Anomaly Detection in Discrete Manufacturing Systems using Event Relationship Tables

Emil Laftchiev[1], **Xinmaio Sun**[2*], **Hoang-Anh Dau**[3*], and **Daniel Nikovski**[1]
[1]Mitsubishi Electric Research Labs
e-mails: laftchiev@merl.com, nikovski@merl.com
[2]Boston University
e-mail: xmsun@bu.edu
[3]UC Riverside
e-mail:hdau001@ucr.edu

Abstract

Anomalies in discrete manufacturing processes (DMPs) can result in reduced product quality, production delays, and physical danger to employees. It is difficult to detect anomalies in DMPs, because sequencing devices such as programmable logic controllers (PLCs) usually do not allow a process engineer to easily determine which sequences of operations are observed and checking against each sequence becomes computationally difficult. This paper proposes a new anomaly detection approach for discrete manufacturing systems. The approach models the normal behavior of the DMP from the PLC output as an event relationship table. This model is then used to determine if new sequences of PLC outputs could be generated by the system. Outputs that do not fit the learned model are labeled anomalous. This method is tested in simulation for DMPs that contain concurrent sub-process with unique or repeated events. The results are compared to a baseline method proposed in prior publications. Experiments show that the proposed algorithms are capable of achieving a higher F-score with less than 10% of the data required by the baseline method. Furthermore, this modeling approach has a linear space complexity in the length of the observed event sequences, as compared to polynomial complexity of prior work.

1 Introduction

Discrete manufacturing is a type of manufacturing focused on producing distinct items through a sequence of discrete operations. A typical example of discrete manufacturing is vehicle assembly. During vehicle assembly, each part of the vehicle is added through a series of operations (events) on the part and the car as a whole. This manufacturing process occurs at high speeds, and is in large part performed through automated actions taken by powerful machines and robots. Because of the high speed and power of the component machines, monitoring the discrete manufacturing process (DMP) is critical to ensure high product quality, minimize production delays, and ensure the safety of human workers.

[*]Work performed during internship at MERL.

Monitoring a DMP for anomalies means checking for incorrect execution of events, incorrect event ordering, and incorrect event timing. However, monitoring a DMP is difficult, because the processes are usually orchestrated by Programmable Logic Controllers (PLCs) whose instructions are programmed during design time, and are often not available later to the process engineer. Still, PLCs, due to their proximity to the actual manufacturing process, are the ideal device for deployment of anomaly detection algorithms.

This paper focuses on the development of anomaly detection algorithms for incorrect event execution order in DMPs using output data from a PLC. Our approach is to develop models of normal behavior inspired by process mining techniques [1; 2; 3; 4], such that anomalous behavior results in event sequences not observed previously. Learning models of normal behavior is an important task when monitoring DMPs, for three reasons. First, it is difficult to collect (balanced) data sets that contain examples of anomalies when the expected mean time between failures is long. Second, even data sets that do contain anomalies are unlikely to contain a sufficiently high sample number of anomalies to describe the full anomaly space of the problem. Third, data sets containing anomalies must be labeled accordingly. This means that a human operator must label all anomalies when they occur, which introduces additional complexity.

To develop these models, this paper evaluates two types of processes found in DMPs: manufacturing processes that are composed of concurrent linear sub-processes containing unique events, and manufacturing processes that are composed of concurrent processes with some events repeated in a loop in each sub-process. In both cases, it's assumed that the discrete event sequences are extracted from the discrete event stream of a PLC on a per-product basis through some existing technology such as RFID tags. Each system that is modeled is demonstrated visually as a Petri Net, although the end goal of the algorithms is anomaly detection and not recovery of the true underlying Petri Net.

There exists prior work in anomaly detection for DMPs [5; 6], but the approaches presented suffer from two significant disadvantages. First, prior work assumes that a complete data set exists such that a very precise anomaly detection algorithm can be trained. Second, the proposed methods scale non-linearly with the number of steps in the manufacturing process. In contrast, the methods proposed in this paper are designed to be efficient with respect to the size of the model, and efficient with respect to the size of training data set. When compared to a baseline method in Sec. 4.4, the approach presented in this paper achieves a better

F-score, while requiring less than 10% of the training set. Comparing the space complexity of the learned model, the approach presented in this paper scales linearly in the number of unique events (N) observed, $O(N)$, while the baseline method has space complexity $O(MN)$, where M is the length of the observed event sequences.

2 Background and Related Work

2.1 Petri Nets

In its classical form, a Petri Net [7] is a directed bipartite graph with two node types called places and transitions. These nodes are connected via directed arcs. Each arc has a defined weight, and can only connect nodes that are of different types. For example, an arc may connect a place and a transition, but may not connect two transitions or two places. A simple Petri Net consisting of two places, one transition, and two arcs is shown in Fig. 1. Here places are represented by circles and transitions by squares. When describing discrete manufacturing processes, places correspond to conditions and transitions correspond to events that could signal the start and end of a manufacturing task. A condition is said to be satisfied when a token is placed in the corresponding place. Events may only take place when the preceding condition has been satisfied. Lastly, when the arc weights are set to 1, they are typically not shown in graphical depictions of the Petri Net.

