-Axes labels of variance, etc

-theta trace: competition between information (from uniform trace distribution) and theta\_trace, …. ??

-noise

-higher value of theta\_anomaly\_trace

-if modify anomaly threshold do we need to change bayes to find it. (plot alpha bayes and theta\_anomaly\_trace). The question is the generalization of the bayes threshold.

-Definition of anomaly: review

-replace anomalous activities with ones elsewhere in the model

-need definition of” anomaly”

Abstract

Process mining of unstructured, noisy environments remains an important area of security and process mining research. This paper discusses an unsupervised, threshold-based method of anomaly detection and process mining using the iterative SUBDUE graph-compression method and the Inductive Miner. We provide an overview of process mining definitions and existing approaches, then evaluate the method on synthetic data. The method generates a taxonomical representation of compressing structural features of a workflow log as a dendrogram, by which further analysis can be performed. We demonstrate one such use by performing anomaly detection with this feature representation of a process log, using a Bayesian threshold to detect unusual substructure in the context of normal behavior, thereby identifying unusual process executions. This method provides a framework for modeling and monitoring unstructured and noisy organizational, technological, and natural processes.

Introduction

As described in [1], a Process Aware Information System 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 are both aware of process data and likewise prescribe tasks and activities via process models. Such systems incorporate process data and process models in a loop by which processes can be defined, tracked, and evaluated via process models; likewise, process models may be derived and analyzed from process data.

However, institutions often rely on overlapping and non-interoperable systems and tribal knowledge to monitor and control processes, making the formal requirements of a PAIS become infeasible or intrusive. This paper focuses on contexts where a PAIS is instead an abstraction consuming process data derived from multiple systems, and from which process models are mined and analyzed. This is amenable to realistic scenarios in which processes execute in an embedded and non-stationary framework of changing people, tools, resources, and institutional knowledge, in the absence of prescribed process models. These scenarios occur frequently since modeling such environments often occurs only due to some ad hoc objective, such as an audit or root-cause analysis. In such scenarios, a PAIS is a collection of operational systems and disparate data sources by which one derives traces characterizing the underlying process-oriented view of a process and its activities.

The ability to mine and analyze normative process patterns in these unstructured contexts is critical for extracting regular activity and detecting irregular activity. Detecting anomalous activity requires prior normative activity models, thus detection and normative pattern mining are complementary tasks. For this, we present a method for mining process patterns from workflow logs with useful anomaly detection properties. We use the Inductive Miner [2] to construct a graphical process model from log data, then apply the SUBDUE [3] graph compression method to iteratively extract a hierarchical dendrogram of normative patterns of this model. In subsequent post-processing, anomalies and other useful features are discovered.

This hybrid approach is useful since the Inductive Miner extracts generality from process log data, outputting a graphical model *M* capable of generating all traces in a process log, regardless of noise in the input log. SUBDUE then extracts from *M* only its most informative components, constructing a hierarchy of sub-structures of *M* most relevant to the log as a dendrogram. In short, for a given trace log, the Inductive Miner is used to mine a general process model, SUBDUE is then used to extract only the most specific components of this model. Using this unsupervised method, one can mine normative process patterns, detect anomalies to those patterns, and perform other analyses.

This captures the spirit and contribution of our method, but requires some definitions. From the control-flow perspective, common process mining terms can be framed in a familiar graphical manner specific to this work:

* 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].
* Process grammar: Recursively-defined constructs for common process patterns. For example, AND-SPLIT is a set of edges branching from a single node and traversing activities in parallel before synchronizing at some later activity or activities. Other basic constructs include OR-SPLIT, XOR-SPLIT, LOOP, and JOIN [14] [15].
* Workflow trace: A single execution of a process as a partially-ordered sequence of activities, following any valid path from a START node to an END 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.
* Process miner: Any algorithm for constructing a process model from a workflow log, usually incorporating criteria for 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 but possibly additional behavior.
* Partial-order property [18]: A property of workflow traces whereby activities may be randomly-ordered with respect to parallel activities. ‘ABCD’ and ‘ACBD’ might be workflow traces from some model, where ‘C’ and ‘B’ are parallel sub-processes, and may recursively embody further parallel sub-processes, and ‘A’ always occurs before ‘D’. The primary task of process mining algorithms is disambiguating the partial-orderings, applying rules and heuristics to generate models with desired properties of complexity, specificity, and generality. The enormous search space of possible graphical models defined over a set of partially-ordered traces is the primary challenge for these algorithms.
* 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 for input to SUBDUE [2].
* SUBDUE: 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 [3].
* 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.

