Sequential learning for case-based pattern recognition in complex event domains

Pablo Gay[1], Beatriz López[1], Joaquim Meléndez[1]

[1] University of Girona,
Campus Montilivi, P4 Building,
E17071, Girona, Spain
{pablo.gay, beatriz.lopez, joaquim.melendez}@udg.edu

Abstract. Large and distributed environments usually generate huge amounts of information. The easiest way to deal with this information in an uncoupled and asynchronous way is using event oriented approaches. These systems are usually implemented to react to the generated information. This paper presents a new track to add to these architectures a mechanism to discover behaviors combined with a reasoning method that predicts the next most probable event. Consequently, the work focuses in two fields: sequence pattern mining and case-based reasoning. The former aims to compress the original large amount of event data by discovering frequent behaviors in the form of sequence patterns. The latter is used to recognize the behaviors and forecast future predictions based on the learnt patterns. The methodology has been tested using real data from a public bike hiring system.

Keywords: case-based reasoning, sequence mining, event, hybrid system, knowledge acquisition, human learning, application.

1 Introduction

Technological advancements have contributed to improve quality of citizens live. For instance, thanks to sensors, an aging person can be monitored 24 hours 7 days a week at home, and when no activity is detected in the house for a long period of time, an alarm is activated in the system. Another example is the use of Radio Frequency IDentification (RFID) cards that allow us to use the public transport systems easily without worrying to bring cash with us all the time.

This type of sensors and devices can detect state changes in systems and objects creating events capable to trigger appropriate services when needed. Event Driven Architectures (EDA) are being used to activate these services triggered by incoming events. These architectures have been designed to operate in domains where the entities that generate the events need total uncoupling and asynchronism. Complex Event Processing (CEP) is a particular case of EDAs, which defines a set of tools and techniques for analyzing and controlling complex series of interrelated events [1][2]. For example, when a user enters in a fully monitored kitchen, and opens the oven, then events "in cuisine" and "oven access" are registered. Those related events, which that take place in consecutive time stamps, are called complex events. Complex events can have actions associated to them; thus, when a complex event is triggered, the associated action is executed. In the example, after the "cuisine+oven" complex event is triggered, the action "caution service" can be executed. A follower event could be "burnt smell", and the action associated whith the new, complex event, "cuisine+oven+burnt smell" could be "fire alarm service". Of course, it could be nice to avoid the "fire alarm" situation, which could be achieved by predicting the "burnt smell" before it happens.

However, defining by hand complex events is a time-consuming task that requires the knowledge from qualified personnel. Moreover, most of the times there is no previous knowledge or evidence available to build them. Therefore, knowledge discovery techniques are helpful in this scenario, in which huge volume of events are continually being generated. Machine learning methods can be used to discover frequent patterns in users' behavior that can then become complex events. Moreover, pattern recognition techniques can then be used upon the learned patterns to predict the next event in a given sequence.

To achieve this two stage process (knowledge discovering and pattern recognition) for complex event processing, we present in this paper a hybrid system that combines sequence pattern mining and case-based reasoning (CBR). Firstly, sequence pattern mining is used to process the raw data generated by an event-driven system, finding sequence patterns of events that could be considered as complex events.

Then, case-based reasoning is used for recognizing the learnt patterns given the current event system information. This recognition is performed as the information is known, so that current events are expected to match the beginning of the recognized sequence patterns, and then event prediction can be performed thanks to the remaining information of the patterns.

For demonstration purposes we use a public bike hiring system (like Vélib' in Paris, Bicing in Barcelona, BIXI in Montreal and Toronto or Barclays Cycle Hire in London, etc.) as example, where users registered in the system can check out bicycles from specified depot stations distributed around the city and return them in another depot. In this domain, it may happen that users find that there are no bicycles available or free slots in the depot (they can't return theirs). This problem is usually solved scheduling a set of vans/truck which moves bikes between stations. With our system, we pretend to learn the most frequent user behaviors, predicting which will be their following stations can be done and consequently re-schedule more efficiently the bike transfer between depots, so the user always finds a bike or a free park place available for him.

This work is organized as follows: Section 2 reviews similar and related works to the one presented here; Section 3 exposes the methodology of our work for discovering and predicting behaviors; Section 4, shows our experimentation process using a specific domain as example; and, finally, Section 5 presents our conclusions and future works.