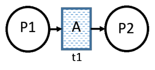

Figure 1: A Petri Net with places p_1, p_2, one transition t_1, and two connecting arcs each with a weight of 1.

Definition: Formally a Petri net is a triple (P, T, F) where:

- P is a finite set of places
- T is a finite set of transitions
- F is a set of arcs ($F \subseteq (P \times T) \cup (T \times P)$)

The number of tokens in a place at any given time is at least zero. The *state* or *marking* of a Petri Net refers to the distribution of tokens in the places. For example, for a Petri Net with four places, $< p_1, p_2, p_3, p_4 >$, a state representation such as $3p_2 + 1p_3 + 4p_4$ indicates there are 3 tokens in place p_2, 1 token in place p_3, and 4 tokens in place p_4. The inclusion of place p_1 is optional. Its absence explicitly indicates it contains no token.

The transitions (or events) can be *fired* (take place) if all of the input places (conditions) have at least one token (are satisfied). A transition that is fired is said to *consume* one token from each of it's input places and *produce* one token for each of it's output places. Thus when a condition is satisfied, an activity is executed, and a new condition is achieved. Of course since multiple activities and conditions occur throughout the Petri Net, each time instant may see multiple changes to the token distribution in the Petri Net.

Each transition has the property of execution time. Ordinary Petri Nets have instantaneous transition time, while timed Petri Nets have a specific duration for each transition. When a timed Petri Net has variable (possibly stochastic) transition times, it is referred to as a stochastic Petri Net. Here the time is usually described by an exponential distribution with parameter λ. From a modeling perspective, this parameter can be used to induce anomalies. For example, increasing λ leads to a tighter distribution of transition durations which could result in a new ordering of events in the final discrete event sequence. Thus anomalies in DMPs can be described by timing anomalies in Petri Nets.

2.2 Anomaly detection for discrete sequences

For a sufficiently finely sampled time step, the changes in the Petri Net token distribution can be observed as a sequence of discrete events that have taken place. This sequence can be analyzed for anomalies using anomaly detection methods for discrete sequences. A comprehensive survey of anomaly detection methods in discrete sequences can be found in Chandola et. al. [8]. The authors discuss three formulations of an anomaly: whole sequence anomalies where the entire sequence is anomalous; long subsequence anomalies within a long discrete sequence; and subsequence anomalies where the subsequence frequency within the larger discrete sequence is different than normal. Sequence-based anomaly detection, which addresses all three formulations, is the most relevant approach to this paper. Sequence-based anomaly detection is addressed through three main approaches. The first approach is kernel-based methods, which compute a similarity matrix for a given training sequence set and evaluate newly acquired sequences against this similarity matrix. The second approach is window-based methods which combine anomaly scores of subsequences of the test sequence for a total test sequence anomaly score. The third approach is to learn a Markovian model from the training sequence, where the goal is to learn the normal distribution of the data and thus assign a probability of being normal to a given test sequence.

A key drawback of similarity-based techniques with respect to the work in this paper is that most require a fixed training and test sequence size to facilitate distance calculations for anomaly scoring. Window-based techniques can be used to overcome this limitation, but they suffer from a high sensitivity to the window length parameter. Both types of techniques suffer from an unacceptably large search space as the number of unique symbols in the discrete sequence grows.

Markovian techniques are advantageous because they incorporate context information into analyzing future events. However, each discrete event in the sequence is not the state of a given system and therefore Markovian methods cannot be used directly. In such cases it is typical to use Hidden Markov Model (HMM)-based techniques to address the problem. The HMM-based techniques consist of two main approaches. The first involves learning an HMM model that best describes the normal training sequence, and then computing the probability of each test sequence given the learned model. The second approach computes the optimal hidden state sequence from each observation sequence in the train and test set. Using the optimal hidden state model and a threshold, each state transition is evaluated as normal or abnormal. The sum of all anomalous state transitions in a test sequence is its anomaly score. This approach suffers from sensitivity to initialization conditions and computational complexity.

2.3 Process mining

An alternative approach to anomaly detection in discrete event sequences is to first learn a generative model and then evaluate each test sequence for adherence to the learned model. Learning of Petri Net models from discrete event sequences has been previously investigated in the field of process mining. Here researchers focused on identifying an underlying process from event log data to learn the typical behavior. Critically, unlike the work in this paper, this research did not focus on anomaly detection, but instead on identifying event log conditions (and methods) under which a process can be recovered. The resulting algorithms are sensitive to uncertainty in the logs. In discrete event sequences, this variability is high due to the variability of duration of activities, and the repetition of some activities.

