ANOMALY DETECTION IN PROCESSES REPRESENTED AS A GRAPH

By

JESSE WAITE

A thesis submitted in partial fulfillment of

the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY

Department of Electrical Engineering and Computer Science

MAY 2018

© Copyright JESSE WAITE, 2018

All Rights Reserved

© Copyright JESSE WAITE, 2018

All Rights Reserved

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of JESSE WAITE find it satisfactory and recommend that it be accepted.

[Committee Chair Full Name, Ph.D., Chair]

[Committee Member Full Name, Ph.D.]

[Committee Member Full Name, Ph.D.]

[Committee Member Full Name, Ph.D.]

[Committee Member Full Name, Ph.D.]

ACKNOWLEDGMENT

I would like to thank my adviser, Lawrence Holder, for his support and instruction over the previous few years. Not only was he a patient and informative adviser for this work, he was my first artificial intelligence instructor, and also previously advised my undergraduate work on language prediction models for alternative and augmentative communication (AAC) devices, the project which established my interest data science and machine learning. I also thank Janardhan Rao Doppa, one of my committee members, for his kind and formal instruction in structured prediction and machine learning. Through his course I learned the problem structure of some of the most difficult machine learning concepts and approaches, which I could not have done without his thorough exposition and implementation examples. Likewise, I thank Ananth Kalyanaraman, whose courses give students a rigorous introduction to sequential problems, their decomposition, and their code implementation, all of which I continue to use frequently. It is because of the patience and formal preparation of WSU professors like Jana, Ananth, and Larry that such advanced concepts are accessible to novices like myself, and can entertain our often stumbling questions. I extend them my sincerest gratitude, and hope they continue to set an example for their respective PhD students.

I would also like to thank the many authors in the field of process mining for their contribution to this thesis, since this project would not have been possible without their open-source software tools, such as ProM. Likewise, I thank the many other process mining and machine learning software developers whose labor made this thesis possible through open-source tools and methods, all of which have radically democratized the application of machine learning and data analysis. These anonymous library authors lead the quiet revolution in machine learning and data analysis, and their contributions to society are dramatically under-credited.

ANOMALY DETECTION IN PROCESSES REPRESENTED AS A GRAPH

Abstract

by Jesse Waite

Washington State University

May 2018

Chair: Lawrence Holder, Ph.D.

Learning structural information from stochastic, noisy environments remains an important area of process mining and graph mining. This work presents an unsupervised, threshold-based method of process mining and anomaly detection using the iterative SUBDUE graph-compression method and the Inductive Miner algorithm. The method generates a dendrogram of compressing structural features of a workflow log, a taxonomical representation by which further analysis can be performed. Via this structured feature representation of a process log, anomaly detection was performed by applying a Bayesian threshold to detect unusual trace components. First, we provide an overview of process mining definitions and existing approaches, then evaluate the method on synthetic data over a range of parameter values and model types. Experimental results show 96% accuracy on an anomaly detection task for reasonable data and algorithmic parameters, reliable performance metrics across a range of these parameters, and competitive performance against a previously studied anomaly detection method known as the Sampling Algorithm. A real-world demonstration is provided for software-testing log data generated from a unit-test suite of function calls of the NASA Crew Exploration Vehicle (CEV) mission platform, with results identifying anomalous components of its design. We close with conclusions and future work.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS iii

ABSTRACT iv

LIST OF TABLES vi

LIST OF FIGURES vii

SECTIONS

SECTION ONE: INTRODUCTION 1

Process Mining Overview 2

Contributions 2

Outline 2

SECTION TWO: BACKGROUND AND RELATED WORK 1

Process Representations 2

The Optimization Problem of Graphical Data Compression 2

A Naïve, Illustrative Problem Formulation 2

The Heuristic View of Graphical Data Compression 2

Graph Representations 2

Anomaly Detection 2

SECTION THREE: GRAPHICAL COMPRESSION OF PROCESS DATA 4

Process Mining 2

Terminology 2

Process Anomaly Detection Problem Definition 2

Problem Complexity 2

Previous Work 2

SECTION FOUR: PROPOSED ALGORITHM 4

Proposed Algorithm 2

Using Graph Compression to Discover Patterns and Cluster Traces 2

Anomaly Detection Method 2

SECTION FIVE: ANOMALY DETECTION EVALUATION 6

Data Generation 2

Experiment 1: Sensitivity 2

Experiment 2: Sensitivity 2

Experiment 3: Multiple Anomalous Structures 2

Variance Analyses for Experiments 1-3 2

Comparison with Existing Methods 2

Real Data Evaluation: NASA CEV Module Test Data 2

SECTION SIX: CONCLUSIONS AND FUTURE WORK 10

REFERENCES 15

APPENDIX

1. Full Experimental Results and Visuals 17

a. Experiment 1 17

b. Experiment 2 17

c. Experiment 3 17

d. Experiment 4 17

d. Experiment 5 17

Sampling Algorithm Evaluation Results and Visuals 17

LIST OF TABLES

Page

Table 1.1: Unfinished section, this will be finalized in later drafts 7

LIST OF FIGURES

Page

Figure 1.1: Unfinished section, this will be finalized in future drafts iii

**Dedication**

For Harlie, a pretty good dog.

SECTION ONE: INTRODUCTION

**Process Mining Overview**

The field of process mining involves the extraction and analysis of process models, which are derived from data characterizing the execution of an underlying, hidden process. Applications often involve business process management, such as tracking a manufacturing production method or a hospital patient admission process. However, the representation of process data and model mining has general application to natural sciences, industrial engineering, and virtually any discrete structured task environment for which a researcher wishes to understand the structural, temporal, or quantitative features of a known or unknown process. Thus, process mining bridges model construction and data analysis using a common set of formalisms for constructing models from data, and for subsequently analyzing these models according to recurring patterns and use-cases within a specific domain.

Although the process mining has many other applications, the two most prominent tasks are model construction and conformance checking. Model construction is primarily used when a process unknown, in the sense that no prior process model exists or simply no process mining system has yet been applied to it. Construction involves inferring a prototype model, *M’*, from a set of process-oriented data, *D*, where *M’* abides a formal language definition (such as a petri-net, as defined later), and *D* is specified by a process-oriented data schema and contains traces of process executions generated from the true underling process, *M\**. These terms are developed later in further detail. For now, it suffices to think of model construction as the abstract task of reconstructing a structured graphical model of a hidden process from traces representing executions of the process. For example, the transaction logs of user interactions with a commerce website could be used to reconstruct the graphical model of an online shopping process and its most frequent customer use-cases. Similarly, a log containing protocol messages for an automated distributed system could be used to reconstruct the industrial process they implement. Hence, model construction has applications that are both organic (noisy, loosely structured) as in the website example, or highly specified, as in the industrial process.

The second task, conformance checking, involves analyzing conformance between the behavior of *D* and some prior model specification, *M\*.* For instance, a bank might specify strict policies on all aspects of a complex financial transaction, to detect fraud or other deviations from the prescribed process model. In this case, conformance checking is likely very straightforward in terms of detecting missing, unexpected, or out-of-order events, such as detecting the approval of transactions without prior digital signatures or detecting a loan approval before a required background check. However, conformance checking can also encompass more continuous schemes for evaluating conformance according to the model’s objectives. In this work, conformance checking is implemented via unsupervised anomaly detection, the task of identifying unusual activity occurring in the context of the normative patterns of a process. These normative patterns represent a form of model construction, but are not in the scope of this work, which solely evaluates anomaly detection.

Model construction and conformance checking remain open problems because the complexity of structural data usually requires heuristic approaches, hence new heuristic methods are always being discovered, and better approximations are constantly being devised. These problems also possess many natural alterations, such as conformance checking processes with various costs, constraints, or bottlenecks assigned to their structural features, which is an active area of research. Such cases arise in operational settings for which a practitioner might wish to predict cost, resource, or other bottlenecks in a process.

Overall, the breadth of process mining is attributable to the multiple objectives it encompasses, as well as structural complexity that does not readily lend itself to algorithms that are both optimal and efficient. Most often, only one of these attributes is attainable, at the expense of the other. However, it is easy to see the value that institutions place on automated process intelligence to guide their decision making. This creates abundant opportunities, but also involves problems that are highly resilient to straightforward, canonical solutions.

**Contributions**

This work contributes to the fields of process mining and graph mining by devising a new method for process feature extraction and anomaly detection using graph compression. Using this unsupervised method, one can gain insight into many different processes and their structural properties. The method allows such processes to be tracked, measured, and improved in a manner which previously required prior knowledge of the process under evaluation. In complementary fashion, the method allows one to detect anomalies in unknown processes also without prior domain knowledge of the process, its implementation, or its prior formal specification.

The primary objective of this research was to further develop graph compression methods for the purposes of process mining and using process mining formalisms. Recursive graph compression was applied previously by [CITE Noble, 8] using a method that did not work in the same manner. The remedy for use in the process mining domain was a slight modification that yielded unexpectedly good results and possesses broader application in unsupervised learning.

The method presented in this work accomplished the following:

* A method for process modeling and structural trace analyses
* A method of anomaly detection capable of causally attributing anomalous structural components
* A hierarchical feature representation of the structural components of unknown processes
* Significant open-source software development
* Synthetic datasets and data generation software for benchmarking process mining tasks

These accomplishments overlap significantly since they involved the same method and the same project. However, the overall framework serves as a starting point for future research using other structural learning methods, and potentially deep learning. Learning structure within high dimensional data embodies the ability to learn compressed representations of a highly dimensional world, a core requirement of general purpose machine learning and artificial intelligence. The method presented in this work advances this goal directly, but also provides useful problem characterizations and datasets for use by implementers of alternative solutions for similar process-oriented tasks.

**Outline**

The remainder of this work is organized as follows. Section two introduces process mining in further detail, then characterizes the task of anomaly detection within this domain. Additionally, graph compression and graph representations are introduced, as they are used throughout this work. With this context established, section three introduces required terminology and defines the specific problem of process anomaly detection and characterizes its complexity. Section four then lays out the proposed method of graphical trace compression and its anomaly detection method. The anomaly detection method is evaluated in section five, along with a description of the datasets and experiments used to do so. The objective of these evaluations was to demonstrate the performance of the method over a range of possible inputs and internal conditions. This was done by defining the necessary data generation schemes, and then evaluating the performance for these datasets over a range of parameters, followed by a discussion of the results. Real world data evaluation is also included in this section. Finally, detailed conclusions and potential future work are discussed in section six.

SECTION TWO: BACKGROUND AND RELATED WORK

**Process Mining**

As described in [1], a process aware information system (PAIS) is, “a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models” (p. 5). This definition formalizes operational management systems as systems that centralize awareness of process data as well as the ability to prescribe tasks and activities via process models. These tasks form a loop by which process models can be defined, tracked, and evaluated; likewise, process models may be derived and analyzed from process data.

However, many institutions instead rely on interwoven, non-interoperable systems to monitor and control processes, making the formal requirements of a centralized PAIS infeasible or intrusive. This work focuses on contexts where a PAIS is instead an abstraction integrating process data derived from multiple systems. This is amenable to realistic scenarios in which processes are embedded within a non-stationary framework of changing people, tools, resources, and tribal knowledge, often in the absence of prescribed process models. These scenarios are frequent, since modeling such environments may only occur due to some ad hoc objective, such as an audit or root-cause analysis of a process failure. In these scenarios, a PAIS is a disparate collection of operational systems and data sources by which to derive traces characterizing the underlying process-oriented view of a system.