2 Related Work

As mentioned before, this paper is related to Sequence Learning and Pattern Recognition. Firstly, the main objective of sequential learning methods is to discover sequential patterns from (very) large databases. Three Apriori-based algorithms, for mining frequent sequence of transactions performed by customers, are reported in [3]. A transaction is then interpreted as constrained sequences of events. The Apriori-based algorithms develop lattices of event sequences that represent the most frequent patterns and discards the remainder. Other algorithms have been developed for improving the efficiency. For example, [4] improved the candidate sequence pattern generation but still needed multiple passes over the data. Our approach is based in [5], on which sequence patterns are organized according to prefixes (ordered subsets of sequences).

Secondly, regarding pattern recognition, this paper is closer to plan recognition and intrusion / detection problems. On the one hand, plan recognition problem [8] uses a sequence of actions performed by an actor and tries to organize them inferring the ultimate goal or goals. In our case we do not pursue to discover these goals but the sequence of events generated by a user, equivalent to sequence of actions or plans, since behaviors usually do not have a clear goal. The objective is to find the relative ordered set of actions that follows the actor in order to recognize its actual state and predict its next step.

On the other hand, a pioneer work on intrusion detection is [9], which also deals with sequences but following an instance based learning approach. In this work, a complementary segmenting process is carried out to obtain sequences from event streams based on windows of a fixed length. Each new subsequence becomes an instance resulting in a significant increase in the number of instances. Therefore, reduction methods are further required to control the case growth. Conversely, we are taking advantage of existing sequence learning algorithms, so sequences are obtained with a given support and confidence, becoming generalized patterns of cases.

There have been other previous synergies between sequence learning and CBR, as in [6], where the authors proposed learning methods to select sequences of repairs in a CBR system, when more than one repair exists. In our case, we are using sequence learning to build cases, so the case base is composed of patterns of sequences, and a new sequence metrics is defined to recover cases from memory to solve the prediction problem. In [14] a similar approach was used, but in addition of using a different sequence metrics than ours, they use a plain memory instead of the hierarchical organization we are proposing. Other approaches, as in [7], use CBR to deal with sequential information, but without hybridizing with sequence learning algorithms.

3 Methodology

Our goal is to learn complex events as sequence patterns, and use case-based reasoning for predicting next events (see Figure 1) by exploiting the recognized patterns as cases. Thus, our methodology consists in three main steps: sequence generation, pattern discovery and sequence recognition for next event prediction. Firstly, we generate sequences from a cloud of events acquired through an event-driven domain. Secondly, a pattern discovery algorithm is applied to find frequent episodes. Finally, duringthe exploitation phase and when monitoring the activity of the system, patterns are recognized and reused for prediction for current users according to their most recent activity (events) registered in the system thanks to case-based reasoning.

3.1 Sequence Generation

The goal of this step is to provide adequate pieces of information for data mining. Thus, given event streams gathered from an event-driven domain, event sequences are obtained.

Definition 1. An event is an observable change in the system and it is usually represented by the action that caused the change.

Events are identified by the timestamp (when the action or change in the system is observed) and they are assumed to be instantaneous. .

Definition 2. An event stream *S* is an arbitrarily sorted array representing events generated by an event-oriented domain.

$$S = < \ldots, s_k. s_{k+1}, s_{k+2}, \ldots >$$

Event streams can be classified according to their source, i.e., the actors who cause the event. Thus, we talk about event streams of actor *a*, *S(a)*.

Definition 3. Given an actor *a*, its associated event stream, *S(a)*, is the subset of events in *S* who has been caused by *a*.

The criteria to sort the events within a stream can be, for example, time and space. In this work we focus on time. Thus, older events are at the beginning of the stream, while the newest are at the end.

To facilitate stream manipulation, streams can be segmented in finite length sequences according to a given criteria. For example, time-sorted event streams can be split according to time windows or space-sorted event streams can be split by considering a maximum distance among events. Since this work focus on time, we define sequences based on time windows.

Definition 4. A time event sequence *Seq* in *S* is a subset of events in *S* whose first and last events defines a time window *w*.

