Kalpa Publications in Computing

Volume 4, 2018, Pages 1–11

28th International Workshop on
Principles of Diagnosis (DX'17)

A Learning Algorithm for Episodes

Tom Obry[1,2]

[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
tobry@laas.fr
[2] ACTIA, 5 Rue Jorge Semprun, 31432 Toulouse, France
tom.obry@actia.fr

Abstract

Sequences of events describing the behavior and actions of agents or systems can be collected in several domains. An episode is a collection of events that occur in a given partial order. By performing a recognition of recurrent episodes in several sequences and comparing them, it is possible to determine a pattern common to all the sequences. In this paper, we propose an approach to recognize episodes that are common in a set of event sequences. The method described is applied to the automotive domain for learning diagnosis procedures.

1 Introduction

In the automotive domain, as vehicles become increasingly complex, it becomes difficult to efficiently train garage mechanics in front of the ever growing technology. Faults become more and more difficult to diagnose and training courses multiply to manage new failures. The purpose of this work is to improve the daily work of garage mechanics by providing them with assistance in the diagnosis and repair tasks.

A diagnosis session corresponds to a set of tests and actions performed by the mechanics via a diagnostic tool performed on a vehicle with a breakdown. Each diagnosis session performed is saved in the form of a trace. The goal is to compare all of these diagnosis traces corresponding to the same symptoms to extract the common actions. The aim of this work is to determine the most common action plan that successfully diagnoses a breakdown or that carries out a repair. We propose an approach based discovery and learning patterns [1] [2]. It is inspired by [3] [4] that proposes an algorithm to learn chronicles. Chronicles are temporal patterns based on a set of events and temporal constraints relating their occurence dates. They are used to represent the dynamic behavior of complex systems in an event-based fashion and have found several applications in the supervision and diagnostic field [5, 3, 6]. In our case, the occurence dates are not recorded in the traces that are given as input sequences for the learning algorithm. The reason is that actions performed by the mechanics are not constrained over time. This is why we turned to a particular type of chronicles called "episodes" [7] [8]. Episodes just specify the precedence order of events.

This paper is organized as followed. Section 2 introduces the notion of episode. In the section 3, a new algorithm to learn episodes (LAE) is described. Section 4 presents the test and results. Section 5 presents a standard algorithm for episode learning and a comparison with the new algorithm. Finally, section 6 concludes with the perspectives of this work. [1]

[1]This work has been supported by the company ACTIA, one of the market leaders for diagnostic tools in the automobile

2 Basic concepts

We consider the input as a sequence of events, where each event has an associated date of occurrence. Given a set E of $event\ types$, an $event$ is a pair (e_i, t_i), where $e_i \in E$ is an event type and t is an integer, the (occurrence) $date$ of the event.

Definition 1. *An $event\ sequence$ s on E is a triple (s, T_s, T_e)*

$$s =< (e_1, t_1), (e_2, t_2), ..., (e_n, t_n) > \tag{1}$$

is an ordered sequence of events such that $e_i \in E$ for all $i = 1, ..., n$, and $t_i \leq t_{i+1}$ for all $i = 1, ..., n-1$. Further on, T_s and T_e are integers: T_s is called the starting time and T_e the ending time, and $T_s \leq t_i \leq T_e$ for all $i = 1, ..., n$ [7].

Episodes are defined as a collection of events that occur frequently in a sequence. An episode is a set of partially or totally ordered event types [7].

Definition 2. *An episode $\psi = (V, \leq, g)$ is a set of nodes V, an ordered partial $\leq$ on V and a mapping $g : V \rightarrow E$ which associates a node with an event type.*

The interpretation of an episode means that the events of $g(V)$ must appear in the order described by $\leq$.
The difference with chronicles is at the level of the temporal bounds: the intervals in episodes are not bounded. This makes it possible to consider only the notion of precedence between the events: $t_j - t_i \in [0, +\infty]$ or $t_j - t_i \in [-\infty, 0]$.

