Article

Anomaly Detection in Discrete Manufacturing Systems by Pattern Relation Table Approaches

Xinmiao Sun 1, Ruiqi Li 2,* and Zhen Yuan 1

1 School of Automation and Electrical Engineering, University of Science and Technology Beijing, Beijing 100083, China; xmsun@ustb.edu.cn (X.S.); g20193024@xs.ustb.edu.cn (Z.Y.)
2 College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China
* Correspondence: lir@mail.buct.edu.cn

Received: 30 August 2020; Accepted: 4 October 2020; Published: 12 October 2020

Abstract: Anomaly detection for discrete manufacturing systems is important in intelligent manufacturing. In this paper, we address the problem of anomaly detection for the discrete manufacturing systems with complicated processes, including parallel processes, loop processes, and/or parallel with nested loop sub-processes. Such systems can generate a series of discrete event data during normal operations. Existing methods that deal with the discrete sequence data may not be efficient for the discrete manufacturing systems or methods that are dealing with manufacturing systems only focus on some specific systems. In this paper, we take the middle way and seek to propose an efficient algorithm by applying only the system structure information. Motivated by the system structure information that the loop processes may result in repeated events, we propose two algorithms—centralized pattern relation table algorithm and parallel pattern relation table algorithm—to build one or multiple relation tables between loop pattern elements and individual events. The effectiveness of the proposed algorithms is tested by two artificial data sets that are generated by Timed Petri Nets. The experimental results show that the proposed algorithms can achieve higher AUC and F1-score, even with smaller sized data set compared to the other algorithms and that the parallel algorithm achieves the highest performance with the smallest data set.

Keywords: anomaly detection; discrete manufacturing systems; pattern relation table; centralized algorithm; parallel algorithm; Timed Petri Net

1. Introduction

Anomaly detection is a hot topic in various domains, such as manufacturing anomaly detection [1,2], cyber attack detection [3–6], and crowded scene video anomaly detection [7,8]. Cyber attacks detection typically detects three types of external attacks, i.e., false data injection attack, denial-of-service attack, and confidentiality attack. The crowded scene video anomaly detection seeks to detect abnormal human behaviors from the video source. This paper emphasizes on the anomaly detection of discrete manufacturing systems due to the surge of smart manufacturing and Industry 4.0. The smart manufacturing systems—equipped with mass sensors, fast networks, and sophisticated controllers—seek to make production efficient and reliable, which inevitably becomes increasingly complex. Such complexity makes it very difficult for even experts to fully understand the behaviors of the systems and identify anomalies in a timely and precise manner. On the other hand, smart devices (i.e., sensors, controllers) and wireless networks make the data collection and distribution to computers easier, thus promoting extensive researches on data-based anomaly detection algorithms. Based on the type of data, anomaly detection algorithms can deal with time-series data [9–11], discrete data [12–15], or mixed types of data [16].
This paper seeks to deal with anomaly detection for discrete event data that result from discrete manufacturing systems. Discrete manufacturing systems produce distinct items, such as automobiles, smartphones, and airplanes, which can be easily identified and counted. Producing one item typically involves multiple operations by different machines. The sequential actions of machines, such as turning on a button and turning off the button, define a series of events, thus generating various discrete event data.

In the production process, there can be some abnormal activities, besides, the conditions of machines that are involved in the production process might also be abnormal. In this paper, we consider three types of abnormal behaviors: missing necessary events, wrong order of events, and false process injection. Discrete manufacturing systems usually consist of parallel processes to produce different parts of the product that will be assembled later. Figure 1 gives an example of producing an L-shaped corner clamp with four parallel processes, see details in [17]. Besides, the whole process may also include loop processes, e.g., a material will be sent back to the same machine to process again when the material should be re-positioned and re-processed. This paper will take these two features into account when designing the anomaly detection algorithms, and such feature-based algorithms facilitate us to reach good performance at low costs: required small-sized data.

![Figure 1. An example of producing an L-shaped corner clamp with four parallel processes (‘Angle-bar with holder L’, ‘Clamp’, ‘Nut’, and ‘Screw’) in Ref. [17].](image)

Chandola et. al. reviewed the classical anomaly detection methods in discrete sequences in [12]. The authors summarized four approaches used in sequence-based anomaly detection problems, namely, kernel-based methods, window-based methods, Markovian methods, and Hidden Markov model (HMM)-based methods. Kernel-based methods are one of the similarity-based algorithms, which detect anomalies by comparing the similarity between the testing sequences and the training sequences. These algorithms are tolerant of false negatives, which may result in huge loss when considering the fact that customs may lose their money once ATM has a problem and manufacturing companies may suffer great losses once severe abnormal behaviors occur. Thus, in this paper, we set a strict bar to anomalous behaviors: as long as anomalous factor appears, the testing sequence will be considered to be abnormal. Window-based algorithms first cut the long sequence by windows of length $k$ and then compare the similarity of each window. The shortcoming of window-based algorithms is that it is sensitive to the selection of $k$, cannot be optimized easily. The Markovian, HMM, or Automata (see [15]) model-based algorithms focus on modeling the transition of states, which cannot reflect parallel processes, since parallel transitions may result in the same state changes as sequential processes. Recently, machine learning and deep learning based algorithms spring up, such as in [10,18]. It is essentially a model-based algorithm and the model can be described as a certain weighted structure of a network. This usually needs a larger data set to train a network.
The aforementioned algorithms mainly focus on the performance (usually measured by F-score), but pay little attention to how much data are required to reach such performance. Nowadays, although data collection is cheap, data cleaning and processing may also be labor extensive and time-consuming. This paper will evaluate both the algorithm performance and size of the data set required.

In this paper, we proposed a centralized pattern relation-based algorithm (Algorithm 1) and a parallel pattern relation-based algorithm (Algorithm 2) to detect anomalies in manufacturing systems, especially for parallel processes and loop processes. Algorithm 1 seeks to build a relation table between patterns—combinations of events that are meaningful in the models. Algorithm 2 seeks to build multiple relation tables for each pattern. Besides, developed algorithms provide two ways in searching for patterns that can describe the loop processes.

