Creating and Using a Pattern Knowledge Graph to Predict Manufacturing Interruptions from Detected Anomalies

Andrew LeClair
dept. name of organization (of Aff.)
Bosch
Toronto, Canada
email address or ORCID

Hyeongsik Kim
dept. name of organization (of Aff.)
Bosch
Sunnyvale, United States of America
email address or ORCID

***Abstract*—This work introduces the Pattern Knowledge Graph (PKG). In this work, a manufacturing setting is evaluated which contains time-series data of two distinct events: anomalies and interruptions. We construct event sequences from these events so that patterns among the sequences can be mind. We then construct a PKG from these patterns. With the PKG, the relationships between anomalies and interruptions can be represented, and thus, queried. This in turn allows for more dynamic responses by users to prepare, or mitigate, interruptions based on the experienced anomalies. Additionally, we provide sound theory using lattice theory to propose how different events can be compared, and events can be obtained through the meet operator. We also provide an example usage of the PKG using sampled data so that example queries or uses can be shown, demonstrating its use and significance.**

***Index Terms*—real-time data, pattern extraction, knowledge graph**

I. Introduction

At any given manufacturing plant, there are numerous lines operating, each with an assortment of machines. Automated systems can be used to detect anomalies at any given location to determine whether the machine is operating erratically [1]. A machine that is not behaving as intended may eventually require a halting order so that employees may address the issue, or the machine may halt on its own, unpredictably. Regardless of whether this halting was planned or unplanned, every second of the machine not operating is a direct loss of productivity, and thus, a loss quantifiable in dollars.

The anomalies that are detected, and the interruptions that they may culminate to, do not have an explicit relationship. It is not the case that experiencing one specific anomaly always results in a specific ensuing interruption. It might instead be that a specific set of anomalies must occur before an interruption does. Or, it might be the case that an untreated, or unresolved, interruption will result in another. Regardless of circumstance, a knowledge system that is able to capture these nuanced relationships, and characterises how specific anomalies may relate to specific interruptions, will allow for a representation that can recognise possible futures the machine may experience.

A knowledge graph aims to capture the knowledge of a domain in a representation that is composed of concepts and relationships between the them [2]. A domain expert is the one tasked with defining these assertions, concepts, and relations. With a well-constructed knowledge graph, it is possible to utilise these concepts and relations in queries, acquiring the 'knowledge' of the domain. In this work, we present a specialised knowledge graph, titled the pattern knowledge graph, as a tool that can be used to assist users in determining when a machine may experience an interruption. With this information, the user may minimise the interruption through preparation, or entirely mitigate the interruption. The data which is used to construct and populate the pattern knowledge graph comes from two different databases. Although these two databases record data for the same location, they record different data: one tracks anomalies, the other tracks interruptions. Since the data is of the same location, they can be unified based on location, and temporally ordered. These orderings are referred to as sequences, and ultimately are used to extract the patterns that construct the pattern knowledge graph.

The structure of this paper is as follows. In Section II we evaluate the literature related to the pattern knowledge graph proposed in this paper. This includes evaluating the current work of knowledge graphs, time-series data representations, and any associated patents for technology. In Section III we formally present the problem statement which the pattern knowledge graph addresses. In this, we provide how specific theory will be used to address the problem. In Section IV we introduce how the sequences are generated, patterns are extracted, and the pattern knowledge graph is constructed. In Section V we present how the pattern knowledge graph can be used, using sample data, to provide enriched answers to queries in the domain about anomalies and interruptions. In Section VI we discuss the results of the pattern knowledge graph and evaluate it to similar technology of knowledge graphs and machine learning. Finally, in Section VII, we conclude the paper with future work and directions of the

Identify applicable funding agency here. If none, delete this.

pattern knowledge graph.

A. Contributions

The contributions of this work are as follows:

1) A framework which ingests time-series event data and uniquely generates sequences that relate anomalies to interruptions in such a way that a correlation between these events can be determined
2) A framework which utilises PrefixSpan and ingests sequences to generate the most frequent patterns
3) A pattern knowledge graph which relates sequence patterns to each other using the followed-by relation, and is generated from the product of the mentioned framework
4) A process for the infusion of domain knowledge to the pattern knowledge graph which utilises domain expertise to enrich the concepts and edges of the knowledge graph with additional data
5) A predictor that utilises the pattern knowledge graph to determine whether an anomaly is considered dangerous or not

II. Background

In this Section, we evaluate the current literature regarding the utilisation of time-series data for the creation and reasoning over knowledge graphs. This evaluation is done in three steps: we first evaluate different methods for extracting knowledge from time-series data; then we evaluate different methods or representations for constructing a knowledge graph; then we evaluate patents of technology that is comparable to the work presented in this paper.

A. Time-series Data Representation and Knowledge Extraction

Utilising time-series data is an established area of research, dating back to the early 2000's and late 1990's [3], [4]. In this work, rather than utilising the raw time-series data for prediction or anomaly-detection, it is first *sequenced*, then mined for patterns. Time series sequencing is commonly used for Machine Learning techniques which require the preprocessing of data into feature vectors [5]. However, this discretization process often involves creating uniformly shaped vectors, such as discretizing the time-series data using uniform windows (e.g., one week windows) [6]. Additionally, this process assumes the input data is a real-time series. In this work, the data we ingest is not real-time, but rather timestamped data. Thus, we can use a modified version of the shifting window to generate the subsequences. Instead of creating a sequence for every event spanning a one week window, we only create sequences for *up to* one week windows that begin with an anomaly and conclude with an interruption. This results in sequences that are not necessarily uniform in temporal length, but are similar in semantic meaning (i.e., they all represent the same type of information). Since the sequences we extract have specific criteria for creation, we also differ from the popular subsequence matching [7], which seeks to find similar subsequences in, for example, time-series data.

The sequences are mined to determine if there are any significant patterns. Pattern mining on sequential data is a well-researched field, with several different techniques for mining [8]. Patterns are subsequences that repeat in the sequences of the dataset, and can be used to interpret relationships between the concepts. For instance, in a shopping domain, where a dataset that contains the transactions of past customers as sequences, a pattern of $\{milk, cookie\}$ could be mined. This pattern shows that those two items are commonly bought together as a part of the same transaction. The *support* of a pattern is the number of sequences in the dataset that follows that pattern. Thus, pattern mining techniques strive to find the patterns with highest supports. There are several pattern mining algorithms, such as Spade [9], Spam [10], CM-Spam [11], and CM-Spade [11]. These algorithms all operate by reading the entire database to create the list of all patterns that meet input parameters, such as minimum support. Other algorithms were developed to improve performance, such as pattern-growth algorithms. These algorithms, such as PrefixSpan [12], improve performance by recursively scanning the database to find larger patterns. They only consider the patterns that exist in the database. Because of the increase in performance of the pattern-growth algorithms, in this work we use PrefixSpan, the predominant pattern-growth algorithm.

B. Knowledge Representations of Extracted Patterns

The patterns that are mined are then related to one another so that knowledge can be extracted. We investigate different similar structures that can be used for extracting knowledge.

A predominant structure for knowledge representation and extraction is a knowledge graph. A knowledge graph is a structure that imbues a simple graph structure with domain knowledge [2]. As a graph, it is composed of a set of concepts, and a set of relations that relates the concepts to one another. The domain knowledge is represented by defining specific relations, and relating the concepts. With a well formed graph, new knowledge can be extracted through reasoning techniques. A common language for writing and representing knowledge graphs is RDF [13].

