Online Probabilistic Interval-based Event Calculus

Periklis Mantenoglou[1] and **Alexander Artikis**[2,1] and **Georgios Paliouras**[1]

Abstract. Activity recognition systems detect temporal combinations of 'low-level' or 'short-term' activities on sensor data. These systems exhibit various types of uncertainty, often leading to erroneous detection. We present an extension of an interval-based activity recognition system which operates on top of a probabilistic Event Calculus implementation. Our proposed system performs online recognition, as opposed to batch processing, thus supporting data streams. The empirical analysis demonstrates the efficacy of our system, comparing it to interval-based batch recognition, point-based recognition, as well as structure and weight learning models.

1 Introduction

Activity recognition systems consume streams of sensor data from multiple sources, such as cameras and microphones, to identify various types of human behaviour. The input data include *short-term* activities (STA)s, e.g. detected on video frames, such as 'walking', 'running', 'active' and 'inactive', indicating that a person is walking, running, moving his arms while in the same position, etc. The output is a set of *long-term* activities (LTA)s, which are spatio-temporal combinations of STAs. Examples of LTAs are 'meeting', 'leaving unattended objects', 'fighting', and so on.

Uncertainty is inherent in activity recognition applications. STAs are typically detected by visual information processing tools operating on video feeds, and often have probabilities attached to them, serving as confidence estimates. Various approaches have been proposed for handling uncertainty in activity recognition—see [3] for a recent survey. Prob-EC [40] is a system based on a probabilistic logic programming implementation of the Event Calculus [22, 21], designed to handle data uncertainty and compute the probability of an LTA at each time-point. The Probabilistic Interval-Based Event Calculus (PIEC) is an extension of Prob-EC that computes, in linear-time, all maximal intervals during which an LTA is said to take place, with a probability above a given threshold [5]. By supporting interval-based recognition, PIEC has proven robust to noisy instantaneous LTA probability fluctuations, and performs better in the common case of non-abrupt probability change.

We present an extension of PIEC, called oPIEC[b], which is capable of online recognition, as opposed to PIEC's batch processing. This way, oPIEC[b] may handle data streams. More precisely, the contributions of this paper are the following. First, we propose a technique for identifying the minimal set of data points that need to be cached in memory, in order to guarantee correct LTA recognition in a streaming setting. Second, we present a way to further reduce the cached data points, paving the way for highly efficient recognition, while at the same time minimising the effects on correctness. Third, we evaluate our algorithm on a benchmark dataset for human activity recognition, and compare it to interval-based batch recognition, point-based recognition, as well as structure and weight learning models.

The remainder of the paper is structured as follows. Section 2 provides an overview of related research, while Section 3 presents PIEC. Then, Sections 4 and 5 introduce our extension of PIEC. The empirical analysis is presented in Section 6, while in Section 7 we summarise our work and outline further work directions.

2 Related Work

Activity recognition is a type of 'complex event recognition' (CER) [24, 16, 12, 19]. Systems for CER accept as input a stream of time-stamped sensor events and identify composite events of interest—collections of events that satisfy some pattern. See [15, 7, 33, 42] for a few CER applications. The event streams that provide the input data to a CER system exhibit various types of uncertainty [45, 13, 3, 8, 18]. Evidence e.g. may be incomplete, as in the case of occluded objects in activity recognition. Other factors contributing to the corruption of the input stream are the limited accuracy of sensors and distortion along a communication channel. Consequently, the input events are often accompanied by a probability value.

A recent survey [3] identified the following classes of methods for handling uncertainty in CER: automata-based methods, probabilistic graphical models, probabilistic/stochastic Petri Nets and approaches based on stochastic (context-free) grammars. In automata-based methods (e.g. [1, 37, 46]), the representation of time is implicit, while hierarchical knowledge, i.e. defining an LTA in terms of some other LTA, is not supported ([44] is an exception). The approaches that use Petri Nets and stochastic grammars do not allow relations to be defined between attributes of STAs and LTAs.