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

Previous Work

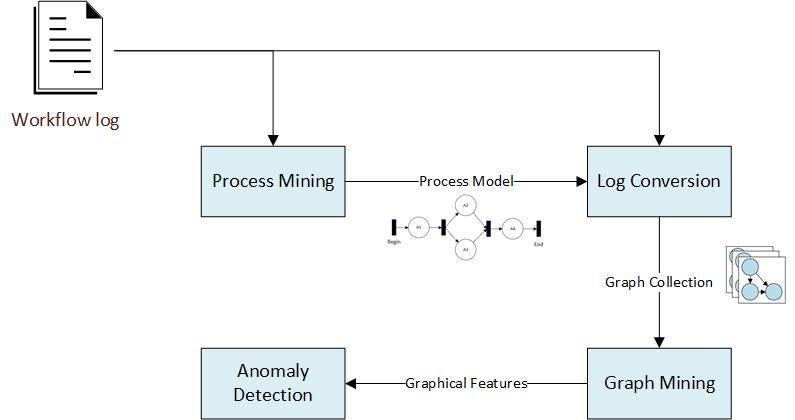
The SUBDUE graph-compression method accommodates any context of mining graph data for normative patterns, and hence works as a graphical feature detector. It was previously used for knowledge representation systems [1], and more recently in security applications for intrusion detection [8]. Using SUBDUE as a process mining tool has been 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.

GBAD formalized SUBDUE’s 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 incorporating three separate 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 detected based on a discriminative threshold. Bezerra’s work examined anomaly detection using several threshold-based approaches within the process mining algorithm itself [11]. Bezerra decomposed this family of process-based anomaly detection into three groups: threshold-based, iterative, and sampling. Our approach does not fit squarely into one these categories since it is compression 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 deeper structural insights into normative patterns and anomalous features. Our work replicates Bezerra’s data generation scheme, but otherwise appends a new method to this work.

The Method

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 such a log to a collection of subgraphs via a mined process model, extracting descriptive normative graphical patterns of the log (with respect to this model) as a dendrogram, and lastly detecting outliers and anomalous behavior.



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

The workflow extends the discovery of overall process characteristics irrespective of prior constraints, such as a prescribed process model. Our approach encompasses more realistic and informal “spaghetti” model scenarios in which processes are organically-defined and highly unstructured: enterprises, communication networks, criminal/fraudulent networks, natural processes, and so on.