The ability to mine and analyze normative process patterns in these unstructured contexts is critical for mining regular activity and detecting anomalies. The latter requires prior normative activity patterns, thus anomaly detection and normative pattern mining are complementary tasks. This work presents such a method for mining process patterns from workflow logs that also possesses useful anomaly detection properties. We use the Inductive Miner [2] to construct a graphical process model from log data, then iteratively apply SUBDUE [3] to extract a hierarchical dendrogram of normative patterns executed on this model. Using this representation, anomalies and significant process features can be discovered in post-processing.

This hybrid approach is practical since the Inductive Miner extracts generality from process log data, outputting a graphical model *M* capable of generating all traces in a process log, including noise. SUBDUE then extracts a collection of *M’*s most informative components, constructing a hierarchy of sub-structures of *M* most relevant to the log as a dendrogram. Using this unsupervised method, one can mine normative process patterns, detect anomalies to those patterns, and perform other analyses.

**Process Representations**

Process modeling encompasses many different process algebras and formal languages, all of which have been developed to specify properties of different kinds of processes. Thus model representations may encompass varying levels of complexity to describe a process, or to satisfy certain properties like verifiability, reachability, and so forth, which are dependent on the choice of representation [CITE SOMETHING]. The canonical example is the Petri-net, whose properties are well-studied but still subject to ongoing research. Under this construction, a process is represented graphically by a set of vertices, “places”, representing activities of the process. Places are separated by “transitions”, which constrain the nature of transitions between places. Edges connect places and transitions to give the model its structure, where transitions connect to places, and places connect only to transitions. The model is given an initial marking, which is a token placed on one or more places to denote the initial state of the process. Lastly, a final marking is assigned in the same manner as the initial marking, to denote the final state of the process.

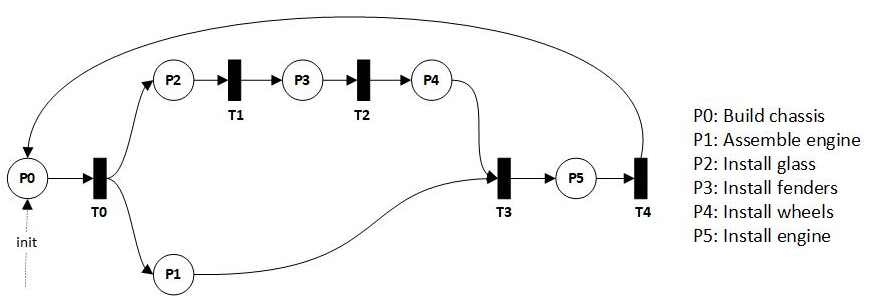
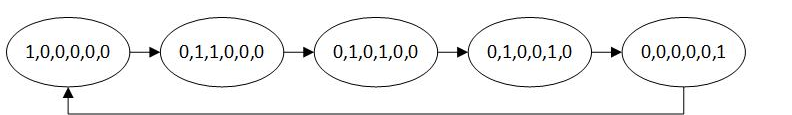


Figure : A petri-net model for a simiplified automotive manufacturing process.

An example process is given above for an extremely simple automotive manufacturing process. A valid walk on this graphical model is constrained by the initial marking and the transitions between places (activities) comprising the core verbs of some process. A walk is given by a token moving scheme, whereby no transition can “fire” until all places pointing to it have completed. These semantics allow a variety of formal analyses, such as verifying completeness, reachability, and satisfiability properties, which may be critical to verify a distributed system.

The visual structure of a process model itself imparts significant information as well. In this model, the path along the body assembly activities likely represents a bottleneck, since engine assembly entails significant waiting before body assembly completes, and thus T4 is allowed to “fire”, and the engine can be installed, completing a manufacturing cycle.

Petri-nets are amenable to a variety of derived graphical representations with useful properties, an overview of which is given by [Murata 1989]. One such derivation is the coverability graph given by a petri-net like the one above. This consists of the graph of all reachable markings given by a fully defined petri-net, which can be simulated by a many-fingered oracle: the oracle begins by placing fingers on each of the places in the initial marking set. Representing the marked places as a binary vector , the initial marking for the above petri-net (without subscripts) is . From this marking, the oracle moves her fingers forward between places and transitions; each unique possible marking spawns a new child node. Hence the only next marking is given by traversing T0 to places , giving marking . From this marking, the token on P1 can reach T3, but must wait until the other arrow pointing from P4 to T3 is active. The complete marking graph is given below. Note that despite the parallel paths in the model, the coverability graph is simply linear.



Ordinarily coverability graphs encompass greater complexity, and can be used for decidability problems, such as reachability, loops (as in the loop in the above example), and other formal properties. The applicability of reachability decisions to process engineering and analysis demonstrates a direct connection between the design of critical systems and many of the most intractable problems in computational theory. Extensive examples of such decidability algorithms are given by [Esparza 1994].

Anomaly Detection in Graphical Process Descriptions

A trivial and perhaps obvious anomaly detection scheme is to query the coverability graph per various decidability questions. Since the coverability graph of a petri-net model provides all valid activity paths through the model, the model-consistency of any trace (a single execution of the process) can be verified by simply checking whether the trace represents a valid walk on the coverability graph: entering, consuming all activities, and exiting correctly (for graphs with defined entrance and exit conditions). This is the most straightforward method for performing conformance checking, as defined previously.

However, in general traces may encompass significant noise and choice behavior, making the simple consistency-checking procedure prone to false positives. In general, such an anomaly detection scheme would only apply to scenarios in which the process model was formally prescribed for a given process. In fact, this begs an important point for the design of critical systems or high security contexts: anomaly detection can be vastly improved by designing prescribed process models (policies), such that detection is a trivial decision task. It is simply easier to design better systems, for which anomaly detection is easy, than to apply anomaly detection methods to a process that was poorly designed, if designed at all.

In most cases, however, and as mentioned in a previous section, processes do not obey a prescribed process model, such that the model itself must be treated as “unknown” and is extracted from data using process mining algorithms. In these cases, distinguishing anomalies, noise, and regular behavior is less trivial since each of these categories encompasses their own statistical, rather than deterministic, distribution. This is because such processes exist in a volatile context, and the process itself may not even be stationary. In fact, for all but critical systems scenarios, it is probably extremely rare for a process to be designed and tracked according to the rigors of the process mining and mindful process modeling practices. Instead, processes develop organically from changing staff, resources, and prototype *ad hoc* methods that evolve over time into the ingrained structure of some process. Software development provides an excellent example, wherein the rigors of process tracking are certainly required and even implemented; but in reality true conformance is regarded by developers and their managers alike as unachievable textbook lore, with every bug submitted.

In this manner, graphical methods can, and should be used to detect anomalies, by applying these methods to the graphical process representations mentioned above. In this work, graphical compression methods are used since they can extract significant, and hence normative, patterns of the graphical representation of a process with respect to the trace data from which it was derived. Hence graphical compression methods lend themselves directly to the analysis of processes and the detection of anomalies.

[MY representations???]

**The Applicability of Graph Compression Algorithms**

The purpose of this section is to introduce the context for graph compression algorithms, and to define the formal problem of compressing a dataset of graphical traces assumed to have been generated from some underlying graphical process model, *M*. The ability to induce the graphical structure of a process from trace data may seem to possess limited application outside of process mining, business process management, or similar operations fields. However, the induction of graphical structure from data encompasses many classical formal problems spanning machine learning, planning, and artificial intelligence, as will be discussed.

Consider a planning problem of a robot navigating a two-dimensional discrete state space , using elementary reinforcement-learning formalisms. Assume the action set contains four discrete actions , the environment is known, the transition model may be known or unknown, and the robot’s task is to navigate to some positive reward location while avoiding obstacles with negative rewards. These problem formulations are amenable to a wide variety of approaches in a class of algorithms modeling the sequential one-step dynamics of discrete action sets, for relatively modestly sized action sets. Many canonical solutions are rooted in Bellman equation formalisms or Monte Carlo methods, an overview of which is provided by [Sutton and Barto (1998)].

Now instead consider that the rewards and system dynamics are determined by *k*-step bounded dynamics, where the robot achieves goals only by executing specific partially-ordered sequences of actions of length *i*, for . For example, or ; additionally, these action structures may be interspersed with actions for unrelated concurrent tasks. Moreover, tasks may execute in parallel, but their independent pathways cannot always be separated without prior knowledge of the underlying process. These representations mimic real-life tasks, where actions possess long-range interdependencies, and tasks recursively decompose to subsets of actions, or “subtasks”, which do not necessarily entail one another despite their adjacency. Moreover, such tasks often can only be described graphically, by process model formalisms such as Petri Nets [Peterson, 1981].

These structured environments violate the clean, Markovian one-step sequential dynamics required by many classical reinforcement learning formulations, and result in exponential search complexity. For instance, the long-range activity set of action sequences in the above example, even excluding parallelism, is exponential in the number of activities, , and the bound on sequence lengths, k, resulting in . This is an intractable space even for this modest activity set, and without any assumptions about the complexity of the action space, which is inherently non-sequential.

These problems are the domain of classical planning problems, such as the block world domain of [Nilsson, 1980], for which a complexity analysis is provided by (Gupta et al, 1992). More accessible examples are the games of Go and Chess, with their high branching factor and multiple game strategies composed of long-range action dependencies. In these problems, the action space, action-sequence space, state space, or combinations of these are intractably large for traditional, sequential learning formulations without search heuristics or cleverly engineered state spaces. Current approaches often implement approaches such as Monte Carlo Tree Search [Brown et al., 2012], which still entail intensive search behavior before upper confidence bounds on action-value estimates yield satisfactory performance.

Graph compression addresses these problems heuristically by extracting the subgraphs characterizing advantageous behavior. Intuitively, a highly-compressing graphical pattern discovered by compression is also a task-meaningful pattern. For instance, in a highly complex combinatorial space such as chess, sub-graphs of useful actions could be learned via long sequences of user actions. Likewise, non-adversarial everyday discrete tasks such as performing home chores or driving may be learned from examples characterizing the underlying graphical model of the task. Given large amounts of such user data for this and similar planning problems, repeated subgraphs can be extracted and can bootstrap learning algorithms to bias their activities toward advantageous structural features. Conversely, the extracted information can be used for tasks such as anomaly detection.

Thus, many process-oriented tasks can be solved heuristically by learning graphical features of processes. These representations can then be used to efficiently learn complex behavior from compositions of subgraphs representing subtasks within a domain. As a result, methods for compressing and extracting structural patterns from graphical data have general applicability to machine learning, planning, and artificial intelligence, and encompass general task-learning.

**The Optimization Problem of Graphical Data Compression**

In the problem of graphical trace compression, we wish to maximally compress a set of graphical data representing directed, possibly-cyclic traces generated from an unknown underlying graphical model, . In the domain of process mining, these models typically have regular, compression-favorable structural properties, such as defined *begin/entry* and *end/exit* points for traces, modest overall average degree, few or zero cycles, and generally, if not always, behaving like directed, acyclic graphs (DAGs). The field of process mining provides many algorithms for mining from some input log ; can then be used to convert a log of partially-ordered traces into a set of corresponding graphical traces (hereafter simply ).

