A Compression-Based Method for Pattern Mining and Anomaly Detection of Graphical Process Data

Abstract

Learning structural information from stochastic, noisy environments remains an important area of process mining and graph mining. In this work, we present an unsupervised, threshold-based method of process mining and anomaly detection using the iterative SUBDUE graph-compression method and the Inductive Miner. The method generates a dendrogram of compressing structural features of a workflow log, a taxonomical representation by which further analysis can be performed. We demonstrate such a use for anomaly detection via this structured feature representation of a process log, 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. 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.

Introduction

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 paper 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 for detecting anomalies. The latter requires prior normative activity patterns, thus anomaly detection and normative pattern mining are complementary tasks. For this, we present 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], a graph compression method, 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.

In the remainder of this work, we define key terms and the problem definition, provide further background on previous graph compression approaches, provide explicit anomaly detection algorithm details, and then demonstrate the algorithm on synthetically generated data over a range of data generation parameters. Finally, we demonstrate the method on real world data derived from a NASA mission model implementation, then close with conclusions and future work.

Problem Definition

The above captures the spirit and contribution of our 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 1 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.

An in-depth overview of process mining terms and methods can be found in [1] and [7]. In this work, we are given a log , 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 .

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.

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 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]. At each iteration, instances of the most compressing subgraph were replaced with a single meta-node, 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.

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 forces 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 3: 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*

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 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. 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). Note the x- and y-axis ranges are reversed in the recall plot (3) for better visualization. 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 higher values reduced the true-negative (TN) rate. Recall, the ability to flag all anomalies, maximized quickly for lower 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. The plots for precision and f1-measure are nearly identical since recall quickly maximized for **>** 0.15, beyond which precision primarily determined the f1-measure. 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 precision 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.

Experiment 2: Sensitivity

The previous experiment analyzed performance over a range of and , but with fixed to 0.05.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 . 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 expected frequency, whereas the high range approached , making anomalous substructures and normal substructures ambiguous.

|  |  |
| --- | --- |
| 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). Note the horizontal x/y axes for recall (3) have been rotated for visualization. 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 dataset D1 included both insertion and deletion anomalies, it did not include substitution anomalies: 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 D2.

|  |  |
| --- | --- |
| 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 axes for recall (3) have been rotated for visualization. ROC curve is shown at left (5). |

Experiment 1 results for D2 are shown above. Experiment 2 results are in the figures below.

|  |  |
| --- | --- |
| XX.1 | XX.2 |
| XX.3 | XX.4 |
| XX.5 | Figure-set XX.x. Performance results for D2, 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 for recall (3) have been rotated 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, most clearly demonstrated by the ROC curves. The decrease in performance is due to the less obvious nature of anomalies when replaced by existing activities. Another contributing factor may have been the random selection of non-anomalous activities: in some cases, anomalous activities could be replaced by non-anomalous activities already very close to the anomalous structure, making anomalous and non-anomalous traces ambiguous.

Experiment 3: Multiple Anomalous Structures

The previous experiments analyzed performance sensitivity with respect to and , but dealt with models for which the number of prior anomalous structures was fixed by the model generation algorithm: the stochastically-generated models contained between 0 and 2 anomalous structures. That is, although larger values of in experiment 2 generated a greater number of anomalous traces, these 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 over anomalous structures.

For this, we generated D3, a third dataset consisting of models with *k* anomalous structures for , each with fixed trace probability. 30 models were generated for each value, hence there were 30 models, each with a single log. Trace parameters were fixed: and , and these outgoing edge probabilities were normalized at every activity vertex. For example, while the model generator might generate an activity vertex with two equiprobable outgoing “OR” edges, the model generation procedure might add an anomalous edge to this vertex, as described later. The result is that the vertex would have outgoing edge probabilities of, which do not sum to 1.0, so the edge probabilities at each activity vertex were simply normalized.

Models were generated using the same procedure as before, until they contained 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. With uniform probability, the method chose to add an OR, LOOP, or null transition structure. A null transition consisted of an edge from some 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 would be 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 vertex or a randomly selected existing (non-anomalous) vertex. The former case creates a substitution anomaly, whereas the latter creates another form of deletion. In sum, 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.

The results for this experiment show…

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 chosen 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 Testing

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.

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

[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.

[19] Song, M., Günther, C. W., & Van der Aalst, W. M. (2008, September). Trace clustering in process mining. In International Conference on Business Process Management (pp. 109-120). Springer, Berlin, Heidelberg.

[20] Akoglu, L., Tong, H., & Koutra, D. (2015). Graph based anomaly detection and description: a survey. Data Mining and Knowledge Discovery, 29(3), 626-688.

[21] Grünwald, P. (2005). A tutorial introduction to the minimum description length principle. Accessed from <https://arxiv.org/pdf/math/0406077.pdf>, December 12, 2017.

[22] ProM: The Process Mining Toolkit. Version 6.6. Retrieved from http://www.promtools.org/doku.php.

[23] M. Leemans. NASA Crew Exploration Vehicle (CEV) software event log. http://doi.org/10.4121/uuid: 60383406-ffcd-441f-aa5e-4ec763426b76, 2017