To test the proposed algorithms, we generated two artificial data sets due to the following reasons: (1) in reality, it is expensive or impossible to design an experiment to get large data sets with balanced data. (2) For new products or the products in the design stage, there are limited data. (3) Artificial data sets can easily provide more scenarios to test the robustness of the proposed algorithms. Data sets are generated through simulation techniques of Timed Petri Net that can model parallel and loop processes.

Overall, the contributions of this paper are threefold. First, two novel approaches are proposed in order to perform the anomaly detection. Second, the proposed approaches are tested for two complex manufacturing systems, one is a parallel system with loop processes, and the other one is a parallel system with nested loop processes. Finally, we investigate the size of the data that are needed to achieve a certain level of performance. The simulation results showed that the AUC values of our proposed algorithms can reach up to 0.92 with as little as 50 training sequences and up to 1 with more sequences.

The paper is organized, as follows. Section 2 formulates the anomaly detection problem and overviews two proposed relation-based anomaly detection approaches. Section 3 describes Algorithm 1 and Section 4 describes Algorithm 2 in detail, respectively. In Section 5, we present the simulation results and conclude in Section 6.

2. Pattern Relation-Based Anomaly Detection Approaches

2.1. Problem Formulation

The anomaly detection problem can be defined as

**Given:** a training data set \( D \) consists of \( N \) normal sequences, each of which contains a series of basic events \( e_i \) (denoted by lower case English letters in the example, which can be extended to the Unicode set).

**Task:** for any new sequence \( \sigma_i \), detect \( \sigma_i \) to see whether it is normal or abnormal.

**Example:** with the training data set \( D = \{afhk, afgfghjfhjkhk, afghjhf g fhk\} \), then for each sequence \( \sigma_i \) in \( D' = \{afghfhk, afghfhk\} \), identify each \( \sigma_i \) as normal or abnormal. This data set will be used to illustrate the proposed algorithms step-by-step through the paper.

2.2. Overview of the Proposed Algorithms

The main idea of the proposed relation-based anomaly detection algorithms follows such a pipeline: first, train a model with normal data sets, and then do the anomaly detection based on the model. The “core piece” of the model is so-called pattern relation tables, where the pattern is a combination of events that can represent a special relation of the events. The differences between the two algorithms proposed are the way of detecting patterns and building the relation tables.

We first define some common notations that are used in describing the models.

Let \( D = \{d_1, \ldots, d_N\} \) be the training data set, and \( d_i (i \in [1, \ldots, N]) \) is a sequence containing events occurred in order in a normal process (e.g., \( d_i \) can be \( afhk \), where each letter represents an event).
Let \( E = \{e_1, \ldots, e_K\} \) be the event set of all the unique events existed in the data set, and \( K \) is the cardinality of set \( E \) (e.g., in the Example, \( E = \{a, f, h, k, g, j\} \), and \( K = |E| = 6 \). Let \( E_n \) be the necessary event set, which contains the events that appeared in every sequence \( d_i \) (thus, in the Example, \( E_n = \{a, f, h, k\} \)). Let \( E_r \) be the repeat event set, where the repeat event appeared at least twice in a sequence. In the Example, \( E_r = \{f, g, h, j\} \).

Let \( LP = \{lp_1, \ldots, lp_n\} \) be the loop pattern set that contains repeatedly appeared sub-sequences with length varies in the range of \([2, K]\) in \( D \). In the two proposed methods, \( LPs \) are different. Let \( P = LP \cup E \) be the pattern set and the pattern set should include the elements that can fully recover the training set \( D \). Note that the choice of \( P \) is not unique. For example, the simplest one is \( LP = \emptyset \) and \( P = E \), and the most useless one is \( LP = D \) and \( P = D \cap E \) which doesn’t provide any new information from the training set. How to extract a suitable \( P \) is the key to the proposed algorithms.

Let \( T \) be the pattern relation table whose row index and column index equals to elements in \( P \) (Algorithm 1) or elements in each \( LP \) (Algorithm 2). A table cell represents an order relation between the two elements in the sequences. The relation tables are also different in the two proposed algorithms: Algorithm 1 builds one table in a centralized way while Algorithm 2 builds \(|LP| + 1\) tables in a parallel way. We will introduce them in detail in each subsection.

Now, we are ready to define the model \( \mathcal{M} = \{E, E_n, E_r, LP, P, T\} \) and we will introduce how we train \( \mathcal{M} \) from \( D \) for different algorithms. Both of the algorithms require four steps to learn \( \mathcal{M} \) and the final step is to do the anomaly detection. Figure 2 provides an overview of the approaches and summarizes the main idea of each step in Algorithms 1 and 2. Note that, not all elements in \( \mathcal{M} \) will be used in each algorithm. Algorithm 1 needs \( \{E, E_n, LP, P, T\} \) and Algorithm 2 needs \( \{E, E_n, E_r, LP, T_{0, .., |LP|}\} \).

![Figure 2](image-url)

**Figure 2.** Overview of major steps in relation-based anomaly detection algorithms and introduce the main idea of each step in Algorithms 1 and 2.

3. **Algorithm 1: Centralized Pattern Relation Table Algorithm**

Algorithm 1 summaries the five steps in the centralized pattern relation table algorithm and the following subsections elaborate each step.
we then come to

Algorithm 1

Centralized Pattern Relation Table Algorithm

Input: Training data set \( D \), test sequence \( \sigma \)
Output: Detect if \( \sigma \) is normal or abnormal
1: Extract the event set \( E \) and the necessary event set \( E_n \).
2: Learn the loop pattern set \( LP \) by Subroutines 1 and 2.
3: Split \( d_i \in D \) by Subroutine 3.
4: Build the Pattern Relation Table \( T \) by Subroutine 4.
5: Detect if \( \sigma \) is normal or not by Subroutine 5.

3.1. Extract the Event Sets \( E \) and the Necessary Event Set \( E_n \)

Traverse all sequences \( d_i \) in the training data set \( D \), extract the event set \( E = \{e_1, \ldots, e_K\} \) (the cardinality \( K = |E| \)) and the necessary event set \( E_n = \{e_{n_1}, \ldots, e_{n_m}\} \). Subsequently, the event set \( E \) can be decoded into two subsets: necessary event set \( E_n \) and the unnecessary event set \( E_u = E \setminus E_n \).