A typical use of probabilistic graphical models in CER concerns Markov Logic Networks (MLN)s [34]. Morariu and Davis [28] e.g. employed Allen's Interval Algebra [4] to determine the most consistent sequence of LTAs, based on the observations of low-level classifiers. A bottom-up process discards the unlikely event hypotheses, thus avoiding the combinatorial explosion of all possible intervals. This elimination process, however, can only be applied to domain-dependent axioms, as it is guided by the observations. Sadilek and Kautz [35] employed hybrid-MLNs [43] in order to detect human interactions using location data from GPS devices. 'Hybrid formulas' i.e. formulas with weights associated with a real-valued function, such as the distance between two persons, de-noise the location data. In contrast to the above, a *domain-independent* probabilistic activity recognition framework via MLNs was presented in [41]. This framework is based on the Event Calculus [22, 29, 30] and tackles LTA definition uncertainty by modelling imperfect rules expressing LTAs.

There are also logic-based approaches to activity recognition that do not (directly) employ graphical models. The Probabilistic Event

[1] Institute of Informatics & Telecommunications, NCSR Demokritos, Greece, email: {pmantenoglou, a.artikis, paliourg}@iit.demokritos.gr
[2] University of Piraeus, Greece

Logic [7, 36] has been used to define a log-linear model from a set of weighted formulas expressing LTAs [39]. Recognition is performed using 'spanning intervals' that allow for a compact representation of event occurrences satisfying a formula. In [2] LTAs are defined in a first-order logic, the input STAs may be deterministic or probabilistic, while their dependencies are modelled by triangular norms [17]. Shet et al. [38] handled uncertainty by expressing the Bilattice framework in logic programming [20]. Each LTA and STA is associated with two uncertainty values, indicating, respectively, a degree of information and confidence. The more confident information is provided, the stronger the belief about the corresponding LTA becomes.

Skarlatidis et al. presented an activity recognition system based on Prob-EC, a probabilistic logic programming implementation of the Event Calculus [22, 21]. Similar to [41], Prob-EC computes the probability of an LTA at each time-point. Unlike [41], Prob-EC is designed to handle data uncertainty. The use of the Event Calculus, as in e.g. [14], allows the development of domain-independent, expressive activity recognition frameworks, supporting the succinct, intuitive specification of complex LTAs, by taking advantage of the built-in representation of inertia. Consequently, the interaction between activity definition developer and domain expert is facilitated, and code maintenance is supported.

Recently, Artikis et al. [5] proposed the Probabilistic Interval-based Event Calculus (PIEC), a method for computing in linear-time all maximal intervals during which an LTA is said to take place, with a probability above a given threshold. PIEC was proposed as an extension of Prob-EC, but may operate on top of any Event Calculus dialect for point-based probability calculation (such as [41, 14]). By supporting interval-based recognition, PIEC is robust to noisy instantaneous LTA probability fluctuations, and outperforms point-based recognition in the common case of non-abrupt probability change.

Note that various Event Calculus dialects allow for durative LTAs by means of durative events or by explicitly representing 'fluent' intervals. Some examples are [11, 9, 31, 32, 10, 27, 6]. These dialects, however, cannot handle the uncertainty of activity recognition.

PIEC was designed to operate in a batch mode, requiring all available data for correct interval computation. We present an extension of PIEC for online recognition where data arrive in a streaming fashion.

3 Background

The Event Calculus is a formalism for representing and reasoning about events and their effects [22]. Since its original proposal, many dialects have been put forward, including formulations in (variants of) first-order logic and as logic programs. In Prob-EC [40] e.g. a simple version of the Event Calculus was presented, with a linear time model including integer time-points. Where F is a variable ranging over 'fluents', i.e. properties that are allowed to have different values at different points in time, the term $F=V$ denotes that F has value V. A key feature of the Event Calculus is the built-in representation of the common-sense law of inertia, according to which $F=V$ holds at a particular time-point, if $F=V$ has been 'initiated' by an event at some earlier time-point, and not 'terminated' by another event in the meantime. A set of LTA definitions may be expressed as an Event Caclulus *event description*, i.e. axioms expressing the STA occurrrences, the effects of STAs, i.e. the way STAs define LTAs, and the values of fluents expressing LTAs.

The Event Calculus has been expressed in frameworks handling uncertainty, such as ProbLog [21] in the case of Prob-EC [40] and Markov Logic Networks [34] in the case of [41], in order to perform probabilistic, time-point-based LTA recognition. The Probabilistic Interval-based Event Calculus (PIEC) [5] consumes the output of such point-based recognition, in order to compute the 'probabilistic maximal intervals' of LTAs, i.e. the maximal intervals during which an LTA is said to take place, with a probability above a given threshold. Below, we define 'probabilistic maximal intervals'; then, we present the way PIEC detects such intervals in linear-time.