The goal of graphical compression is to reduce a set of graphical traces to a minimal collection of prototype subgraphs , or edge subsets, maximally compressing :

This objective defines the search for the minimum *k-*bit length set, , given log , though other criteria could be devised. Once the set is found, each trace can be encoded as a *k*-length binary vector indicating its member subgraphs, and thus the trace can be encoded and decoded via and . Thus, a lossless compression method can be devised by finding the minimal , converting each trace to a bit vector indicating its subsets in , and transmitting these vectors along with the edge-subsets by which to decode them.

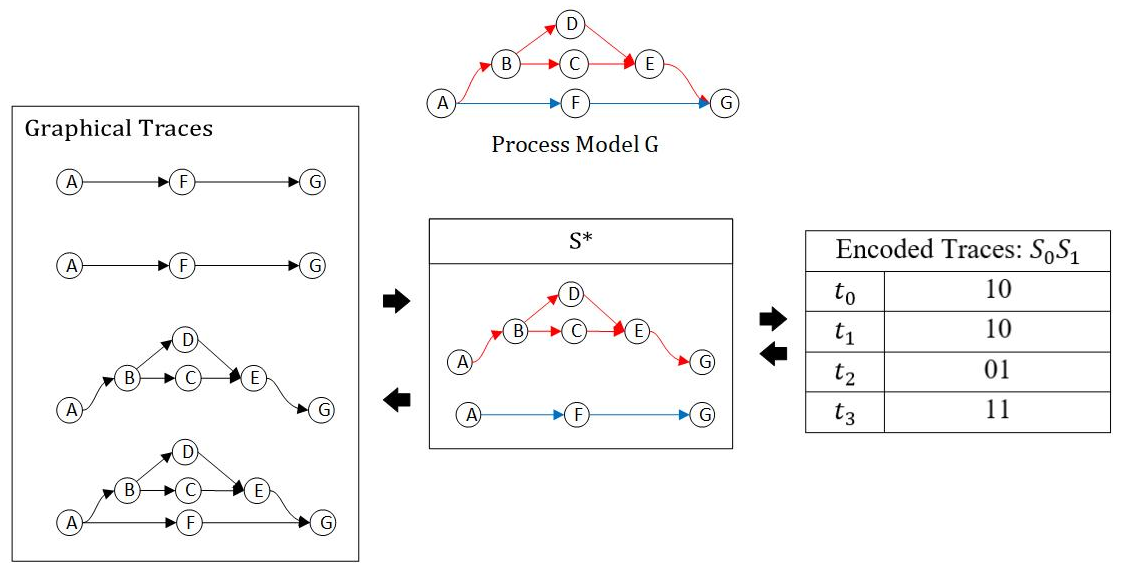


Figure : Graphical encoding and decoding. Trace set 'T' of four traces is shown at left, compressing set S\* (center), and the encoding vectors for each of the four traces (right), where a '1' indicates the presence of Si. S\* provides the encoding/decoding dictionary.

The general method is illustrated above in figure 1. Assume that any trace executes only one (or both) of the colored paths shown as the process graph . Then the colored components form substructures by which the entire log of traces may be compressed (encoded), by converting each trace to a binary vector. The colored substructures provide the mapping to and from these vectors.

This is also a very simple instance of graph grammars and graph parsing. Figure XX provides a direct example of a simple graph parsing output, where the encoded binary vector high-bits indicate the production rules by which to convert any trace back into its graphical form via the production rule set *S*. The similarity is coincidental, since graph grammars encompass more complex recursive rules than the example in figure 1. However, many graph grammar induction algorithms have been published, given their relevance to tasks such as compiler construction and various mathematical problems. An example of heuristic graph grammar induction is given by [Jonyers et al., 2002] using a similar approach to the one demonstrated later in this work for dendrogram-induction from process data.

**A Naïve, Illustrative Problem Formulation**

The formal problem of finding the compressing set reduces to iteratively selecting columns from the unfolded binary adjacency matrices of all traces in the directed graphical data. For a dataset in the form of a trace log, , of size , each trace is a subgraph of the super-graph . represents the graphical behavioral model inclusive of all traces, which is the union of all edges in . Each trace can be represented as an adjacency matrix whose rows and columns are composed of the vertices of . Concatenating the rows of this matrix in row-major order gives a binary vector whose non-zero components indicate the directed relations (edges) present in the trace, as shown in figure 1.

This representation is solely for illustration, since this binary representation only captures first-order structure. For instance, it does not quantify edge transitions, such as multiple loop executions. However, it is sufficient to describe many graphical structural properties via labeled, directed graphs with unweighted, undecorated edges.

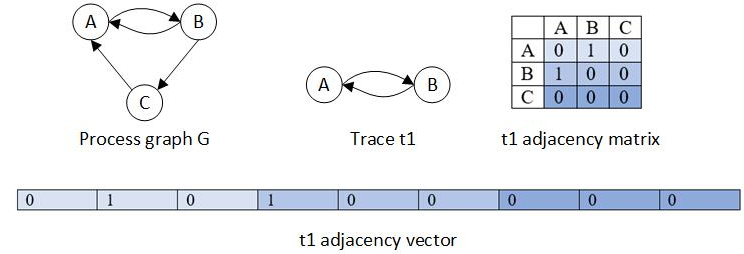


Figure 3: Shown above are the graph G, the process model inclusive of all traces. Trace t1 is an example subgraph of the execution of this process, with its associated adjacency matrix and adjacency vector representation.

The entire set of traces can be converted to a matrix, of adjacency ‘vectors’ like the one shown above in figure 2. A hypothetical matrix is shown below in figure 3, whose shaded regions are explained further along. This data representation is illustrative because it demonstrates the dimensionality of the input space, which is quadratic in the number of vertices, for a complete, directed graph *G*.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Edges, | | | | | | | | | | |
| Traces |  | A-A | A-B | A-C | B-A | B-B | B-C | C-A | C-B | C-C |
| t1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| t2 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| t3 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| t4 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| t5 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| t6 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

Figure : A log data representation in which each trace’s adjacency matrix is unfolded into a row. The complete log includes all such rows. The light blue regions represent column and row selection, and the darker blue regions are their intersection.

Discovering the maximally compressing subgraph within this data representation reduces to finding the largest subset of columns containing all 1’s and encompassing the greatest number of rows. Such a column set of edges represents a subgraph , if connected. The size of the column set represents the size of the subgraph in edges, likewise, the number of rows is its frequency.

In this manner, a maximum-compressing subgraph can be found at each iteration by searching the columns of this matrix for the largest collection of all 1’s. The shaded columns in figure 3 depict such a search; the highlighted rows are rows containing all 1’s for the given column selection. For any fixed choice of columns, all traces must be traversed to count the number of rows for which the conjunction of this fixed choice of columns evaluates to true, and there are possible choices of columns. Defining the set of all edges in graph as , the set of all vertices and using the identity (Cormen et al, 2001), the complexity of this procedure is given by:

Equation : Search complexity

Thus, this column selection procedure is exponential in the number of edges, which in the worst-case for a directed, fully-connected, and non-reflexive graph, is:

Hence,

Equation : Bounded search complexity

This search procedure illustrates the problem’s structure and brute force complexity for only a single compression iteration. It is flawed since it needlessly searches over all combinations of subsets of edges of size *i*, whereas we are only interested in connected components of the . Graphical problems frequently involve sparse graphs, so we can expect to reduce complexity by restricting iterations to the subsets of columns representing connected components. For special kinds of graphs like DAG’s, components can be enumerated using basic, elementary graph-search procedures. Lastly, real graphical data typically has high redundancy, such that can also be reduced significantly by a de-duplication strategy of storing each unique row with its frequency, and further by omitting columns of all zeroes, such as the ‘C-B’ and ‘C-C’ columns in figure 2.

Critically, this search characterization overlooks how subgraph size and subgraph frequency affect the optimality of the resulting set . The search over columns (substructures) and rows (frequency) introduces a compression tradeoff between size and frequency: it may be easy to find a very large and infrequent subgraph, or conversely a very small but frequent subgraph (such as a single edge). Consider two candidate prototypes, and . Let have and , and let have and . Which of or should be chosen to obtain optimal compression, ? How do criterions of and affect the optimality of the resulting set ? A straightforward heuristic is to select the subset with the largest sum of 1’s as a measure of information gain.

Worse yet, dependencies exist between the selection of subgraphs since candidate subgraphs for compression are not disjoint. The selection of a prototype at the t*th* iteration can affect which candidates are available in subsequent iterations, which may affect the compression of the resulting set . Optimal compression is defined as minimizing the description length of the trace-graph codes sufficient for losslessly decoding all trace-graphs from their encodings via . Due to these tradeoffs, maximally encoding the subgraphs requires correctly making the complete sequence of decisions per the size and frequency of each prototype subgraph. This problem is akin to bin-packing [Korte, 2008], an NP-hard combinatorial problem, but harder due to the overlapping dependencies between prototype selection. Loosely, each prototype’s size and frequency define each object’s abstract dimensions, while the objective is to fit as many of these objects as possible into the smallest bin, thus globally minimizing . The similar sequential problem of generating the smallest context-free grammar generating a sequence of symbols (e.g. traces) is also known to be NP-hard [Charikar et al., 2005]. Fortunately, the optimal formulation of this problem is not the subject of this work. But understanding the problem structure of the simple brute-force graph compression formulation given in this section is important for designing alternative methods.

**The Heuristic View of Graphical Data Compression**

In a relaxed version of the trace-graph compression problem the encoding need not be optimal but may instead heuristically generate an approximately optimal encoding much faster than the optimal . Such procedures can still be lossless, such that any trace can be completely reconstructed from its encoding and the set to decode it. Many heuristics are possible, since the task is relaxed to that of iteratively finding frequent subgraphs under some compression criterion, such as favoring the size or frequency of graphs. Many of these heuristics can be viewed as greedy methods for reducing the column selection search or for reducing the leading constant in equation 2. An overview of such methods can be found in [Maneth et al., 2017]. Many graph problems involve graphs with node attributes or other additional information that may also be incorporated into the information-theoretic definition of their encoding.

The prior problem of discovering the smallest context-free grammar for generating sequences relates to graph grammars and graph parsing. The example in figure 1 provided a direct example of a simple graph parsing output, where the encoded binary vector high-bits flag the ‘rule productions’ by which to convert any trace back into its graphical form via the production rule set *S*. The similarity is coincidental, as graph grammars normally encompass complex recursive rules. However, many graph grammar induction algorithms have been published, given their relevance to computing tasks such as compiler construction and various mathematical problems. An example of heuristic graph grammar induction is given by [Jonyers, 2002] using a similar approach to the one demonstrated later in this work for dendrogram-induction from process data.