A Markov Chain is a mathematical system composed of a set of states, and a set of edges over the set of states [14]. They can be modelled as a finite state machine. Each edge is assigned a probability as a weight and describes how one state may transition to another. Thus, the sum of all outgoing edges from any state must be equal to 1. The defining characteristic of a Markov Chain is that it is *memoryless*. This property states that the probability of the outgoing edges of a state are only determined by the present state. It is not influenced by *how* it got to that state. Markov Chains are suitable mathematical systems for representing any stochastic process that has the memoryless property, such as queuing theory. However, in any system where past events influence future possibilities, a different system is needed.

Lattices are structures that can be represented either with algebra or logic [15]. A partially ordered set $(L, \leq)$ is a lattice if it is both a join- and meet-semilattice. This is represented

by for every $a, b \in L$ there exists a join ($a \vee b$) and a meet ($a \wedge b$). A system which satisfies only the existence of a join (or meet) is called a join-semilattice (or meet-semilattice). A system represented as a lattice is granted the properties of reflexivity, idempotency, and associativity between the elements with the join and meet operators. This in turn allows the comparison of any two elements within the set L. Additionally, the algebraic representation allows one to use a theorem prover to provide basic automated reasoning processes, such as Coq [16]. However, for the system to follow a lattice structure, there must be a partial order that orders the elements of the set.

C. Review of Patents

In reviewing patents that relate to the creation and utilisation of a knowledge graph from patterns for the use of detecting interruptions we find the following.

[17] is a patent regarding the use of creating a knowledge graph from data to detect anomalies. However, they do not construct a knowledge graph from patterns, nor do they utilise the knowledge graph for detecting interruptions (that result from an anomaly).

[18] is a patent which uses a knowledge graph for anomaly detection. However, the knowledge graph belongs to the repository and is already created, rather than created from the data like the PKG.

[19] uses a knowledge graph to detect interruptions based on anomalous data, however, the knowledge graph is not created by the data, like the PKG. Additionally, security logs are used as input rather than sequences of data.

[20] uses time-series data and proposes a tool for anomaly detection. However, knowledge graphs are not used.

[21] proposes a directed knowledge graph that is created from time-series data. However, it is not used for detecting anomalies or interruptions, and it is not formed from patterns that exist within the time-series data.

[22] proposes the detection of anomalies in time-series data by taking sequences and comparing them to one another. However, the sequences are not mined for patterns, and no knowledge graph is formed.

III. Problem Formulation

A. Knowledge Graph

Here we introduce the knowledge graph, a fundamental structure necessary to understand the utilise the PKG. As introduced, a knowledge graph is a graph of concepts (or objects) and relations [2]. The knowledge graph is then represented by writing these concepts and relations using a language such as RDF [23] or OWL [24].

A concept in a knowledge graph can be an abstract idea defined by a domain expert that has been explicitly declared in the graph, whereas an entity can be a specific instance of data. An example would be to declare the concept *Dog* and assert an instance of it, such as *Fido*. That is, the entity *Fido* is an instance of the concept *Dog*. We can then create relations between concepts, such as *chases* which relates two concepts *Dog* with *Cat*. In this way, we can explicitly write out concepts, their instances, and how they relate to each other in a structured way. These relations can be interpreted as predicates over the set of concepts. The knowledge graph can then be used as a tool for to answer rich queries. For example, in a movie domain, there exists numerous sets of data about movies such as year they were made, directors, and their accolades. With a knowledge graph, concepts such as a movie genre can be defined, and thus can perform such a query as 'Which drama directed by Robert Benigini won the Best Actor Award?' [25]. In this example, the movie 'Life is Beautiful' would be the response.

In this work, we produce the PKG which is a knowledge graph, and as such, is formed using concepts and relations. As the name suggests, the concepts of the PKG are patterns of anomalies and interruptions. They are then related to each other such that queries can be performed using the PKG.

B. The Event Timeline

To introduce and define the PKG, we must first introduce preliminary terms and how they relate to the data that ultimately forms the PKG.

The foundational element that populates the PKG are *events*. An event is any single discrete event that occurs at a specific location at a specific time (denoted using a UTC timestamp). These events are saved into their respective tables in various databases. Two particular tables of interest are the *Interruptions* table and the *Predictions* table. The former table tracks events called *interruptions*. An interruption is a failure at the location identified which causes a delay. These delays result in a halting of manufacturing parts, and has a quantifiable monetary impact, in addition to requiring intervention by a worker to address what caused the interruption. The interruptions table tracks relevant details pertaining an interruption, such as where it occurred, when it occurred, a human-input text description of the interruption, whether it was a planned or unplanned interruption, and the duration of the interruption. The second table tracks events called *anomalies*. An anomaly is autonomously predicted by watching the manufacturing lines and detecting any irregularities or trends in measured data. A detected anomaly is then submitted to the anomalies table. Unlike the interruptions table, the anomalies table does not have human-input fields. Along with the anomaly, additional data is submitted to the table, such as where it occurred, when it occurred, what caused the detection or prediction, and an automatically generated description of the anomaly.

Although these two datasets are distinct from one another, and exist in two different databases, there is a relationship between them. The occurrence of specific anomalies, or sequence of anomalies, may correlate with the occurrence of certain interruptions. By using the timestamp and location of each event, we can create a timeline of events for each location. In Figure 1, we show a timeline that allows for the visualisation of anomalies and interruptions. With this timeline, a floor-worker or engineer is able to visualise the events of that location. The timeline visualisation is a powerful tool itself,

as it shows the temporal relationship between the events from various datasets in a single view. The temporal relationship that is visually displayed on the timeline allows one to identify links between anomalies and interruptions on the timeline. However, this process requires cumbersome human action of identifying how and if specific anomalies relate to a specific interruption.

The timeline visualisation tool allows for a user to determine relationships between anomalies and interruptions, but it is neither feasible nor scalable in determining all possible relationships. Any single location streams events at an extreme pace, and there are numerous locations monitored. To create a timeline for each location, and have an individual who monitors each timeline, at all hours, for any possible relationships between anomalies and interruptions is impossible. However, to remove the human element, we require the substitution of their domain expertise– being able to identify these relationships– with a knowledge system that can identify these relationships.

C. Pattern Knowledge Graph Theory

In this section, we introduce the Pattern Knowledge Graph (PKG) and the theory that is used to define it.

Let $\mathbb{P}$ be a finite set of patterns, the universe of our domain, and let E be a set of edges defined as follows:

$$E \subseteq \{(p_1, p_2) \mid (p_1, p_2) \in \mathbb{P}^2 \text{ AND } p_1 \neq p_2\} \tag{1}$$

We define a subset of $\mathbb{P}$, denoted as P, as the subset of all patterns which can be related to each other. We define it as follows:

$$P = \{p \in \mathbb{P} \mid \exists(p_0 \mid p_0 \in \mathbb{P} : (p, p_0) \in E \vee (p_0, p) \in E)\} \tag{2}$$

We define a subset of edges, denoted as E_P as the set of edges over the subset P. It is as follows:

$$E_P \subseteq \{(p_1, p_2) \mid (p_1, p_2) \in P^2 \text{ AND } p_1 \neq p_2\} \tag{3}$$

Then we define $G = (P, E_P)$ as a directed graph of patterns. We define the relation E (and thus, E_P as well) using the subset relation as follows. Let $p_1, p_2 \in \mathbb{P}$, then

$$(p_1, p_2) \in E \iff p_1 \subseteq p_2 \tag{4}$$

In other words, two patterns in $\mathbb{P}$ are related to each other if one pattern is a subset of another. Thanks to the partial-ordering of the subset relation, we can further define G as a directed acyclic graph, or a tree. More formally, since it is a tree of the PKG, we call it a *PKG Tree*. The definition is as follows.