Definition 1 *The probability of interval $I_{LTA}=[i,j]$ of LTA with $length(I_{LTA})=j-i+1$ time-points, is defined as*

$$P(I_{LTA})=\frac{\sum_{k=i}^{j} P(\textsf{holdsAt}(LTA,\ k))}{length(I_{LTA})},$$

where $\textsf{holdsAt}(LTA,k)$ is an Event Calculus predicate expressing the truth value of LTA at time-point k.

In other words, the probability of an interval of some LTA is equal to the average of the LTA probabilities at the time-points that it contains. A key concept of PIEC is that of probabilistic maximal interval:

Definition 2 *A probabilistic maximal interval $I_{LTA}=[i,j]$ of LTA is an interval such that, given some threshold $\mathcal{T}\in[0,1]$, $P(I_{LTA})\geq\mathcal{T}$, and there is no other interval I'_{LTA} such that $P(I'_{LTA})\geq\mathcal{T}$ and I_{LTA} is a sub-interval of I'_{LTA}.*

Probabilistic maximal intervals (PMI)s may be overlapping. To choose a PMI among overlapping ones, PIEC computes the 'credibility' of each such interval—see [5].

Given a dataset of n instantaneous LTA probabilities $In[1..n]$ and a threshold $\mathcal{T}$, PIEC infers all PMIs of that LTA in linear-time. To achieve this, PIEC constructs:

- The $L[1..n]$ list containing each element of In subtracted by the given threshold $\mathcal{T}$, i.e. $\forall\ i\in[1,n]$, we have that $L[i]=In[i]-\mathcal{T}$. Note that an interval I_{LTA} satisfies the condition $P(I_{LTA})\geq\mathcal{T}$ iff the sum of the corresponding elements of list L is non-negative.
- The $prefix[1..n]$ list containing the cumulative or prefix sums of list L, i.e. $\forall\ i\in[1,n]$ we have that $prefix[i]=\sum_{j=1}^{i}L[j]$.
- The $dp[1..n]$ list where, $\forall\ i\in[1,n]$ we have that $dp[i]=\max_{i\leq j\leq n}(prefix[j])$. The elements of the dp list are calculated by traversing the $prefix$ list in reverse order.

Table 1 presents an example dataset $In[1..10]$, along with the lists calculated by PIEC for $\mathcal{T}=0.5$. In this example, there are three PMIs: $[1,5]$, $[2,6]$ and $[8,10]$.

Table 1: PIEC with threshold $\mathcal{T}=0.5$.

Time	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
In	0	0.5	0.7	0.9	0.4	0.1	0	0	0.5	1
L	-0.5	0	0.2	0.4	-0.1	-0.4	-0.5	-0.5	0	0.5
prefix	-0.5	-0.5	-0.3	0.1	0	-0.4	-0.9	-1.4	-1.4	-0.9
dp	0.1	0.1	0.1	0.1	0	-0.4	-0.9	-0.9	-0.9	-0.9

PIEC processes a dataset sequentially using two pointers, s and e, indicating, respectively, the starting point and ending point of a potential PMI. Furthermore, PIEC uses the following variable:

$$dprange[s,e]=\begin{cases} dp[e]-prefix[s-1] & \text{if } s>1\\ dp[e] & \text{if } s=1\end{cases} \qquad (1)$$

$dprange[s,e]$ expresses the maximum sum that may be achieved by adding the elements of list L from s to some $e^*\geq e$, i.e.:

$$\max_{e\leq e^*\leq n}(L[s]+\cdots+L[e^*]).$$

Thus, a non-negative $dprange$ value for some pair of s and e pointers indicates that $\exists e^*$: $e^*\geq e$ and $\sum_{s\leq i\leq e^*}L[i]\geq 0$. Consequently,

$[s, e^*]$ is a potential PMI. In this case, PIEC increments the e pointer until *dprange* becomes negative. When *dprange* becomes negative, PIEC produces the following PMI: $[s, e{-}1]$. Once a PMI is computed, PIEC increments the s pointer and re-calculates *dprange*. By repeating this process, PIEC computes all PMIs of a given dataset.