Notably, the adjacency-matrix based data representation in the prior section is amenable to a wide range of supervised and unsupervised learning approaches. Unsupervised approaches, such as neural autoencoders, hold great promise in terms of automating graphical pattern discovery. This framework trains a neural network using the input as the target output. Using various training and architectural strategies, these networks learn the hidden structural patterns of the data, by which normative and anomalous patterns can be determined. A recent example is given by [Nolle et al, 2016], in which the authors used a denoising autoencoder model for both anomaly detection and normative pattern discovery. The authors presented the traces to their networks as linear activity sequences, which is the canonical process mining trace representation, rather than trace adjacency matrices. They reported their method perfectly split the trace log into normal and anomalous traces. Similar work is possible using recurrent neural networks [Ellman, 1990] [Jordan, 1986] by presenting the traces to the network as linear activity sequences, although the hidden layers of such a model would be difficult to interpret. More recent neural models combine recurrent networks and autoencoders for a variety of similar structured input/output tasks, such as the seq2seq model of [Kalchbrenner et al., 2013] or the long-short-term memory (LSTM) autoencoder framework described in [Srivastava, 2015].

More traditional supervised learning models can also be adapted to unsupervised pattern discovery. By appending a +1 to each binary adjacency ‘vector’ as a dummy target “output” for a learning model, the unsupervised dataset can be mapped to a supervised learning representation. Each vector can likewise be replicated by its negation, possibly with additive noise, to generate a semi-synthetic supervised learning dataset dividing the input space into two classes: positive examples, and ‘negative’ synthetic examples sampled outside the set via some distribution facilitating a specific learning model. Some distant examples of such data extension/generation strategies are the negative sampling used by some implementations of the word2vec algorithm (Mikolov et al, 2013), various structured learning algorithms like the DAgger algorithm (Bagnell, 2015), or (very distantly) the generative adversarial networks of (Goodfellow et al, 2014).

The benefit of such a supervised data representation is that many supervised learning models have been developed, especially generative ones, by which normative patterns or other model parameters can be learned to perform secondary tasks like anomaly detection and normative pattern extraction. The simplest approach is to run linear regression on the preceding semi-synthetic dataset . The result output is a weight vector whose non-zero components correspond to the collection of edges which maximally “compress” the data by minimizing the mean-squared error (MSE) loss. The corresponding columns would then be removed from the input data, and the procedure would be re-run on the remaining examples to find the next set of such edges, and so on, until the data is completely compressed. Notably, the edge collection found on any iteration might not represent a connected subgraph, but regularization strategies could be devised to bias the learning algorithm toward connected, rather than disconnected edge subsets.

SUBDUE

While many data representations and strategies could be explored, this work’s primary focus is on the SUBDUE method for discovering the maximally compressing components of graphical data. In contrast to matrix representations of graphical data, SUBDUE is search-based and focuses on the vertex perspective to search for compressing substructures. In this manner, SUBDUE proceeds by “growing” candidate substructures within some search beam of size *k*, maintaining only the *k* most highly compressing components in the beam at any time. Compression is measured by the reduction in the description length of the data with respect to compressing components, via the minimum-description length (MDL) principle [CITE]. This metric can encompass additional information, such as node labels, which is beyond the scope of this work since activities are treated as single symbols, hence every vertex is unique and every vertex label is the same effective description length.

From the SUBDUE perspective, compressing components are found not by solving a brute-force global search over edges, but rather by growing compressing components from the neighborhood surrounding promising nodes. This work’s focus is on structural compression, hence SUBDUE’s ability to efficiently discover compressing patterns maps directly to the compression of process data to its most relevant structural features, and thereafter the discovery of anomalies. The reason SUBDUE performs well for anomaly detection is that using this process-feature discovery mechanism to compress process data, eventually all that remains are the “sore thumb” poorly compressing components of the data. The beam size additionally provides a hyperparameter for constraining search complexity, where a larger beam size allows SUBDUE to discover more compressing patterns.

SECTION THREE: PROBLEM DEFINITION

Whereas the previous section was intended to characterize the formal problem of graphical compression, this section and the remainder of this work involves its application to process data and anomaly detection. This section introduces the application domain of process mining, the method developed, followed by experimental results and conclusions.

**Terminology**

The above captures the spirit and contribution of the method but requires definitions within the scope of this work. From the control-flow perspective, common process mining terms can be framed in a graph theoretic manner:

* Process model: A graph with vertices representing activities, and edges representing one-step transitions between activities. Processes can contain many constructs representing linear and non-linear constructs, and a variety of notations and languages have been defined over the space of process models. A canonical example is the Petri-Net [4] [5], shown below at left. Its simpler control-flow counterpart is shown at right, and is used throughout this work to describe the structural activity patterns of traces



Figure 5 Petri model (left) and its simpler control flow counterpart (right).

* Workflow trace: A single execution of a process as a partially-ordered sequence of activities, following any valid path from a START/BEGIN to an END/EXIT node on a process model. These are represented as a string composed of letters representing the activities traversed.
* Workflow log: A set of workflow traces, for which various measures are taken to mitigate properties of noise or incompleteness. Synonymous with ‘trace log’, ‘process log’, or often just ‘log’.
* Process miner: Any algorithm for constructing a process model from a workflow log, and often incorporating criteria for model complexity, specificity, and generality. Specificity favors restrictive models including only or even less than the behavior described by the workflow log, whereas generality favors larger models describing all traces and possibly additional behavior.
* Partial-order property [18]: A property of workflow traces whereby activities may be randomly-ordered with respect to parallel sub-processes. ‘ABCD’ and ‘ACBD’ might be workflow traces from some model, where ‘C’ and ‘B’ represent parallel sub-processes, or may recursively embody further parallel sub-processes, and ‘A’ always occurs before ‘D’. The primary challenge of process mining algorithms is disambiguating these partial-orderings, applying rules and heuristics to generate models with desired properties of complexity, specificity, and generality. The enormous space of possible graphical models defined over a set of partially-ordered traces constitutes the primary search problem for these algorithms.
* Process grammar: Recursively-defined constructs for common process patterns. An AND-SPLIT is a set of edges branching from a single node and traversing activities in parallel before synchronizing at some later activity. Other basic constructs include OR-SPLIT, XOR-SPLIT, LOOP, and JOIN [14] [15].
* Replayability: The ability for a partially-ordered trace *t* to be generated from some process model *M*. The string ‘ABCD’ would be replayable on the model above (figure 1), but ‘ACDB’ would not.
* Spaghetti model: A workflow defined by highly diverse, informal, and disorderly behavior, typically containing many scattered, repetitive events. These represent unstructured business processes, in contrast to orderly “lasagna” processes with prescriptive, stratified behavior.
* Inductive miner: A process mining algorithm capable of generating the most general, all-inclusive process model of the traces in some log. For our purposes, this model is used to convert a workflow log of traces into a collection of subgraphs as structured input for SUBDUE.
* SUBDUE [2]: Short for “Substructure Discovery”, this method implements a subgraph beam search over a graph collection and, by applying the minimum-description length (MDL) heuristic, returns the top-k most compressing sub-graphs of the collection.
* GBAD: Acronym for “graph-based anomaly detection” [6], this method internally calls SUBDUE, then implements methods for detecting anomalies occurring in the context of discovered patterns.
* Dendrogram: A graphical representation of the non-disjoint compressing features of some graph *G*, with respect to a set of traces. Each vertex represents a compressing subgraph of *G*, and edges between features indicate familial relationships, such as indicating that some trace includes two separate compressing subgraphs. Such a construction can be represented multiple ways: directed or undirected, and with or without edge weights. Devising edge/vertex quantifications of this representation is a crucial task, since it facilitates how the representation can be used. An example is quantifying edge weights per trace counts between two compressing subgraphs, by which regularity statistics can be measured, and thus unusual behavior detected. The term “dendrogram” is used here only for its hierarchical implications, since this representation can contain vertices with multiple parents, whereas most dendrograms only allow disjoint single parent-child relationships. An explicit example dendrogram is given later and further refined.

An in-depth overview of process mining terms and methods can be found in [1] and [7].

**Anomaly Detection Problem Definition**

In this work, one is given a log , consisting of traces generated from some unknown process model *,* a graphical process miner , and a graph-compression method . The formal problem is to mine a graphical process model by which to convert into a collection of graphs via (a preliminary step), and then to mine the normative patterns of using s. The patterns can then be used for post-processing tasks, such as anomaly detection: given , identify anomalous traces .

**Process Anomaly Detection Problem Definition**

**Input** *mine*: A process mining algorithm

*L*: A trace log from some process

*compress:* A graph compression method

*detect:* An anomaly detection method

**Output** : The set of normative graphical patterns

: A set of anomalous traces in L

1. = *mine*(miner, *L*) #mine the graphical process model
2. traceGraphs = convert(, *L*) #regenerate the log traces as graphs, using *model*
3. = *compress*(traceGraphs)
4. = *detect*(, *L*)
5. return ,

This problem definition is overly verbose and could be reduced to solely outputting the set of anomalous traces to reflect the original problem statement. Outputting reflects that in many cases an anomaly detection method will be able to provide normative pattern information along with the anomalies it discovers, though this is not always the case. Likewise, both *compress* and *detect* could be reduced to the single task of anomaly detection.

Based on this problem definition, a solution’s performance can be evaluated based on how well it classifies anomalous/non-anomalous traces in logs with known anomalies, per standard binary classification. Alternatively, from the compression perspective, one could ignore the binary classification problem of anomaly detection, and evaluate a solution in terms of to determine how well a method compresses a log according to some compression-evaluation function. This is a subject of future work and would require different experimental designs for evaluating log compression performance. But it is important to distinguish these formal tasks: anomaly detection and compression-evaluation. The latter is an important task for evaluating machine learning approaches to structural learning in terms of the compactness of their learned representations.

**Problem Complexity**

The complexity of finding anomalies is a function of their structural complexity. Whereas many anomaly detection works focus on statistical events, this work deals with graphical anomalies: the unusual modification of graphical structure. The term “unusual” refers to the fact that anomalies are defined in the context of normative patterns, hence the complexity of finding graphical anomalies reduces to finding normative graphical patterns, which was established in a previous section. That is, even for the case of unquantified edge data containing only binary structural information, the optimal search for normative patterns was shown to be where *d* was the size of the log , and *V* was the number of vertices of the graphical model.

Thus in the new problem domain of process anomaly detection, *V* is the number of activities in a graphical process model, and therefore complexity is largely determine by the size of a process model. Again, the complexity estimate is a gross over-estimate, which simple heuristics can overcome. Additionally, processes in this domain are expected to be of modest size, consisting of fewer than 50 activities on average. Data containing models larger than this would likely need to be decomposed to specific processes, such as the complete log data of an entire company and all its separate departments. Oftentimes the number of activities in process data is view-specific to how “activities” are represented and what they represent in the domain under test. For instance, within some data one might choose for activities to represent event types or finer event names, such as a patient’s transition between the different areas of a hospital versus the fine-grained treatment tasks associated with their entire passage through a hospital. The former evaluates traces with respect to an organizational view, and is likely more appropriate, whereas the latter is very fine grained and will likely contain lots of noise and fewer recurring patterns. Another good example is process modeling an operating system for which activities represent function calls. A typical operating system contains thousands of functions, so it would be more appropriate to model interfaces, each of which likely implements a much more modest number of functions.