A comprehensive overview of process discovery techniques is provided in De Weerdt et. al. [9]. In this paper the authors note that learning Petri Nets is only tractable for specialized kinds of Petri nets such as Sound Workflow Nets (SWN). SWNs are a class of Petri nets with a dedicated start and end nodes. Algorithms to learn SWN from process event logs were first introduced by W.M. van der Aalst [10; 11]. The first proposed algorithm was termed by the α-algorithm [12]. This algorithm was shown to be sensitive to noise, incompleteness of event logs, and repeated events. To solve the robustness problem of the α-algorithm, HeuristicsMiner algorithm was developed in [4]. HeuristicMiner addresses: noise in event logs by adding weights to the graph arcs, and event path uncertainty by using a Causal Matrix instead of a Petri Net. The authors also argue that the F-score is the best metric to evaluate algorithm performance.

A successor to the α-algorithm, the α^+ algorithm, focuses on detecting repeated events in short loops [2]. Here the authors first identify which elements are in a loop of length 1 and then proceed with the original α-algorithm. A third algorithm, the α^{++} algorithm is designed to identify more complex event flows such as AND/XOR splits and joins [3]. Lastly, the β algorithm specifically focuses on identifying concurrent processes based on their start and end events [13].

3 Learning Models of Normal Operation using Event Relationships Table

This section introduces model learning of normal behavior of two types of processes found in DMPs: manufacturing processes that are composed of concurrent sub-processes, and manufacturing processes that are composed of concurrent processes with some events repeated in a loop in each sub-process. In the first case, all events that occur in the manufacturing sequence are unique, while in the second case, some events are repeated in a loop. Here we restate our assumption that the discrete event sequences are extracted from the discrete event stream of a PLC on a per widget basis through some existing technology such as RFID tags.

3.1 Algorithm I: Anomaly Detection *without* Event Repetition

Concurrent Processes in a Petri Net

The first discrete manufacturing process under consideration is one that is composed of concurrent sub-processes with unique events that occur only once during the DMP for each product. When modeling this DMP as a Petri Net,

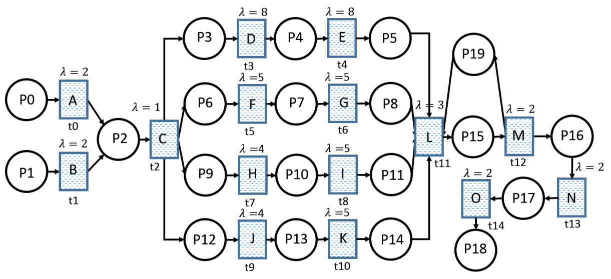

Figure 2: A Petri Net representing three concurrent processes in a DMP.

the multiple concurrent processes are modeled as simultaneously executing branches. To execute these branches, the first condition in each branch is triggered by a single event whose input is one or more origin conditions. The execution of events within each branch is independent of other branches. It is assumed that the concurrent processes operate on a single product, which means that a single final event (or series of events) denotes the end of the DMP. An example of such a Petri Net that describes a DMP with concurrent processes is given in [14]. This model is depicted in Fig. 2, and is used to demonstrate the proposed algorithm.

The Petri Net in Fig. 2 has 2 initial places P0 and P1 (either of which can start the process) and a final place, P18. The initial places are the input to transitions A and B, which both share an output place, P2. The token at place P2 is consumed by transition C (t2), which in turn triggers four concurrent processes. The first process is represented by events D and E; the second process is represented by events F and G; the third process is represented by events H and I; and the forth process is represented by events J and K. All four processes must complete in order to trigger event L (t11), which is followed by a sequence of events M, N, and O. This is a real-world example of concurrency in a DMP, and it's easy to see that the complexity of this system can be scaled while preserving the underlying concept.

In Fig. 2, λ is the parameter of the exponential distribution describing the execution time of each transition. For example, in the figure, transition C has a λ value of 1. This means that any single duration of this transition is a realization of a random variable with an exponential distribution with mean $\lambda = 1$.

Finding the Discrete Event Sequence in PLC Data

We now describe the design of an anomaly detection algorithm that is capable of finding anomalies in discrete event sequences. The algorithm presented herein is inspired by previous work in process mining algorithms presented in Sec. 2. Note here that the previously published algorithms focused on first identifying event successions in the form of event pairs and then applying sets of rules to the discovered events such that a valid Petri Net could be recovered. In the case of anomaly detection, it is not necessary to recover the final Petri Net, instead it is sufficient to focus on the succession of events as evidenced by the discrete event sequences in the training data set. By removing the need to find a valid Petri Net, this approach reduces the sensitivity to noise in modeling. We begin by briefly describing how to recover discrete event sequences from PLC output. For this purpose we use the following procedure.

First, the data is loaded into a table or matrix. The number of columns in this table is N and the number of rows is M. Here N denotes the number of number digital outputs of the PLC. The rows correspond to the number of samples observed from the DMP. For sufficiently short processes (a small number of events, N) with reasonable sample rates and sufficiently short time duration (small M), it is possible to load the data entirely into memory. In such cases, it is simpler to remove missing or corrupted data values. PLC digital outputs are particularly straight forward to clean, because the digital signal consists of only two levels: 0 or 1. If the data is too large to fit into memory, then cleaning the data must be interwoven with the second step, change detection.