Definition 3. *An episode $\beta = (V', \leq', g')$ is a sub-episode of $\alpha = (V, \leq, g)$, denoted $\beta \leq \alpha$ if there exists an injective mapping $f: V' \rightarrow V$ such that $g'(v) = g(f(v))$ for all $v \in V'$ and for all $v, w \in V'$ with $v \leq' w$ also $f(v) \leq g(w)$.*

An episode α is a super-episode of β if and only if $\beta \leq \alpha$. There are two classes of episodes: serial episodes and parallel episodes.

Definition 4. *An episode α is serial if the relation $\leq$ is a total order (i.e., $x \leq y$ or $y \leq x$ for all x, $y \in V$). If the partial order $\leq$ is trivial (i.e., $x \not\leq y$ for all x, $y \in V$ such that $x \neq y$), the episode is parallel [9].*

domain. This paper is done as part of a Ph.D supervised by Louise Travé-Massuyès and Audine Subias.

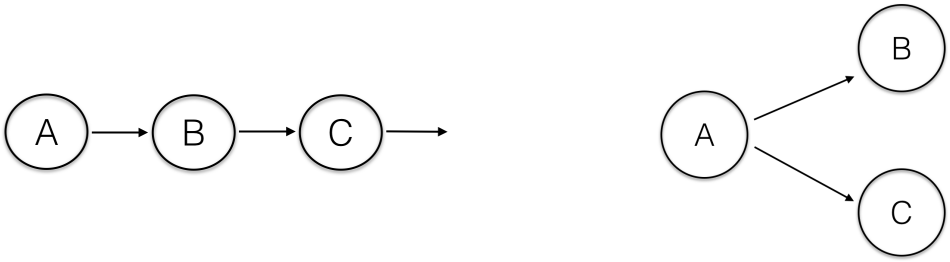

Figure 1: Example of a serial episode and a parallel episode

Figure 1 shows two examples of episodes. The episode on the left is a serial episode because the order is total between A, B and C. The other episode is parallel. Indeed, there is no constraint on the partial order on the event types A, B or C [7]. Episodes issued from the diagnostic tool traces are parallel episodes because the order between the event types is not total. This means that the order of the actions for diagnose a vehicle is not total.

3 Learning Algorithm for Episodes (LAE)

We present in this section, a new algorithm which can compute a set of parallel episodes from a set of input sequences. The solution includes events that are common to all input sequences. We detail the different steps that make up the algorithm. For a better comprehension of the steps, we use 3 sequences, $s_1 \in S$, $s_2 \in S$ and $s_3 \in S$ (where S is the set of the input sequences):

$$s_1 =< (A,1),(B,2),(C,3),(D,4),(A,5) > \tag{2}$$

$$s_2 =< (A,1),(B,2),(C,3),(D,4),(E,5),(A,6) > \tag{3}$$

$$s_3 =< (B,1),(C,2),(A,3),(D,4),(A,5),(E,6) > \tag{4}$$

The sequences are stored in a table. Each cell contains 2 other cells, where the first contains the event types and the second, the dates of occurrence.

3.1 Searching for common events

The first step consists in sorting the events that are common and those that are not. Common events are searched in the input sequences then, sequences of entries are modified to keep only the common events. In this context, events are different screens used during the diagnosis session. Experienced mechanics receive training on how to diagnose a vehicle. They know how to use the diagnosis tool and which screens to navigate. The events that are not in all input sequences are considered as beginner mechanic's actions.

A	B	C	D

Table 1: List Com of common events in s_1, s_2 et s_3

Table 1 represents all the common event types located in S. Here, the common event types are A, B, C and D. Since event E is not present in all the sequences, it is not present in the list Com.
With these common events, events that are not in the initial sequences are removed. The k^{th} event e_k of the i^{th} sequence s_i is compared with the j^{th} event e_j of the collection of common events Com[j]. When the two events compared are identical, the algorithm indexes the event in the first cell and the date of occurrence in the second cell on s_i.

s_1		s_2		s_3	
ABCDA	12345	ABCDA	12346	BCADA	12345

Table 2: s_1, s_2 et s_3 with only common events

Table 2 summarizes, for each sequence, the event types that are common in all sequences and their occurrence positions in the sequence. All the events located in s_1 are in $Com[1]$ because they are present in all the sequences. The event type E is not present in the s_2 and s_3 because E is not present in all the sequences.

