Log Analysis via Space-time Pattern Matching

Anne Bouillard — *Nokia Bell Labs France*
Marc-Olivier Buob — *Nokia Bell Labs France*
Maxime Raynal — *Université de Grenoble*
Achille Salaün — *Nokia Bell Labs France, Telecom Sud Paris*

***Abstract*—This paper proposes a new methodology inspired from pattern matching and able to find alarm correlations with or without prior knowledge about the monitored system. The data structure can store every observed pattern of correlated alarms by processing logs online. It can be queried to extract the patterns of alarms leading to an arbitrary failure. First, we propose a framework able to represent alarm logs according to spatio-temporal dependencies. Second, we design a new scalable data structure, able to store every observed pattern of alarms, and validate it by simulation. Third, we show how to exploit this data structure for fault diagnosis.**

***Index Terms*—Fault diagnosis, pattern matching, online algorithm.**

I. Introduction

Telecommunication networks are more and more complex: they involve different pieces of equipment and technologies, support heterogeneous and heavy traffic, and have stronger requirements. Network management has then become a complex, yet crucial task. To detect misbehaviors of the system, network elements trigger alarm messages that are aggregated into logs. There can be thousands of alarms fired in a day.

Most fault diagnosis techniques rely on models built from prior knowledge or by processing and analyzing alarm logs.

A quick overview of fault diagnosis techniques:

a) Chronicles: they are timed causal patterns [10], presented in the form of acyclic graphs, Petri nets or logical functions. Some works focus on the construction of chronicles by processing alarm logs, thus on searching frequent patterns of a log [7], [9], or a collection of logs [5], [18].

b) Bayesian networks: They are usually built using prior knowledge of the system. To cope with scalability issues, various techniques have been developed, like introducing case-based reasoning [3] or clustering [4].

c) Clustering: [2], [13] group alarms sharing similar attributes (such as their location or message); [12], [16] cluster non-overlapping blocks of consecutive alarms by relying on PCA; and [6] groups alarm blocks sharing similar patterns using a timed version of the Smith-Wasserman algorithm [17], but an expert is required to interpret each cluster.

d) Learning techniques: [11] uses random forests in order to build models from a collection of alarm logs.

All these approaches are based on some statistics and focus on frequent patterns. On the contrary, our aim is to help for fault diagnosis, even for rare patterns of events. To do so, we need to store all the possible patterns. Our proposal is inspired from pattern-matching data structures. [1] provides a tree storing a collection of patterns and define suffix links so that the presence of these patterns in a text can be efficiently detected. Works related to building a suffix-tree from a given input text are also relevant since logs of alarms can be modeled by a word. Building such trees can be done in linear time [14], [19], [20].

Contributions: In this paper, we propose a data structure, called DIG-DAG, able to store all the potential chains of correlations based on the temporal and spatial characteristics of alarms, and possibly on the prior knowledge of experts. Our main contributions are:

1) Defining a framework to represent spatio-temporal dependencies in alarm logs (cf. §II).
2) Designing a scalable data structure storing all the observed sequences of correlated alarms and the corresponding online update algorithm (cf. §III).
3) Exploiting this data structure for fault diagnosis and summarizing a log of alarm (cf. §IV).
4) Evaluating the efficiency of our approach through simulations on real data and toy examples (cf. §V).

II. Modeling logs of alarms

An alarm log is a collection of events appearing in chronological order. Our goal is to find all potential sequences of correlated alarms based on the observation of alarm logs, that are processed online.

In the rest of the paper, we use formal language notations. Given a finite set Σ, Σ^* denotes the set of the finite sequences, called words, on Σ, and ϵ is the empty word. If w is a word, $|w|_a$ is the number of occurrences of a in w and $w_{\leq k}$ is the prefix of w of length k.

A. Modeling segmented events

An event $e = (a, [s,t])$ is described by three parameters:

- The interval $[s,t]$ describes when the event is occurring: the event starts at time s and ends at time t.
- The symbol a represents the alarm type and its attributes. A spatial attribute can indicate where the event is occurring. We denote by Σ the set of all symbols representing this information. By a slight abuse of language, we will call *spatial* all the information that is not temporal.

Let $\mathcal{E} = (e_i)_{i \in \{1,\dots,n\}}$ be the collection of events of a log. For all $i \in \{1,\dots,n\}$, let us denote $e_i = (a_i, [s_i, t_i])$. We make the following assumptions:

(H1) All the starting and ending times of events are distinct.
(H2) Two events with the same symbol have disjoint emission period: $[s_i, t_i] \cap [s_j, t_j] \neq \emptyset \Rightarrow a_i \neq a_j$ or $i = j$.