In the Example, \( E = \{a,f,g,h,j,k\}, K = 6, E_u = \{a,f,h,k\}, \) and \( E_u = \{g,j\} \).

3.2. Learn the Loop Pattern Set

We observed that, in practical discrete manufacturing systems, the events that are involved in the loops must be repeated at least twice in the sequence. Accordingly, we define a loop pattern \( lp \) as some sequential events (at least two events) repeatedly appeared in any single sequence \( d_i \). The learning process of the loop pattern set is implemented by Subroutines 1 and 2.

Subroutine 1 illustrates the process of extracting a loop pattern candidate list \( C \). Candidate \( c_i \) is obtained by traversing all \( d_i \) to find sub-strings of length 2 to \( K \) and appearance frequency \( > 1 \). Besides, the elements in \( C \) is ordered by its importance value—defined by the length of \( c_i \) multiply its appearance frequency—in the training data. We use the Example to show the details.

In the Example, there are no sub-strings with length \( \geq 2 \) appeared for more than once in \( d_1 = a f h k \), so no loop pattern candidate can be extracted. Then for \( d_2 = a f g f g f h j f h j h k \), we start with the length-of-2 sub-string \( af \), and find its appearance frequency is only 1, so discard it; according to the running basis, we then come to \( fg \), and add it to \( C \) since its frequency is 2; append \( gf, hj, jh \) due to the same reason; go on until the end of \( d_2 \). And then come to traversing the length-of-3 sub-strings; \( afg \) appears only once, so discard it, as well as \( fgf \) whose frequency is only 1; continue this process until the end. And no patterns with length \( > 3 \) that appeared more than once. Perform the same process for \( d_3 \). So eventually, the \( C = \{fg, gf, hj\} \). Meanwhile, calculate the importance value for each element in \( C \) and reorder \( C \) by the importance (the importance values of elements in \( C \) are \( 8(fg), 8(gf), 4(hj) \), respectively. Subsequently, the ordered \( C = \{fg, gf, hj\} \). It is worth noting that, although \( af \) has appeared in all three sequences \( d_i \) in \( D \), yet it does not appear twice or more in any single sequence, so it was not added into \( C \).

Subroutine 1

Candidate Patterns Extracting \((D, |E|)\)

Input: Data set \( D \), event cardinality \( K \)
Output: Ordered candidate pattern set \( C \)
1: \( C ← \emptyset \), \( Map = \text{set(seq.importance)} \)
2: \textbf{for} \( d_i \in D \) \textbf{do}
3: \hspace{1em} \textbf{for} \( w ← \text{sub-string of } d_i \) with \(|w| \in [2, \min\{|d_i|, K\}] \) \textbf{do}
4: \hspace{2em} \( num ← \text{frequency of } w \) in \( d_i \)
5: \hspace{2em} \textbf{if} \( num > 1 \) \textbf{then}
6: \hspace{3em} \( Map.add(w,0) \) \textbf{if} \( w \notin Map.keys() \)
7: \hspace{3em} \text{importance} ← \text{importance} + \text{len}(w) \times num
8: \hspace{2em} \textbf{end if}
9: \hspace{1em} \textbf{end for}
10: \textbf{end for}
11: \text{Order the } Map \text{ based on importance in a descending order}
12: \textbf{return} Map.keys()
Note that the candidate C may include redundant elements, since, if a longer sub-sequence $c_i$ repeated more than twice, then all of the sub-sequences $c_j$ will be repeated more than twice and become the candidates automatically. Therefore, we need a eliminate routine to select the most proper loop pattern elements among them. Moreover, a loop pattern element may exist in the form of $c_i,c_j$ in any sequence. Accordingly, the next step is to filter the candidates by Subroutine 2 in order to obtain the true loop pattern set $LP$. Steps 1–6 in Subroutine 2 is to eliminate latter candidates that can be represented by a former candidate together with some basic elements. Steps 7–14 are to eliminate candidate $c_i$ that never appeared with the form $c_i,c_j$ in any sequence $d_i$.

Continued the above example, the candidate loop pattern set $C$ has three elements $f g,g f,h j$. We then check the first one (i.e., $f g$) to see whether it is a subset of any other pattern, if so, then the latter element will be marked as $-1$ and won’t be added into the loop pattern set $LP$ (in the Example, $g f$ is a subset of $f g$, so $g f$ will be marked as $-1$); then, check the second one ($g f$), and do the same process, and if it is marked as $-1$, we will skip it and go on. Eventually, We add all of the candidate patterns not marked as $-1$ to the loop pattern $LP$. So far, the loop pattern candidate is $LP = \{f g, h j\}$. Subsequently, test whether there exists the form $l p_1, l p_1$ for the loop element $l p_i \in LP$ in the split list $SL$ obtained from splitting $d_i$ by the current $LP$ with Subroutine 3.

In the Example, $SL_1 = [a, f, h, k], SL_2 = [a, f, g, f, f, h, f, h, h, k], SL_3 = [a, f, g, j, h, f, g, g, f, f, k]$, then if sequential $l p_1, l p_1$ exists in $SL_1$, then $l p_1$ will be retained; otherwise, it will be discarded. Here, $f g, f g$ exists in $SL_2$ and $SL_3$, but $h j, h j$ doesn’t exist in any $SL_i$, so, eventually, $LP = \{f g\}$.

---

**Subroutine 2 Pattern Learning Subroutine** ($C$)

**Input:** Ordered candidate pattern set $C$

**Output:** Loop pattern set $LP$

1. $LP \leftarrow \emptyset$
2. for candidate $c_i!=-1$ in $C$ do
   3. for $c_j$ in $C[i + 1 : len(C)]$ do
      4. $c_i \leftarrow -1$ if $c_i \subseteq c_j$
   5. end for
5. end for
8. for $d_i$ in $D$ do
9. $SL \leftarrow$ Sequence Processing Subroutine($d_i, LP, E$)
10. for $l p_i$ in $LP$ do
11. if no sequential $l p_i, l p_i$ in $SL$ then
12. $LP_.remove(l p_i)$
13. end if
14. end for
15. end for
16. return $LP$

---

### 3.3. Sequence Processing Subroutine

The Sequence Processing Subroutine ($\sigma, S, E$) (Subroutine 3) returns an ordered list $SL$ by splitting any given sequence $\sigma$ with a given ordered list $S$ and the basic event set $E$. This subroutine will be applied in getting the loop pattern set $LP$ and building the relation table $T$ later.