Analogously, we can define a time event sequence for a given actor *a*, *Seq(a)*. For example, the time window used in this work correspond to a day interval since we consider that the information from two different days can be considered independent according to the domain problem we have dealt with. So a time event sequence *Seq(a)* would contain the events generated in an specific day by the actor *a*.

Observe that event sequences can be directly used to build complex events. According to [10], a complex event is an abstraction of event relationships. The event relationships that can be abstracted into complex event are basically three: time relationship, when event *A* happens before event *B*; causal relationship, when event A happens because of *B*; and aggregation relationship, when events A_1, A_2, ..., A_n means *B*. These abstract relationships are represented by the ":" symbol in Figure 1.

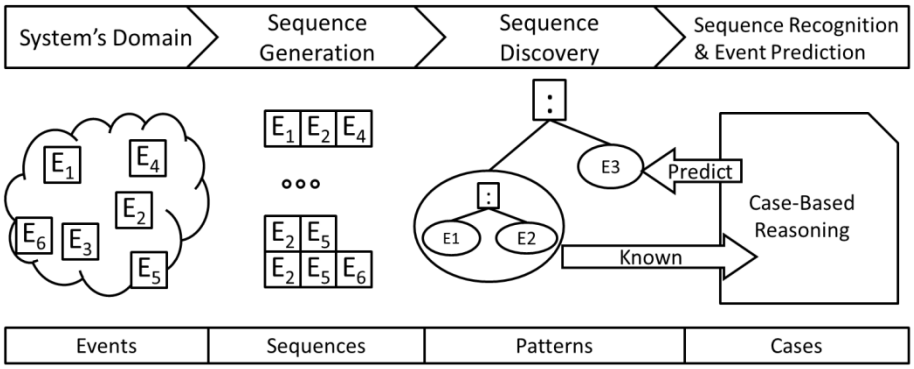

Figure 1 Methodology overview.

Thus, in the sequence generation step, we are given with a set of streams *S*, and we obtain a set of time event sequences, Seq_1, ..., Seq_m according to a time window *w*. In case of being interested on a specific actor *a*, we can obtain also, the corresponding sequences particularized to this actor, $Seq_1(a)$, ..., $Seq_m(a)$. For the sake of simplicity, we assume from now sequences without any particularization to an actor, since the methodology does not change the results, as we show in Section 4.

3.2 Pattern Discovery

Pattern discovery can be seen as a step where we compress huge volume of data, and remove the useless information by mining the most common sequences. Given a set of time event sequences, this step find sequence patterns corresponding to a high number of similar sequences, which represents similar actor activities which are hidden inside all the information gathered in the initial streams.

Definition 5. An event sequence pattern *P* is an arbitrarily sorted array of events p_i supported by a high number of event sequences.

$$P = < p_1, p_2, p_3, ..., p_n >$$

Discovering these patterns can be done using several pattern sequence mining algorithms like the ones commented in Section 2. The main idea of all of them is to explore all the sequences and find the most repeated elements and their combination. In our case we have chosen to work with Prefixspan [5] because of two reasons: its simple basis and its inherent pattern tree structure. On the one hand, its simplicity allows us to easily implement and add modifications to the algorithm to map our requirements. On the other hand, the tree structure helps in the next step of the methodology.

The basic idea of Prefixspan, from which our implementation derives, is building a search tree. Nodes represent possible patterns and they are annotated with their support; this is, the number of sequences in the database that supports the pattern. Figure 2 gives an example. The root of the tree contains the empty pattern sequence which is supported by all sequences. The next level of the tree adds a node for each possible event occurring in the system. In the Figure, the sequences that appear in the node as supporting them have been modified with underscores, meaning where the pattern is located (i.e., elements iterated in the sequence until the pattern has been found). This process is called "projection" of the node in the database. Be aware that a pattern *AB* fulfills both, *ABC* and *AC*" sequences, since it is not mandatory that the events occur in the same position as they are in the pattern but following the same order. This is why the information stored in nodes is called "prefixes". Branches of the tree that do not have a minimum support τ (in the Figure τ =2/3) are pruned. The algorithm ends when no more projections are possible. Figure 2 uses the toy stream case base < *ABBA, ABCA, CAAB* > to illustrate this. The minimum support for the example is 2/3 and the final patterns are < *A, AA, AB, ABA, B, BA, C, CA* >.