978-3-903176-14-0 © 2018 IFIP

These assumptions are not restrictive: we can have a total order on the emission dates considering the event order in the log. Overlapping events of the same type are aggregated.

B. Modeling potential causalities between events

The temporal and the spatial information can be used to decide whether two events have a potential relation. Let $e = (a, [s, t])$ and $e' = (a', [s', t'])$ be two events.

- The *temporal relation* $\mathcal{R}_t$ tests whether two events are temporally related: $e\mathcal{R}_t e'$ iff $s' \in [s, t]$.
- The *spatial relation* $\mathcal{R}_s$ tests if two events are spatially related, given a subset $\mathcal{C} \subseteq \Sigma^2$ representing valid spatial causalities: $e\mathcal{R}_s e'$ iff $aa' \in \mathcal{C}$.

Definition 1. *An event e is a potential cause of event e' if $e\mathcal{R}_t e' \wedge e\mathcal{R}_s e'$. In this case, we write $e \to e'$.*

C. Two representations for the aggregation of alarm logs

1) Directed interval graph (DIG): We have defined events and a potential causality relation between them. This leads to a first natural graphical representation for an alarm log: let $\mathcal{E}$ be the set of events of the log, and $\lambda : \mathcal{E} \to \Sigma;\ (a, [s, t]) \mapsto a$ be a labeling function. The triple $(\mathcal{E}, \to, \lambda)$ is a labeled directed graph, called DIG, whose nodes are the events, labeled by their symbol, and the arcs are exactly the potential causalities. Due to the temporal relation, this graph is acyclic.

2) Word representation: As we want an online construction of our data structure, we need to represent the start and end of events. To this end, we introduce the alphabet $\overline{\Sigma}$, a disjoint copy of Σ. Each event $e = (a, [s, t])$ emits a symbol a at time s and a symbol $\bar{a}$ at time t. The word corresponding to a collection $\mathcal{E}$ of events is the word composed of all the emitted symbols in their emission order. Note that this word is independent of the relation $\mathcal{C}$ and represents only the potential temporal causalities. Because of assumption (H1), this word is uniquely defined, and the following properties hold:

(P1) $\forall a \in \Sigma,\ |w|_a = |w|_{\overline{a}}$;
(P2) $\forall j \in \{1, \ldots, |w|\},\ |w_{\leq j}|_{\overline{a}} \leq |w_{\leq j}|_a \leq |w_{\leq j}|_{\overline{a}} + 1$.

The right inequality of the second property is a direct consequence of assumption (H2): alarms with the same symbol do not overlap.

Definition 2. *A word is* correct *if it satisfies properties (P1) and (P2) and is a* correct prefix *if it satisfies property (P2). We denote by $\mathcal{L}$ the language of correct prefixes.*

One can show that every DIG has unique word representation. Conversely, one can show that a canonical DIG can be built from a correct word and $\mathcal{C}$.

D. The language of words to store

Our aim now is to store all the words corresponding to the paths in a DIG. The word of a path is the sequence of labels on the nodes along this path. In Fig. 1a, the set of words is $\{\epsilon, a, b, c, d, cb, ba, cd, db, cba, dba, cdb, cdba\}$.

Definition 3. *Let $w = w_1 \cdots w_n \in \mathcal{L}$ be a correct prefix. the sub-word $w_{\phi(1)} w_{\phi(2)} \cdots w_{\phi(k)} \in \Sigma^k$ of w is admissible if:*

- $\forall j \in \{1, \ldots, k-1\},\ w_{\phi(j)} w_{\phi(j+1)} \in \mathcal{C}$;
- $\forall j \in \{1, \ldots, k-1\},\ |w_{\phi(j)} \cdots w_{\phi(j+1)}|_{\overline{w_{\phi(j)}}} = 0$.

The language of admissible sub-words of w *is denoted* $\mathcal{L}(w)$.

Theorem 1. *Let $\mathcal{E}$ be a collection of events, and w be the correct word built from it. Then the set of labeled paths in $(\mathcal{E}, \to, \lambda)$ is exactly $\mathcal{L}(w)$.*

A recursive construction of $\mathcal{L}(w)$: One can distinguish two types of admissible sub-words: the *continuable* and non-continuable sub-words. Informally, the continuable sub-words, $\mathcal{L}^c(w)$, are those that can potentially be extended when considering the next symbols of w.