Definition III.1. *PKG Tree Let $P \in \mathbb{P}$ be a set of patterns and let E_p be a set of edges over P. We then define a PKG Tree as $G = (P, E_P)$ where the following properties are true:*

- $P \subseteq \mathbb{P}$
- $E_P \subseteq E$
- $\forall(p_1, p_2 \mid p_1, p_2 \in \mathbb{P} : (p_1, p_2) \in E \implies p_1, p_2 \in P \wedge (p_1, p_2) \in E_P)$

This definition states that the set of patterns in a PKG Tree contains *all* patterns that can be related to one another. Therefore, every PKG Tree must be maximal as outlined in the following claim.

Claim III.1 (Maximal PKG Tree)**.** *Let $G = (P, E_P)$ be a PKG Tree. Then, there is no PKG Tree $G_0 = (P_0, E_0)$ such that $P_0 \subset P$.*

Proof. Proof by contradiction. Assume that $G_0 = (P_0, E_0)$ is a PKG Tree such that $P_0 \subset P$. Then,
$P_0 \subset P$
$\iff \exists(p \mid p \in \mathbb{P} : p \in P \wedge p \notin P_0)$
Using the definition of $p \in P$ implies there exists another pattern, p_1, such that either $(p, p_1) \in E$ or $(p_1, p) \in E$. The $p \notin P_0$ implies that neither of these edges exist. Therefore, we have a contradiction: both cannot be true. □

From this claim, we know that any PKG Tree will include all patterns that can be related to one another.

Figure 2 is an example of a PKG tree from four patterns: $P1, P2, P3, P4$. In this Figure, we note the root, $P1$, is the pattern that is a subset of all the other patterns, and that there is no cycles in the tree.

Thanks to the partial order provided by the subset relation, we can also represent a PKG tree as a meet-semilattice. For a given PKG Tree $G = (P, E_P)$, E_P is defined using the subset relation. If two elements, $p_1, p_2 \in P$ are related to each other such that $(p_1, p_2) \in E_P$, then we say that pattern p_1 is *followed by* pattern p_2. We then define the meet operator, $\wedge$, as the following for any patterns $p_1, p_2, p_3 \in P$:

$$p_1 \wedge p_2 = p_3 \iff p_3 \subseteq p_1 \text{ AND } p_3 \subseteq p_2 \tag{5}$$

The meet operator is defined as determining the pattern that precedes both input patterns. There are two scenarios for what can happen in a tree: either the two patterns belong to two different branches, or the two patterns come from the same branch. Referring to Figure 2, we illustrate the first scenario. If p_1 = P4 and p_2 = P3, we have two different patterns from two different branches. The meet is the event that comes before both p_1 *and* p_2, which in this case, is the initial P1. Therefore, we can see that for two patterns from two different branches, the meet is pattern that is common to both branches. The second scenario is for when two patterns are from the same branch. Let p_1 = P2 and p_2 = P4, then $p_1 \wedge p_2$ = P2. This is due to the reflexivity of the subset relation. Therefore, in the scenario where both patterns are from the same branch, the meet of two patterns is smaller one.

With this, we can define the PKG meet-semilattice, $L = (P, \wedge)$. We distinguish a the first pattern of the tree, the initial P1 in Figure 2, as the zero element 0. It is easy to show that the following is true for any $p \in P$

$$p \wedge 0 = 0 \tag{6}$$

In other words, when comparing any pattern to the root of the PKG meet-semilattice, the meet will always be the root; the root is not a subset of any other patterns. This relates to the

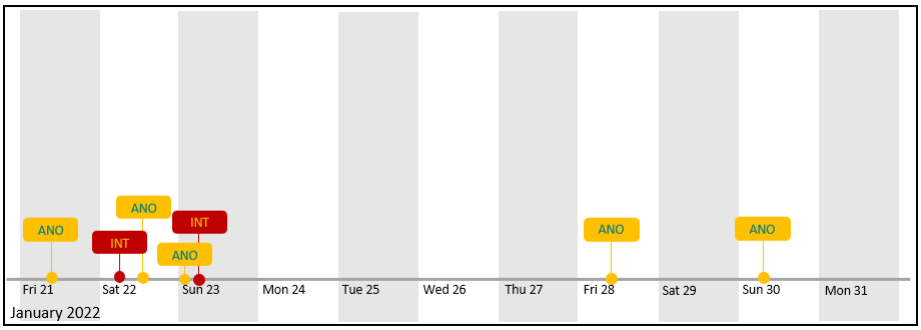

Fig. 1. Depiction of a timeline of events.

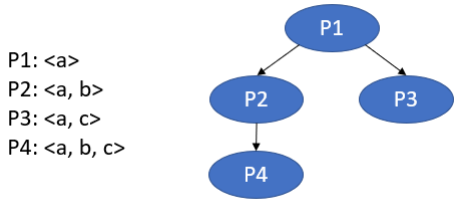

Fig. 2. Depiction of a PKG Tree with four patterns.

PKG Tree representation because through the transitivity of the subset relation, the zero element will be related to *every* pattern in the tree. Formally, for any PKG Tree $G = (P, E_P)$, and its semi-lattice representation, $L = (P, \wedge)$, we have

$$\forall (p \mid p \in P : (0, p) \in E_P) \tag{7}$$

The significance of defining the meet-semilattice structure lies in the ability to ensure the computation of the meet. So long as we have a meet-semilattice, then we can compare any two patterns. Additional to this, we guarantee the properties of commutativity, idempotency, and associativity. We provide the definitions for each below. For any three events, $p_1, p_2, p_3 \in P$:

$$\begin{aligned} \text{Commutativity: } & p_1 \wedge p_2 = p_2 \wedge p_1 \\ \text{Idempotency: } & p_1 \wedge p_1 = p_1 \\ \text{Associativity: } & (p_1 \wedge p_2) \wedge p_3 = p_1 \wedge (p_2 \wedge p_3) \end{aligned} \tag{8}$$

With these properties, in addition to the zero element, we are equipped to relate different patterns in the meet-semilattice to each other. Additionally, we can easily demonstrate that each PKG Tree is disjoint using the zero element, shown in the following claim.

Claim III.2 (Disjointedness of PKG Trees)**.** *Let $G_1 = (P_1, E_1)$ and $G_2 = (P_2, E_2)$ be two different PKG trees. Then the two sets of patterns are disjoint such that $P_1 \cap P_2 = \emptyset$.*

To avoid confusion between the semilattice meet operation ($\wedge$) and the logical 'and', we denote the logical 'and' with '*and*' in the proof.