The original PrefixSpan algorithm (and their predecessors) used sets of items instead of events and forbade item repetition inside these sets. That means that event sequence patterns as *AAC* could not be represented as sets of items. For that purpose, we have studied the algorithm and concluded that feeding PrefixSpan adequately, the algorithm can work according to our target. The learnt event sequence patterns represent the expected registered activity of actors in the event-driven system.

3.3 Sequence recognition and event prediction

Given the actual state of the event-driven domain, in this stage we predict the most probable events to happen thanks to the discovered patterns. Actual state is provided by an input stream in which the most recent activity of the domain has been registered. Usually, prediction is actor-based, that is, we are interested in predicting the next activity of a given actor.

We map this procedure using a Case-Based reasoning (CBR) strategy [11], where from a case-base of past experiences, a new solution is generated, by reusing solutions of past cases retrieved thanks to a specifically designed metric. In our approach, the case-base is composed by the event sequence patterns discovered. Then, a new distance method for retrieving and reusing cases has been designed to meet our prediction needs.

Retrieving Patterns. This stage of a CBR compares the input stream with the patterns in the case-base, providing a ranked list of similar patterns. Our innovation for this step relies in the distance algorithm, the metric function used to assert if a stream contains a pattern.

The algorithm has been designed with the purpose to return a weighted output depending on the following properties:

- A complete match of a pattern within a sequence returns a lower value than a partial match.
- A recent match (end of the sequence) returns a lower value than an old match (beginning of the sequence).
- A match where the events are closer between them returns a lower value than a match between distant events.
- The lowest value, the better.

This algorithm sets distances close to zero for patterns that just happened "right now". Consequently, retrieving the closest patterns can be considered equivalent to recognize the actual state.

Our notation assumes that within sequences and patterns, events located at position *0* will be the last event recorded, the newest ones, and the ones located at $|S|-1$ or $|P|-1$ the oldest ones. Basically, the algorithm to estimate the distance is composed of two parts (see Table 1). The first one is the *index* function, which returns an integer indicating the position where the pattern event p_j is located inside the stream sequence *S*. Given a current position *i* (where the previous p_{j-1} event was found), the search is started it. If p_j cannot be found in stream *S*, it will be assigned the maximum value possible $|S|$. The second part is the distance algorithm itself, where, for each event pattern inside *P*, positions retrieved by function *index* are exponentially accumulated. If the last pattern event was not found, the index *i* remains the same and we continue looking from there.

Algorithm	Calculate distance (S, P)
1:	$\Delta = 0$
2:	$i = 0$
3:	for j = 0 until |P|-1 do
4:	aux = index(S, p_j, i)
5:	$\Delta = \Delta + 2^{aux}$
6:	if aux ≠ |S| then
7:	i = aux
8:	end if
9:	end for
10:	return Δ

$$index(S, p_j, i) = \begin{cases} |S| \; if \; \nexists k > i \; such \; that \; s_k = p_j \\ min_{k>i}(\{s_k = p_j\}) \; otherwise \end{cases}$$

Table 1: Distance algorithm

Reusing Patterns. Reuse uses the ranked list of cases generated at the retrieve stage in order to predict the next event in the system. For that purpose, the organization of cases in memory (tree structure generated by the sequence pattern mining algorithm) is used.

Firstly, the pattern in the ranking with the lowest value (the closest one) is selected. In case of finding two or more patterns with the same value, the longest pattern is selected, since it represents the actual state with more information. Once the pattern has been selected, using the tree structure, its immediate descendant with the highest support is chosen as the candidate solution. This is, the candidate should contain an additional event in the sequence, the predicted event. In case of the pattern selected does not have any descendant (leaf pattern), the next pattern in the ranking is used, and so on, until a prediction is found.

For example, using Figure 2 as the tree pattern, if *AB* is our closest pattern to the input stream, its immediate descendant would be *ABA* and therefore the final prediction *A*.