In sum, the complexity of anomaly detection in processes reduces to the size of the underlying model, in terms of the number of activities (vertices) it encompasses. Real world processes usually meet these requirements, having fewer than 50 activities or so, although this threshold is loose and is view-specific. Heuristic normative graphical pattern methods will surely vary in terms of how accurately and how quickly they can discover normative patterns, and subsequently anomalies, as processes grow in size.

SECTION FOUR: PROPOSED ALGORITHM

**Previous Work**

The SUBDUE graph-compression method mines normative patterns from graph data, working as a graphical feature detector. It was previously used for knowledge representation systems [1], and more recently in security applications for intrusion detection [8]. Using SUBDUE as a process mining tool was successfully performed by [9] and more recently by Genga [10], whose results demonstrated the method’s utility for “spaghetti processes” describing more realistic institutional processes. An extensive overview of graphical compression and anomaly detection is provided by Akoglu [20].

GBAD builds on SUBDUE to provide anomaly detection capabilities, particularly within the immediate proximity of normative graphical patterns. This is appropriate for safety-critical and security contexts possessing some underlying process model by which normative patterns can be assumed to have a ground-truth behavioral policy, but less so when there is no such policy or model. An application is given by Holder and Eberle, in which GBAD was used for insider threat detection by combining three anomaly detection algorithms [16].

Process-mining anomaly detection focuses primarily on the mining process itself and on trace-scoring schemes. W. van der Aalst [12] details scoring schemes, by which work traces are replayed on a discovered model, assigned a numeric fitness score, and anomalies flagged based on a discriminative threshold. Bezerra’s work [11] examined anomaly detection using several threshold-based approaches within the process mining algorithm itself. Bezerra decomposed this family of process-based anomaly detection into three groups: score based, iterative, and sampling. Our approach does not fit squarely into these categories since it is compression/feature based: a generic process model is mined, graphical features detected, and anomalies are detected and reported in post-processing. Likewise, whereas previous works focused on individual traces, our feature-based approach provides structural insights into normative patterns and anomalous features. Our work replicates Bezerra’s data generation scheme, but otherwise adds a new method to this work.

**Method Overview**

Under our relaxed definition, a PAIS is a composition of process monitoring systems by which workflow traces are extracted for process mining in a standard log format, such as Extensible Event Stream (XES) [17]. Our method decomposes to three tasks: converting a trace log to a collection of subgraphs via a mined process model, extracting descriptive normative graphical patterns of the log with respect to this model, compiling these patterns into a dendrogram of graphical features, and lastly detecting outliers and anomalous behavior. The overall data-flow of our approach is shown below in figure XX.



Figure XX: Data-flow model of our approach

For the first task, the Inductive Miner was suitable for mining the most general graphical process model described by some log. This model is typically overly-inclusive, whereas the second and third tasks discover the patterns and features precisely relevant to the log. For this, we used the SUBDUE graph-compression method to discover normative behavioral patterns, subsequently enabling anomaly detection.

The workflow extends the discovery of process characteristics irrespective of prior constraints, such as a prescribed process model or a formal PAIS. It is tolerable for the process mining algorithm to produce overly general models, since significant graphical features are extracted in post-processing, rather than within the mining algorithm itself. Decoupling the feature extraction and mining steps simply offers greater tuning for noisy or poorly structured process data. This facilitates more realistic and informal “spaghetti” model scenarios in which processes are ad hoc and highly unstructured, such as enterprises, communication networks, distributed systems, software system executions, or natural processes. For instance, the mining algorithm or the graph-feature extraction components shown in figure XX can be readily replaced with components to better fit the statistical properties or scale of a specific dataset.

**Using Graph Compression to Discover Patterns and Cluster Traces**

SUBDUE discovers compressing patterns in graph data via the minimum description length (MDL) principle and a beam search over candidate subgraph patterns. This satisfies the requirement for an unsupervised method of discovering a hierarchy of meaningful components of a graphical process model, since a workflow log is also a set of subgraphs generated by a process model. The Inductive Miner complements this approach by providing the super-graph for converting a log of partially-ordered traces into subgraphs; these subgraphs are passed to SUBDUE.

Prior work showed strong anomaly detection results when running SUBDUE iteratively on a set of graphs [8]. In these author’s work, instances of the most compressing subgraph were replaced with a single meta-node at each iteration, and the method was repeated until no further compression was possible. At the end, the authors obtained a recursive and hierarchical description of a set of graphs, by which they modeled their anomaly detection scheme. Several other previous works also used SUBDUE for hierarchical graphical clustering, applying the same pattern of iteratively replacing compressing substructures with single-node prototypes to generate a hierarchical decomposition of graphical data [123] [124] [125] [126].

We tested a similar approach using GBAD, by which workflow traces were iteratively recompressed using the most-compressing subgraph found at each iteration. The three anomaly detection methods of GBAD were used to detect anomalies at each iteration. While successful at discovering patterns, this method suffered a high false positive rate for anomaly detection. Ultimately the issue was iterative recompression: on successive iterations, the most highly compressing subgraph was often only a small alteration (node substitution, deletion, or insertion) to a compressing subgraph found by the previous iteration. GBAD’s primary deficiency in this context is that its anomaly detection methods apply to the local vicinity of the compressing pattern discovered by SUBDUE. Hence, the search space was highly redundant, repeatedly analyzing the same regions of the process model, progressing very slowly toward the outlying model regions where compressing structure decays and anomalies often lie.

Since the desire was for SUBDUE to analyze new regions, the remedy was to delete all instances of the most-compressing subgraph from the traces at each iteration. This forced SUBDUE to discover regularity in new regions, and thus to discover dissimilar graphical features at each iteration. Compressing graphical features (substructures) are thus compressed away in order of decreasing information, as shown below in figure XX.



Figure : Constructing a dendrogram of graphical features from graphical process data

Above, iterations are portrayed from left to right, with the original activity traces shown leftmost as strings, and their graph counterparts in blue. In the first iteration, S1 is discovered, a graph of four nodes. S1 is then deleted from all traces (notably, along with any incident or outgoing edges). Next, S2, a substructure of two nodes, is discovered and deleted. In this fashion the entire log is compressed away, building a hierarchical dendrogram of substructures as a directed acyclic graph, as shown rightmost in green with some hypothetical frequency labels. Edges of the dendrogram represent immediate ancestry: above, S2 is found in 45 traces; S3 is then found on a later iteration within 2 traces, both of which were previously compressed in S2’s 45 trace set. The links reflect immediate ancestry of successive compression iterations, so reconstructing a trace from its structural components amounts to an upstream walk on its ancestral edges.

The approach loosely resembles data dimensionality reduction, in which data is compressed via an ordered set of vectors of decreasing information. Except vectors are replaced by graphical substructures, forming a lossy hierarchical derivation of process substructures as a dendrogram. Lossy, since any edges incident to or exiting from a compressing substructure are deleted with the substructure and in general cannot be deterministically reconstructed from the dendrogram. Otherwise the dendrogram comprises the entire behavior of the log, with the ancestral components reflecting its most relevant and compressing graphical features. This is amenable to anomaly detection since the less compressing a feature is, the greater its deviation from normative patterns and normal overall behavior, and hence it will be located “deeper” in the dendrogram and with lower frequency.

This gives the following process-oriented pattern-mining algorithm:

**Algorithm 1: SUBDUE-based Process Log Compression**

**Input** *mine*: A process mining algorithm (e.g., the Inductive Miner)

*log*: A trace log from some process

**Output** *dendrogram*: A graphical decomposition of the log’s structural features

1. model = *mine*(miner, log) #mine the graphical process model
2. traceGraphs = convert(model, log) #regenerate the log traces as graphs, using *model*
3. *dendrogram* = {}
4. while not empty(traceGraphs):
5. bestSubstructure = MineBestSubstructure(SUBDUE, traceGraphs)
6. *dendrogram* = AddSubstructure(*dendrogram*, bestSubstructure)
7. traceGraphs = DeleteSubstructure(traceGraphs, bestSubstructure)
8. return *dendrogram*

As described, the Inductive Miner takes a workflow log and returns a process model by which each trace is converted to a graph, and hence the entire log is converted into a collection of graphs. This collection is iteratively fed to SUBDUE to discover the most compressing substructure, which is appended to the dendrogram and deleted from all traces. This step repeats until all traces have been compressed to their most elementary substructures. The dendrogram is returned, whose vertices represent compressed substructures, and whose edges represent substructure ancestry between compressing substructures.

The strength of this method lies in the dendrogram as a descriptive model of the input log. The dendrogram can be analyzed in post-processing for frequent process features, redundant behavior, outliers, and anomalies. Diamintini *et al.* [13] successfully implemented a variety of uses for similar SUBDUE-based dendrograms, especially in the context of spaghetti processes.

This method of frequent-subgraph mining of workflow logs belongs to the family of dendrogram- or tree-induction methods in process mining literature [19], for which anomaly detection is only one application. For instance, while the low-frequency, outlier components of the dendrogram characterize anomalies, outliers, and noise, the ancestral components encode the most relevant substructures of a log. Using this information, the process model returned by the Inductive Miner could be reduced for greater specificity. The recurrent behavior of an unstructured institution is discovered, and thereby important processes can be identified, measured, and improved via business process formalisms.

In sum, coupling SUBDUE with the generality feature of the Inductive Miner provides a framework for concise modelling of unstructured “spaghetti” process environments. Similarly, an analyst may examine highly similar components of the dendrogram, likely indicating duplicate work or poor cohesion among business processes. The dendrogram offers a range of pattern mining and other enterprise uses.

**Anomaly Detection Method**

Anomaly detection provides an illustrative example because of the structural characteristics of the dendrogram: given that anomalies are assumed to be infrequent events in the context of regular structure, subgraphs containing anomalies will be among its lower-frequency components. A second effect is that the size of the dendrogram components decreases smoothly then drops suddenly, such that the only remaining traces/subgraphs are those representing anomalies, outliers, and noise.

This property is useful for anomaly detection since many discriminating metrics can be devised to differentiate anomalies, noise, and regular patterns. Given that anomalies occur in the context of regular behavior, the anomalous structures tend to have sharply lower frequency than their parent substructures in the dendrogram. They are also distinguished from noise in the input log, which tends to result in poorer structural decomposition of a trace, and as such, substructures characterized by noise and their parents both tend to have lower frequency. Hence, detecting anomalies resolves to finding these sharp boundaries between high frequency substructures and relatively lower frequency substructures adjacent to them.

A local Bayesian metric was selected to capture this property, based on the frequency of substructures and their parent (immediate ancestor) substructures in the dendrogram. The metric was chosen because it discriminatively quantifies the relationships between parent and child substructures so that unusual child substructures have very low probability. Under this model, each substructure is assigned a Bayesian probability defined as:

Equation 1

Where unconditional prior substructure probabilities like are defined in terms of the global probability of a substructure in any trace, or , where the ‘#’ operator returns the frequency of substructure *s*. It is worth mentioning that this metric is subject to criticism due to its local nature: the edge-relationships in the dendrogram only loosely represent parent-child relationships via their compression order, whereas a child’s probability could be estimated from all of its ancestor vertices for a global characterization. For parentless root nodes, we simply define such nodes as their own only parent, such that , which is needless, since root nodes are nearly always highly compressing and frequent initial features.