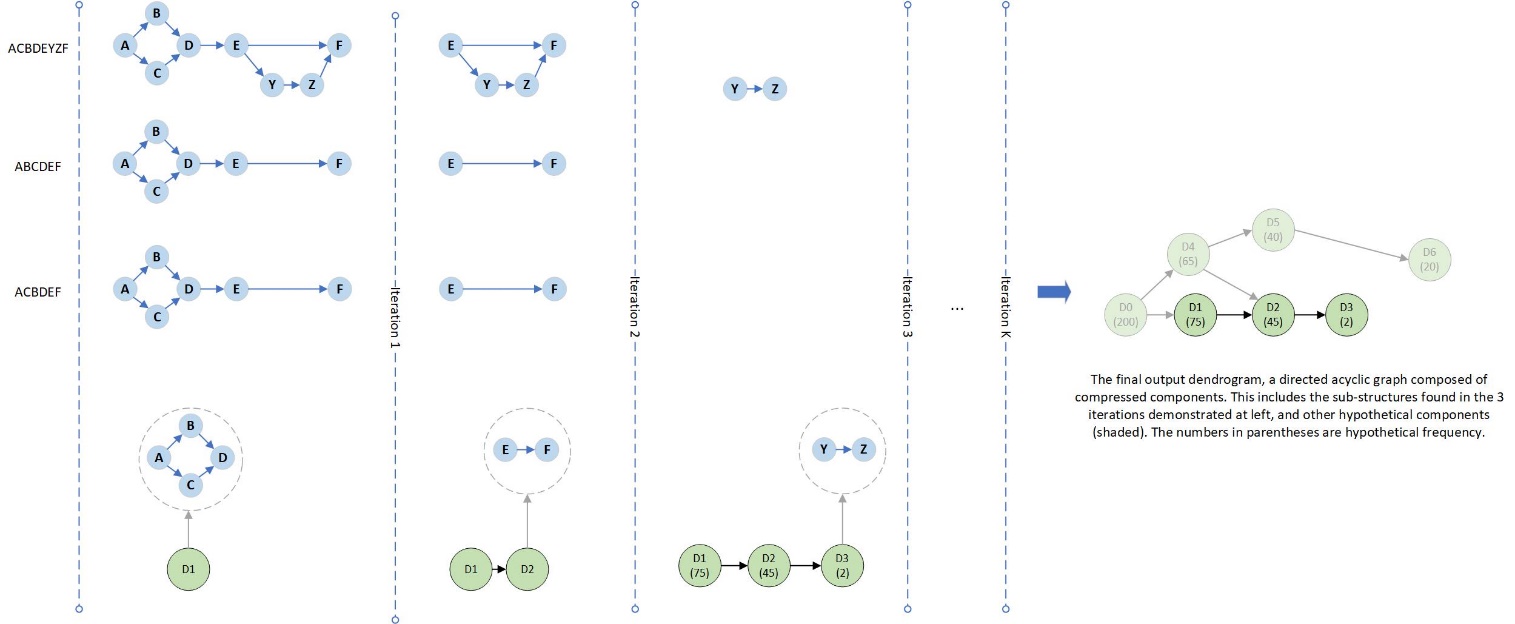
Using Graph Compression to Discover Patterns and Cluster Traces

SUBDUE discovers highly compressing patterns in graph data via the minimum description length (mdl) principle and a beam search over candidate subgraphs. This satisfies the requirement for an unsupervised method of discovering the most 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 to discover meaningful graphical patterns.

Prior work showed strong anomaly-detection results when running SUBDUE iteratively on a set of graphs [8]. At each iteration, the most compressing subgraph discovered by SUBDUE was used to replace all such instances with a single node, and the method 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 modelled their anomaly-detection activities.

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 for 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 a 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, but failing to reach the further reaches where compressing structure decays, and where anomalies often lie.

Since the requirement was to force SUBDUE to analyze new regions, the remedy was to simply delete all instances of the most-compressing subgraph from the traces at each iteration. This effectively forces SUBDUE to discover regularity in new regions at each iteration, and thus to discover dissimilar graphical features, compressing away graphical features of decreasing information content.



The approach loosely resembles data dimensionality reduction, which compresses data 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. The dendrogram comprises the entire behavior of the log, with the ancestral components reflecting the most relevant graphical features of the log. This is amenable to anomaly detection since the less compressing a feature is, the more deviation it represents with respect to normative patterns and normal overall behavior, and hence will occur “lower” in the dendrogram.

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

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

**Input** *miner*: 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. until 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 the traces are converted to a collection of graphs. This collection is iteratively fed to SUBDUE to find the most compressing substructure, which is appended to the dendrogram, then deleted from all traces in which it occurs. This repeats until no further progress can be made, and all traces have been compressed to their most elementary substructures. The dendrogram is returned, whose vertices represent compressed substructures, and whose edges represent trace-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 common process features, redundant behavior, outliers, anomalies, and so on. Diamintini *et al* [13] have successfully detailed the 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, and anomaly detection is only one of many applications of the dendrogram. 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 the log. Using this information, the process model returned by the Inductive Miner could be reduced for greater specificity. For instance, the underlying properties of an unstructured institution could be discovered, and thereby the process could be identified, measured, and improved via business-process formalisms.

In this regard, coupling SUBDUE with the generalization feature of the Inductive Miner lends an extensible framework for more 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 amongst business processes. In short, the dendrogram extends to a range of pattern mining and other enterprise uses, beyond the scope of anomaly detection.

Anomaly Detection Method

Anomaly detection lends 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 the lower-frequency components. As a result, the size of the dendrogram components decreases smoothly, then drops suddenly, such that the only remaining traces/subgraphs are those representing anomalies, outliers, or log noise.