The second step in this procedure is to detect changes in the digital outputs, or simply change detection. Specifically, for each of the N signal traces observed in the PLC output, we find the instances at which the trace changes from 0 to 1 or from 1 to 0. For large files, change detection can be performed iteratively, scanning a single line of the data file, and finding the columns whose state has changed. This means that instead of iterating through a large table, here, a state change table is created whose rows correspond to time stamps of at least 1 sensor state change, and whose columns correspond to sensors in the PLC output.

An important question is whether only activation state changes or activation and deactivation state changes should be recorded. That is, should sensor A switching from 0 to 1 be an event A1, and sensor A switching from 1 to 0 also be an event, A0. For the Petri Net model proposed in this paper, we argue that it is possible to only extract the sequence of activations, ex. A1. However, it is easy to imagine other cases where the deactivation events are equally important. In this case both activation and deactivation events can be incorporated into the discrete event sequence.

The third step, having isolated the changes in state for each sensor in the PLC data file, is to determine the sequence of changes. To find the sequence of changes, the sensor columns are first ordered with respect to time. The table is rebuilt by sequentially appending columns in order of occurrence. For example, if sensor A has been triggered at time 0, this is placed as the first column in the rebuilt table. If the next sensor to observe a change is sensor C, this column is placed as the next column in the table.

Having processed the data, the discrete event sequence observed from the PLC is simply the ordered sequence of column headers from the event change table. If the PLC data files include data that is specific to a particular product, then repeating this procedure across data files will results in a set of discrete event sequences observed across the manufacture of a set of products.

Building an Event Relationship Table

Having extracted the discrete event sequence from PLC data, this section proposes a simple model for the normal operation of a DMP: the Event Relationship Table (ERT). The ERT is square and has rows and columns corresponding to all unique events in the observed discrete event sequences. Its entries represent the temporal relationship between any pair of events. An example of a relationship table with three unique events: *A*, *B*, and *C* is shown in Table 1.

The relationship table is populated using the algorithm shown in Alg. 1. Here the discrete sequence of events (t_i) is denoted by σ. The succession of events is described by the following: a $A \rightarrow B$ denotes that event *B* follows event *A*; a $A \leftarrow B$ denotes that event *A* follows event *B*; and || denotes that events *A* and *B* follow one another but can happen in any order.

	A	*B*	*C*
A		$\rightarrow$	
B	$\leftarrow$		||
C		||	

Table 1: A simple event table describing the relationship among three events: A, B, and C.

Algorithm 1 Create Event Relationship Table

```
Input: a sequence of unique events, $\sigma = t_1, t_2, ..., t_n$
Output: an event relationship table
1: create an N+1 x N+1 dimensional table
   LOOP Process:
2: for for $n \in [0, len(\sigma) - 2]$ do
3:     place $\rightarrow$ in row $t_n$ and column $t_{n+1}$
4:     place $\leftarrow$ in column $t_n$ and row $t_{n+1}$
5:     if $\leftarrow$ and $\rightarrow$ in cell then
           replace with ||
6:     end if
7: end for
8: return event relationship table
```

Table 1, built using Alg. 1, shows the following event relations: event *A* always precedes event B, and events *B* and *C* always occur after one another but the order is not defined. Example sequences that can be generated by this table are *ABCB*, *AB* or *BCBC*. A sequence that cannot be generated according to this table is *BAC*.

To obtain a relationship table that fully describes a given DMP, the learning of the relationship table must be iterated through a training data set of discrete event sequences. Importantly, this data set must contain a sufficient sampling of the possible pairwise event sequences exhibited by the DMP to learn a complete event relationship table. This is not the same as observing all possible discrete event sequence generated by the DMP. Section 4.4 shows that relatively few sequences need to be observed to generate a full event relationship table.

Anomaly Detection Using the Event Relationship Table

Performing anomaly detection using a relationship table such as the table shown in Table 1 is straightforward. All that must be determined is if the the observed sequence can be the result of the pairwise event sequences as shown in the event relationship table. To see if this is the case, simply read the discrete event sequence from the first event to the last event, checking at each pairwise event sequence if it is an allowed transition in the table. If any transition is not allowed, then an anomaly is declared and the specific process in the DMP can be investigated.

The Weakness of the Proposed Algorithm

The algorithm presented in this section is attractive because of its simplicity and interpretability. However, the developed algorithm has been strictly limited to sequences that contain only unique discrete events that appear only once during a sequence. The reason for this is that anomaly detection using this approach may fail when events in the se-

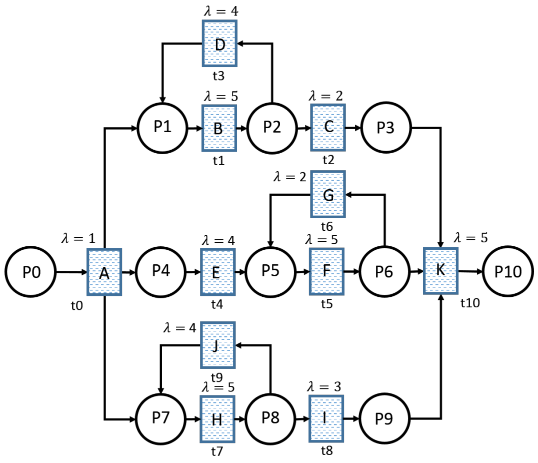

Figure 3: A Petri Net representing three concurrent processes in a DMP with events repeated in a loop.

quence are not unique, and specifically, when some events are repeated in a loop.