3.2 Computing the number of occurrences of an event type in n sequences and computing the pairs of events

In this step, the number of occurrences of each event type is found in the n input sequences. These information are stored in a table called Set. It is composed of k columns and i rows, where k corresponds to the event type and i, the input sequence number. Each cell is composed of 2 more cells. The first contains the frequency of e_k located in the i^{th} sequence. The second cell contains a vector of "0" and "1". "1" indicates the position of the event e_k in the sequence i.

A		B		C		D	
2	[100010]	1	[010000]	1	[001000]	1	[000100]
2	[100001]	1	[010000]	1	[001000]	1	[000100]
2	[001010]	1	[100000]	1	[010000]	1	[000100]

Table 3: Synthesis of the information in Set

Table 3 shows the event types common in Set, their occurrence positions for each sequence and the position of each event in each sequence.
To determine the maximal frequency of each event, noted $Freq(e_k)$, the minimal number of occurrences of each event, noted $min(Occ(e_k))$, for each sequence s is required.

$$Freq(e_k) = min(Occ(e_k) \in s, s \in S, \forall s) \tag{5}$$

Each frequency is stored in a table $Freq$, under the format (e_k, $Freq(e_k)$).

A	B	C	D
2	1	1	1

Table 4: Maximal frequencies in the table $Freq$ of each event types in s_1, s_2, s_3

In the table 4, we can see the maximal frequency of each common event type. In this example, this information equals 2 for the event A and 1 for the others event types. To calculate the pairs available in each sequence, all combinaisons of $Freq$[i] with $Freq$[j] must be performed.

AB	AC	AD	BC	BD	CD
2	2	2	1	1	1

Table 5: Set of pairs available in sequences s_1, s_2, s_3

All the pairs and their maximal frequencies in S are presented in the table 5. In this case, the maximal frequency for pairs (A, B), (A, C) and (A, D) is 2 and for pairs (B, C), (B, D) and (C, D), the value of the maximal frequency is 1.

3.3 Determination of the direction of the pairs

We are going to explain, for each pair of event types, a relation of unique precedence, that is to say present on the n input sequences. In other words, it is a matter of determining the $direction$ of a pair of events. Let's take our 3 examples:

$$s_1 =< (A,1),(B,2),(C,3),(D,4),(A,5) > \tag{6}$$

$$s_2 =< (A,1),(B,2),(C,3),(D,4),(E,5),(A,6) > \tag{7}$$

$$s_3 =< (B,1),(C,2),(A,3),(D,4),(A,5),(E,6) > \tag{8}$$

For the pair (A, B), A is before and after B in s_1 and s_2. The label "$\leftrightarrow$" is allocated to the pair (A,B) to indicate that the pair (A, B) can occur in any order, i.e. A can precede B or B can precede A. For example, the pair A $\leftrightarrow$ B means it is possible to go to the event B from A and return to the event A. Pairs (B, C), (B, D) and (C, D) keep always the same direction in all sequences. The label is given for those two pairs of events "$\rightarrow$" because C always follows B, D is always after B and D always occurs after C. To summarize, the label "$\rightarrow$" can only be assigned if A is followed by B in all the sequences (conversely, "$\leftarrow$" for B is followed by A in the all the sequences). The label "$\leftrightarrow$" is assigned to a pair:

- if there is a label "$\rightarrow$" and a label "$\leftarrow$" in at least one sequence. It indicates that both directions are possible.
- if the two directions are possible in the same sequence.