4 Experimentation

For the experimentations process, we have used real information from a public bike hiring system. The testing objective in this domain is to prove the effectiveness of our method while trying to predict which will be the following event in the system. This predictions represents the next station where a user will interact with the system, and they can be used in future works to move bikes between depots stations using optimization models that know where the bikes are going to be, not where they are.

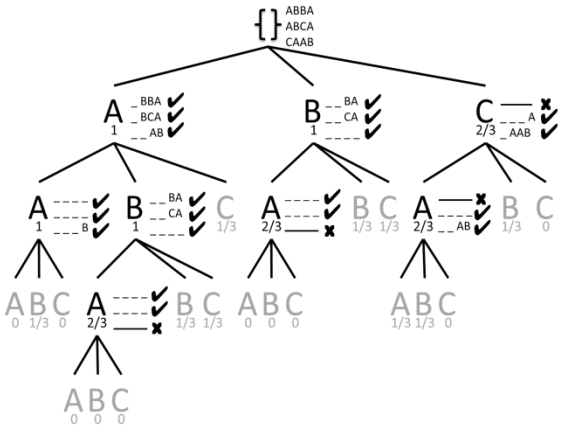

Figure 2 Simple example of Prefixspan. The toy stream set ABBA / ABCA / CAAB is used to mine patters with a minimum support of 2/3. Gray nodes are explored but since they don't have minimum support the branch is pruned.

4.1 Data

Data originally was a 30 days database bulk of a real public bike hiring system, specifically January 2008, where, for each user, there were all the bike's uses performed (source, destination and timing). This usage information is in fact a record of the events generated by the interaction of users with depots: each time a user picked or dropped a bike, the visited depot station generated an event with information about who was generating the event; where this event was coming from, the type of action (picking or dropping) and a timestamp.

Once reorganized as *user streams*, each stream contained only the sequence of events of one user. After that, a clean-up process was done for removing noises and outliers, for instance removing user streams with low amounts of events. The reason we removed users with low usage of bikes is because when our approach mines the patterns, they don't provide any valuable information but they still count when estimating the support, decreasing it and hindering the selection of a proper threshold. Finally, 260 user streams with at least 150 events each one remained. The information about the visited depot stations was included so the time event streams represent ordered sequences in time of the visited stations. This means that the predictions done at this point concerns the future depot stations a user will visit, leaving for future woks the inclusion of time and action of those predictions.

4.2 Scenarios

To compare the effectiveness of our approach we have prepared a set of four scenarios. In Scenario 0, or baseline scenario, predictions are performed using statistical information from the original data. Scenarios 1 to 3 use our approach, and the difference among them is the way the original data is divided. All scenarios are run under the same configuration while mining patterns: the minimum support for a sequence to become a pattern is *10%* of their sequences.

Scenario 0 (Baseline). The worst case scenario would be to predict randomly any other station as future prediction.

Scenario 1. In this scenario we want to check the possibility of generating prediction for each user only with their individual information, this means that when a prediction for a user is requested only the patterns found in his data are used and only his past information is reused for the prediction (see Figure 3.top).

Scenario 2. This scenario complements the previous one. In this case instead of using only the information of each user for his prediction now we mine the patterns for each single user and combine them in order to create a common pattern case-base for the CBR (see Figure 3.middle).

With this configuration we check how useful can be sharing the patterns. For instance, if a user starts a new behavior, predictions could fail in the previous scenario, since it is a new pattern that does not exist in his past information. But sharing the patterns another user could already have this pattern and therefore, with the unified pattern case-base, be predicted as well.

Scenario 3. The latest scenario, the set of all user streams feeds the sequential pattern learning instead of using independent sets of patterns mined from the users (see Figure 3.bottom).

Now, using all the user streams as cases for the pattern discovery, we check if the global trends of the system are also useful to make individual predictions.

4.3 Experimentation sets

Original data has been split into training and test sets. To avoid biased results, we have generated 100 experimentation sets randomly, with 20% and 80% of test and train cases correspondingly. The final results are the mean amount of times a user finally uses the predicted station. This means that the prediction can be few events in the future, not exactly the next one.

Concerning test streams, they have been modified to map as closely as possible how the real system process the events. That is, given a test stream $S^t = < s^t_1,\ldots, s^t_k >$, k test are performed: $< s^t_1 >$, $< s^t_1, s^t_2 >$, $\ldots < s^t_1,\ldots, s^t_k >$., starting with a single event stream and increasing the number of events in it until the total stream length. Thus, prediction results depend on the on the amount of events (stream length) registered by a user in the system.