Characterizing involves defining the probability of a parent relating to one of any of its children, where parents may have multiple children, and children may have multiple parents. Hence, one must sum over all parents of a given child, a set of independent events (since substructures assumed independent), and weight each event by its likelihood :

Equation 2

The repeated looks unusual, but correctly weights each summation component as required to obtain a proper probability distribution. Substituting ‘c’ for child, ‘p’ for ‘parent’, and ‘P’ for ‘parents’, the fully-defined metric becomes:

Equation 3

An awkward looking probability, but the properly weighted Bayesian definition of a child’s probability given its parents. Anomalous substructures are expected to have a low value for , and a substructure is flagged as anomalous when , where is the anomaly threshold in . Traces containing the anomalous substructure are then flagged, as follows:

**Algorithm 2: Dendrogram-based Anomaly Detection, Using a Bayesian Threshold**

**Input** *dendrogram*: A dendrogram, as output by Algorithm 1

*bayesThreshold*: The anomaly detection threshold

**Output** *anomalyIds*: A set of trace id’s flagged as anomalous

1. *anomalyIds* = {}
2. for vertex in *dendrogram*:
3. bayesProbability = (vertex)
4. if bayesProbability < *bayesThreshold*:
5. traceIds = GetVertexTraceIds(vertex)
6. *anomalyIds* = *anomalyIds* traceIds
7. return *anomalyIds*

SECTION FIVE: ALGORITHM EVALUATION

**Algorithm Evaluation**

Although real process-oriented datasets are available, they do not offer controlled conditions sufficient to compare the characteristics of algorithms over a range of data parameters. Instead we used a synthetic data generation algorithm as found in appendix A of [11], modified to generate data directly from probability distributions embedded into the generated models. This approach generates random process models from which synthetic traces are generated, and thus the performance of an anomaly detection method can be assessed with respect to a known model and known trace-generation parameters. To cohere to a stable performance baseline, we also used the same experimental set up as described in [11]: 60 randomly-generated process models and 1000 traces per log.

**Data Generation Algorithm**

Data generation consisted of two steps: generating process models and generating traces from them. The parameters described the probability of recursively generating various structural features, including SEQ, OR-SPLIT, AND-SPLIT, and LOOP, defined as follows.

SEQ: the appending of a single activity.

OR-SPLIT: a single activity splitting to one of two successors.

AND-SPLIT: a single activity splitting to two parallel activities, both of which are traversed.

LOOP: An activity splitting to an optional loop, then returning to the activity.

These recursive operators generate directed, potentially cyclic graphs of arbitrary complexity, with the constraint that the graph start at a single START node and all paths eventually terminate at a single END node. Additional complexity results by including the null transitions in the set of “activities”, and as such the split constructs may divert to more than two activity paths or may bypass components of a model. The parameters constrain model complexity to a probability distribution over these operators, replicated from *Bezerra et al.* [11] and fixed throughout this work.

Since probabilistic model-generation allows for the possibility of unusual or task-trivializing models, additional basic tests were applied to ensure sufficient complexity. These included verifying that models contained a minimum START to END path length of one, maximum of one anomalous structures, maximum of four anomalous edges within an anomaly to (constrain anomaly size), minimum of 10 unique activities, and minimum of 10 unique paths from START to END for adequate model complexity.

A second parameter set defined the trace-generation distribution constraining the graphical walks of traces, which is only defined for the choice operators OR-SPLIT and LOOP. These parameters determined trace diversity, from very uniform to very non-uniform. A value of implies the trace-generation scheme assigned a 0.9 probability of traversing one branch and a of its alternative. By varying from very uniform (0.5) to very non-uniform (0.9), one derives a less uniform distribution of traces, making anomaly detection more difficult as anomalies and regular behavior become ambiguous. To ensure maximum partial-order entropy, activities lying ambiguously within the same timestep were shuffled for uniform random partial-order.

Lastly, the parameters defined the probability of generating anomalies, and encompassed both the generation of anomalous structures, and their embedded traversal probability when generating traces. Anomalies in this context are defined as unusual behavior occurring in the context of regular behavior, hence in this work we desired to generate insertion, substitution, or deletion anomalies in the context of frequent behavior. LOOP and OR constructs were marked as anomalous with fixed probability 0.3. Anomalous paths were marked with a traversal probability that was experimentally varied between 0 and 0.2 in increments of 0.02. This overall method generated insertion, substitution, and deletion anomalies (since OR branches may be mere null transitions). Notably, embedding anomalies probabilistically allows for generated logs to contain zero anomalies. This was important to include in synthetic data, to verify that the method was not simply over-generalizing and flagging anomalies even with none present.

In this manner, the models output by this method were guaranteed to achieve sufficient complexity, and to generate an exponential distribution of unique traces.

**Experiment 1: Sensitivity**

In our first experiment, 60 models were generated under fixed , from which 1000 traces were generated separately for values of 0.5, 0.6, 0.7 and 0.9. We refer to this complete dataset as D1. For each value, the method was evaluated for values of in increments of 0.02. A large runtime bottleneck is that the method is currently implemented via script components that construct complete application processes (such as ProM) for each log and is not yet implemented in a standalone execution environment. Thus, the run-time per log was around 1 minute, and the experiment required about 6 hours total, some of which was merely result compilation. Accuracy, precision, recall, and f1-measure were averaged over the 60 test models for the cross product of and values, giving the following plots and ROC curve:

|  |  |
| --- | --- |
| XX.1 | XX.2 |
| XX.3 | XX.4 |
| XX.5 | Figure-set XX. Dataset D1, experiment 1 result plots for accuracy (1), f1-measure (2), recall (3), and precision (4). The x/y axes of all curves were oriented for best visualization, so note their orientation carefully. ROC curve (5) is shown at left. |

Notably, for all four performance curves, performance degraded only slightly along the axis for larger values, indicating the method worked well for very skewed trace distributions. Clearly, impacted performance most strongly. From the top-left, accuracy maximized around , then tapered gradually as larger values increased the false positive (FP) rate. Recall, the ability to flag all anomalies, maximized quickly for low values of ,as expected for a probabilistic anomaly detection threshold. Precision and the f1-measure were more informative, since precision clearly dominated the f1-measure compared with recall. Precision, the proportion of flagged traces that were anomalies, was likewise maximized for small values and informs us where the greatest saturation of anomalies occurred. The f1-measure results are most informative in terms of selection, suggesting one choose an value of around 0.07, with a corresponding accuracy of 96%.

Additionally, we plotted the receiver-operator characteristic (ROC) curve TPR/FPR values for all values of in 0.02 increments, averaged over all 60 models and all values of . The area under the ROC curve is clearly very near 1.0, indicating a high true-positive rate.

The results show lower values (0.04-0.10) of are preferable and can be tuned to suit recall vs. precision objectives. From a risk perspective, recall is the most important in terms of capturing all anomalies, at the expense of decreasing accuracy and precision. On the other hand, precision and f1-measure provide better comparative performance metrics, and f1-measure shows room for improvement. However, the sharp maximum of the precision curve along the axis indicates the method and the Bayesian metric worked as intended, distinguishing anomalies from regular structure with a sharp boundary, for the synthetic data parameters.

Lastly, this experiment analyzed performance over a range of and , but with fixed to 0.05. The results demonstrated that an algorithmic parameter, ,determined the results far more than the data parameter, . Thus, more experiments were required to stress properties of the data.

**Experiment 2: Sensitivity**

A second experiment analyzed the sensitivity of the method with respect to ,and with fixed. This was to verify that the previously selected did not trivialize the task of anomaly detection. In this case, , and was varied between 0.0 and 0.2 in increments of 0.02, then in increments of 0.05 between 0.25 and . This choice of yields a more uniform distribution of traces. The low range of was expected to trivialize the discovery of anomalies due to their low expected frequency, whereas the high range approached , making anomalous substructures and normal substructures ambiguous. The same models as dataset D1 were used, but new traces were generated for the new parameter values, generating dataset D2.

|  |  |
| --- | --- |
| XX.1 | XX.2 |
| XX.3 | XX.4 |
| XX.5 | Figure XX. Dataset D1, experiment 2 results for (from top-left) accuracy (1), f1-measure (2), recall (3), and precision (4). The x/y axes of all curves were oriented for best visualization, so note their orientation carefully. ROC curve is shown at left (5). |

As expected, the four metrics show performance worsening along the axis approaching 0.5, with performance diminishing rapidly above approximately 0.3. However, the decay was smooth, showing the method works satisfactorily for a range of very rare and somewhat frequent anomaly occurrence rates, with respect to a somewhat regular process. The ROC curve bears this out, and likely anticipates expected performance on real world data, for which  and are not known in advance.

Experiments 1 and 2 were performed with a different anomaly characteristic. Whereas the previous datasets D1 and D2 included insertion, substitution and deletion anomalies, they excluded a specific type of substitution: anomalous structure consisting of existing model activities. Such anomalies represent activities occurring out of context. The same two experiments were performed on the same data, but with the anomalous activities in each model replaced by an existing activity randomly chosen from the model’s non-anomalous activities. New traces were generated as for D1, 60 models and 1000 traces for , giving dataset D3. Dataset D4 was generated likewise but for various values of , as per experiment 2.

|  |  |
| --- | --- |
| XX.1 | XX.2 |
| XX.3 | XX.4 |
| XX.5 | Figure-set XX.x. Dataset D2, experiment 1 results for (from top-left) accuracy (1), recall (2), precision (3), and f1-measure. The x/y axes of all curves were oriented for best visualization, so note their orientation carefully. ROC curve is shown at left (5). |

Experiment 1 results for D3 are shown above. Experiment 2 results are in the figures below for dataset D4.

|  |  |
| --- | --- |
| XX.1 | XX.2 |
| XX.3 | XX.4 |
| XX.5 | Figure-set XX.x. Performance results for D4, experiment 2 over a range of theta-anomaly values for (from top-left) accuracy (1), f1-measure (2), recall (3), and precision (4). Again note that the horizontal x/y axes have been oriented to improve visualization. ROC curve is shown at left (5). |

As shown, the curvature of the four performance metrics showed no significant changes, but overall performance decreased measurably, most clearly demonstrated by the respective ROC curves. The decrease in performance was due to the less obvious nature of anomalies when replaced by existing activities.

**Experiment 3: Evaluation of Multiple Anomalous Structures**

The previous experiments analyzed performance sensitivity with respect to and , but dealt with models for which the prior number of anomalous structures was distributed according to the model generation algorithm. As a result, most models contained between 0 and 2 anomalous structures by which to generate anomalous traces. These model classes are denoted as , where *k* defines the number of anomalous structures in model *M*. Despite that larger values of in experiment 2 generated more anomalous traces, they were generated from models containing only a low number of anomalous structures by which to generate them. Therefore, a remaining task was to analyze performance sensitivity to a controlled range of .

This required generating D5, a final model-oriented dataset consisting of models with *k* anomalous structures for . For each *k*, 30 models were generated each with a single log generated from fixed trace parameters: and . These values were not truly fixed, since the outgoing edge probabilities of all vertices were normalized. For example, while the model generator might generate an activity vertex with two equiprobable outgoing “OR” edges, it might also add an anomalous edge to this vertex, as described later. This vertex’ outgoing edge probabilities would then be, which do not sum to 1.0, so the edge probabilities at each activity vertex were simply normalized. An example model with 16 anomalies is shown below in figure [XX].