The table Set is organized by events in columns indexed k. Every event e_k gives rise to a macro-column with two subcolumns indexed k_1 and k_2. The lines indexed i correspond to the input sequences. The vectors of "0" and "1", located in the table Set are studied because they contain the occurrence positions of the events. $Set(i, k_1)$ provides the number of occurence of the event of column k in the sequence i and $Set(i, k_2)$ is the vector from which we can retrieve the event position in the sequence. From this information, we can obtain the direction of every pair in every sequence. To save the direction of the pairs, the variable $Inter$ is defined and can take several values:

- $Inter_i(k, l)$ =+1 if e_l follows e_k in the i^{th} sequence,
- $Inter_i(k, l)$ =-1 if e_k follows e_l in the i^{th} sequence,
- $Inter_i(k, l)$ = 0 if e_k precedes and follows e_l in the i^{th} sequence.

To find the direction of the pairs in the n sequences, we proceed as follows:

- if $\sum_{i=1}^{n} Inter_i(k, l) = n$, the label assigned to the pair (e_k, e_l) is "$\rightarrow$",
- if $\sum_{i=1}^{n} Inter_i(k, l) = -n$, the label assigned to the pair (e_k, e_l) is "$\rightarrow$",
- otherwise, the label assigned to the pair (e_k, e_l) is "$\leftrightarrow$".

3.4 Test of consecutivity

We aim to reduce the set of solutions by adding the notion of consecutiveness. If two events are always consecutive on the n sequences (i.e there is no other event between them) then we can say that the pair is consecutive. In this case, a "No_Event" constraint between the two events is set [4]. The vectors of the occurrence positions $Set(i, k_2)$ of the first pair events e_k and e_l are studied. When the first "1" of the e_k vector is detected, the value of the following column of the vector of the event e_l is studied: if it is a "1", then the pair studied is consecutive at least once in one sequence and the algorithm continues the test with the other sequences and with the others pairs. When all pairs are calculated, they are assembled to be interpreted with a contraints graph. A contraints graph represents the result with a graphic way. Each node represents an event and the index next to the event indicates the maximal frequency of each event. The arc corresponds of the no bounded time contraints. A dashed line represent the specific relation "No_Event" between a pair of events. Arrows on the arcs model the direction of each pair.

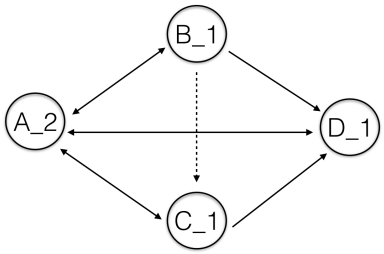

Figure 2: Constraints graph of the 3 test sequences

Figure 2 represents the constraints graph of the 3 tests sequences. We can remark that we have 2 occurrences of the event type A and 1 occurence for the event types B, C and D, 3 pairs with a double direction ((A, B), (A, C), (A, D)), 3 pairs with an unique direction ((B, C), (B, D), (C, D)). The pair (B, C) have an dashed arrow to represent the label "No_Event" because it is consecutive on the 3 input sequences.

4 Tests and results

We applied the LAE algorithm with the ACTIA's traces. The actual traces were not available during the tests, they were simulated on a diagnostic tool. The diagnostic tool allows the mechanic to carry out his tests and measurements on the vehicle in real time. The mechanic selects actions displayed on the screen. The tool works by changing screens. The users can interact via the tool to test the various functions of the vehicle.

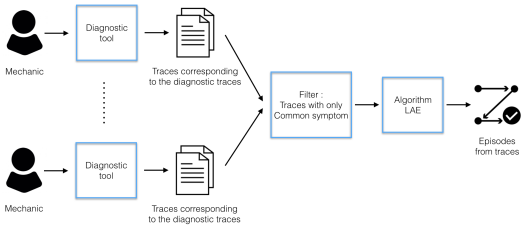

Figure 3: Overview of the context

The figure 3 explains the architecture of the context. The mechanic tests the vehicle using the diagnostic tool. The tool provides a trace summing all the actions the garage owner did in the diagnosis session. The LAE algorithm uses all the diagnosis traces corresponding to common symptoms. They are compared to provide the common approach to all entries in the form of parallel episodes. For a good precision, we went into 4 different garages to ask owners the procedures to follow to diagnose a failure with a common symptom.

```
1 (Btn_C,1)
2 (Menu_C,2)
3 (Btn_C4PB78,3)
4 (Pop_VIN,4)
5 (Btn_OK,5)
6 (Infos,6)
7 (Btn_OK,7)
8 (Menu_App,8)
```