Example 1 Consider the dataset presented in Table 1 and a threshold $\mathcal{T}=0.5$. Initially, $s=e=1$ and PIEC calculates that $dprange[1, 1]=0.1 \geq 0$. Then, PIEC increments e as long as *dprange* remains non-negative. This holds until $e=6$ when $dprange[1, 6]=-0.4$. At that point, PIEC produces the PMI $[1, 5]$ and increments s. Subsequently, PIEC calculates that $dprange[2, 6]=0.1$ and thus increments e, i.e. e becomes 7 while s remains equal to 2. $dprange[2, 7]=-0.4 < 0$ and, accordingly, PIEC produces the PMI $[2, 6]$ and increments s, i.e. $s=3$. The condition $dprange[s, 7] \geq 0$ is not satisfied until $s=8$ when $dprange[8, 7]=0$. Next, PIEC increments e as long as $dprange[8, e] \geq 0$. This holds for every subsequent time-point of the dataset. Hence, PIEC produces the PMI $[8, 10]$. Summarising, PIEC computes all PMIs of $In[1..10]$. □

By computing PMIs, PIEC outperforms point-based recognition in the presence of noisy instantaneous LTA probability fluctuations, and in the common case of non-abrupt probability change. See [5] for an empirical analysis supporting these claims. On the other hand, PIEC was designed to operate in a batch mode, as opposed to an online setting where data stream in to the recognition system. Consider the example below.

Table 2: PIEC operating on data batches.

Time	1	2	3	4	5	6	7	8	9	10
prefix	-0.5	-0.5	-0.3	0.1	-0.1	-0.5	-1	-1.5	0	0.5
dp	0.1	0.1	0.1	0.1	-0.1	-0.5	-1	-1.5	0.5	0.5

Example 2 Assume that the dataset shown in Table 1 arrives in three batches: $In[1..4]$, $In[5..8]$ and $In[9, 10]$. Table 2 shows the elements of the *prefix* and *dp* lists in this case. $prefix[5]$ e.g. is now equal to $L[5]=-0.1$, since the values of the $L[1..4]$ list are not available at the second data batch. Note that the elements of list L do not change (and are presented in Table 1). Given the first data batch $In[1..4]$, PIEC, starting from time-point 1, increments pointer e as long as $dprange[1, e] \geq 0$. This condition holds for every time-point in the first batch. Hence, PIEC computes the interval $[1, 4]$. Considering the second data batch $In[5..8]$, PIEC does not compute any intervals, as every probability in $In[5..8]$ is lower that $\mathcal{T}$. For the third batch $In[9, 10]$, PIEC initiates with $s=e=9$ and subsequently computes the interval $[9, 10]$, as $dprange[9, 10] \geq 0$. □

None of the intervals $[1, 4]$ and $[9, 10]$ computed in the above example is a PMI. The former could have been extended to the right by one time-point, if the next data batch was foreseen. The latter could have started from time-point 8, if that time-point had been stored for future use. Additionally, the PMI $[2, 6]$ was ignored entirely. One way to address these issues would be to re-iterate over all data received so far upon the receipt of each new data batch. The computational cost of such a strategy, however, is not acceptable in streaming environments (as will be demonstrated in our empirical analysis).

4 Online PIEC

We present an extension of PIEC, called *online Probabilistic Interval-based Event Calculus* (oPIEC), which operates on data batches $In[i..j]$ with $i \leq j$, i.e. oPIEC processes each incoming data batch and then discards it. oPIEC identifies the minimal set of time-points that need to be cached in memory in order to guarantee correct LTA recognition. These time-points are cached in the 'support set' and express the starting points of potential PMIs, i.e. PMIs that may end in the future. For instance, after processing the first data batch $In[1..4]$ in Example 2, oPIEC would have cached time-points $t=1$ and $t=2$ in the support set, thus allowing for the computation of PMIs $[1, 5]$ and $[2, 6]$ in the future. On the contrary, oPIEC would not have cached time-points $t=3$ and $t=4$, because, given that $\mathcal{T}=0.5$, a PMI cannot start from any of these points, irrespective of the data that may arrive after the first batch.