Figure : A synthetically-generated model with 16 anomalies (yellow edges) and ~40 activities overall (red vertices).

Models were generated using the same procedure as for D1 and D2 until one satisfied the target number of anomalous structures. Models with were very unlikely under the previous model generation parameters, so in this case anomalous structures were manually added until the target was reached. To achieve a nearly uniform distribution of insertion, deletion, and substitution anomaly structures, this process used a stochastic method of its own, using the decision tree shown below in figure [XX].



Figure : The anomalous structure generation decision tree (top). At bottom, the three types of anomalous structures added. Null transitions '^' represent execution paths which bypass (delete) normal behavior. LOOP structures transition and return from a newly inserted activity, creating an insertion. Lastly, OR branches decompose to two types os substitution, whereby normal behavior is replaced by either some existing activity () or a new activity ().

With uniform probability, the method chose to add an OR, LOOP, or null-transition anomalous structure. A null transition consisted of an edge from a randomly selected vertex to some downstream vertex, resulting in a deletion anomaly. A LOOP was inserted into the model by adding edges between a new activity vertex and a randomly selected vertex, creating an insertion anomaly. For OR, a new edge was added to a randomly selected vertex, rejoining the model at a random downstream vertex. However, along this OR edge, a single vertex was inserted which was either a new activity or a randomly selected existing (non-anomalous) activity. Both cases generate substitution anomalies, but the former adds a new activity that might be easily detectable, whereas the latter adds ambiguity by using an existing activity. Adding anomalous structures via this stochastic decision tree resulted in models with a fixed number of *k* anomalies, each of which was selected to achieve a nearly uniform distribution of insertion, deletion, and substitution anomalies.

Given dataset D5, generated by the procedure described above, the method was re-run and evaluated over the cross product of values in terms of accuracy, recall, precision, and f1-measure, averaged over thirty models/logs. Here and in increments of 0.02. The results are shown below.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | Figure-set XX.x. Performance results for D3, experiment 3 over a range of k-anomaly values for (from top-left) accuracy (1), f1-measure (2), recall (3), and precision (4). These visuals were oriented to convey the curvature of each metric, so attention must be paid to the orientation of the x/y axes. The *k* parameter is given by the axes labeled “Anomalous Structures.” ROC curve is shown at left (5). |

Overall, these results are most like experiment 1, since and were fixed, and experiment 1 showed little variance over the range of . The overall curvature shows accuracy decaying in linear fashion for larger *k,* due to the larger numbers of anomalous traces in the logs. Again, recall quickly maximized since small values detected most anomalies. Precision remained high across a range of *k* anomalous structures despite decreasing accuracy, since for larger *k* values the set of flagged traces became saturated with real anomalies simply because there were many more in the data. The ROC curve bears out the overall performance, which did not suffer substantially compared with the previous experiments.

In sum, experiment 3 results show that the method could handle lots of anomalous structures, and subsequently many anomalous traces. However, the method arguably benefited from the ease of finding anomalous structures when more anomalous structures were added, but those anomalies arose from separate-but-rare events. In some ways this benefits results for larger *k* by simply adding more of the anomalous class to the data, but under a distribution that remained favorable to finding them. Hence there could be more anomalies to find, but without substantially decreasing the probability of finding them. By increasing we can expect worsening performance for larger k, as experiment 2 demonstrated. This shows that has the greatest negative impact on the performance of our algorithm, which is to be expected since larger values essentially make anomalies indistinguishable from normal behavior and noise.

**Comparison with Existing Methods**

To provide context for these results, we tested the sampling algorithm from (Bezerra et al, 2013), upon which our data generation method was based. The authors reported their best results using this algorithm with optimized parameters, using similar synthetic data and anomalies. The sampling algorithm is defined as:

|  |  |
| --- | --- |
| **Algorithm 3: Sampling Algorithm** | |
|  | **Input** A log, *L*  Sampling proportion,  Mining algorithm, *mine,* outputting a process model |
|  | **Output** *TA*, the set of traces flagged as anomalous |
| 1. | T = set of all unique traces from the log *L* |
| 2. | TC = { } #used to contain anomalous candidate traces |
| 3. | *TA* = { } #used to contain traces flagged as anomalous |
| 4. | **for** **do:** |
| 5. | if then: |
| 6. |  |
| 7. | **for**  **do:** |
| 8. | *S* = sample of *s%* of traces of *L* |
| 9. | *M = mine(S)* |
| 10. | if t is not replayable on M: |
| 11. | *TA* += t |
| 12. | **return** *TA* |

To detect anomalies, the sampling algorithm expects that an anomalous trace will deviate significantly from the expected process model derived from a randomly selected subset of the traces. Anomalous traces are defined as outliers that are also inconsistent with the expected behavior of a log. To exploit this property, the sampling algorithm begins by gathering low-frequency outlier traces from a log. For each trace in this set, a process model is mined from a randomly selected subset of all the traces. The trace is added to the anomalous trace set if it is not replayable on the mined model. In contrast with our method, the sampling algorithm flags anomalies without providing causal, structural context for the flag, although additional processing could provide it.

To test the sampling algorithm, we implemented and ran it on datasets D1 and D2, for using a frequency threshold of 0.02 and a sampling rate of 0.7, parameters suggested by the authors’ highest performance results. Our goal was only to derive a straightforward performance baseline with which to compare our results, and we did not exhaustively test over a range of values. Exhaustive testing was difficult because of implementation dependencies on ProM [22], which required significant test run time. This was not a fault of the sampling algorithm nor of ProM, but of our system’s integration with ProM’s command line features. As a result, we only tested and averaged performance over 30 0f the 60 models, which is still a confident set of models. Results for D1 and D2 were virtually identical, so only D1 are shown for brevity.

|  |  |
| --- | --- |
| XX.1 | XX.2 |
| XX.3 | XX.4 |
| Figure-set XX.x: From top-left, dataset D1 Sample Algorithm results, accuracy (1), f1-measure (2), recall (3), and precision (4) for . | |

Table 1 below compares performance of our method (Algorithm 2) with a generic value of , beside the sampling algorithm (Algorithm 3), for dataset D1. Results were simply averaged over all values under test.

|  |  |  |  |
| --- | --- | --- | --- |
| Table 1: Dataset D1 Results | | | |
| Algorithm | Algorithm 2 | Algorithm 3 |  |
| Accuracy | 0.972 | 0.717 | +0.255 |
| Recall | 0.892 | 0.924 | -0.032 |
| Precision | 0.700 | 0.186 | +0.514 |
| F1-Measure | 0.721 | 0.233 | +0.488 |

As shown in Table 1, the sampling algorithm performed well, but below our method for all metrics except recall. The performance discrepancies are attributable to differences in their objectives: our method targets anomalous behavior in the context of regular behavior with greater precision, whereas the sampling algorithm is concerned with detecting deviation with respect to expected model structure. Its authors described high recall as a primary performance objective, which is justified by these results and explains the lowe precision. The sampling algorithm’s frequency threshold and sampling rate were the best values reported by the authors, but could be optimized to improve performance on this data.

**Real Data Evaluation**

For real system data evaluation, we applied our method to a dataset consisting of 2,566 traces representing activity-sequences of code function calls made by software unit-tests of the NASA Crew Exploration Vehicle (CEV) [33]. The CEV system implements the complete UML-design of a structured aerospace mission model, and the unit-test suite executes components of this implementation. Thus, the test data provided a description of called components and code paths, albeit triggered by unit test calls, providing a context to use our pattern mining and anomaly detection method to evaluate discrepancies between system design and behavior, and to detect unusual code executions.

Unit-testing emphasizes code coverage, which entails repetitive software component calls and extreme value testing. Hence the distribution of these traces models the test design, whereas traces representing normal system operation are more desirable. Nonetheless, the dataset comprised a normative view of the system from the design perspective, and provided a suitable demonstration of the method’s model checking and anomaly detection potential. The demonstration is unsupervised, since the traces in this dataset were not labeled as anomalous. We are instead claiming that the findings of the method are ‘anomalous’ per unit test design or system behavior, in terms of unusual behavior in the context of normative patterns. We tracked the number of anomalies detected various Bayesian thresholds, shown below in Table 1. The total run time for these results was approximately two minutes.

|  |  |
| --- | --- |
|  |  |
| Figure XX: The mined process model of function calls, clearly very connected in terms of core functionality. | Figure XX: The mined dendrogram of 33 substructures. Each red node represents a subset of blue nodes/edges from the model at left. |
|  |  |
| Figure XYZ: A highly compressing substructure, SUB2, demonstrating the encapsulated structure of rocket burn code. | Figure ZZZ: SUB27, an anomaly including a call to an error function amidst critical calls to a rocket burn function. |

|  |  |  |
| --- | --- | --- |
| Table 1: NASA CEV test suite results | | |
| Number of traces: 2,566 | | |
| Number of activities: 34 | | |
|  | Anomalies detected | % of log |
| 0.01 | 18 | 0.7 |
| 0.03 | 183 | 7.1 |
| 0.05 | 966 | 37.6 |
| 0.07 | 1,348 | 52.5 |
| 0.09 | 1,620 | 63.1 |

The results revealed that from the synthetic data experiments did not generalize well for this log, flagging over half of the traces in this data. This was due to an extremely heavy tailed trace distribution, with many low frequency outliers. Evidently this was because the data consisted of white-box unit tests, rather than data reflecting the distribution of function calls during normal system operation. By reducing in increments of 0.02, we reduced the subset of anomalous traces to those of more critical importance; such as substructure 27 (figure ZZZ), which exhibited error behavior by an error-log call between rocket burn calls. Conversely, the most compressing patterns, such as figure XYZ, represented well-encapsulated code components.

Such structures allow system evaluators to identify exception-like behavior unintentionally included in a system, a critical task in software model verification. For instance, “anomalies” and normative patterns provide useful insights into the distribution of risk across code regions, by identifying “spaghetti” code in need of more rigorous testing, refactoring, or code interfaces straying from design. Non-anomalous substructures, such as figure XYZ, identify reusable components defined by recurring subgraphs of function calls (frequent code paths), hence, high-cohesion. Loosely, the graphical similarity of substructures with respect to design components lends insight into code quality (high similarity, strong cohesion), or the need to refactor (low similarity, poor cohesion). In summary, this data demonstrates how the method can be used as a grey-box verification aid, providing insight into system behavior that is overlooked by granular white- and black-box test perspectives.

**SECTION FOUR: CONCLUSIONS AND FUTURE WORK**

**Conclusions and Future Work**

Overall, the results indicate our method succeeds for a range of model complexity, trace complexity, and anomaly characteristics. The requirement is that real-world data contains enough traces and sufficient regularity to discover regular graphical patterns. Additionally, the method is tunable via the parameter to suit different datasets or performance objectives. Another notable advantage is that such an unsupervised approach requires no prior process model, nor exceptional tuning to derive normative patterns. This makes it an extensible analysis tool when applied to processes with no prior definition or pre-defined policy. Such scenarios occur frequently for computer networks, distributed systems, and communication protocols, for which detecting anomalies in system behavior is crucial. A final advantage is that the method is capable not only of flagging anomalous traces, but also of causally identifying anomalous features.