Proof. Proof by contrapositive. Let $G_1 = (P_1, E_1)$ and $G_2 = (P_2, E_2)$ be two PKG Trees, and let 0_1 be the zero element of G_1 and 0_2 be the zero of G_2, then

$p \in P_1 \cap P_2$
$\langle$ Definition of zero element $\rangle$
$\iff 0_1 \subseteq p$ *and* $0_2 \subseteq p$
$\langle$ Definition of $E \rangle$
$\iff (0_1, p) \in E_1$ *and* $(0_2, p) \in E_2$
$\langle$ Definition of $P \rangle$
$\implies (0_1, p) \in E_2$ *and* $(0_2, p) \in E_1$
$\langle$ Definition of zero element $\rangle$
$\implies (0_1, 0_2) \in E$ *or* $(0_2, 0_1) \in E$
$\langle$ Equation 7 $\rangle$
$\implies P_2 \subseteq P_1$ *or* $P_1 \subseteq P_2$
$\langle$ Claim III.1 $\rangle$
$\implies P_1 = P2$ □

Therefore, as a result of this claim, the PKG will be composed of a set of disjoint trees; it is a *forest*. The property that each tree is disjoint also preserves the meet-semilattice. Any two patterns in a tree can be compared to each other using the meet operation, but no pattern can be compared with another that belongs to another tree.

IV. Construction

A. Preliminaries

A key motivation for using the pattern knowledge graph is to provide a holistic approach to the possible futures, as well as explainability for machine learning predictions. A knowledge graph is typically used to represent the concepts and relations that compose a domain using a set of graph nodes and edges. This graph can then be queried for answers that go beyond what traditional data querying can provide. This work goes beyond typical knowledge graphs because it incorporates patterns extracted from event-sequences to the knowledge graph. With this, the knowledge graph captures the knowledge of how the events in a manufacturing setting relate to each other in a sequential way. Queries that inquire about possible futures, given an input sequence of events, is possible thanks to the pattern knowledge graph. Additionally, further knowledge can be extracted, such as the urgency of upcoming events, or the expected time before the next event. This holistic

approach synergises with machine learning. Machine learning, most prevalent in anomaly detection fields, can be used to predict whether a provided event sequence is expected to end with an interruption. However, a prediction of interruption does not provide the information such as, how urgent the interruption is, what type it is, or what events might occur before the interruption. The pattern knowledge graph can be used to provide this explanation in accompaniment of the machine learning prediction.

However, our formulation of the pattern knowledge graph is not suitable for straightforward knowledge graph development. Whereas traditional knowledge graph development techniques involve several domain experts and the creation of concepts and relations, our approach receives the concepts from patterns. We require the determination of the patterns from the real-time datasets, and we require the creation of a pattern knowledge graph from these patterns. In Figure 3 we present a high-level workflow of the construction process. The process proceeds as follows. First, with Step 0, the data about the interruptions or anomalies, respective to the manufacturing line and factory they occur in, is recorded in a remote database. Following this, Step 1 marks the beginning of the PKG Creation workflow by ingesting this time-series data and homogenizing the data into a single database for events. The homogenization processes the anomalies and interruptions and defines them as types of events so that they may be recorded in a similar schema. Once the database has been established, in Step 2, we generate event sequences from this time-series data. These sequences are then submitted for pattern mining in Step 3 to produce the most frequent patterns among the generated sequences. Finally, in Step 4, we create the PKG from the patterns through a weaving process.

B. Sequence Generation

To address the issue of determining the patterns that ultimately generate the pattern knowledge graph, we first require sequences which the patterns are mined from. The sequences must be a structured sequence of ordered events. We define an event E_v as any anomaly or interruption that occurs at any location. These events can be temporally organised on a timeline using their timelines. However, we require an understanding on how to organise the events into a sequence such that the sequences provide significant results.

As introduced, we have two types of events: *anomalies* and *interruptions*. It is the case that an anomaly may be related to a following set of interruptions. Thus, a sequence must be defined such that it begins with an anomaly, and includes the events up to the interruption(s), for a given time range. For our work, we state that the time range is one week as we assume an interruption that occurs outside this window is no longer related to the anomaly. We provide the formal definition of a sequence as follows:

Definition IV.1 (Sequence)**.** *Let $I = i_1, i_2, ..., i_n$ be the set of all interruptions, and $A = a_1, a_2, ..., a_n$ be the set of all anomalies. A sequence is defined as $S = \langle s_1, s_2, ..., s_n \rangle$ where there is some index $k < n$ such that for all $i \leq k$, $s_i \subseteq A$, and for all $j > k$, $s_j \subseteq I$.*

The rationale for including all successive interruptions ($s_{k+1}...s_n$) is that if multiple interruptions occur, with no anomaly occurring between them, then they might all be related to the initial anomaly. Once a new anomaly happens, we begin a new sequence is formed and future interruptions belong to the new sequence instead.

In Figure 4, a graphical example of a sequence is shown. In this mock timeline, we show anomalies (denoted by blue squares with an 'A') and interruptions (denoted by orange diamonds with an 'I'), and the time they occurred at. At time t_0, the first anomaly occurs, followed by 3 more anomalies at times t_1, t_2 and t_3. The first interruption occurs at time t_4. At this point, the anomalies that occur *within* one week of t_4 are included in the sequence. For illustration, we state that t_1, t_2 and t_3 are all within one week, and so they are included to the sequence, denoted by the yellow box. t_0, although occurring before the interruption, falls outside of the 1 week window, and is thus not included. Following this, another interruption occurs at time t_5. Since no anomaly has yet occurred to signal a new sequence, this interruption is included to the current sequence. Finally, at t_6 a new anomaly occurs, signalling the end of the current sequence and potential creation of a new sequence.

C. Patterns Extraction

With the sequences formed, we can begin the pattern extraction process. Sequences detail the anomalies that occur before a set of interruptions, but there is no way to tell one anomaly from another as being more or less significant. The one week time window only provides us with temporal significance. Thus, we beyond this and analyze the sequences for patterns.

To mine the patterns from the sequences we created, the pattern mining algorithm PrefixSpan [12] is used. This algorithm mines the most prevalent subsequence patterns from the input set of sequences. In this work, we refer to a subsequence pattern as simply a *pattern*, and define it as follows:

Definition IV.2 (Pattern)**.** *Let $S = \langle s_1, s_2, ..., s_n \rangle$ be a sequence of events. We define $S' = \langle s'_1, s'_2, ..., s'_k \rangle$ as a subsequence, or pattern, of S, for $k \leq n$, if there exists an integer $1 \leq i_1 \leq i_2 \leq ... \leq i_k \leq n$ such that $s'_1 \subseteq s_{i_1}, s'_2 \subseteq s_{i_2}, ..., s'_k \subseteq s_{i_k}$.*

The output of the algorithm is a set of tuples of form (*frequency*, *pattern*). The pattern is the pattern found in the sequences, represented as a list, and the frequency is an integer denoting the number of sequences the pattern was found in. An example sample of results is shown in Figure 5. Here, we can see the first pattern has frequency 178, meaning that this pattern can be found in 178 sequences. The pattern itself is the single event titled 'i114'. This is a codified string which represents a unique interruption. The third row shows a pattern of 'a1768, i114'. This describes that in 101 sequences, the