Upon the arrival of a data batch $In[i..j]$, oPIEC computes the values of the $L[i..j]$, $prefix[i..j]$ and $dp[i..j]$ lists. To allow for correct reasoning, the last *prefix* value of a batch is transferred to the next one. Consequently, the *prefix* value of the first time-point of a batch, $prefix[i]$, is set to $prefix[i-1]+L[i]$. (For the first batch, we have, as in PIEC, $prefix[1]=L[1]$.) This way, the computation of the values of $prefix[i..j]$ and $dp[i..j]$ is not affected by the absence of the data prior to i. Subsequently, oPIEC performs the following steps:

1. It computes intervals starting from a time-point in the support set.
2. It computes intervals starting from a time-point in the current batch.
3. It identifies the elements of the current batch that should be cached in the support set.

Step 2 is performed by means of PIEC (see Section 3). In what follows, we present first Step 3 and then move to Step 1.

4.1 Support Set

The support set comprises a set of tuples of the form $(t, prev\_prefix[t])$, where t is a time-point and $prev\_prefix[t]$ expresses t's previous *prefix* value, and is defined as follows:

$$prev\_prefix[t] = \begin{cases} prefix[t-1] & \text{if } t > 1 \\ 0 & \text{if } t = 1 \end{cases} \tag{2}$$

With the use of $prev\_prefix[t]$, oPIEC is able to compute $dprange[t, t']$ for any future time-point t', and thus determine whether t is the starting point of a PMI. For example, the arrival of a time-point $t' > t$ for which $dp[t'] \geq prev\_prefix[t]$ implies that $dprange[t, t'] \geq 0$ (see eq. (1) and (2)), and hence indicates that t is the starting point of a PMI that may end either at t' or at a later time-point.

Algorithm 1 support_set_update$(ignore_value,\ support_set)$

```
for t ∈ In[i..j] do
    if prev_prefix[t] < ignore_value then
        support_set ⇐(add) (t, prev_prefix[t])
        ignore_value ← prev_prefix[t]
    end if
end for
return (ignore_value, support_set)
```

Algorithm 1 identifies the time-points of a data batch $In[i..j]$ that should be cached in the support set. For each time-point $t \in In[i..j]$, we check whether $prev\_prefix[t]$ is less than the $ignore\_value$—this variable expresses the lowest *prev_prefix* value found so far. If $prev\_prefix[t]$ is less than the $ignore\_value$, then we append $(t, prev\_prefix[t])$ to the support set, and set the $ignore\_value$ to $prev\_prefix[t]$.

Example 3 Consider the dataset of the previous examples, arriving in three batches, $In[1..4]$, $In[5..8]$ and $In[9, 10]$, as in Example 2. The values of the $prefix$ list are shown in Table 1—recall that operating on data batches, as opposed to all data received so far, does not affect oPIEC's computation of the $prefix$ list. Algorithm 1 processes every time-point of each batch sequentially. For $t = 1$, we have $prev\_prefix[1] = 0 < ignore\_value$, since, initially, $ignore\_value = +\infty$. Thus the tuple $(1, prev\_prefix[1] = 0)$ is added to the support set and the $ignore\_value$ is set to 0. Next, $t = 2$ and $prev\_prefix[2] = -0.5 < ignore\_value$. Therefore, Algorithm 1 caches the tuple $(2, -0.5)$ and updates the $ignore\_value$. The remaining time-points of the first batch are not added to the support set as their $prev\_prefix$ value does not satisfy the condition $prev\_prefix[t] < ignore\_value$. By processing the remaining batches in a similar way, we get the following support sets:

- $[(1, 0), (2, -0.5)]$; computed after processing batch $In[1..4]$.
- $[(1, 0), (2, -0.5), (8, -0.9)]$; computed after processing batch $In[5..8]$.
- $[(1, 0), (2, -0.5), (8, -0.9), (9, -1.4)]$; computed after processing batch $In[9..10]$.

□

This example illustrates that oPIEC caches the time-points that may be the starting points of PMIs, and no other time-points. A time-point t may be the starting point of a PMI iff:

$$\forall t_{prev} \in [1, t) \text{ we have } prev\_prefix[t_{prev}] > prev\_prefix[t] \quad (3)$$

Theorem 1 *If $[t_s, t_e]$ is a PMI then t_s satisfies condition* (3).