The drawback to this method is its noise intolerance, a recurring problem faced by mining algorithms. The graphical patterns discovered by SUBDUE become the only patterns by which the log is recompressed, such that even small deviations to a normative pattern are ignored and may later be flagged as anomalies. This strongly discriminatory property is in fact why the method works as desired, and the high dimensionality of graphical data typically requires such heuristics. However, a frequent goal of process mining is to create a more noise-tolerant balance between specificity and generality. Essentially, the substructure decomposition of our approach is both a strength and a potential criticism, yielding the recurring process-mining discussion on specificity-generalization tradeoffs. Future work lies in making the approach more noise tolerant, much like the GBAD system determines acceptable deviations in the local context of a normative pattern using graph distance metrics.

A final detraction is that the method exemplifies a “purpose-built” approach, designed from the perspective of searching over discrete graphical representations. The use of the Bayesian anomaly detection metric also incorporates foreknowledge of expected data characteristics. In short, although the method is unsupervised, its design was heavily tailored to its intended purposes. However, emerging graphical deep learning models, such as auto-encoders, can encode normative patterns in real-valued parameters, softening the discrete problems mentioned prior [CITE A BUNCH: process encoder work, deep walk, graph2vec, etc]. These models generally require less hand tuning and fewer hyper-parameters, a primary objective of deep learning. From a theoretical perspective, the most promising work lies in adaptating such learning models to process mining, since they can potentially perform end-to-end normative pattern mining and anomaly detection without as much engineering.

REFERENCES

[123] I. Jonyer, D. J. Cook, and L. B. Holder, Discovery and Evaluation of Graph-Based Hierarchical Conceptual Clusters, Journal of Machine Learning Research, 2, pages 19-43, 2001.

[124] I. Jonyer, L. B. Holder, and D. J. Cook, Hierarchical Conceptual Structural Clustering, International Journal on Artificial Intelligence Tools, 10(1-2), pages 107-136, 2001.

[125] I. Jonyer, L. B. Holder, and D. J. Cook, Graph-Based Hierarchical Conceptual Clustering in Structural Databases['](http://hydra/subdue/publications/jonyer_aaai2000sa.ps)', In the Proceedings of the Seventeenth National Conference on Artificial Intelligence, 2000.

[126] I. Jonyer, L. B. Holder, and D. J. Cook, Graph-Based Hierarchical Conceptual Clustering, In the Proceedings of the Thirteenth Annual Florida AI Research Symposium, 2000.

Esparza, J., & Nielsen, M. (1994). Decidability issues for Petri nets. *Petri nets newsletter*, *94*, 5-23.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541-580.

APPENDIX

**I. Experiment One Results and Visuals**

This section contains the full results and visuals for experiment one. Some of the results are reproduced from the previous section on experiment one, but others are included as well, such as false positive, true positive, false negative, and true negative counts.

[Begin the body of your first chapter here. There is a variety of methods for dividing your dissertation. You might use chapters, sections, or manuscript numbers. Please work in close consultation with your committee chair to determine the most appropriate means of organizing your dissertation for your discipline. The most important thing to remember is to be consistent. Below please find a series of subsections detailing the Graduate School’s formatting requirements for various aspects of the dissertation.

**Graduate School Formatting Requirements:**

Thesis and dissertation authors must follow Graduate School formatting requirements exactly. This is particularly true of the title page, abstract, signature page, and table of contents. There is no allowance for creativity or deviation on these pages and the Graduate School will not approve theses or dissertations that deviate from the required formatting.

**More on Formatting the Title Page**

The “Degree Name” field is strictly “DOCTOR OF PHILOSOPHY” for all PhD graduates and “DOCTOR OF EDUCATION” for all EdD graduates. Only masters’ graduates include the field of study, such as “MASTER OF ARTS IN HISTORY” or “MASTER OF SCIENCE IN CHEMISTRY”.

You must use your full, official name of record on file at Washington State University in all places that require your name. This name appears on your unofficial transcripts (available at no cost via myWSU). If you wish to use a different name, you must provide the necessary legal documentation to WSU’s Payroll Office.

The “Department Name” field must include the official unit name that is granting the degree. Some examples are “Department of History,” “School of Engineering and Computer Science”, and “Department of Criminal Justice and Criminology.” If you are not sure of your department’s official name, contact your department or gradschool@wsu.edu for assistance.

**More on Formatting the Signature Page:**

This is arguably one of the most important pages to have formatted correctly BEFORE your final defense. You must secure ORIGINAL black or blue ink signatures from all committee members for submission to the Graduate School within five business days of your successful defense. The Graduate School cannot accept signed signature pages that are not formatted correctly; you must be sure that this page is correct before taking it to your examination for signatures. It is your responsibility to ensure your signature page is numbered as page ii and that it matches the example in this template EXACTLY.

**Final Document Submission:**

Following your successful defense, you must submit a physical copy of the abstract, title page, and signature page printed on 100% cotton paper to the Graduate School. Along with these 100% cotton pages, you must also submit a signed Hold Harmless/Copyright Acknowledgment Form. Doctoral students are also responsible for submitting a completed Survey of Earned Doctorates. Your 100% cotton pages are filed for permanent retention with Manuscripts, Archives, and Special Collections (MASC) after cataloging with WSU Libraries is complete.

CHAPTER TWO: METHODOLOGY

**Chapter Headings:**

The Graduate School requires that new chapters always begin at the top of a new page within the thesis or dissertation, regardless of where text ends on the previous page.

**Professionalism and Consistency:**

Authors should use the formatting and citation style guide that is appropriate for their discipline for the body of the thesis or dissertation. Whenever there is a formatting conflict between Graduate School requirements and a style guide, authors must adhere to the Graduate School’s requirements. In the body of the thesis or dissertation, there are two primary formatting considerations to keep in mind: professionalism and consistency. For example, it is perfectly acceptable to bold your subheadings (as in this template) if you treat all subheadings in the entire dissertation in the same manner. You cannot bold or underline some subheadings and not others. In all cases, be certain that your formatting decisions convey a sense of professionalism and always be consistent in how you apply your decisions.

**Page Margins:**

All pages must have at least a 1” margin with a 1.2” margin on the bottom of the page. The larger bottom margin allows you to have ½” of clear space above and below the page number. This is mandatory – please ensure that no text, charts, graphs, images, etc. infringe upon your document’s margins.

**Font Requirements:**

The Graduate School requires that dissertations bear a professional appearance to the reader. While we do not enforce a set font or font size requirement, your choices must align with the following three principles:

1. You must use ONE font in your dissertation. Wherever possible, use the same font in your graphs, tables, and charts, but there is some leeway given if this is not possible.
2. Your font and font size choices must look professional. The Graduate School will instruct students to change their font or font size if the reviewer feels these elements are not in keeping with the professional appearance requirement. In all cases, dissertations must be in black font.
3. You must be consistent. Should you decide to increase the font size of your chapter headings from 12 (the setting in this template) to 14, you must be sure to do so for ALL chapter headings in the dissertation. This principle applies to the use of font effects (**bold**, underlining, *italics*, etc.) as well.

**Page Numbers:**

This template is designed to help you with the most challenging aspects of pagination. Please do not edit these settings. If you need to adjust the pagination, keep the following requirements in mind. Page numbers must have at least ½” of cleared space on all sides of the number. Page numbers must either be centered at the bottom of the page or in the upper right-hand corner. If you decide to move your page numbers to the upper right hand corner in this template, you need to reduce the bottom margin back to 1” and increase the top margin to 1.2” to allow for enough space around your page numbers.

Page numbers must be in the same font as the rest of your dissertation. You may edit the font size to be slightly smaller if desired. Each page must be assigned a page number with the exceptions of a half-title page (see an example half-title page later in this template). Half-title pages have a number counted for them; however, the number is not visible to the reader. Front matter pages (signature page, abstract, table of contents, etc.) must be numbered using lower case roman numerals (ii, iii, iv) whereas all body pages, beginning with the first page of the introduction, must be numbered with Arabic numerals (1, 2, 3).

**Blank Pages and Running Headers**

The Graduate School does not permit students to have blank pages or running headers within their dissertation. Please ensure that you do not include these in your document.

**Block Quotes:**

Use a single or double-space for block quotes. Remember; always be consistent with your formatting choices!

**Handwriting:**

Handwriting is not acceptable within a thesis or dissertation. Any graphics, drawings, figure labels, etc. must be generated by computer, typewriter, Kroy lettering, or a professional draftsperson.

**Hyperlinks:**

Anytime you include a website, such as www.gradschool.wsu.edu, you need to deactivate the hyperlinks in your text. Your dissertation needs to be viewable today, next week, next month, next year, in twenty years, and beyond. In the long term, hyperlinks will no longer point to anything and will not provide much meaning for readers. The Graduate School *suggests* you incorporate any such referenced material into the body of your dissertation, but we *require* that you deactivate the hyperlink. In Word, right-click the link and select “Remove Hyperlink”.

CHAPTER THREE: ANALYSIS

**Landscape Pages:**

Sometimes, it becomes preferential to insert a landscape page into your dissertation. In these cases, the page number must be reoriented so, when printed, it appears in the same location as the rest of your page numbers. For instructions on how to make this change, please proceed to the next page.

This is a pre-formatted, example landscape page. There is a variety of ways to format these page numbers. This template used the directions found here (<http://guides.lib.umich.edu/c.php?g=283073&p=1886009>) to create this example, but if you are having difficulty, you could also use <http://support.microsoft.com/?kbid=211930>. Regardless of the method used to create your landscape pages, they must follow this example in formatting.

APPENDIX

<*This is a sample half-title page>*

*<Half-title pages have a page number counted, but it is not displayed>*

**Example Image:**

Images, charts, and graphs must not enter the mandatory 1” margins. Captions for these elements may be either single or double-spaced so long as you are consistent. Additionally, be sure that your reader can easily tell the difference between your caption and the body text of your dissertation.



Example caption: Captions for figures and charts may be single or double-spaced. Remember, be consistent and format all captions in the same manner.

REFERENCES

Authors may use whichever style guide is appropriate for their field. Remember to be consistent and use the same citation style throughout the dissertation. Additionally, you may either place references at the end of each chapter or at the end of the document. In all cases, references pages always start at the top of their own page and the page heading must be formatted in the same manner as a chapter title.

**Additional Resources**

For additional help with thesis and dissertation formatting, please see the Graduate School’s forms website, available at https://gradschool.wsu.edu/facultystaff-resources/18-2/. The “Thesis and Dissertation Formatting and Submission Requirements” provides more in-depth explanations of thesis/dissertation formatting as well as instructions and checklists to help you through your final semester at WSU.

If you need help with content, clarity, form, flow, organization, or writing, contact the Graduate Professional Writing Center at gpwc@wsu.edu. If you need help with grammar, proofreading, or copy editing, contact the Professional Editing and Service Center at prof.edit@wsu.edu. Finally, feel free to direct any questions regarding formatting to gradschool@wsu.edu.