Theorem 2. $\mathcal{L}(\epsilon) = \mathcal{L}^c(\epsilon) = \{\epsilon\}$. *For all $w \in \mathcal{L}$ and $a \in \Sigma$,*

- *if $wa \in \mathcal{L}$, then $\mathcal{L}^c(wa) = \mathcal{L}^c(w) \cup (\mathcal{L}^c(w)a \cap \Sigma^*\mathcal{C})$ and $\mathcal{L}(wa) = \mathcal{L}(w) \cup (\mathcal{L}^c(w)a \cap \Sigma^*\mathcal{C})$.*
- *if $w\bar{a} \in \mathcal{L}$, $\mathcal{L}(w\bar{a}) = \mathcal{L}(w)$ and $\mathcal{L}^c(w\bar{a}) = \mathcal{L}^c(w) \setminus \Sigma^* a$.*

III. Storing sequences of alarms

We now define an efficient data structure storing all admissible sub-words $\mathcal{L}(w)$ of a correct prefix w. The construction should be online, discovering symbols of w from left to right. We call it DIG-DAG for *directed interval graph - directed acyclic graph*

A. The DIG-DAG data structure

A natural way to store a set of words is to use a graph, where words can be read on the paths. To this end, we propose a data structure $\mathcal{D} = (V, E, \lambda, \mathcal{A})$ with the following properties:

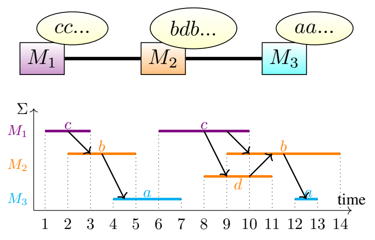
(D1) (V, E) is a directed acyclic graph with node set V and arc set E, with a unique node q_0 with in-degree 0.
(D2) λ is a labeling function $\lambda : V \to \Sigma \cup \{\epsilon\}$ with $\lambda(q_0) = \epsilon$ and $\lambda(q) \in \Sigma$ for any other node.
(D3) $\mathcal{A} \subseteq V$ is the set of *active nodes*, and $q_0 \in \mathcal{A}$ always holds. We denote by $\mathcal{A}_a$ the set of the active nodes labeled by a.
(D4) For each symbol a, a node p has at most one successor q with label a, in which case we denote $q = g(p, a)$. The arc (p, q) is called an a-transition;
(D5) The set of words of all paths from the root is exactly $\mathcal{L}(w)$, and the set of words of all paths from the root to any active node is exactly $\mathcal{L}^c(w)$.

Definition 4. *A DIG-DAG for a correct prefix w is a data structure $\mathcal{D} = (V, E, \lambda, \mathcal{A})$ satisfying properties (D1)–(D5).*

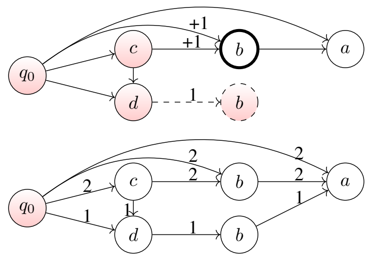
This data structure is very close to the notion of automaton. Here labels are on the nodes instead of on the arcs, g plays the role of the transition function, q_0 is the initial state. Property (D4) states that the automaton is deterministic. From this viewpoint, $\mathcal{L}(w)$ is the language recognized when every state is final and $\mathcal{L}^c(w)$ is the language recognized when the final states set is $\mathcal{A}$.

B. Online construction of a compact DIG-DAG

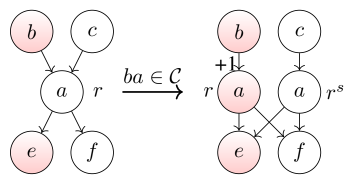
Multiple instances of DIG-DAG satisfy the above properties for a given word w. Our aim is to build a compact structure that can be efficiently updated at the reception of each symbol.



(a) An example of DIG.