Proof. Suppose that t_s does not satisfy condition (3). Then,

$$\exists t'_s : t'_s < t_s \text{ and } prev\_prefix[t'_s] \leq prev\_prefix[t_s] \quad (4)$$

We have that:

$$\begin{aligned} dprange[t'_s, t_e] &= dp[t_e] - prefix[t'_s - 1] \\ &= dp[t_e] - prev\_prefix[t'_s] \\ &\overset{\text{from ineq. (4)}}{\geq} dp[t_e] - prev\_prefix[t_s] \\ &= dprange[t_s, t_e] \geq 0 \end{aligned}$$

Note that $dprange[t_s, t_e] \geq 0$ because $[t_s, t_e]$ is a PMI.

The fact that $dprange[t'_s, t_e] \geq 0$, as shown above, indicates that $\exists t'_e : t'_e \geq t_e$ and $[t'_s, t'_e]$ is a PMI. Additionally, $[t_s, t_e]$ is a sub-interval of $[t'_s, t'_e]$ since $t'_s < t_s$. Therefore, by Definition 2, $[t_s, t_e]$ is not a PMI. By contradiction, t_s must satisfy condition (3). ■

Note that a time-point t may satisfy condition (3) and not be the starting point of a PMI in a given dataset. See e.g. time-point 9 in Example 3. These time-points must also be cached in the support set because they may become the starting point of a PMI in the future. Consider again Example 3 and assume that a fourth batch arrives with $In[11] = 0$. In this case, we have a new PMI: $[9, 11]$.

Theorem 2 *oPIEC adds a time-point t to the support set iff t satisfies condition* (3).

Proof. oPIEC adds a time-point t to the support set iff the condition of the if statement shown in line 2 of Algorithm 1 is satisfied. When this condition, $prev\_prefix[t] < ignore\_value$, is satisfied, the $ignore\_value$ is set to $prev\_prefix[t]$. Since Algorithm 1 handles every time-point in chronological order, we deduce that in its i^{th} iteration, $ignore\_value$ would be equal to the minimum $prev\_prefix$ value of the first i time-points. So, a time-point's t $prev\_prefix$ value is less than the $ignore\_value$ iff t satisfies condition (3). ■

According to Theorem 2, therefore, oPIEC caches in the support set the *minimal* set of time-points that guarantees correct LTA recognition, irrespective of the data that may arrive in the future.

Algorithm 2 interval_computation$(dp,\ support_set,\ intervals)$

1: $s \leftarrow 1$, $e \leftarrow 1$, $flag \leftarrow false$
2: **while** $s \leq support\_set.length$ and $e \leq dp.length$ **do**
3:   **if** $dp[e] \geq support\_set[s].prev\_prefix$ **then**
4:     $flag \leftarrow true$
5:     e += 1
6:   **else**
7:     **if** $flag == true$ **then**
8:       $intervals \xLeftarrow{add} (support\_set[s].timepoint,\ e-1)$
9:     **end if**
10:     $flag \leftarrow false$
11:     s += 1
12:   **end if**
13: **end while**
14: **if** $flag == true$ **then**
15:   $intervals \xLeftarrow{add} (support\_set[s].timepoint,\ e-1)$
16: **end if**
17: **return** $intervals$

4.2 Interval Computation

We now describe how oPIEC computes PMIs using the elements of the support set. Algorithm 2 shows this process. oPIEC uses a pointer s to traverse the support set, and a pointer e to traverse the dp list of the current data batch. (The elements of the support set and all lists maintained by oPIEC are temporally sorted.) Algorithm 2 starts from the first element s of the support set and the first time-point e of the current data batch, and checks if the interval $[support\_set[s].timepoint, e]$ is or may be extended to a PMI. The condition in line 3 of Algorithm 2 essentially checks whether

$$dprange[support\_set[s].timepoint, e] \quad (5)$$

is non-negative. If it is non-negative, then a PMI starts at $support\_set[s].timepoint$. The Boolean variable $flag$ is set to $true$ and, subsequently, e is incremented as an attempt to find the ending point of the PMI starting at $support\_set[s].timepoint$.