This property is useful for anomaly detection objectives since many discriminating metrics could be devised to exploit it. Given that anomalies occur in the context of regular structures, the anomalous structures tend to have sharply lower frequency than their parent substructures in the dendrogram. Further, they are also distinguished from noise in the input log, since noise tends to result in poorer structural decomposition of a trace, and as such, substructures characterized by noise and their parents tend to have lower frequency. Hence, finding anomalies resolves to finding the sharp boundaries between high frequency substructures and very low-frequency substructures adjacent to them.

A Bayesian metric was selected to capture this property, based on the frequency of substructures and their parent (immediate ancestor) substructures in the dendrogram. Under this model, each substructure is assigned a Bayesian probability defined as

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 its argument. Characterizing is trickier, as it 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 joint set of independent events (as substructures are independent), with each event weighted by its likelihood :

The repeated looks unusual, but is the weight of each parent’s contribution to the overall probability, as required to obtain a proper probability distribution. Substituting ‘c’ for child, ‘p’ for ‘parent’, and ‘P’ for ‘parents’, the fully-defined metric becomes:

An awkward looking probability, but nonetheless the properly weighted result from defining the Bayesian interpretation for the probability of a child given its parents. Anomalous substructures are expected to have a low value for , and thus any substructure is flagged as anomalous when , where is the anomaly threshold. All traces containing the anomalous substructure are then flagged as well, as follows:

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

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

bayesThreshold: An anomaly detection threshold

**Output** *anomalies*: A set of trace-id’s identified 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

Algorithm Evaluation

Although real process-oriented datasets are available, they do not offer the controlled conditions sufficient to compare the characteristics of different algorithms. Instead we used a synthetic data generation algorithm as found in appendix A of [11], modified slightly to generate data directly from probability distributions embedded in 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. Likewise, to cohere to a stable performance baseline, we also used the same model generation parameters as described in Bezerra: 60 randomly-generated process models, 1000 traces [11].

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 from including the null transitions in the set of “activities”, and as such the split constructs may divert to more than two activity paths, may bypass parts of a model, and so on. The parameters constrain model complexity to a probability distribution over these operators, replicated from Bezerra [CITE] 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 the models contained a minimum START to END path length of 1, maximum of 1 anomalous structures, maximum of 4 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 determine trace diversity, from very uniform to very non-uniform. A value of implies the trace-generation scheme has a 0.9 probability of taking one branch and a probability of taking its alternative. Thus, by varying from very uniform (0.5) to very non-uniform, one derives a less uniform distribution of traces, making anomaly detection more difficult. To ensure maximum partial-order entropy, activities lying ambiguously within the same timestep were shuffled to achieve uniform random partial order.

Lastly, the parameters defined the probability of generating anomalies. These encompassed both the generation of anomalous model structure 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 more frequent behavior. LOOP and OR constructs were marked as anomalous with fixed probability 0.3. Anomalous paths were marked with traversal probability that was experimentally varied between 0.0 and 0.2 and evaluated 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 no anomalies. This is important to include in synthetic data, to verify that a method is not simply over-generalizing and flagging anomalies when none exist.

In this manner, the models output by this method were guaranteed to achieve sufficient complexity and to generate an exponential distribution of unique traces [COULD INSERT TRACE-DIST VISUAL].

Noise Generation [obsolete?]

The model and trace generation methods are sufficient to generate spaghetti-like process data in terms of model complexity via and activity complexity via , but remain constrained to the given model. For example, under this data generation method there will be no log ambiguity as to whether anactivity occurs before another if it always lies upstream in the paths contained in the model; hence, the underlying model strongly constrains the generated data. Although likely unusual, one might expect unstructured work environments to inevitable lead to occasional repetitions, whereby sub-components of some model are repeated or activities do not always follow one another. Such a scenario occurs, for instance, if prior to some software release a bug is detected after final testing, causing components of the testing activities to be repeated arbitrarily with respect to the software development process.