To see why this might be the case, consider the following. Suppose that event *A* always precedes event *B*, and both occur only once. Then, the presented algorithm would learn the relationship $A \rightarrow B$. Now, suppose event *A* occurs first, then event *B* takes place but this time event *A* is repeated after event *B*. Then, the learned relationships are both $A \rightarrow B$, and $B \rightarrow A$ which means that the learned relationship will be $A||B$. This relationship has low diagnostic value. This is because the relationship applies regardless of the event order, which means that both *AB* or *BA* are valid event sequences. Importantly, this relationship cannot help to specifically identify the sequence *ABA* as normal. Thus, the algorithm will fail to detect *AB* and *BA* as abnormal sequences (a miss and a false negative in detection, respectively). Such failures will result in an increased missed detection rate. To address this issue the section to follow extends the developed algorithm to improve its performance for the case of repeated events in a discrete event sequence.

3.2 Algorithm II: Anomaly Detection *with* Event Repetition

Real-life DMPs are often much more complicated than the concurrent process DMP presented in the last section. This is because events may be repeated within any of the DMP processes. Consider the example DMP for phone manufacturing in [13]. This DMP executes multiple concurrent processes, and is completed with an event loop that checks the quality of the final product. Such repetition of events results in entries of the event relationship table that have low diagnostic value.

In this section, we improve the previously proposed algorithm so that it is capable of identifying event loops within each of the concurrent processes. To develop this algorithm, this section proposes a more complex Petri Net model, shown in Fig. 3, that contains both concurrent processes and events repeated in a loop within each process. In contrast to the model in Sec. 3.1, this Petri Net is not published. To the best of the authors' knowledge, this model is more complex than published DMP models with event loops.

Here, a new transition (*D*, *G*, or *J*) is inserted into each process such that some events are repeated in a loop. The addition of these transitions has the effect of adding repeated events to the observed discrete event sequences. This results in a possibly infinite set of event sequences that can be generated by this Petri Net, or equivalently the DMP. With respect to the event relationship tables discussed in the last section, this leads to the following observation. The dimensions of the event relation table (NxN) will be smaller than the number of observations of the PLC output signal (M). This is the case because the event relationship table contains only unique events in the discrete event sequence. Thus we observe that the size of the event relationship table is bounded by the number of unique events in the DMP and not the length of the observed discrete event sequences.

Learning Sequences of Events

To improve anomaly detection in the presence of repeated events, we expand the concept of an event from a single atomic event to a complex event that consist of short sequences of atomic events that repeat with a high frequency within the discrete event sequences. For example, suppose that three distinct discrete events occur in a sequence: *A*, *B*, and *C*. Suppose further that events *B* and *C* repeat in a loop as *BCBCBC* such that the observed sequence is *ABCBCBC*. Then instead of using three single events, a better approach is to learn that the basic repeated loop is *BC*, and that there are two events in this sequence: *A* and *BC*, the latter of which is a complex event.

Learning the smallest set of event subsequences that describe a discrete event sequence is generally an NP-hard problem [15]. For this reason, this paper focuses on finding short event subsequences that occur more than once, and uses the remaining single events to complete the set of patterns that describe the discrete event sequences. Obtaining a computationally feasible solution to the problem is aided by two assumptions:

- If events are repeated in a loop, the events may appear more than once, and may appear contiguously in a sequence.
- Events that are not repeated in a loop appear as unique events in the sequence.

The first assumption is important, and it follows from the stated goal of separating the discrete event sequences by product (manufactured item). If events are allowed to repeat in a non-contiguous fashion, this would imply that multiple products are entering the discrete manufacturing process during the given event sequence.

Finding Repeated Event Sequences

Build on the previously introduced notation, the set of all contiguous subsequences within σ is W. The set of subsequences that occur with a frequency of at least 2 are part of the set W_{hf}. The set of unique events that are not part of a subsequence and occur only once is W_e. The union of sets W_{hf} and W_e is the set P which contains the set of all unique events and high frequency subsequences that describe a discrete event sequence. Given this notation and assumptions, event subsequences repeated with high frequency can be found as shown in Alg. 2.