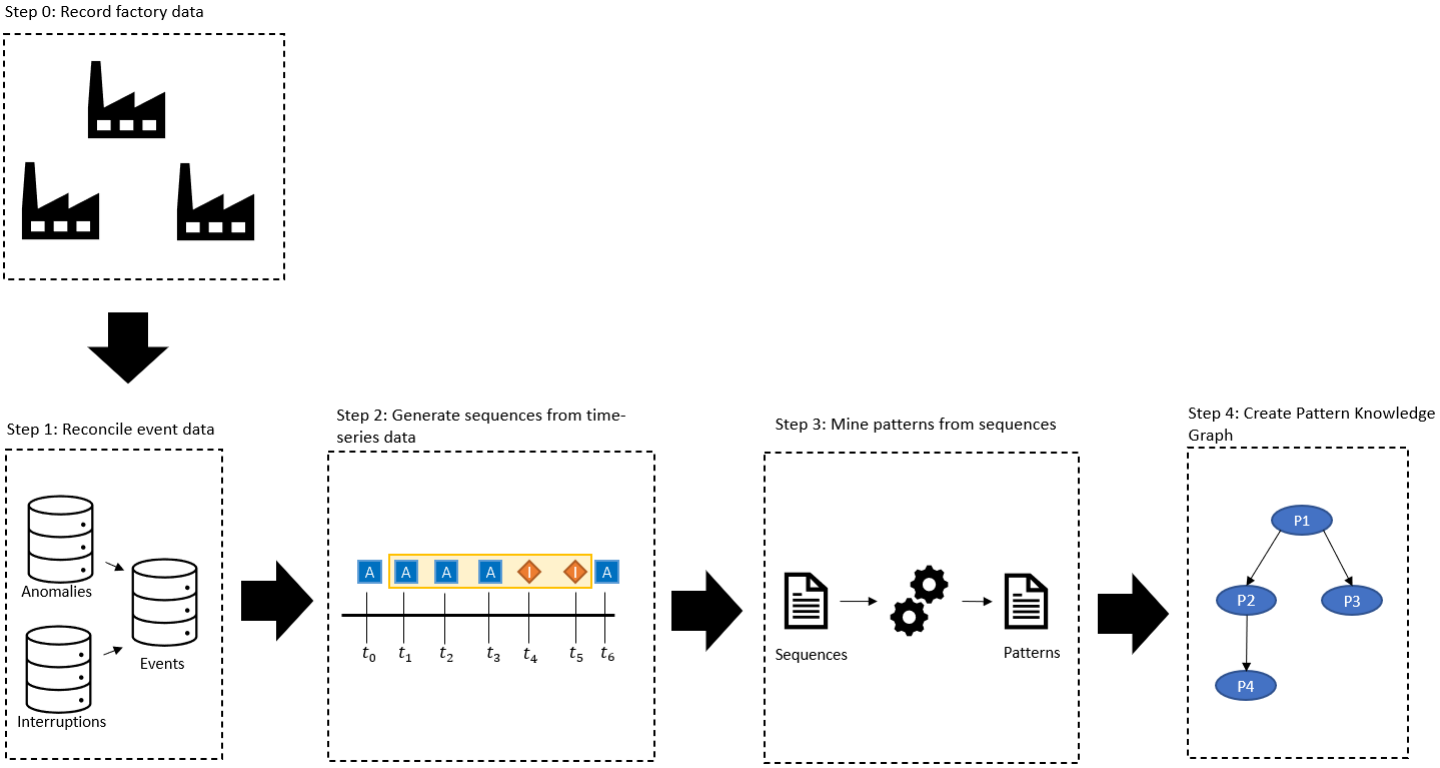

Fig. 3. The workflow for creating a PKG from the initial set of data.

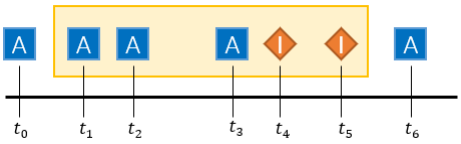

Fig. 4. Depiction of anomalies (A) and interruptions (I) in a sequence on a discrete timeline with events at time intervals t_i.

```
(178, ['i114'])
(112, ['a1768'])
(101, ['a1768', 'i114'])
(77, ['i212'])
(72, ['i114', 'i114'])
```

Fig. 5. Example set of results from PrefixSpan.

anomaly coded 1768 is followed by the interruption coded 114.

D. Crafting the Pattern Knowledge Graph

Once the patterns have been mined, the pattern knowledge graph can be formed by *weaving* them together. Patterns can be weaved if they satisfy the pattern overlap condition, defined as follows:

Definition IV.3 (Pattern Overlap)**.** *Let* $P_1 = \langle s_0, ..., s_i \rangle$ *and* $P_2 = \langle s'_0, ..., s'_j \rangle$*, for some* $i, j > 0$*, be two patterns. Then* P_1 *overlaps with* P_2 *if* $\langle s_0, ..., s_k \rangle \subseteq P_2$ *for some* $k \leq j$

In other words, if the two sequences begin with the same k events, then they can be weaved into a PKG Tree. The weaving process is conducted using the 'followed by' relation to relate the events to one other, and is defined using the $\subseteq$ relation. After comparing all patterns to determine all possible PKG Trees, the pattern knowledge graph is created.

Figure 6 illustrates the process of weaving two patterns to create a single PKG Tree. In the top of the figure there are two patterns, simply labelled as Pattern 1 and Pattern 2. Each pattern has a number of anomalies, denoted as A#, where the number denotes what type of anomaly it was, and an interruption. Since both patterns begin with the same type of anomaly, A1, we can merge these two patterns into a single tree, with root A1. Had their second anomalies also been identical, then it could also have been merged. Since the two patterns diverge at the second anomaly, the PKG Tree branches at this point.

The example in Figure 6 is a PKG Tree, same as what is shown in Figure 2. In Figure 2, each node represents a pattern, and the edges are the subset relation. We omit the labels of the edges in this figure to ensure readability. In Figure 6, each node is an event, and the edges are the 'followed by'. However, it is denoted this way to be more human-readable. In actuality, the single event of 'a1' is a pattern. Specifically, it is the zero element. The pattern 'a1, a2' is also a pattern, and one that contains the zero element. In this notation, we can see the

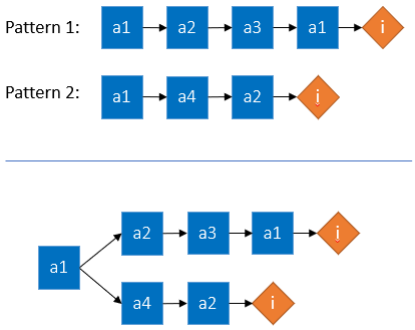

Fig. 6. Depiction of creating a PKG Tree from two patterns which share the same initial anomaly, A1.

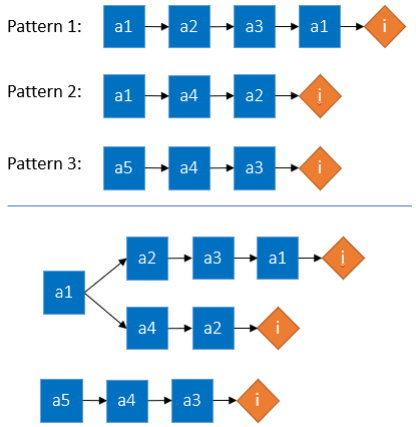

Fig. 7. Depiction of how two distinct PKG Trees are formed from three patterns in the PKG.

similarity between Figure 6 and Figure 2. However, for the sake of brevity, we only write the final event of the pattern, such as 'a2' instead of 'a1, a2'. We carry this notation for the remaining figures.

The PKG Tree in Figure 6 can be used to interpret or determine possible futures. For instance, when we first experience an anomaly of type a1, we have two possible futures according to our patterns: either an a2 can occur, or an a4. Depending on what the next event is, we will be traversing down that path of the PKG Tree. In this way, the PKG Tree is similar to a Markov Chain. The future possibilities of a node are expressed through the relationships of that node to other nodes. However, unlike a Markov Chain, a PKG Tree is not memoryless. This is because of the crucial aspect of the PKG in which the future possibilities are dependent on the previous, already occurred, events.