Figure 4: Extract of an Actia's sequence

The extract from figure 4 represents the beginning of a simulated trace on the diagnostic tool. The first accessible screen offers a choice on the brand of the vehicle to be diagnosed, "Peugeot" or "Citröen". This example was carried out on a vehicle "Citröen" and the model was a "C4 Picasso". The first line corresponds to the selection of the "Citröen" button of the diagnostic tool. A change of screen displays a list of vehicles of the mark to be selected. The second line indicates that the mechanic was on the "Citröen" vehicle selection menu and the third line shows he selected the "C4PB78" button, which corresponds to the "C4 Picasso" model, etc. The simulated sequences have an average of 65 events and there are 4. The result consists of 78 pairs including 13 elements in common. Note that in this case of study, two actions (i.e. events) without preference of order cannot be done sequentially in one or the other order equivalently. The generated result should be filtered to take this feature into account. The algorithm is able to consume traces of ACTIA and generate a satisfactory result from a temporal and qualitative point of view. The execution time is 0.7 seconds with ACTIA traces. The tests were carried out on an ASUS PC, under windows 7 with an Inter Core i7 processor - 2670QM, 2.2GHz, 6GB of RAM. The work carried out is considered as a proof of concept. Nevertheless, this work need to be continued by a deeper analysis of the results. Indeed, the constraints graph represents the raw episode of the n input sequences. As the result was not confronted with the mechanics, the pertinence of the episode was not evaluated.

5 A standard algorithm for episode learning

5.1 Presentation of the algorithm

In this section, we present an existing algorithm that allows learning episodes in n sequences [7]. This algorithm allows to extract the different serial episodes and parallel episodes that are located in a sequence s. The sequences provided by the diagnostic tool being parallel, we only present the part dealing with this type of episodes.
The algorithm recognizes parallel episodes in a sequence s. This step creates the database that contains all the episodes of the s sequence. The algorithm uses two windows $\mathbf{w} = (w, t_s, t_s + win)$ et $\mathbf{w}^{'} = (w^{'}, t_s + 1, t_s + win + 1)$ where w and $w^{'}$ are the sequences contained mutually in $\mathbf{w}$ and $\mathbf{w}^{'}$, t_s is the starting time of the window and win is the dimension of a window such that $width(\mathbf{w}) = win$ [7]. The sequences contained in $\mathbf{w}$ et $\mathbf{w}^{'}$ will therefore be similar. After the recognition of the episode in w, an update of the data structure is performed to get the window to move to the $w^{'}$ episode in the same sequence s. The recognition of a candidate parallel episode α is done with a counter $\alpha.event\_count$. This counter indicates how much events of α are present in the window. When $\alpha.event\_count$ is equal to the length of α, the totality of α is present in the window. $\alpha.freq\_count$ is incremented by 1. That counter represents the number of windows where α is contained entirely. Candidates episodes are indexed by the number of events of each type they contain. All episodes that contain a type A events will be in the $contain(A, a)$.

An episode α is stored in a table of events sorted alphabetically. Each event of α is indexed between brackets. For example, an episode β with events types B, D, D, E is represented by a table β with $\beta[1] = B$, $\beta[2] = D$, $\beta[3] = D$, $\beta[4] = E$. Collections of episodes are sorted alphabetically in a table $\mathscr{F}$, where the i^{th} episode of the collection is represented by $\mathscr{F}$[i]. All the episodes that share the first event type are consecutive in the episode collection. A maximum sequence of consecutive episodes of size l, sharing the $l - 1$ first events is called a block. Candidate episodes are identified by creating all possible combinations of two episodes of the same block [7].

The collection of frequent episodes is a list featuring episodes that most frequently appear in the n sequences. The algorithm computes the collection of frequent episodes that make up the sequence s from the ϵ class of parallel episodes. The search is done by levels of length l on episodes α in the episode class following the sub-episode relationship. The search begins with the most general episode (one containing only one event). At each level, the algorithm calculates the collection of episodes and their frequencies.