This algorithm begins with a sequence of events $\sigma = t_1, t_2, t_3, \ldots, t_n$ which has K unique events, where K is smaller than the sequence length ($K \leq n$). In step 1 the algorithm generates all subsequences w' with length $len(w') \in [2, K]$ and places these subsequences in a set

W. W has a size of $(n-1)+(n-2)+\ldots+(n-K) = Kn-K(K+1)/2$ subsequences. Step 2 finds a subset W_{hf} whose component subsequences have a frequency of occurrence in $\sigma \geq 2$. Steps 3 through 5 are a loop which is used to remove subsequences of high occurrence from the original event sequence σ. This loop returns $\hat{\sigma}$ which contains all single events that do not belong to high frequency patterns. Step 6 places all events in $\hat{\sigma}$ in W_e. Lastly, step 7 returns the intersection of W_e and W_{hf}, P.

Algorithm 2 Find the set of high frequency patterns/single events, P, that covers a sequence of events

```
Input: Sequence of $K$ unique characters, $\sigma$
Output: $P = W_e \cap W_{hf}$
 1: $W \leftarrow$ Find $w' = t_{n_k}$ where $n_1 < n_2 < \ldots$ st. $len(w') \in [2, K]$
 2: $W_{hf} \leftarrow w' \in W$ with freq > 2
    LOOP Process:
 3: for each $w_i \in W_{hf}$ do
 4:     $\hat{\sigma} \leftarrow$ remove $w'$ from $\sigma$
 5: end for    return $\hat{\sigma}$
 6: place all events in $\hat{\sigma}$ in $W_e$
 7: return $P = W_e \cap W_{hf}$
```

Build a Pattern Relationship Table

Using the set P, it is now possible to build a relationship table as shown in Section 3.1. Because the events here may be subsequences of events, the new table is referred to as a pattern relationship table. The word pattern specifically denotes the fact that both unique events and high frequency subsequences occur in P. Patterns that occur with high frequencies are denoted by a new event relationship type, *. The procedure to build the relationship table is shown in Alg. 3. An example of a relationship table that encodes events *A*, *B*, *C*, and *DE* is shown in Table 2.

Algorithm 3 Create Pattern Relationship Table

```
Input: $\sigma$, $P$
Output: a pattern relationship table
 1: create an $len(P)$+1 x $len(P)$+1 dimensional table
    LOOP Process:
 2: while $\sigma$ do
 3:     find patterns $p_1$ and $p_2 \in P$ s.t. $p1$, $p2$ are consecutive and are at the beginning of $\sigma$
 4:     place $\rightarrow$ in row $p_1$ and column $p_2$
 5:     place $\leftarrow$ in column $p_1$ and row $p_2$
 6:     if $\leftarrow$ and $\rightarrow$ in cell then
 7:         replace with $\parallel$
 8:     end if
 9:     if $p_i$ for $i \in [1, 2]$ repeats in a loop then
10:         place $*$ in row $p_i$ and column $p_i$
11:     end if
12:     remove $p_1$ from $\sigma$
13: end while
14: return pattern relationship table
```

Note here that this event relationship table contains four base elements: *A*, *B*, *C*, and *DE*, while the number of unique elements in the sequence is five. Furthermore, the length of the discrete event sequences generated by this table can be arbitrary. Thus, this table illustrates that the approach presented in this paper creates compact representations of the discrete event sequences.

	A	*B*	*C*	DE
A		$\rightarrow$		
B	$\leftarrow$		$\parallel$	$\rightarrow$
C		$\parallel$		$\rightarrow$
DE		$\leftarrow$	$\leftarrow$	*

Table 2: A simple relationship table that includes event patterns repeated in a loop.

4 Experimental Results

This section describes an empirical verification for the proposed algorithms in simulation. In addition, we discuss a suitable evaluation error metric to be used, the minimum number of experiments performed, and the cross-validation method used to ensure generalizability of the results.

4.1 Choosing the Performance Metric

Anomaly detection is a classification problem with binary classes: normal (negative) or abnormal (positive). Anomaly detection algorithms are typically described by their sensitivity and precision. Sensitivity is the number of times that the positive class was correctly identified divided by the number of all positive class instances (true positive rate). Precision is the number of truly anomalous cases identified relative to the number of all cases identified as anomalous (positive) by the algorithm.

Precision and sensitivity can be combined in a single statistic called the F-score, defined as the harmonic mean of precision and sensitivity. The F-score allows a simple 1-dimensional comparison to be made between algorithms. In this paper, the F-score is used to compare the presented algorithms with a known anomaly detection algorithm based on a prefix tree, which is a baseline method [6].

4.2 Choosing the Number of Experiments

Typically, algorithm evaluations assume that the training data set is complete, which means that it contains a thorough enumeration of all possible normal and abnormal cases. Given this assumption, an algorithm's error rates are representative of the shortcomings of an algorithm's ability to capture the problem domain. However, in the case of anomaly detection algorithms for DMPs, this assumption may not hold, for several reasons. First, events in a DMP can be stochastic in duration. Second, DMPs are designed to be highly reliable, providing few opportunities to observe anomalies. Third, deployment cost of DMPs is tightly coupled to deployment time which means that training data set collection must be as short as possible. We should also note that some DMPs with loops in their execution logic would produce an infinite number of valid sequences, so exhaustive enumeration of all of them would be impossible.