E. Enriching with Domain Knowledge

As described, a PKG Tree can be represented as a meet-semilattice. Thus, the meet-semilattice properties and relations can be used to compare events to one another. For example, two patterns can be compared to find which event is common to both of them. However, we can utilize the domain knowledge from the sequences and patterns to further elaborate the PKG Trees. For example, we can use the following information:

- the average number of events between pattern events (e.g., 1)
- the average time between pattern events (e.g., 1000s)
- the average anomaly level of a pattern event (e.g., 7)
- the probability of occurrence between pattern events (e.g., 80%)

It is important to remember that a pattern is a subsequence of a sequence. The first two items of information of that list help understand what type of sequence one might be following. For instance, in Figure 7, although Pattern 3 states that a4 follows a5, it does not state that a4 must *immediately* follow a5. A sequence of form ⟨a1, a5, a2, a4, a1, a3⟩ does follow Pattern 3, despite a1 existing in between a5 and a4. Thus, in this example sequence, *one* event occurs between a5 and a4. By gathering all sequences that follow Pattern 3, we can determine the average number of non-pattern events that occur between each pattern event. A similar average can be determined for the number of seconds between the two pattern events. For instance, if we take all sequences that follow Pattern 3, and average the time elapsed between 'a1' and 'a5', we can determine on average how long 'a5' will occur after 'a1' if a sequence follows this pattern. Supplying this information to the PKG Tree allows one to better understand what a sequence that follows the pattern might look like. For instance, if the average number of events between a5 and a4 is 0 and the average time between events is 60s, then the worker is made aware that when 'a5' occurs, 'a4' will likely occur next within a minute. If the average number of events were ten, then the worker would know that many events, that are not a4, will likely occur.

The third piece of information is domain knowledge from an autonomous system that measures the impact of an anomaly. The anomaly level details how significant it was, and how much attention should be given. A higher anomaly level implies a more impactful event has occurred at the location. By taking the average of all the anomalies of the events in a sequence that follows the pattern, we can determine how significant the anomaly in that pattern is. For instance, in Figure 7, we have an a4 in both Pattern 2 and Pattern 3. We take the average of all anomaly levels of the a4s that follow Pattern 2 and do the same for all a4's that follow Pattern 3 and provide that information to the nodes of the respective PKG Trees. With this information, we can quantify whether a4 is more, on average, significant when preceded by either a1 or a5.

The final piece of information is the probability of different occurrences happening, i.e., which future is more likely. When investigating a1 in Figure 7, we note the PKG Tree has two futures: one with the next event as A2, and the other with the next event as a4. By accounting for all of the sequences that follow the two patterns of Pattern 1 and Pattern 2, we can

determine which one is more likely. For instance, if between all sequences that follow either Pattern 1 or Pattern 2, 95% follow Pattern 1, then it is much more likely that a2 will follow a1. This information can be imbued into the weight of the edges.

V. Utilisation

In this Section, we describe how an example PKG was created, and demonstrate how it can be used.

For our case study, we investigated a dozen manufacturing stations located over three manufacturing plants. The data regarding the events from February 2021 to February 2022 was used. In total there were millions of events, which were organised into sequences according to location. The sequences were generated as described in Section IV-B. A sequence begins with an anomaly and includes all anomalies up to an interruption that occurs no more than 1 week after the anomaly. All subsequent interruptions are also included, so long as no other anomaly occurs. From this data, 450 unique sequences were generated. These sequences were collected from all stations. After applying PrefixSpan to this set of sequences, the top 8000 patterns were extracted.

The patterns that are extracted are then weaved based on the process described in Section IV-D. There are two representations that can be stored for the PKG. The first is shown in Figures **??** 9. In these figures, two example interfaces are provided to illustrate how the timeline and PKG operate together to provide the user with domain knowledge of events that have occurred, and possible futures. In Figure 8, we see the timeline introduced earlier. Inside the timeline, we see the events that have occurred, placed appropriate to when they occurred. However, new to this timeline is that the most recent anomalies in a sequence are diagnosed with a danger level, indicated with a green or red border around the anomaly in the timeline. In the example, the first anomaly is diagnosed as not being dangerous, indicated with a green border. The second anomaly is diagnosed as dangerous, indicated with a red border, as well as a modal displaying additional information about the anomaly. In this modal we see that the type of anomaly is 'a3', and we can find additional domain information supplied, such as the parameters which are acting anomalous. Below the timeline, there are two windows. The information from these windows explains the reason for why it was diagnosed as dangerous. The first window is knowledge supplied by the PKG. It provides the knowledge about the future events that may occur given the sequence of events that have already occurred, as well as any particular interruptions to be aware of. The second window provides the result of an accompanying machine learning prediction models. For each event that occurs, the model predicts whether an interruption is likely to occur in the next 7 days. The window is populated by the event of the sequence that was submitted, in addition to the result of the prediction. In this case, the model predicts an interruption will occur after a3 within 7 days, and so it is denoted with the red box. Together, the PKG and machine learning model predict an interruption, and so the event 'a3' is diagnosed as dangerous. Figure 9 is an extension of the first example interface. In this figure, we show how the PKG Tree itself is displayed, allowing the user to further explore the possible dangers that have been predicted. In this example, since 'a3' has occurred, the PKG window displays the PKG Tree appropriate for this sequence, with 'a3' as the root. This window is interactive, allowing the user to select specific nodes or edges based on what they are curious about. Any selected node will have further information about that node displayed in the accompanying window. For instance, selecting the child node of the root, 'a3', populates the adjacent window with it's data. The selected node is indicated by the yellow node, and the corresponding node name is 'a1 a3 a3'. This is because the first event to occur in the sequence was 'a1', and the second event was 'a3', the one diagnosed as dangerous. Selecting the child node 'a3' is evaluating the sequence 'a1 a3 a3', which in this example, corresponds to a pattern since it exists as a node within the PKG Tree. The information of this node, shown in the second window, allows the user to learn more about this specific future. In this case, the user can learn that there is an 80% of 'a3' being the next event to occur, and that if it does, it is also considered dangerous. The PKG Tree is also stating that this event, if it were to occur, would be the next event since the average number of events between 'a1 a3' and 'a1 a3 a3' is 0. The user can also see in the PKG Tree, that if by chance the next event that occurs is 'a2', then they are likely safe. This is because no interruption exists in a pattern where 'a2' follows 'a1 a3', shown by no interruption existing in that branch of the PKG Tree. It is important to note that the PKG Tree does not describe what future *will* happen, but rather, will *likely* happen, or what will *eventually* happen, given previous sequences of events. For instance, although we have experienced 'a1 a3', it is not for certain that either 'a3', 'i2', or 'a2' will occur next. The PKG Tree can provide knowledge on whether the probability of one of these events being the next immediate event, but another event outside of these 3 may occur. For example, it might be the case that another anomaly, 'a4' occurs. Since the sequence 'a1 a3 a4' is still following the pattern, the PKG would update accordingly. In this case, since 1 event has occurred, then 'a1 a3 a3' may no longer be the likely future as typically 0 events occur between. Thus, the PKG could indicate that 'i2' or 'a2' is the more likely next event.