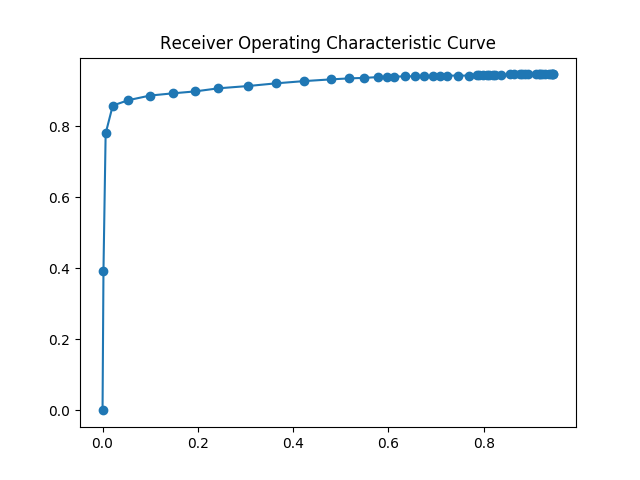
The simplest way to simulate worst-case noise is to randomly inject activities into the traces after trace generation. For every activity in each trace, with probability a random activity is inserted. The chosen activity is selected at uniform random from the set of all activities defined over the log. This method of noise addition is necessary to test our approach’s resilience to noise, and is justified since it creates many data outliers that would be too easily declared anomalous by an anomaly detection method that does not distinguish outliers from anomalies. Note that even small values of tend to create highly obfuscated partially-orderings for a process mining algorithm to mine: for a log of 1000 traces, for which the average trace length is 25, an value of 0.1 will have an expected number of “noise” activity occurrences of 0.1 \* 25 \* 1000 = 2500. This is an extreme number of random activities, since even a single misplaced activity with respect to the underlying model, breaks the logical rules and heuristics applied by process mining algorithms to mine the regular features of the underlying process model.

Experiment

To test our method, we desired datasets achieving “spaghetti” like behavior described previously. For each of 60 models generated under , 1000 traces were generated under for 0.5, 0.6, 0.7 and 0.9. For each of these, the method was run for all values of in increments of 0.02. The run-time per log was around 1 minute, and the experiment required about 6 hours total, some of which was merely result-gathering. Accuracy, precision, recall, f1-measure were then averaged over the over the 60 test models for each combined value of and , giving the following plots and ROC curve:

|  |  |
| --- | --- |
| C:\Users\jesse\AppData\Local\Microsoft\Windows\INetCache\Content.Word\accuracy.png | C:\Users\jesse\AppData\Local\Microsoft\Windows\INetCache\Content.Word\fMeasure.png |
| C:\Users\jesse\AppData\Local\Microsoft\Windows\INetCache\Content.Word\precision.png | C:\Users\jesse\AppData\Local\Microsoft\Windows\INetCache\Content.Word\recall.png |

Notably, for all four performance curves, performance degraded only slightly along the axis for higher values, indicating the method worked well for very skewed trace distributions. Instead, impacted performance most strongly. From the top-left, accuracy maximized around , then tapered gradually as higher values decreased the true-negative rate (TN). Recall, the ability to catch all anomalies, was quickly maximized for lower values of as one might expect, for larger anomaly-detection thresholds. Precision and the f-1 score are perhaps more informative, since precision clearly influenced the f-1 score much more than recall. The plots for precision and f1-measure are nearly identical since recall quickly maximizing for **>** 0.15, and subsequently precision dominated the f1-measure. The f-1 score results are most informative in terms of selection, suggesting that 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 tunable to suit performance objectives. From a risk perspective, recall is most important in terms of capturing all anomalies, at the expense of decreasing accuracy and precision. On the other hand, precision and f1-measure appear more faithful academic performance metrics. 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 dataset. As is often the case for anomaly detection, the test threshold is tunable for the desired performance objective in terms of either maximizing recall and tolerating false positives, or maximizing accuracy at the expense of a few false negatives.

The previous experiment analyzed performance over a range of and , but with fixed to 0.05.A second experiment was required to analyze the sensitivity of the result with respect to , andwith fixed. This was needed to verify that the previously fixed didn’t trivialize the task of anomaly detection. In this case, , and was varied between 0.0 and 0.2 in increments of 0.02. Qualitatively, the choice of yields a more uniform distribution of traces, and the low range of was expected to trivialize the discovery of anomalies due to their low frequency, whereas the high range approaches the normal distribution of substructures within traces. In fact, the effect of varying did little to effect the result, although precision dropped a bit. Otherwise, accuracy remained high in the reasonable range of **,** around 0.05-0.08.