The high reliability of DMPs, coupled with desired short deployment times, means that anomaly detection algorithms that are designed for DMPs would have typically have to work with less than sufficiently well sampled training data set. The stochastic duration of events in the DMP means that there is a much larger number of possible discrete event sequences that can be generated than one would expect from the number of events in the DMP.

Prior work has modeled the stochasticity in event duration of Petri Nets using exponential, Gaussian, or triangular distributions [16]. In this paper, an exponential distribution is used to describe the event duration, and the parameter of the exponential distribution is described by λ. Given the stochasticity in a process, it is instructive to calculate the number of possible discrete event sequences that could be generated.

Here we use the real DMP shown in Fig. 2. In this DMP the variation in discrete event sequences is produced by the four concurrent processes. These processes start with transition *C* and end with transition *L*. Then, the number of possible generated sequences is equal to the number of feasible permutations of *DEFGHIJK*. The total number of the permutations of these 8 events is $8!$ Among all the permutations, *D* must come before *E*, *F* must come before *G*, *H* must come before *I*, and *J* must come before *K*. The probability of this ordering is $0.5 * 0.5 * 0.5 * 0.5 = 0.0625$, and the number of possible sequence is $8! * 0.0625 = 2,520$.

This means that any indexing method, for example prefix trees (the baseline method of this paper), must be trained on all $2,520$ unique event sequences to achieve the best possible performance (F-score). In practice, when evaluating the performance of algorithms, many more data sequences (simulated experiments) must be collected. This is because each new sequence is not necessarily unique. For this reason, the experimental results in Sec 4.4 show a succession of training data set sizes ranging up to $10,000$ training sequences. This succession shows the F-score improvement as a function of training data set size.

4.3 Cross-Validation/Experiment Method

Lastly, it is important to ensure that the algorithm evaluation is not biased to any single training data set. That is, the experimental method should ensure that the algorithm performance statistics do not depend on a single data set, but rather represent generalized statistics that might be observed on a new test data set.

In machine learning, such generalized statistics are customarily determined using a technique called cross-validation (CV). In CV, models are trained iteratively on partitions of the available data, and tested on a held out data set. Both the training and testing data sets are changed for each iteration, and the performance statistics are averaged over all iterations. This paper uses CV, and therefore the reported rates in Sec. 4.4 are averaged across the number of tested partitions.

For example, suppose that $1,100$ normal discrete event sequences are generated in simulation, and 100 abnormal discrete event sequences are also generated in simulation. Anomalies are generated from normal sequences by: introducing an erroneous event transition (ex. normal: A $\rightarrow$ B, abnormal: B $\rightarrow$ A), removing an element of the discrete event sequence, and introducing a repeated event. These types of anomalies are placed in 33, 33, and 34 discrete event sequences, respectively.

During CV, 100 normal sequences are set aside for testing, and the remaining $1,000$ normal sequences are split into 10 partitions. For each of the 10 partitions, each algorithm is trained, and it's performance is tested using the 100 normal and 100 abnormal held out discrete event sequences. For each partition, the F-score of all algorithms under testing is calculated. After the 10th iteration, the F-scores are averaged and the average score is returned.

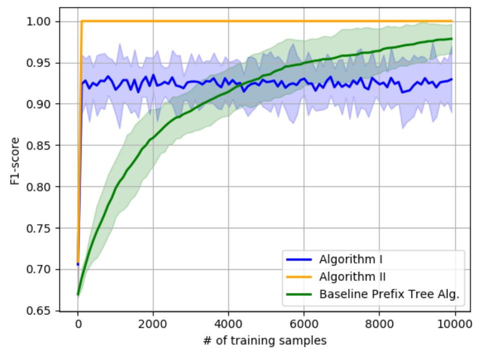

Figure 4: Experimental results for a DMP composed of concurrent sub-processes each with unique events.

4.4 Experimental Results

This section presents the performance of the proposed algorithms as compared to the performance of a baseline method. Here the baseline method is the prefix tree method, which was chosen because of its relation to prior published work.

Algorithm performance with unique events in concurrent processes

The first set of experiments demonstrate the performance of the developed algorithms, when the simulated DMP has concurrent processes, but no repeated events (Fig. 2). The experiments are performed on training sets of up to $10,000$ discrete event sequences using the method of cross-validation with 10 partitions. The results of this experiment are shown in Fig. 4. In this figure, the pattern relationship table algorithm is represented by an orange line, the event relationship algorithm is represented by a blue line, and the prefix method is represented by a green line. Each point on the plot represents an averaged F-score for the given method, and given the number of discrete event sequences in the total training data set. About each line, the shaded region represents two standard deviations in the F-score within the cross-validation rate set.