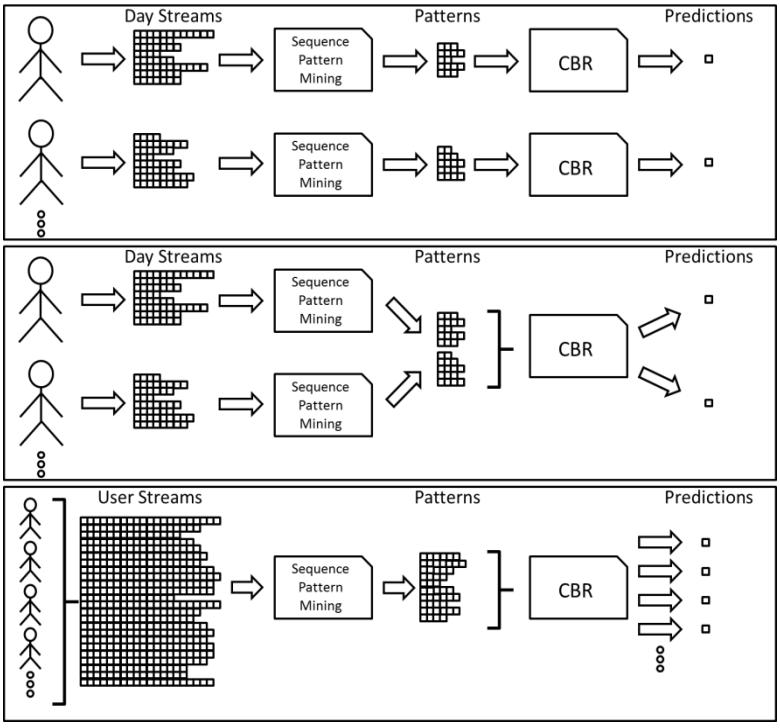

Figure 3 Top figure represents Scenario 1, where independent predictions for each user are generated. Middle figure represents Scenario 2, where users share the same pattern case-base for the generation. Bottom figure is Scenario 3, where whole user streams are used to generate the patterns case-base.