This subroutine starts with splitting $\sigma$ by $s_i$ in $S$ one by one (so $s_1$ is of the highest priority, then $s_2$ and so on) in Steps 1–5. For the remaining part that not match any $s_i$ will be split by basic events $e_j$ in $E$, see Steps 6–12. Note that $SL$ is an ordered list and concatenate all the elements in $SL$ can obtain the original sequence $\sigma$. In the Example, if $S$ is $[f g, h j]$, then the split of $d_1 = a f h k$ will be $SL_1 = [a, f, h, k]$, since there is no sub-sequence includes any element in $LP$, $d_2 = a f g f h j f h j h k$ will be $SL_2 = [a, f g, f g, h j, f, h j, h, k]$, and $d_3 = a f h g j h f g f k$ will be $SL_3 = [a, f, h, g, j, h, f g, f g, f, k]$. 

Subroutine 3 Sequence Processing Subroutine \((\sigma, S, E)\)

**Input:** Any sequence \(\sigma\), given ordered list \(S\) and event set \(E\)

**Output:** A split-list \(SL\) obtained from splitting \(\sigma\) by \(S\) and \(E\) sequentially

\[
\begin{align*}
c_0 & \leftarrow \sigma \\
SL & \leftarrow \emptyset \\
\text{for } s_i & \in S \ (i \in [1, |S|]) \ \text{do} \\
& \sigma_i \leftarrow \text{replace } s_i \text{ by } \text{’}+	ext{’}+s_i+\text{’} \text{ in } \sigma_{i-1} \\
\text{end for} \\
\sigma \leftarrow \sigma | S \ . \ \text{replace(’’’,’’‘)} \\
\text{for each symbol } sp & \text{ in } \sigma[|1 : -1|] . \ \text{split(’’‘)} \ \text{do} \\
& \text{if } sp \ \text{in } S \ \text{then} \\
& \quad SL . \text{append}(sp) \\
& \text{else} \\
& \quad SL . \text{append}(sp \ \text{split by basic events } e_i) \\
& \text{end if} \\
\text{end for} \\
\text{return } SL
\end{align*}
\]

3.4. Build a Relation Table

Define the pattern set \(P = LP \cup E\) as the union of the loop pattern set and the basic event set. Now, we are ready to build a pattern table \(T\) to describe the relationship between pattern events for a given data set \(D\). There are three types of relations:

- \(p_i \to p_j, i \neq j\), meaning that pattern \(p_i\) appears ahead of \(p_j\), but \(p_j\) cannot be ahead of \(p_i\).
- \(p_i \| p_j, i \neq j\), meaning that \(p_i \to p_j\) and \(p_j \to p_i\) both exist in the normal data set.
- \(p_i \ast p_j\), meaning that this pattern can repeat itself as many times as possible. We can infer that any loop pattern \(lp_i\) must have this relation.

Subroutine 4 provides more details of building such a relation table and Table 1 shows the result for the data set \(D\) in the Example.

Subroutine 4 Pattern Table Building Subroutine \((LP, E, D)\)

**Input:** Loop Pattern \(LP\), Event Set \(E\), Training Data Set \(D\)

**Output:** Pattern Table \(T\)

Pattern set \(P \leftarrow LP \cup E\), Pattern table \(T[|P| \times |P|] \leftarrow \emptyset\)

**for** sequence \(d_i\) in \(D\) **do**

\[
\begin{align*}
SL & = \text{Sequence Splitting Routine}(d_i, LP, D) \\
\text{for } (sl_i, sl_{i+1}) & \text{ in } SL \ \text{do} \\
T(sl_i, sl_{i+1}) & = * \ \text{if } sl_i \text{ equals to } sl_{i+1} \\
T(sl_i, sl_{i+1}) & = \to \ \text{if } T(sl_i, sl_{i+1}) \ \text{is empty or } \to, \ \text{and } \ T(sl_{i+1}, sl_i) \ \text{is empty} \\
T(sl_i, sl_{i+1}), T(sl_{i+1}, sl_i) & = \| \ \text{if } T(sl_i, sl_{i+1}) \ \text{is empty or } \to, \ \text{and } \ T(sl_{i+1}, sl_i) = \to \\
\text{end for} \\
\text{return } T
\end{align*}
\]
Table 1. Pattern Relation Table $T$ of $D$.

| fg | h | f | a | k | g | j |
|----|---|---|---|---|---|---|
| *  | → | → | → | → | → | → |

3.5. Anomaly Detection Algorithm

Now, the model $M = \{E, E_n, LP, PT\}$ has been trained from the normal data set $D$. Accordingly, three rules are proposed to detect anomalous behaviors in a new sequence $\sigma$:

- $\sigma$ lacks any element in the necessary event set $E_n$.
- $\sigma$ has any element that is not in the basic event set $E$.
- Split $\sigma$ by Subroutine 3 to obtain $SL$, check in $T$ and the cell $T(SL_i, SL_{i+1})$ is empty.

The first rule is to ensure that the normal sequence should have the necessary events in $D$. The second rule is to ensure that there is no event out of the basic event set $E$. For the third one, we believe the normal orders between the pattern elements have existed in $T$. See the details in Subroutine 5.

The centralized algorithm shows all of the relations between pattern elements in one table $T$. Consider that the systems may be complex and the dimension of $T$ is large, this algorithm will be time-consuming. Subsequently, we propose a parallel algorithm that allows for multiple such $T$s to be used simultaneously.