This plot shows that the baseline method's F-score eventually approaches that of the pattern relationship table algorithm, but only as the training data set size approaches $10,000$ sequences. In contrast, the algorithms presented in this paper achieve their maximal F-score with only 110 (1.1%) of the training discrete event sequences required by the baseline method. The reason for this is that the methods presented here are able to learn event transitions even in partially unique sequences, while the prefix method requires all possible event sequences to be observed and stored in a prefix tree.

Lastly, note that the pattern relationship table has the higher F-score even after all $10,000$ training sequences are observed. The reason for this is that pattern relationship table has learned subsequences of events that occur with high frequency, and therefore has learned event transitions that are of more importance than single events.

Algorithm performance with repeated events in concurrent processes

The second set of experiments showcases the performance of the developed algorithms, when the simulated DMP has

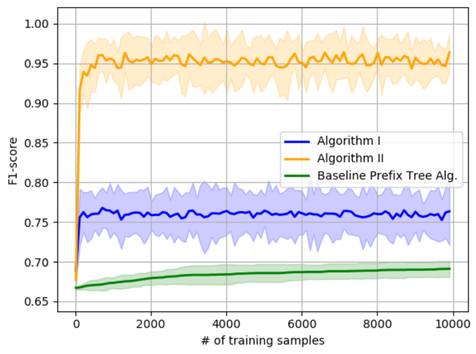

Figure 5: Experimental results for a DMP composed of concurrent sub-processes with events repeated in a loop in each process.

concurrent processes each with one repeated event (Fig. 3). Here, we use the same experimental parameters as in the previous section. The results of the experiment are shown in Fig. 5.

Once again, the pattern algorithm and the event relationship table algorithm both achieve their respective maximal F-scores with less than $1,000$ (10%) sequences in the training data set. In contrast, the prefix tree method, even after $10,000$ training data sequences, still has an F-score that is lower than the F-score of the event relationship table algorithm. These experiments show that when events are repeated in a loop, the variable length of the resulting discrete event sequence is a limitation for indexing methods like the prefix tree, most likely because of the size of the required training data set.

5 Conclusions

In this paper, we propose a novel approach for anomaly detection in discrete manufacturing processes (DMPs). The approach is to learn models of normal operation of a DMP in the form of relationship tables between pairs of events. These tables describe compactly the sequences of discrete events that can be observed in a DMP. The relationship tables can then be used to determine if a new sequence of events occurring in the DMP is anomalous or not.

The algorithms proposed herein are tested in simulation on two types of DMPs: DMPs with concurrent processes that have only unique events per process, and DMPs with concurrent processes and repeated events within each process. The performance of our algorithms is compared to a baseline method similar to that proposed in previously published work. The results show that the presented algorithms are capable of achieving a high F-score even for relatively small training data sets. Furthermore, the presented models are compact, reducing the space complexity of the DMP model from polynomial complexity to linear complexity.

References

[1] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, Sept 2004.

[2] AK Alves de Medeiros, BF Van Dongen, WMP Van Der Aalst, and AJMM Weijters. Process mining: Extending the alpha-algorithm to mine short loops. Technical report, BETA Working Paper Series, 2004.

[3] Lijie Wen, Wil MP van der Aalst, Jianmin Wang, and Jiaguang Sun. Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, 2007.

[4] AJMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166:1–34, 2006.

[5] Sicco Verwer. *Efficient Identification of Timed Automata: Theory and practice*. PhD thesis, Delft University of Technology, 2010.

[6] Oliver Niggemann, Benno Stein, Alexander Maier, Asmir Vodenčarevič, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. In *AAAI Conference on Artificial Intelligence*, AAAI'12, 2012.

[7] Christos G Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2012.

[9] Jochen De Weerdt, Manu De Backer, Jan Vanthienen, and Bart Baesens. A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Information Systems*, 37(7):654 – 676, 2012.

[10] Wil MP Van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[11] Wil MP Van der Aalst. Verification of workflow nets. In *International Conference on Application and Theory of Petri Nets*, pages 407–426. Springer, 1997.

[12] Wil Van der Aalst. Process discovery: An introduction. In *Process Mining*, pages 163–194. Springer Berlin Heidelberg, 2016.

[13] Lijie Wen, Jianmin Wang, Wil M. P. van der Aalst, Biqing Huang, and Jiaguang Sun. A novel approach for process mining based on event types. *Journal of Intelligent Information Systems*, 32(2):163–190, Apr 2009.

[14] Dejan Gradišar and Gašper Mušič. Petri-net modelling of an assembly process system. In *Proceedings of the 7th International Ph.D. Workshop: Young generation viewpoint*, volume 7, page 16. Institute of Information Theory and Automation, 2006.

[15] Danny Hermelin, Dror Rawitz, Romeo Rizzi, and Stéphane Vialette. The minimum substring cover problem. In Christos Kaklamanis and Martin Skutella, editors, *Approximation and Online Algorithms*, pages 170–183, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[16] Subhash Chandra, Muhammad Al Salamah, and Vakkar Ali. Stochastic simulation of assembly line for optimal sequence using petri nets (pn). 11:26–33, 01 2014.