Conclusions and Future Work

A notable advantage of this method is that such an unsupervised approach requires no prior process model, nor requires exceptional tuning to derive normative patterns for a process. This makes it an extensible exploratory tool when applied to process views without a prior definition or pre-defined policy. Such processes occur often in computer networks distributed systems, or in the context of communication protocols, for which detecting anomalies in the extrinsic process behavior of such systems may be a critical test case. A final advantage is that the method is capable not only of flagging anomalous traces, but also of causally identifying their unusual features.

The drawback to this method is its specialized noise-tolerance, a common problem faced by mining algorithms. SUBDUE finds graphical patterns in an unsupervised way, but these become the only patterns by which the log is further compressed; such that even small deviations to a normative pattern are ignored, and may be flagged as anomalies later. This strongly discriminatory behavior is desirable for anomaly detection. However, within process mining, the goal is sometimes to mitigate such strong discrimination, balancing a noise-tolerant tradeoff between specificity and generality. Essentially, this substructure decomposition of our approach is both a strength and a potential criticism, and yields the recurring process-mining discussion on specificity-generalization tradeoffs.

Future work lies in making the approach more noise tolerant, as the GBAD system determines acceptable deviations in the local context of a normative pattern discovered by SUBDUE using graph distance metrics. Likewise, the runtime of the algorithm may be increased by specializing the complexity of the beam search over candidate compressing substructures to be more specific to process mining.

References

[1] M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley & Sons, 2005.

[2] Leemans, S., Fahland, D., van der Aalst, W.: Discovering block-structured process

models from event logs containing infrequent behaviour. In: Business Process Man-

agement Workshops. pp. 66–78 (2013).

[3] L. B. Holder. Empirical Substructure Discovery. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 133-136, 1989.

[4] Petri, Carl Adam; Reisig, Wolfgang (2008). "Petri net". Scholarpedia. **3** (4): 6477. doi:10.4249/scholarpedia.6477.

[5] Murata, Tadao. "Petri nets: Properties, analysis and applications." *Proceedings of the IEEE* 77.4 (1989): 541-580.

[6] Eberle, W. and Holder, L., Anomaly Detection in Data Represented as Graphs, *Intelligent Data Analysis, An International Journal*, Volume 11(6), 2007.

[7] Wil M.P. van der Aalst. *Process Mining Discovery, Conformance and Enhancement of Business Processes.* Heidelburg, Germany: Springer, 2011.

[8] Noble, Caleb C., and Diane J. Cook. "Graph-based anomaly detection." *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003.

[9] Diamantini, Claudia, et al. "Pattern discovery from innovation processes." *Collaboration technologies and systems (CTS), 2013 international conference on*. IEEE, 2013.

[10] Genga, Laura, et al. "Subgraph mining for anomalous pattern discovery in event logs." (2016).

[11] Bezerra, Fábio, and Jacques Wainer. "Algorithms for anomaly detection of traces in logs of process aware information systems." *Information Systems* 38.1 (2013): 33-44.

[12] Bezerra, Fábio, Jacques Wainer, and Wil MP van der Aalst. "Anomaly detection using process mining." *Enterprise, business-process and information systems modeling* 29 (2009): 149-161.

[13] Diamantini, Claudia, Laura Genga, and Domenico Potena. "Esub: Exploration of subgraphs." *Proceedings of the BPM Demo Session* (2015): 70-74.

[14] N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. *BPM Center Report BPM-06-22* , BPMcenter.org, 2006.

[15] Control-Flow Patterns. Retrieved on September 13, 2017, from http://www.workflowpatterns.com/patterns/control/.

[16] Eberle, W., & Holder, L. (2009, April). Graph-Based Approaches to Insider Threat Detection. In *Proceedings of the 5th annual workshop on cyber security and information intelligence research: cyber security and information intelligence challenges and strategies* (p. 44). ACM.

[17] http://www.xes-standard.org/

[18] Herbst, Joachim. "A machine learning approach to workflow management." *ECML*. Vol. 1810. 2000.