Subroutine 5 Anomaly Detection of Any Sequence $\sigma$

Input: Model $M = \{E, E_n, LP, P, T\}$, test sequence $\sigma$

Output: Detect if $\sigma$ is normal or abnormal

1. Extract the event set $\hat{E}$ of $\sigma$.
2. if $\hat{E} \subseteq E_n$ or $E \subseteq \hat{E}$ then
3. return $\sigma$ is abnormal.
4. end if
5. $SL \leftarrow$ Sequence Splitting Routine($\sigma$, $LP$, $E$)
6. for pattern pair ($p_i$, $p_{i+1}$) in $SL$ do
7. if $T(p_i, p_{i+1})$ is $\emptyset$ then
8. return $\sigma$ is abnormal.
9. end if
10. end for
11. return $\sigma$ is normal.

4. Algorithm 2: Parallel Pattern Relation Table Algorithm

The parallel pattern relation table algorithm in Algorithm 2 has the same work flow as in Algorithm 1. However, in this algorithm, we take a different point of view of the loops in the data set, i.e., when the loop process happens, there must exist some events that will be repeated and the combinations of repeated events show the patterns of the loops. Therefore, a repeated event set needs to be learned. Besides, to extract multiple pattern relation tables in a parallel way, the subroutines to process a sequence and learn the loop pattern set will be changed accordingly. We will introduce the difference in the following subsections.
Algorithm 2 Parallel Pattern Relation Table Algorithm

Input: Training data set $D$, test sequence $\sigma$
Output: Detect if $\sigma$ is normal or abnormal
1: Extract the event set $E$ and the necessary event set $E_n$.
2: Extract the repeated event set $E_r$ by Subroutine 6.
3: Learn the loop pattern set $LP$ introduced in Section 4.2.
4: Process the sequence $d_i \in D$ in parallel introduced in Section 4.3.
5: Build multiple relation tables in parallel introduced in Section 4.4.
6: Detect if $\sigma$ is normal or not by Subroutine 7.

4.1. Extract the Repeated Event

Similar to Algorithm 1, this algorithm also needs to extract the event set $E$ and the necessary event set $E_n$ from the normal training data set $D$. Subsequently, a repeated event set $E_r$ whose elements appear at least twice in a single sequence is extracted. Subroutine 6 provides a way to extract the $E_r$. In the Example, $E_r = \{f, g, h, j\}$.

Subroutine 6 Repeated Events Extracting Subroutine ($D, E$)

Input: Training data set $D$, Event set $E$
Output: Repeated event set $E_r$
1: $E_r \leftarrow \emptyset$
2: for sequence $d_i$ in $D$ do
3: for event $e_i$ in $E$ do
4: if pattern $e_i \ldots e_i$ in $d_i$ and $e_i \notin E_r$ then
5: $E_r.add(e_i)$
6: end if
7: end for
8: end for
9: return $E_r$

4.2. Learn the Loop Pattern Set

In this step, first, process the data set $D$ get a modified data set $\hat{D}$ by traversing over $D$ and removing all events not in $E_r$ while keeping the original order. Subsequently, the remaining process can be done in a parallel way. For each element in $E_{r\prime}$, extract the candidate loop pattern set $C(E_{r\prime})$ from $\hat{D}$ where each element in $C(E_{r\prime})$ is started and ended with the same single event $E_{r\prime}$. Then sort $C(E_{r\prime})$ according to the length of the elements therein and the shortest one in $C(E_{r\prime})$ is the pattern of $C(E_{r\prime})$. In the Example, $\hat{D} = \{fh, fgfghjfhjhfj, fhgjhfgfgf, f, \{fgf, fgfgf, \ldots, f\}, (f) = f, (g) = \{fgf, gfhfgf, \ldots, f\}, (c) = \{fh\}, (g) = \{fgf, gfhfgf, \ldots, f\}, (h) = \{hjh, hjfjhj, hjh, hjhj\}$, so far the loop pattern set $LP$ is $LP = ffg, fgf, hjh, jfh$. However, for each pattern element, the order of the events is not necessary, just keep the event set of the pattern elements and remove the redundant ones and get the final loop pattern set. Continue the Example, $set(fgfg) = set(fg) = f$, so $fgf$ is identified as redundant and it will be removed from $LP$, eventually, $LP = [fg, hj, jfh]$.

4.3. Sequence Processing Subroutine

The third step is to define a subroutine to process any given sequence $\sigma$ based on each element in $LP$ and obtain a corresponding new sequence $\sigma(LP)$. It is still a removal process that is based on $LP$, for each $\sigma$ remove all elements not in $LP$, and keep the order of remaining elements. Subsequently, apply it to every sequence in the training data $D$ and obtain $D(LP_i) = \{d_1(LP_i), d_2(LP_i), d_3(LP_i)\}$, and add artificial start (S) and end(E) symbols to every $d_i(LP_i)$ at its two ends. For example, $LP_1 = fg$, then $D(LP_1) = D(fg) = \{SfE, SfgfE, SfgfgfE\}$, $D(LP_2) = D(hj) = \{ShE, ShjhE, ShhE\}$, and $D(LP_3) = D(jfh) = \{SfthE, SffhjhjhfjE, SfhhjhhffFE\}$.
4.4. Build Multiple Relation Tables

Afterwards, the fourth step is to build multiple relation tables $T_i$ based on $D(LP_i)$ (then the number of $T_i$ equals to $|LP|$). In addition, we also build a basic event relation table $T_0$ (see Table 2) based on consecutive event relation in the original training data set $D$. The process of building the relation table is the same, as shown in Subroutine 4, based on the $D$. Accordingly, for $D(LP_1)$, the corresponding relation table $T_{fg}$ has indices of $[S - f, f - E, f - g, g - f]$ (see Table 3); for $D(LP_2)$ and $D(LP_3)$, the corresponding $T_{hi}$ and $T_{jfh}$ are shown in Tables 4 and 5, respectively.