The frequency of an episode is defined as the ration between the number of the window where the episode α appear entirely and the total window number. With a sequence of events s, a window of dimension win, the frequency of an episode α in s is:

$$fr(\alpha, s, win) = \frac{|\{w \in \mathscr{W}(s, win) | \alpha \in w\}|}{|\mathscr{W}(s, win)|} \tag{9}$$

where s is the number of windows is the α episode is fully included, $\mathscr{W}$ is the set of all windows w on s. For a defined threshold frequency thres_fr, α is frequent if $fr(\alpha, s, win) \geq min\_fr$.
A rule for predicting frequent episodes that respect s, win, et min_fr is denoted $\mathscr{F}(s, win, min\_fr)$. A prediction rule is a $\beta \Rightarrow \gamma$ expression where β and γ are episodes such as $\beta \leq \gamma$. The ratio $\frac{fr(\gamma, s, win)}{fr(\beta, s, win)}$ is called the confidence of the episode prediction rule. This confidence can be interpreted as a conditional probability that the entire γ is present in the window, with a given β.

5.2 Comparaison of LAE and the standard algorithm

In this section, we compare the algorithm LAE presented in section 3 to the standard algorithm of [7] presented in section 5.1. The first step of the standard algorithm is to find all candidate episodes of the n sequences. Once all the episodes are found, they are stored and classed alphabetically in order to start creating all possible combinations of two episodes of the same block. Once the candidate episodes are generated, they are stored in the $\mathscr{F}(s, win, min\_fr)$ episode collection. The prediction and confidence rules are then deduced from the previous result. The LAE algorithm proceeds quite differently: it begins by looking for events that are not in common in the n sequences and then it removes them from the sequences to keep only the common events. It determines the maximum occurrence number of each event, which allows us to easily find the maximum frequency of each pair of events on the n sequences.

The difference with the standard algorithm is that it finds the episodes in the form of a constraint graph describing all the possible episodes. The standard algorithm makes it possible to obtain an episode table directly.
The algorithms have their differences, however: with the $\mathscr{F}(s, win, min\_fr)$ event collection, it is possible to predict or anticipate the behavior of the sequence in its integrity. The algorithm thus provides a set of episodes as well as their prediction rules.
On the other hand, the LAE algorithm provides important information thanks to the consecutive test. It makes it possible to express the fact that there can be no intermediate events between the two events of a pair on the n sequences of inputs. An intermediate event is an event of a different type than those defined in the pair.
The comparison with the Mannila algorithm will be pursued when we'll have a larger set of data. Then, real tests and further analysis could be done.

6 Toward the Ph.D work

The thesis is the continuation of the presented work. The Ph.D is a collaboration with the ACTIA company which developed a software named ACTI-DIAG destined to mechanics. It allows to dialogue with the set of calculators (airbag, injectors, ...) of a vehicle and to pick up all the available information inside the car. These information can then be exploited in conjunction with information from the manufacturer or independent actors to finally allow the mechanic to locate the component to be replaced. In the field of automotive diagnostics, the number of identical vehicles circulating makes it possible to capitalize a great amount of knowledge. This capitalization then makes it possible to repair a vehicle more quickly by taking advantage of the experience gained on similar vehicles having already encountered the same problem. Currently, the information collected on the vehicle are:

- The type of the vehicle.
- The set of calculators et their actual software version.
- The set of default codes (also called Data Trouble Code) on calculators with the occurrence time.
- The context (values of parameters judged relevant).
- etc...