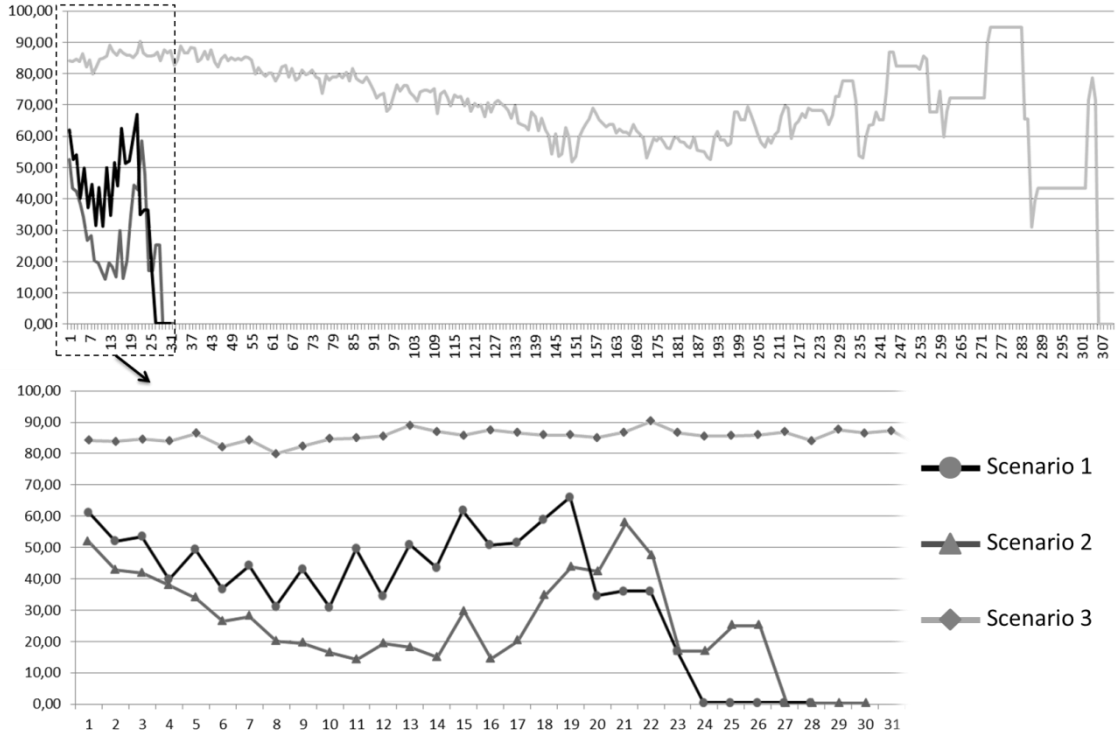

Figure 4 Mean success rate (y-axis) according to the stream length (x-axis) in Scenarios 1 to 3. Baseline Scenario 0 has not been represented due its poor success ratio would be a flat line colapsing with the x-axis.

4.4 Results

Figure 4 shows the results obtained, except for Scenario 0. This baseline scenario has been created to compare the results and it is based on the fact that the system has *296* stations, so the odds of randomly choose correctly the next station to be visited by the user is *100* (1/296) ≈ 0,338%*. If we try to improve this result using some knowledge from the data, we could recommend only the stations that are within a *k-radius*. We have estimated that the mean trip time is *14min 47sec*. Using a sensible biking speed of *12 Km/h* (note that bikes ride in urban environments with pedestrians and cars in the way) we have that the mean distance traveled is approx. *2,597 Km*. With this information we have estimated that the mean amount of stations within this *2,597-radius* from any station is *124*. So the probabilities of predicting a station now is *100* 1/124 ≈ 0,806%*. This result is outperformed by the other scenarios and if they were represented in Figure 4 they would look like a flat line so close to zero that they would collapse to the x-axis in that figure.

Scenario 1 can be seen in Figure 4. Streams with length 1 have an approximated *60%* of success but it descends until almost *30%* with lengths *8* and *9*. When historic information is too large (more than *19* events or *4* days of information approx.) predictions drops their success rate drastically.

The zigzagging shape is due to the domain, since users alternate piking and dropping bikes. X-axis represents the stream length used to perform the prediction. When the system is predicting with a stream of length one, the single event in the stream represents the first action of a user recorded, which can only be picking a bike. Therefore, the second event represents the event which records where the user dropped the bike and so on. Using this reasoning, Scenario 1 in Figure 4 exhibits a better performance predicting the following depot stations visited when the user has just picked a bike (and so there will be a closer dropping action). Also, it is interesting to observe that in the long run, using a bigger amount of background information does not improve the performance. It is easily seen how the prediction fails after more than 19 events of historic data. But this can also be caused because there are too few day streams with this length, what directly affects the amount of support and prevents a pattern to be discovered.

Scenario 2 presents similar results than Scenario 1 but an approximately *10%* worst. The exception would be when predictions are done with around *21* events of information. In this case, the zigzagging

prediction according if the user is taking or dropping a bike disappears. Also, with big amounts of background information in stream predictions tend to fail always. It was sensible to think that this scenario should outperform the previous one, but the combination of patterns from different users in this case provides to the CBR case-base more noise than useful information. Huge amounts of slightly different patters can be selected, confusing the system.

Finally, Scenario3 shows better results than the previous ones. In this case, the prediction success rate slowly decreases from an almost *90%* using around *25-length* streams until an almost *50%* when length is around *150*. Neither can be the zigzagging appreciated, but this time, after 200 length the prediction, success rate becomes erratic. This could be a combination of two symptoms: the first one is the same one that happened in the previous scenarios when finding patterns within the longest streams. The existing patterns at the end of the longest streams have problems to be detected, hindering the prediction at this point. The second symptom can be a side effect of our distance algorithm. It represents an exponential sum and when an event from a pattern is not found in the user stream it receives a $2^{|S|}$ penalty, where $|S|$ represents the total length of the stream. In this scenario there are user streams longer than 300, hereby 2^{300}, what makes the process lose a lot of precision.