**Table 2. Pattern Relation Table $T_0$.**

|   | a |   | f |   | h |   | g |   | j |   | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
|   | → |   |   |   |   |   |   |   |   |   |   |
| f |   |   | → |   |   |   |   |   |   |   |   |
| h |   |   |   | → |   |   |   |   |   |   |   |
| g |   |   |   |   | → |   |   |   |   |   |   |
| j |   |   |   |   |   | → |   |   |   |   |   |

**Table 3. Pattern Relation Table $T_{fg}$.**

|   | S |   | f |   | g |   | E |
|---|---|---|---|---|---|---|---|
| S | → |   |   |   |   |   |   |
| f |   | → |   |   |   |   |   |
| g |   |   | → |   |   |   |   |
| E |   |   |   | → |   |   |   |

**Table 4. Pattern Relation Table $T_{hi}$.**

|   | S |   | h |   | j |   | E |
|---|---|---|---|---|---|---|---|
| S | → |   |   |   |   |   |   |
| h |   |   | → |   |   |   |   |
| j |   |   |   | → |   |   |   |
| E |   |   |   |   | → |   |   |

**Table 5. Pattern Relation Table $T_{jfh}$.**

|   | S |   | j |   | f |   | h |   | E |
|---|---|---|---|---|---|---|---|---|---|
| S | → |   |   |   |   |   |   |   |   |
| j |   | → |   |   |   |   |   |   |   |
| f |   |   | → |   |   |   |   |   |   |
| h |   |   |   | → |   |   |   |   |   |
| E |   |   |   |   | → |   |   |   |   |

4.5. Parallel Anomaly Detection Algorithm

In the parallel algorithm, the anomaly detection is based on $|LP| + 1$ relation tables $T_i(i \in \{0, \ldots, |LP|\})$. Only when all of the relation pairs obtained from processing $\sigma$ are shown to be normal, the sequence will be a normal one; otherwise, it is abnormal. Subroutine 7 summarizes this process.

It is worth noting that after extracting the loop pattern set LP, the algorithm can be run in parallel based on the element in LP to speed up the algorithm. That’s why we called it “Parallel Anomaly Detection Algorithm”.

Subroutine 7 Parallel Relation Table Anomaly Detection Algorithm

Input: Model $\mathcal{M} = \{E, E_n, E_r, LP, T_0, \ldots, T_{|LP|}\}$, test sequence $\sigma$

Output: Detect if $\sigma$ is normal or not

1: Extract the event set $\hat{E}$ of $\sigma$.
2: if $\hat{E} \not\subseteq E_n$ or $E \not\subseteq \hat{E}$ then
3: return $\sigma$ is abnormal.
4: end if
5: for event pair $(e_i, e_{i+1})$ in $\sigma$ do
6: if $T_0(e_i, e_{i+1})$ is $\emptyset$ then
7: return $\sigma$ is abnormal.
8: end if
9: end for
10: for Loop pattern $lp_i$ in $LP$ do
11: $\sigma(lp_i) = \text{Sequence processing}(\sigma, lp_i)$
12: for event pair $(e_i, e_{i+1})$ in $\sigma(lp_i)$ do
13: if $T_i(e_i, e_{i+1})$ is $\emptyset$ then
14: return $\sigma$ is abnormal.
15: end if
16: end if
17: end for
18: return $\sigma$ is normal.

5. Simulation

In this section, we test the proposed algorithms with two artificial data sets. The data sets are generated by a discrete event model Timed Petri Nets and we begin with briefly introduce the definitions and mechanisms of the Timed Petri Net. A detailed introduction can be found in the book [18].

5.1. Timed Petri Net

Petri Net is a classical model of Discrete Event Systems. When compared with Automata, it is more straightforward to demonstrate discrete manufacturing systems, such as parallel processes and loop processes.

A Timed Petri Net is a six-tuple $(P, T, A, w, x, V)$, where

- $P$ is a set of places, that is conditions required to enable transitions.
- $T$ is a set of transitions.
- $A \subseteq (P \times T) \cup (T \times P)$ contains all arcs from places to transitions and from transitions to places.
- $w: A \rightarrow \{1, 2, 3, \ldots\}$ is the weight function on the arcs.
- $x$ is the marking function of the Petri Net, where a marking represents the number of tokens in each place in the net.
- $V = \{v_j : t_j \in T\}$ is a clock structure and the clock sequence $v_j$ is associated with a transition $t_j$.

The clock structure $v_1 = \{v_{j_1}, v_{j_2}, \ldots\}$ means that, if the $k$-th fired time of the transition $t_j$ is $t$, then the next fired time could be $t' = t + v_{jk}$.

A Petri net graph can visualize a Petri net. Figure 3 shows a Petri net with $P = \{p_1, p_2, p_3\}$, $T = \{t_1\}$, $A = \{(p_1, t_1)(p_3, t_1), (t_1, p_2)\}$, $w(p_1, t_1) = 1$, $w(p_3, t_1) = 1$, $w(t_1, p_2) = 1$. The marking function $x = [1, 0, 1]$. A possible clock structure is $v_1 = \{1, 2, 1, 2, \ldots\}$.
A transition \( t_j \in T \) is defined as enabled or feasible if \( x(p_i) \geq w(p_i, t_j) \) for all \( p_i \in I(t_j) \), where \( I(t_j) = \{ p_i \in P : (p_i, t_j) \in A \} \) contains the input places to transition \( t_j \). In other words, transition \( t_j \) is enabled when the number of tokens in \( p_i \) is at least as large as the weight of the arc connecting \( p_i \) to \( t_j \), for all places \( p_i \) that are inputs to transition \( t_j \). A transition \( t_j \) is fired if it happened at some time and then moved tokens to the next places. For example, in Figure 3, we can say \( t_1 \) is enabled during the period \([0, 1)\), and fired at \( t = 1 \).

When a transition is fired, then we can say an event occurred. To obtain a sequence of events, we need the dynamics of the Petri Net, i.e., the way of moving tokens from places to places. It is defined by \( x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i) \) and one can easily identify it through a Petri Net graph.

Given an initial state \( x_0 \) and a Timed Petri Net, a sequence of fired events can be obtained by repeating the following steps: check the enabled transitions, fired the transition that is of the minimal scheduled time, and changes of states. This is the main idea of generating an event sequence, and a detailed description can be found in Chapter 10 of Ref. [18].