The objectives of the thesis is to analyze (semi-) automatically this set of data to reduce the updating time of the diagnosis and repair methods by the manufacturer, but also to reduce the time of the diagnosis and repair stages when a vehicle is in a workshop. This reduction can be addressed by:

- An update of existant knowledges (the list of possible causes for a given symptom, the probability of each fault occurrences about a vehicle, ...).
- A ranking of knowledges proposed to the mechanic (most probable hypothesis related to a symptom, the "incident" files having most often completed, ...).
- The creation of a new incident file about the most recurrent fails in the network.
- Optimize existing diagnosis procedures to minimize their journey time by taking into account the time required to complete each of their tests and the probability of occurrence of each fault.
- The learning of new diagnosis procedures by using collected data (actual DTCs, the context of the raise of DTCs, ...) in order to distinguish different possible faults for a or several given symptoms.

ACTIA is setting a network of 50 workshops for collecting data from different diagnosis with the symptoms and causes labels. This operation will allow to do real tests and comparisons with standard algorithms. The deployment of this network is planned for the end of November. Until that time, the study is performed without the label information. In first step, for a given ECU (Electronic Control Unit), the goal is to find co-occurrence of DTCs on vehicles. As DTC are discrete data (for example, P318D is the DTC for a communication error for the Nissan Leaf model), learning algorithms like clustering with non-ordered discrete data [10] and k-Nearest Neighbor Searching in non-ordered discrete data space [11] are investigated.

7 Conclusion and perspectives

The work presented in this paper is part of a learning process for generic automotive diagnostics. The objective is to automatically generate a model of the diagnosis procedures used by mechanics to diagnose a given fault. This is performed from a set of action sequences carried out during several diagnosis sessions referring to the fault. Since the traces recorded by the diagnostic tool of our case study are timeless, we have directed ourselves to the development of our own learning algorithm: Learning Algorithm for Episodes. This algorithm allows to learn temporal patterns where time information is captured only through the notion of precedence. The tests carried out made it possible to validate the first version of the algorithm presented in the paper.

This work has reinforced the interest of automatic learning to exploit the feedback from the diagnosis sessions. Several perspectives have been identified. First, it is necessary to test the algorithm with a larger volume of input data and to evaluate its efficiency. Also, the consecutive test need consolidation. From an application point of view, the idea is to use the results of LAE to improve the training tools of garage mechanics. As the complexity of vehicles increases, paper is not suffisant anymore. Our industrial partner (ACTIA) develops a Serious Game to facilitate the learning of new methods of diagnosis and repair. A Serious Game is a software that combines a serious intention with playful springs. Episodes provided by LAE can be turned into training scenarios for the Serious Game for the diagnosis and repair tasks.
The problem of knowledge capitalization addressed in this work is a real challenge for ACTIA company. The collaboration with ACTIA on this research area is going in the context of a Ph.D.

Acknowledgements

We would like to thank our contacts in the company ACTIA for their valuable comments and recommendations.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile*, pages 487–499, Jan 1994.

[2] Mitsa and Theophano. *Temporal data mining*. CRC Press, 2010.

[3] A. Subias, L. Travé-Massuyès, and E. Le Corronc. Learning chronicles signing multiple scenario instances. In *The 19th World Congress - The International Federation of Automatic Control*, Cape Town, South Africa, August, 24-29 2014.

[4] *Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems*, July 21-26 1999.

[5] C. Dousson, P. Goborit, and M. Ghallab. Situation recognition : representation and algorithms. In *IJCAI : International Joint Conference on Artificial Intelligence*, pages 166–172, Chambéry, France, August 1993.

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

[7] A.Inkeri Verkalo, H. Mannila, and H. Toivonen. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, September 1997.

[8] K. Amphawan, J. Soulas, and P. Lenca. Mining top-k regular episodes from sensor streams. In *7th International Conference on Advances in Information Technology*, pages 76–85, 2015.

[9] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *KDD-96 Proceedings*, 1996.

[10] A. Watve, S. Pramanik, S. Jung, B. Jo, S. Kumar, and S. Sural. Clustering non-ordered discrete data. *Journal of Information Science And Engineering*, 30:1–23, 2014.

[11] D. Kolbe, Q. Zhu, and S. Pramanik. On k-nearest neighbor searching in non-ordered discret data spaces. In *IEEE 23rd International Conference of Data Engineering*, Istanbul, Turkey, April 2007. IEEE.