(b) DIG-DAG construction



(c) Split operation

Fig. 1: (a) top: a network involving three machines. The spatial causality is based on the topology, $\mathcal{C} = \Sigma^2 \setminus \{ac, ca\}$; (a) bottom: the DIG corresponding to a collection of 7 events. Arc (a, c) is discarded according to the relation $\mathcal{C}$. The word corresponding to the DIG is $cb\bar{c}a\bar{b}c\bar{a}db\bar{c}\bar{d}a\bar{a}\bar{b}$. Consider $w = cb\bar{c}a\bar{b}c\bar{a}d$ its 8 first symbols, we have $\mathcal{L}(w) = \{\epsilon, a, b, c, d, ba, cb, cd, cba\}$ and $\mathcal{L}^c(w) = \{\epsilon, c, d, cd\}$. Th. 2 gives $\mathcal{L}^c(wb) = \{\epsilon, b, c, d, cb, db, cd, cdb\}$ and $\mathcal{L}(wb) = \{\epsilon, a, b, c, d, ba, cb, cd, db, cdb, cba\}$.
(b) top: The DIG-DAG evolution from word w to wb. The (bold) node labeled b has only active (colored) predecessors and $cb \in \mathcal{C}$, so this node is activated. The node with label d has no b-transition, so a new node is created (dashed); (b) bottom: the DIG-DAG of the correct word. Weights their update are explained in §IV. (c) Split operation, with update of the weights

UPDATE$(\mathcal{D}, a)$: For sake of clarity, we decompose the update procedure in three steps, even if they can be implemented in a single step. Suppose that $\mathcal{A}$ is the set of active nodes and that the next symbol in the word is $a \in \Sigma$.

a) Split: Let q be an active node such that $r = g(q, a)$. Note that necessarily $\lambda(q)a \in \mathcal{C}$. Directly activating r would induce inconsistencies in the set of stored words if r also has a predecessor p with $p \notin \mathcal{A}$. To avoid this, we split the node into two nodes, r and a new node r^s. The predecessors p of r all satisfy $p \in \mathcal{A}$, and the predecessors p of r^s all satisfy $p \notin \mathcal{A}$. Node r^s has the same successors as r. This construction is illustrated on Fig. 1c.

b) Extend and activate: For each active node q with $\lambda(q)a \in \mathcal{C}$, after the split operation, we activate $g(q, a)$ if it exists, or otherwise create a new node successor of q labeled by a. This step is similar to the DIG-trie update.

c) Merge: Merge all the active leaves with label a into one single node. The language of correct sub-words is not modified by this operation.

Theorem 3. *Let $w \in (\Sigma \cup \overline{\Sigma})^*$ and $a \in \Sigma \cup \overline{\Sigma}$. If wa is a correct prefix and if $\mathcal{D}$ is a DIG-DAG for w, then the* UPDATE$(\mathcal{D}, a)$ *is a DIG-DAG for wa.*

C. Minimal DIG-DAG

The DIG-DAG is generally not minimal: in Fig. 1b (bottom), the two nodes labeled by b could be merged. Minimization techniques from automata theory [8], [15] can be adapted to our context with very minor modifications. We found that the DIG-DAG construction scales rather well compared to the minimal DIG-DAG. Note that even if there are efficient minimization algorithms in this case, their execution time is significant, which is a reason why we do not use them.

IV. Finding causes of a failure

In this section, we present some preliminary works on exploiting the DIG-DAG data structure to find the most relevant causalities, hence the causes of a failure. A first step, done during the DIG-DAG construction is to *weight* the data structure. The second step is to exploit these weights to quantify the correlation between a given context and an alarm.

A. Weighting the DIG-DAG

Let q be a node and a be a symbol. We call *context of* q the set $\mathcal{P}(q)$ of words labeling paths from q_0 to q, and *context of* qa the set $\mathcal{P}(q)a$. We denote by $n_{q,a}$ the number of times the context $\mathcal{P}(q)a$ appears in the DIG, that is, the number of times there is a node labeled by a that has paths to it labeled by the words of $\mathcal{P}(q)a$. For example, in Fig. 1a, the context $\{cba, ba, a\}$ appears twice, and the context $\{cdba, bda\}$ appears once. In this paragraph we show how to maintain the weight $n_{q,a}$ assigned to each arc $(q, g(q, a))$ in an online fashion. In arc (p, q), we call q its *head*.

Intuitively, the following properties should hold:

- The weight of a newly created arc is 1, as it corresponds to a pattern seen for the first time;
- The weight of an arc is incremented by 1 whenever its head is re-activated.

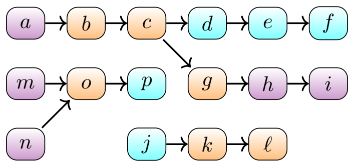
Split and merge operations only imply copying arcs and their weights. Minimizing weighted DIG-DAGs is not possible since arcs with different weights could be merged.