5.2. Experimental Systems

As we mentioned, we are interested in detecting any anomaly that exists in parallel processes, loop processes, and even nested loops and combinations of all these factors in more complex systems. Besides, the operation time (the interval time between two events) of a machine might be stochastic. Figures 4 and 5 show the Petri Net graphs of two such manufacturing systems. Figure 4 represents a system that consists of three parallel processes and each process has a loop process, which is the system used in [14] for comparison purposes. Figure 5 contains parallel processes and nested loop processes that make the systems more complex. We will show that the proposed algorithms are also capable of dealing with such a system. In both systems, the stochastic operation time is modeled by a Poisson process with parameter \( \lambda \).

5.3. Data Generation

With these two systems and the data generation approaches, we can generate normal data sets. However, in order to test our algorithms, we also need to generate abnormal data sets. We introduce three types of abnormal behaviors, as follows:

- Introduce an erroneous event transition to a normal sequence.
- Remove a necessary event from a normal sequence.
- Introduce some unrecorded/unknown events in a normal sequence.

Each type of anomalies accounts for \( 1/3 \) of sequences in the abnormal data set. Assume that the number of sequences in the training data set is \( N \), in the testing data set, half of them are normal sequences, and the other half are abnormal ones.
5.4. Compared Approaches

We will compare our algorithms with the prefix tree algorithms and the basic event table algorithm, as introduced in [14].

The prefix tree algorithm strictly compares the difference between the normal sequences and test sequence. When the testing sequence differs from any of the normal sequences, it is abnormal. In order to record the normal sequences in less space and to loop them up at high speed, the “prefix tree” data structure is used, hence its name.

The basic event table-based algorithm builds an event relation table between any two consecutive events in a normal sequence. When a testing sequence is given, if any two consecutive events within it do not exist in the relation table, then it is identified as abnormal.

5.5. Performance Metrics

Anomaly detection problems can also be viewed as a binary classification problem. Typical metrics for evaluating a classification algorithm include accuracy, sensitivity, precision, F-1 score, and AUC (area under curve). In reality, manufacturing systems can generate much more normal data than the abnormal one, so the data set will be highly unbalanced. We choose the F-1 score and AUC score as
the performance metrics, as the accuracy does not work well under such circumstances. The F-1 score is calculated from the precision and recall, namely

\[ F-1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \] (1)

where the precision is defined as

\[ \text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \] (2)

and the recall is defined as

\[ \text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \] (3)

The AUC is the area under the ROC (Receiver Operating Characteristic) curve, where an ROC curve plots the true positive rate vs. false positive rate at different classification thresholds. AUC represents the probability that a random positive example is positioned to the right of a random negative example. When \( \text{AUC} > 0.5 \), the classification result is better than a random classification method.

Besides, we plot the relationship between the number of sequences (\( N \)) in the training data set and the AUC and the F-1 score in order to investigate the efficiency of the proposed algorithm.

### 5.6. Simulation Results and Discussion

Figures 6–9 show how the AUC and F-1 score vary with the number of sequences \( N \) in the training data sets, where the dots represent the average values, and the upper and lower bars represent the standard deviations. These figures are obtained by the following steps: (1) for each \( N \), we ran the data generation routine for 10 times to get 10 data sets, which contain \( N \) sequences of each set; (2) performed the anomaly detection algorithms; (3) calculated AUC and F-1 Score on each data set, and obtained the average and standard deviation; and, (4) draw the figures.

From the figures, we can conclude that the newly proposed algorithms-parallel pattern relation table algorithm and the centralized one-outperform the prefix tree algorithm and the basic event table algorithm. This is because there are inherent shortages in the latter two algorithms. In the prefix tree algorithm, the criterion of being detected as normal is strict that only the sequences that existed in the training data set can be identified. However, because there are loop processes, the number of normal sequences is unlimited. Therefore given a small size of training data, some actual normal cases (especially the loop ones) cannot be identified, thus making the F1-score and the AUC-score low. With the size of the normal data set grows, the performance will be better, but will not be able to achieve 1.

In the basic event table algorithm, it only considers the event relationship between the consecutive events in a sequence. This might misidentify the cases where some necessary events are missing as normal. Taking the case that is shown in Figure 4 as an example, the sequence ‘abcfefgfk’ and ‘abcfefgik’ are mistakenly identified as normal, since there exist all of the consecutive events, yet ‘abcfegfk’ misses ‘i’ and ‘abcfegik’ misses ‘f’ between ‘fg’ and ‘f’.

In the centralized pattern relation table algorithm and the parallel pattern relation algorithm, they are designed to capture the loop patterns by focusing on the loop patterns or the repeated events. In the centralized algorithm, it overcomes the shortage of the basic event table algorithm by checking if all of the necessary events are included in the tested sequence. However, it still fails to detect the cases ‘abcfefgk’, since ‘fg’ is also normal to be followed by ‘i’. To address this issue, the parallel algorithm only focuses on the relationship between the events within the loop patterns by the deletion process of the sequences (sequence processing subroutine); moreover, it also builds the relationship with an artificial start (S) and end (E). For instance, only ‘i’ can be connected to ‘S’ and ‘E’ while ‘g’ cannot,
as shown in Table 3. Applying the sequence processing subroutine, ‘abhcefgik’ will be ‘SfgE’ and it is detected as abnormal. That is why it can correctly identify the ‘abhcefgik’ case.

In addition, it is worth noting that the parallel algorithm can achieve higher performance with a smaller size of the data set. This is also owing to the sequence processing subroutine. In the parallel algorithm, as long as repeated events appear, the relationship between the repeated events can be built in the relation table. However, in the centralized algorithm, the repeated events should be first detected as a pattern and then appear repeatedly. Because the latter case is a subset of the former case, the parallel algorithm will be easier to capture the loop patterns that are based on a smaller sized data set.

Finally, we also note that the relation table based algorithms have their own limitations: the performance converges no matter how the number of the training data set grows. The reason is that, if the number of patterns or the number of the repeated event is fixed, the number of the relationship in the pattern relation tables is limited. As long as all of the relationships appear, the table is fully built.

Figure 6. AUC score for System in Figure 4.

Figure 7. F-1 score for System in Figure 4.