The second representation for the PKG is as a serializable format, such as XML or JSON. Numerous XML formats, specifically for graphs, exist, such as GraphML [26]. Python libraries, such as Networkx [27] offers means of translating between GraphML to JSON, or to numerous other formats. These serialized formats can then be saved in a database, for example, a JSON record can be saved withing a Mongo collection. Regardless of specific format, each PKG Tree is saved such that the domain knowledge of the nodes or edges are saved as attributes and are easily accessible with queries. Additionally, metadata about its graphical structure, such as it being directed, allows for visualisation with any graph visualisation library, such as cytoscape [28].

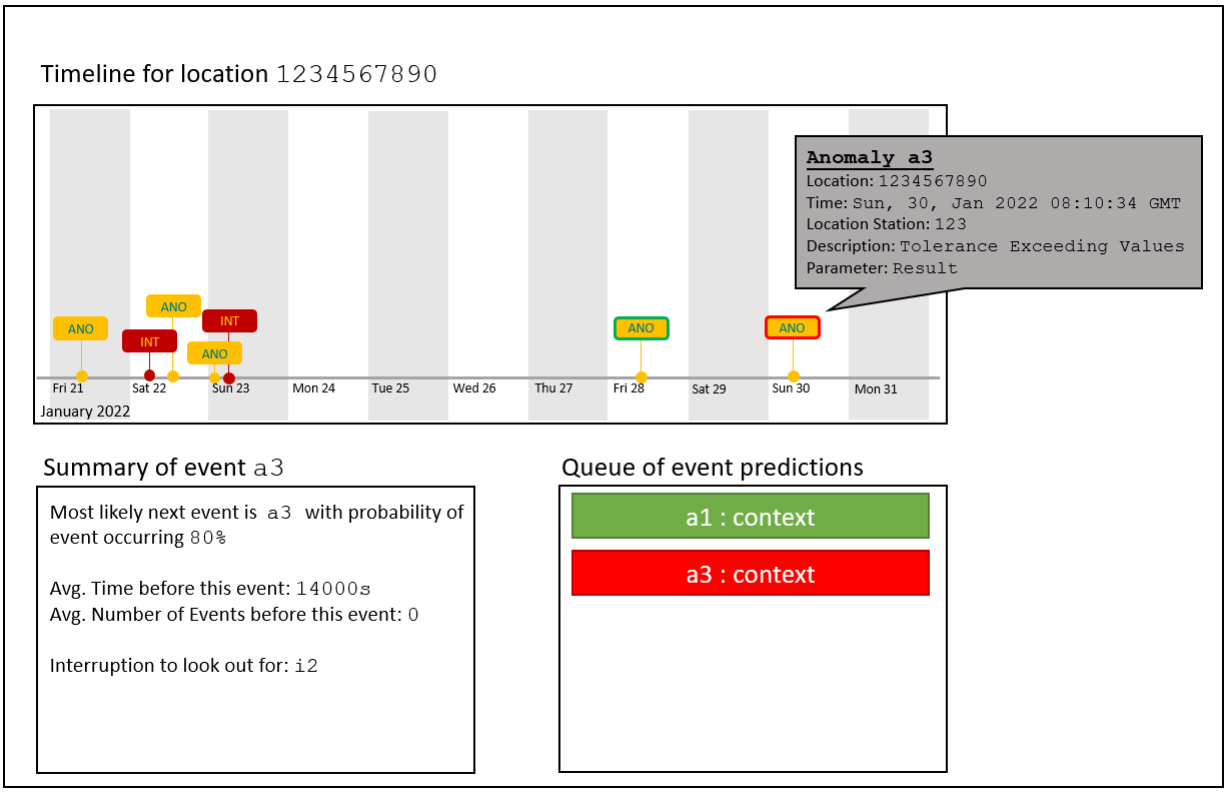

Fig. 8. Example interface of a timeline supported with the PKG, providing contextual data to the user based on occurring events.

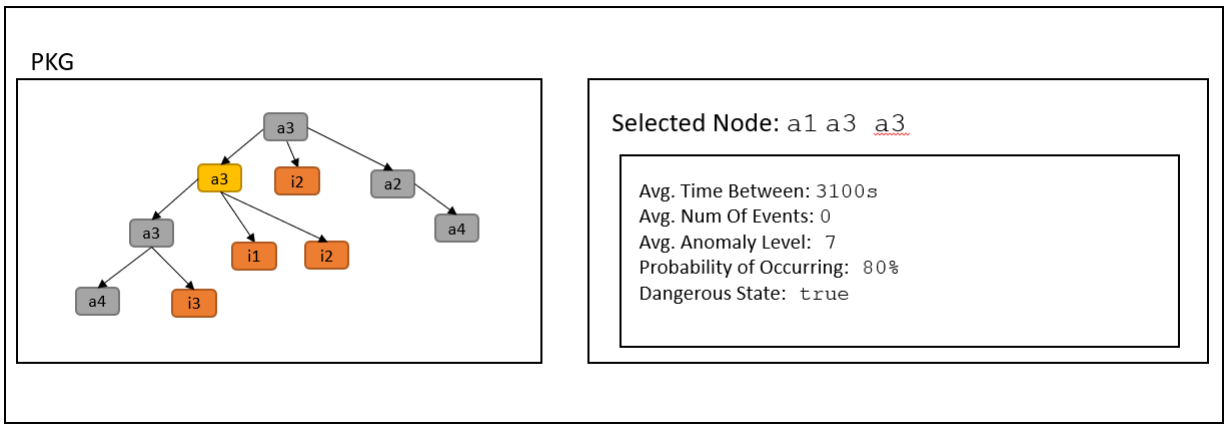

Fig. 9. Example of a PKG Tree represented graphically with accompanying data.

Since the PKG can be represented as a graph, then generic graph-based algorithms can be applied to it to extract the imbued domain knowledge. For example, we can use a search algorithm such as depth-first or breadth-first search to answer more intricate queries. For example, the query of

> *"Given that we have experienced 'a1 a3', is it possible for an anomaly with average anomaly level greater than 6 to occur either in the next 24 hours or as the next expected event?"*

To answer this, we simply need to traverse the PKG from the node of 'a1 a3' to determine if an anomaly directly related to it has average anomaly level greater than 7, or an anomaly, possibly indirectly, related to it meets the time condition. In this case, a simple breadth-first search would return that the second 'a3' satisfies the condition of having an average anomaly level greater than 6, and it is immediately related to it. Thus, the result to this query would be true.

VI. Discussion

In this work, a novel approach for detecting interruptions in a manufacturing setting using previous data points to extract patterns from sequences was developed. Specifically, a pattern

knowledge graph is proposed. This knowledge graph can be used for two purposes: to provide an interactive experience to users to explore the relationship between different anomalies and interruptions in the domain, and to provide explanation to a prediction from a machine learning model.

The first use-case is typical for a user that has experienced some event, or sequence of events. The PKG can be queried to provide insight to which pattern they are experiencing, if there is one. For instance, Figure 9 would be the result of searching for the sequence 'a1 a3'. Although simple, it provides the user with possible futures. They can explore possible futures by selecting nodes or edges to retrieve information about these futures. They can then determine whether certain futures are dangerous, or what events to look out for. Alternatively, a query, similar to the earlier example, could provide the user with more exact knowledge, if they know what they are looking for.