B. Analysis of the log

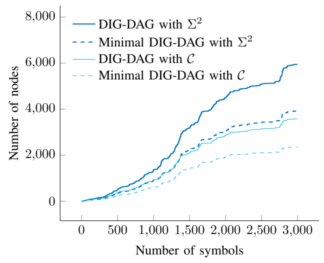
We now illustrate how these weights can be exploited. Our goal is to extract the most likely contexts explaining an alarm. Let q be a node of the DIG-DAG and $a \in \Sigma$ such that $g(q, a)$ exists. Let us define the ratio $n_{q,a}/n_{q_0,a}$. Without rigorously defining the probability $\mathbf{P}$, we can give the interpretation

$$r_{q,a} = \frac{n_{q,a}}{n_{q_0,a}} \approx \frac{\mathbf{P}(q, a)}{\mathbf{P}(a)} = \mathbf{P}(q \mid a),$$

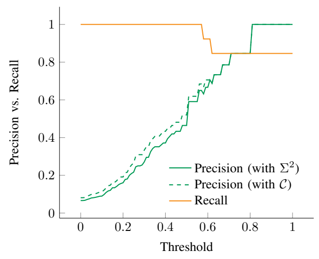
and this ratio represents the probability that, given the observation of a, the context $\mathcal{P}(q)$ appeared before. If this ratio approaches 1, then a is strongly correlated with $\mathcal{P}(q)$.



(a) Model of the toy example presented in §V.



(b) Impact of the spatial relation $\mathcal{C}$.



(c) Precision and recall.

Fig. 2: Scalability of the DIG-DAG structures.

1) Finding correlations: One can infer a correlation graph by exploring the DIG-DAG from the root's successors and keeping only arcs above a given threshold ρ. Consider $\rho = 0.8$ and the DIG-DAG from Fig. 1b (bottom). The correlation graph keeps the chain $c \to b \to a$ and arc $c \to d$.

2) Fault diagnosis: A DIG-DAG can also be used to find the origin of a failure related to an alarm a. At reception of a, after updating the DIG-DAG, one might consider the relevant paths from the root to the newly activated nodes. Those paths are all possible explanations of a.

V. Simulations

We now evaluate the performances of our solution. We first performed our experiments on a log issued from an experimental platform, where failures are manually triggered. It contains 1409 events (hence 2818 symbols) involving 27 machines (physical or virtual), 29 types of alarms, and resulting to an alphabet of size 73. The underlying system topology is unknown. The DIG-DAG corresponding to the whole log has 1726 nodes the construction took 16s. The size of the DIG-DAG differs from that of the minimal DIG-DAG by a small multiplicative constant (1.18). The consistency of our results was validated by experts.

As we do not have ground-truth about the real dataset, we implemented a discrete event simulator to produce artificial logs. Informally, our log generator is Markovian and relies on a causal graph (see Fig. 2a), where a symbol is assigned to each node. Symbols a, h, i, m, n are emitted from M_1, symbols b, c, g, k, ℓ, o, from M_2 and d, e, f, j, p from M_3, and the topology is that of Fig. 1a. The relation $\mathcal{C}$ is built from this topology ($ad, ae, \ldots \notin \mathcal{C}$), which discards 21% of the potential causalities.

Fig. 2b highlights the impact of the expert knowledge on scalability. Here, we considered the DIG-DAG construction of an artificial dataset when $\mathcal{C}$ or Σ^2 is used. We can observe that the DIG-DAG is 40% smaller with $\mathcal{C}$, as reducing the potential correlations limits the growth of the DIG-DAG. The gain on the minimized DIG-DAG is the same. These curves also show that the DIG-DAG scales very well on this toy example, both in terms of memory and processing time.

One of the strengths of our approach is to store chains of causality. We now illustrate the ability of the DIG-DAG to infer the causality model of Fig. 2a. We use the approach of §IV-B and extract a sub-graph from the DIG-DAG according a given threshold ρ. We denote by $F(\rho)$ the arc set of the extracted DIG-DAG and by G that of the causality graph.

Then, we compare how the extracted DIG-DAG and the causality graph overlap. The accuracy is evaluated according to the *precision* P and *recall* R performance metrics:

$$P(\rho) = \frac{|G \cap F(\rho)|}{|F(\rho)|} \quad \text{and} \quad R(\rho) = \frac{|G \cap F(\rho)|}{|G|}.$$

Informally, the recall is the proportion of missed arcs, and the precision is the proportion of arcs not in the causality graph.

We build a DIG-DAG based on a log of length 1000 and then measure the precision and the recall as functions ρ.