Figure 8. AUC score for System in Figure 5.
6. Conclusions

We investigated anomaly detection problems in complex discrete manufacturing systems and proposed two effective methods for detecting anomalies of wrong orders, missing events, and unrecorded/unknown events. By fully taking the system structure (not the entire model) into account, our algorithms work well, even with quite small-sized training data sets. This implies that our methods can apply to the fields, where only limited data can be obtained and partial structure information is given, such as in the test of prototypes of intelligent sensor/control devices. In reality, such structure information is usually easier to be observed.

Note that, although our proposed algorithms seek to be applied in discrete manufacturing systems, they can also be used to other fields if parallel processes and loop processes exist, such as ATM fraud detection. Besides, the idea of searching for patterns in the centralized pattern relation algorithm may be applied to the natural language learning fields to learn the frequent words and grammar between the words.

Future work could be extended to several directions: (1) anomaly detection for totally unlabeled data, which includes both normal data and abnormal data; and, (2) event duration anomaly detection with normal data or unlabeled data.

**Author Contributions:** Conceptualization, X.S. and R.L.; methodology, X.S. and Z.Y.; software, X.S. and Z.Y.; validation, R.L.; formal analysis, X.S. and R.L.; writing, X.S. and R.L.; supervision, R.L.; project administration, X.S.; funding acquisition, X.S. and R.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Fundamental Research Funds for the Central Universities (Grant No. FRF-TP-19-034A1), by National Natural Science Foundation of China (Grant No. 61903020), and by Guangdong Basic and Applied Basic Research Foundation (Grant No. 2019A1515111039). BUCT Talent Start-up Fund (Grant No. BUCTRC201825).

**Conflicts of Interest:** The authors declare no conflict of interest.

**Abbreviations**

The following abbreviations are used in this manuscript:

HMM Hidden Markov model
ATM Automatic Teller Machine
AUC Area Under Curve
ROC Receiver Operating Characteristic

**References**

1. Kammerer, K.; Hoppenstedt, B.; Pryss, R.; Stöcker, S.; Allgaier, J.; Reichert, M. Anomaly Detections for Manufacturing Systems Based on Sensor Data—Insights into Two Challenging Real-World Production Settings. *Sensors* **2019**, *19*, 5370. [CrossRef] [PubMed]
2. Pittino, F.; Puggl, M.; Moldaschl, T.; Hirschl, C. Automatic Anomaly Detection on In-Production Manufacturing Machines Using Statistical Learning Methods. *Sensors* **2020**, *20*, 2344. [CrossRef] [PubMed]
3. Dong, Y.; Zhang, Y.; Ma, H.; Wu, Q.; Liu, Q.; Wang, K.; Wang, W. An adaptive system for detecting malicious queries in web attacks. *Sci. China Inf. Sci.* **2018**, *61*, 032114. [CrossRef]

4. Zhang, Y.; Fang, B.; Zhang, Y. Identifying heavy hitters in high-speed network monitoring. *Sci. China Inf. Sci.* **2010**, *53*, 659–676. [CrossRef]

5. Tu, H.; Xia, Y.; Chi, K.T.; Chen, X. A hybrid cyber attack model for cyber-physical power systems. *IEEE Access* **2020**, *8*, 114876–114883. [CrossRef]

6. Hosseinzadeh, M.; Sinopoli, B.; Garone, E. Feasibility and Detection of Replay Attack in Networked Constrained Cyber-Physical Systems. In Proceedings of the 57th Annual Allerton Conference on Communication, Control, and Computing, Allerton, IL, USA, 24–27 September 2019; pp. 712–717.

7. Nguyen, T.N.; Meunier, J. Anomaly detection in video sequence with appearance-motion correspondence. In Proceedings of the IEEE International Conference on Computer Vision, Seoul, Korea, 27 October–2 November, 2019; pp. 1273–1283.

8. Saligrama, V.; Chen, Z. Video anomaly detection based on local statistical aggregates. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 16–21 June 2012; pp. 2112–2119.

9. Malhotra, P.; Vig, L.; Shroff, G.; Agarwal, P. Long short term memory networks for anomaly detection in time series. In Proceedings of the 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, Bruges, Belgium, 22–24 April 2015; pp 89–94.

10. Munir, M.; Siddiqui, S.A.; Dengel, A.; Ahmed, S. DeepAnT: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access* **2018**, *7*, 1991–2005. [CrossRef]

11. Laptev, N.; Amizadeh, S.; Flint, I. Generic and scalable framework for automated time-series anomaly detection. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, Australia, 10–13 August 2015; pp. 1939–1947.

12. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly Detection for Discrete Sequences: A Survey. *IEEE Trans. Knowl. Data Eng.* **2012**, *24*, 823–839. [CrossRef]

13. Golait, D.; Hubballi, N. Detecting anomalous behavior in VoIP systems: A discrete event system modeling. *IEEE Trans. Knowl. Data Eng.* **2016**, *12*, 730–745. [CrossRef]

14. Laftchiev, E.; Sun, X.; Dau, H.A.; Nikovski, D. Anomaly Detection in Discrete Manufacturing Systems using Event Relationship Tables. In Proceedings of the 29th International Workshop on Principles of Diagnosis, Warsaw, Poland, 27–30 August 2018; pp 17–24.

15. Klerx, T.; Anderka, M.; Büning, H.K.; Priesterjahn, S. Model-Based Anomaly Detection for Discrete Event Systems. In Proceedings of the 2014 IEEE 26th International Conference on Tools with Artificial Intelligence, Limassol, Cyprus, 10–12 November 2014; pp. 665–672.

16. Feremans, L.; Vercruyssen, V.; Cule, B.; Meert, W.; Goethals, B. Pattern-based anomaly detection in mixed-type time series. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Würzburg, Germany, 16–20 September 2019; pp. 240–256

17. Gradišar, D.; Mušic, G. Petri-net modelling of an assembly process system. In Proceedings of the 7th International Ph. D. Workshop: Young generation viewpoint, Hruba Skala, Czech Republic, 25–30 September 2006; p. 16.

18. Cassandras, C.G.; Lafortune, S. *Introduction to Discrete Event Systems*; Springer Science Business Media: New York, NY, USA, 2009.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).