* Registry
2010 - 2013	Captured data provenance by instrumentation, logging, and using performance measurement tools such as PerfSONAR on planetary-scale networks (e.g., PlanetLab), configurable networks (e.g., Emulab), and Mobile Wireless networks (e.g. ORBIT WiMAX). Developed visualization tool that supports interactive exploration and comparison of large scale provenance graphs. Proposed a temporal representation for large scale data provenance, on which we applied data mining techniques to discover failure executions and variants.

Intelligent Information Processing Group, State Key Laboratory for Novel Software Technology, China

Team leader	*A Rich Internet Application of Facial Expression*

	Recognition
2009.9 - 2010.6	Responsibilities included training classifiers based on Japanese Female Facial Expression (JAFFE) Database, developing user interface using Adobe Flex, and organizing seminars and assigning tasks.

Knowledge Grid Group, Institute of Computing Technology, Chinese Academy of Sciences, China

Research Intern	*Data Visualization in eMerging rEsource Space nAvigator (Mesa)*
Summer 2009	Developed an Eclipse Rich Client Platform (RCP) program to support visualization and navigation of a Hasse-graph with thousands of nodes.

TEACHING

School of Information Technology, University of Cincinnati

Guest Lecturer	*Information Technology Graduate Seminar (Online Course)*
Fall 2015	Applying research in industry from a Ph.D. intern's perspective

School of Informatics and Computing, Indiana University, Bloomington

Guest Lecturer	*Data Management for Big Data (Online Course)*
Fall 2014	Introducing and comparing different NoSQL databases
Teaching Assistant	*Data Management for Big Data (Online Course)*
Spring 2014	Assisted in course design, content development and website construction.

PROFESSIONAL WORK

Software Engineering Intern Summer 2014	*Facebook Ads Payment Infrastructure, Seattle*
Software Engineering Intern Summer 2013	*Facebook Data Infrastructure, Seattle*

SCHOLARSHIP AND AWARDS

2010 - Present	Research Assistantship, Indiana University, USA
2015	CCGrid 2015 Student Travel Award, NSF
2010	Outstanding Graduates of Nanjing University, China (top

	5%)
2006 - 2009	National Inspiration Scholarship of Nanjing University, China
2007 - 2008	Outstanding Youth Volunteers Association Commissioner of Nanjing University, China
2006	National Physics Olympiad Anhui Province First Prize, China
2006	Pan-Pearl River Delta Physics Olympiad Competition First Prize, China

UNIVERSITY SERVICES

School of Informatics and Computing, Indiana University, Bloomington

2012 - 2014	Computer Science Club Officer
2012 - 2013	Graduate Education Committee Student Representative

Nanjing University, China

2007 - 2009	Youth Volunteers Association Commissioner