5 Conclusions

In this paper we have presented a sequence discovery and recognition method specifically designed for event-oriented domains that register the activity of actors. Event streams are mined to extract the most important information as sequence patterns. Such sequence patterns represent the discovered behavior of the actors. Then, the system recognizes current event information thanks to a CBR approach, in which cases are the learnt sequence patterns. Thanks to the recognition, a prediction of future events can be performed. Such prediction can benefit the overall system to anticipate undesirable future system situations (fire alarms, resource unavailability).

We have tested our methodology in a public bike hiring system. Experiments have been performed with isolated and global actor information, obtaining better predictions in the latter case. Thus, in the lack of information for making predictions for a user, using information from similar users helps.

Nevertheless, results drop when there is a lot of historical information involved. This should be a matter of research in future works in which we analyze which is the adequate time window needed to make predictions, conversely, to establish the adequate forgetting mechanism.

Moreover, we need to include time interval durations within the events in order to generate more precise predictions. The current predicted events are carried out by actors but not necessarily right away, they can occur several events forward. We need to extend our sequence representation from $< s_i, s_j >$ to $< s_i \, time\_dur \, s_j, >$ which poses new challenges to the pattern learning step, in which other works like [12] or [13] could be an starting point.

References

1. Aggarwal, C. C. & Yu, P. S. (2009), 'A Survey of Uncertain Data Algorithms and Applications', IEEE Trans. on Knowledge and Data Engineering 21(5). Pages 609-623
2. Gero Mühl, Ludger Fiege, P. P. (2006), Distributed Event-Based Systems, Springer.
3. Agrawal, R. & Srikant, R. (1995), Mining Sequential Patterns, in 'Proc. of the 11th International Conference on Data Engineering', IEEE Computer Society, Washington, DC, USA, pp. 3-14.
4. Srikant, R. & Agrawal, R. (1996), Mining Sequential Patterns: Generalizations and Performance Improvements, in Peter M. G. Apers; Mokrane Bouzeghoub & Georges Gardarin, ed., 'Proc. 5th Int. Conf. Extending Database Technology, EDBT', Springer-Verlag, , pp. 3-17.
5. Pei, J.; Han, J.; Mortazavi-Asl, B.; Wang, J.; Pinto, H.; Chen, Q.; Dayal, U. & Hsu, M.-C. (2004), 'Mining sequential patterns by pattern-growth: the PrefixSpan approach', Knowledge and Data Engineering, IEEE Transactions on 16(11), 1424 - 1440.
6. Cox, M. T. (1997), Loose Coupling of Failure Explanation and Repair: Using Learning Goals to Sequence Learning Models, in 'ICCBR', pp. 425-434
7. Martin, F. (2004), 'Case-Based Sequence Analysis in Dynamic, Imprecise, and Adversarial Domains', PhD thesis, Universitat Politenica de Catalunya.
8. Schmidt, C.; Sridharan, N. & Goodson, J. (1978), 'The plan recognition problem: An intersection of psychology and artificial intelligence', Artificial Intelligence 11(1-2), 45 - 83.

9. Lane, T. & Brodley, C. E. (1999), 'Temporal sequence learning and data reduction for anomaly detection', ACM Trans. Inf. Syst. Secur. 2, 295-331.
10. Luckham, D. (2008), The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, 'Rule Representation, Interchange and Reasoning on the Web', Addison-Wesley, Pearson Education, pp. 3.
11. Aamodt, A. & Plaza, E. (1994), 'Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches', Artificial Intelligence Communications 7, 39-59.
12. Bettini, C.; Sean, X.; Sushil, W.; ling Lin, J. J.; Words, I. K. & Wang, P. X. S. (1998), 'Discovering Temporal Relationships with Multiple Granularities in Time Sequences', IEEE Transations on Knowledge and Data Engineering 10.
13. Marascu, A. & Masseglia, F. (2005), Mining Sequential Patterns from Temporal Streaming Data, in 'Proc. of the 1st ECML/PKDD Workshop on Mining Spatio-Temporal Data (MSTD'05), held in conjunction with the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'05). Pages 291-307.
14. Compta, M., López, B. Integration of sequence learning and CBR for complex equipment failure prediction. In Poc. ICCBR 2011, pages 408-422.