The precision increases with ρ, while the recall decreases. On Fig. 2c, the precision can reach 65% with a recall equal to 1, by choosing $\rho = 0.55$. Thus, the ratio $r_{q,a}$ is indicative of the relevance of the potential observed causalities.

The recall decreases for high values of ρ, hence relevant dependencies will be discarded. In our example, as o can be consequence either of m or of n, alarm o can appear under two distinct contexts, thus ratios $r_{g(q_0,m),o}$ and $r_{g(q_0,n),o}$ are slightly above 0.5. Note that $r_{g(q_0,u),v} = 1$ for all other implications $u \to v$, $v \neq o$, in our toy model.

Adding spatial information $\mathcal{C}$ increases the precision only for $\rho \leq 0.6$. Actually, most false positives for higher ρ are shortcuts of arcs present in the model, and spatial filtering has little impact on it: if prior knowledge slightly improves performances in this example, its main benefits reside in improving the scalability and discarding irrelevant correlations.

VI. Conclusion

In this article, we presented a pattern matching approach dedicated to fault diagnosis. We process *online* logs of alarms and build a graph structure, called DIG-DAG. It stores all the *spatio-temporal* patterns. It can be used to summarize large logs of alarms involving multiple components and to extract the patterns of alarms raised before a given outage.

As future work, we plan to generalize the notion of correlation (e.g. $a \wedge b \to c$) to detect more elaborated patterns. Extending the notion of symbol to embed numerical measurements (and adapting split and merge primitives consequently) could also be an interesting improvement. At last, we believe that using our solution after having filtered noise from its input could improve its accuracy.

References

[1] A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975.

[2] S. O. Al-Mamory and H. Zhang. Intrusion detection alarms reduction using root cause analysis and clustering. *Computer Communications*, 32(2):419–430, 2009.

[3] L. Bennacer, Y. Amirat, A. Chibani, A. Mellouk, and L. Ciavaglia. Self-diagnosis technique for virtual private networks combining bayesian networks and case-based reasoning. 12:354–366, 01 2015.

[4] L. Bennacer, L. Ciavaglia, S. Ghamri-Doudane, A. Chibani, Y. Amirat, and A. Mellouk. Scalable and fast root cause analysis using inter cluster inference. In *Proceedings of IEEE International Conference on Communications, ICC*, pages 3563–3568, 2013.

[5] J. V. Capacho, A. Subias, L. Travé-Massuyès, and F. Jimenez. Alarm management via temporal pattern learning. *Engineering Applications of Artificial Intelligence*, 65:506 – 516, 2017.

[6] Y. Cheng, I. Izadi, and T. Chen. Pattern matching of alarm flood sequences by a modified Smith–Waterman algorithm. *Chemical engineering research and design*, 91(6):1085–1094, 2013.

[7] D. Cram, B. Mathern, and A. Mille. A complete chronicle discovery approach: application to activity analysis. *Expert Systems*, 29(4):321–346, 2012.

[8] J. Daciuk, S. Mihov, B. W. Watson, and R. E. Watson. Incremental construction of minimal acyclic finite-state automata. *Computational linguistics*, 26(1):3–16, 2000.

[9] C. Dousson and T. V. Duong. Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In *IJCAI*, volume 99, pages 620–626, 1999.

[10] C. Dousson, P. Gaborit, and M. Ghallab. Situation recognition: Representation and algorithms. In *IJCAI: International Joint Conference on Artificial Intelligence*, pages 166–174, 01 1993.

[11] J. M. N. Gonzalez, J. A. Jimenez, J. C. D. Lopez, and H. A. P. G. Root cause analysis of network failures using machine learning and summarization techniques. *IEEE Communications Magazine*, 55(9):126–131, 2017.

[12] M. C. Johannesmeyer, A. Singhal, and D. E. Seborg. Pattern matching in historical data. *AIChE journal*, 48(9):2022–2038, 2002.

[13] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM transactions on information and system security (TISSEC)*, 6(4):443–471, 2003.

[14] E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, Apr. 1976.

[15] D. Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1):181–189, 1992.

[16] A. Singhal and D. E. Seborg. Matching patterns from historical data using PCA and distance similarity factors. In *Proceedings of the American Control Conference*, volume 2, pages 1759–1764. IEEE, 2001.

[17] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[18] A. Subias, L. Travé-Massuyès, and E. Le Corronc. Learning chronicles signing multiple scenario instances. *IFAC Proceedings Volumes*, 47(3):10397–10402, 2014.

[19] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[20] P. Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (swat 1973)*, pages 1–11, Oct 1973.