The second use-case combines the PKG technology with machine learning prediction models. This use-case is first explored in Figure 8 with the 'Queue of event predictions window'. Machine learning models can be developed to predict whether an interruption will occur, given an input sequence. This prediction will contribute to the diagnosis of whether an anomaly is deemed dangerous or not. Although the insight to whether an interruption will occur is extremely useful, it does not provide the necessary information of which interruption, or what events may occur leading up to the interruption. This is why the combination of machine learning prediction technologies with the PKG is significant. Take, for example, the input sequence 'a1 a3'. The machine learning prediction model will return either a true or false to whether an interruption will occur. If the output is true, then using the PKG we can assume that 'a2' will not occur next because that particular branch of the tree does not conclude with an interruption. Thus, we should be bracing for either interruption 'i2' or for the anomaly 'a3' which might still eventually conclude with an interruption. Thus, the PKG can be used to provide explainability to the model's prediction with an explorative and interactive interface.

The PKG of this work, compared to other knowledge graphs or machine learning technique, is a unique approach. The PKG is the product of a pipeline which starts with the time-series datasets. Sequences are generated from this data-set so that patterns can be extracted and then weaved together to create the PKG. The PKG is then enhanced with the domain knowledge of the sequences to get knowledge about the individual events of the patterns. In this way, the PKG is not a knowledge graph formed from the insight of domain experts, but rather, from the domain itself. Additionally, it incorporates traits of a Markov chain, in that notion of traversing the PKG when events occur. It is also a set of meet-semilattices, which allows for a formal comparison between two events in any given single PKG Tree. Finally, the PKG Tree is best exemplified when used with machine learning techniques. It is not intended to replace machine learning, as it does not provide a prediction, but rather, it accompanies machine learning. It provides the explainability and traceability that machine learning techniques often lack.

VII. Conclusion and Future Work

The PKG is a tool which is built to accompany machine learning prediction tools. It empowers users with the ability to explore how the events relate to one another, as well as potentially explain the output of a machine learning model. However, there is still open questions with the PKG.

The first source of future work is with developing an automated pipeline so that, as new events occur, new sequences are generated and the PKG is autonomously updated. As more sequences are generated, and more patterns mined, the significance of PKG results will increase. More possible futures will be displayed.

The second source of future work is adapting the PKG to specific locations, or incorporating topologies of manufacturing plants. Currently, the PKG is agnostic to *where* events occur. If a specific event occurs, it will produce the PKG Tree that corresponds with that pattern. However, certain locations might be prone to certain patterns over others. In this case, the PKG should factor in the location of the sequence so that the user is aware that one future is more likely than the other based on location data. Additionally, the sequences that are generated are isolated to only one specific location. However, it is likely the case that an anomaly at one location will have an effect on locations further down the manufacturing line. Thus, the sequences should be generated with the topology of the plant in mind. Since more sequences would be generated, an increase in patterns would be found, and thus, grow the PKG.

References

[1] N. Laptev, S. Amizadeh, and I. Flint, "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*, pp. 1939–1947, 2015.

[2] D. Fensel, U. Şimşek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich, and A. Wahler, "Introduction: what is a knowledge graph?," in *Knowledge Graphs*, pp. 1–10, Springer, 2020.

[3] P. Esling and C. Agon, "Time-series data mining," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–34, 2012.

[4] C. A. Ralanamahatana, J. Lin, D. Gunopulos, E. Keogh, M. Vlachos, and G. Das, "Mining time series data," in *Data mining and knowledge discovery handbook*, pp. 1069–1103, Springer, 2005.

[5] E. S. Dimitrova, M. P. V. Licona, J. McGee, and R. Laubenbacher, "Discretization of time series data," *Journal of Computational Biology*, vol. 17, no. 6, pp. 853–868, 2010.

[6] A. R. Post and J. H. Harrison Jr, "Temporal data mining," *Clinics in Laboratory Medicine*, vol. 28, no. 1, pp. 83–100, 2008.

[7] S. Zolhavarieh, S. Aghabozorgi, and Y. W. Teh, "A review of subsequence time series clustering," *The Scientific World Journal*, vol. 2014, 2014.

[8] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–77, 2017.

[9] K. M. Kumar, P. Srinivas, and C. R. Rao, "Sequential pattern mining with multiple minimum supports by ms-spade," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 5, p. 285, 2012.

[10] R. Boghey and S. Singh, "Sequential pattern mining: A survey on approaches," in *2013 International Conference on Communication Systems and Network Technologies*, pp. 670–674, IEEE, 2013.

[11] D. Maylawati, H. Aulawi, and M. Ramdhani, "The concept of sequential pattern mining for text," in *IOP Conference Series: Materials Science and Engineering*, vol. 434, p. 012042, IOP Publishing, 2018.

[12] G. Aloysius and D. Binu, "An approach to products placement in supermarkets using prefixspan algorithm," *Journal of King Saud University-Computer and Information Sciences*, vol. 25, no. 1, pp. 77–87, 2013.

[13] H. Arnaout and S. Elbassuoni, "Effective searching of rdf knowledge graphs," *Journal of Web Semantics*, vol. 48, pp. 66–84, 2018.

[14] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, *Handbook of markov chain monte carlo*. CRC press, 2011.

[15] J. B. Nation, "Notes on lattice theory," 1998.

[16] Y. Bertot and P. Castéran, *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[17] B. Jia, C. Dong, Z. Chen, K.-C. Chang, N. Sullivan, and G. Chen, "Method and system for pattern discovery and real-time anomaly detection based on knowledge graph," June 9 2020. US Patent 10,679,007.

[18] S. Hans, S. Z. H. Shaikh, R. Ananthanarayanan, D. Saha, A. Aggarwal, G. Singh, P. K. Lohia, M. A. Bhide, and S. Mehta, "Domain aware explainable anomaly and drift detection for multi-variate raw data using a constraint repository," May 3 2022. US Patent 11,321,304.

[19] G. Apostolopoulos, "Graph-based network security threat detection across time and entities," Feb. 12 2019. US Patent 10,205,735.

[20] X. Zhang and K. Yu, "Anomaly detection in time series data using post-processing," June 9 2016. US Patent App. 13/826,994.

[21] R. S. Krajec and A. G. Gounares, "Force directed graph with time series data," Sept. 5 2013. US Patent App. 13/757,598.

[22] A. J. Oliner, J. La, C. Kinross, H. Zhang, J. Leverich, S. Cai, M. Ganea, A. Cruise, T. Boubez, M. Sainani, *et al.*, "Anomaly detection based on relationships between multiple time series," Aug. 6 2019. US Patent 10,375,098.

[23] O. Lassila, R. R. Swick, *et al.*, "Resource description framework (rdf) model and syntax specification," 1998.

[24] D. L. McGuinness, F. Van Harmelen, *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[25] D. Hakkani-Tür, L. Heck, and G. Tur, "Using a knowledge graph and query click logs for unsupervised learning of relation detection," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8327–8331, IEEE, 2013.

[26] U. Brandes, M. Eiglsperger, J. Lerner, and C. Pich, "Graph markup language (graphml)," 2013.

[27] A. Hagberg and D. Conway, "Networkx: Network analysis with python," *URL: https://networkx. github. io*, 2020.

[28] M. E. Smoot, K. Ono, J. Ruscheinski, P.-L. Wang, and T. Ideker, "Cytoscape 2.8: new features for data integration and network visualization," *Bioinformatics*, vol. 27, no. 3, pp. 431–432, 2011.