If the value of expression (5) is negative, then oPIEC realises that the interval $[support\_set[s].timepoint, e]$ is not a PMI, and that there is no point in extending it to the right. Consequently, oPIEC checks whether the s, e pointers were pointing to a PMI in the previous iteration. If they were, oPIEC adds the interval of the previous iteration, i.e. $[support\_set[s].timepoint, e-1]$, to the list of PMIs (lines 7-8). Then, it sets $flag$ to $false$, and increments s, as no other PMI may be found from the current element of the support set.

Table 3: oPIEC operating on data batches.

Time	1	2	3	4	5	6	7	8	9	10
prev_prefix	0	-0.5	-0.5	-0.3	0.1	0	-0.4	-0.9	-1.4	-1.4
dp	0.1	0.1	0.1	0.1	0	-0.4	-0.9	-1.4	-0.9	-0.9

Example 4 We complete Example 3 by presenting the interval computation process for the same dataset, arriving in batches $In[1..4]$, $In[5..8]$ and $In[9..10]$. Table 3 displays the values of the dp list

as computed by oPIEC, as well as the $prev\_prefix$ values for aiding the presentation of the example. Upon the arrival of the first batch $In[1..4]$, the support set is empty, and thus Algorithm 2 does not compute any interval. When the second batch $In[5..8]$ arrives, the support set is $[(1, 0), (2, -0.5)]$ (see Example 3). Hence, Algorithm 2 initializes pointer s to 1 and pointer e to 5. Since $dp[5] \geq prev\_prefix[1]$, the *flag* becomes $true$ and e is incremented (see line 4–5 of Algorithm 2). In the following iteration, $dp[6] < prev\_prefix[1]$ and thus Algorithm 2 produces the PMI $[1, 5]$. Next, s is set to 2. Because $dp[6] > prev\_prefix[2]$, Algorithm 2 decides that there is a PMI starting from $t = 2$. However, it fails to extend it in the following iteration. Therefore, Algorithm 2 produces the PMI $[2, 6]$ and terminates for this data batch. When the third batch $In[9..10]$ arrives, the support set is $[(1, 0), (2, -0.5), (8, -0.9)]$. Since $dp[9] = -0.9$ is less than the $prev\_prefix$ values of the first two elements of the support set, Algorithm 2 skips these elements, and sets s to the third element, i.e. $s = 8$. Following similar reasoning, Algorithm 2 increments e and eventually produces the PMI [8, 10]. □

In this example dataset oPIEC computes all PMIs. This is in contrast to PIEC that does not compute any of the PMIs (see Example 2).

5 Bounded Support Set

The key difference between the complexity of oPIEC and PIEC is that the former takes into consideration the support set in interval computation. The size of the support set depends on the data stream of instantaneous probabilities and the value of the threshold $\mathcal{T}$. In brief, high threshold values increase the size of the support set and vice versa. In any case, to allow for efficient reasoning in streaming environments, the support needs to be bound. To address this issue, we present oPIEC^b, i.e. oPIEC with bounded support set. oPIEC^b introduces an algorithm to decide which elements of the support set should be deleted, to make room for new ones. Consequently, when compared to PIEC and oPIEC, oPIEC^b may detect shorter intervals and/or less intervals.

When a time-point t satisfying condition (3) arrives, i.e. t may be the starting point of a PMI, oPIEC^b will attempt to cache it in the support set, provided that the designated support set limit is not exceeded. If it is exceeded, then oPIEC^b decides whether to cache t, replacing some older time-point in the support set, by computing the 'score range':

$$score\_range[t] = [prev\_prefix[t], prev\_prefix[prev_s[t]]) \quad (6)$$

The $score\_range$ is computed for the time-points in set S, i.e. the time-points already in the support set, and the time-points that are candidates for the support set. All the time-points in S satisfy condition (3) and thus are in descending $prev\_prefix$ order. $prev_s[t]$ is the time-point before t in S.

With the use of $score\_range[t]$, oPIEC^b computes the likelihood that a time-point t satisfying condition (3) will indeed become the starting point of a PMI. Suppose e.g. that a time-point $t_e > t$ arrives later in the stream, with $dp[t_e] \in score\_range[t]$ i.e. $prev\_prefix[t] \leq dp[t_e] < prev\_prefix[prev_s[t]]$ (see eq. (6)). In this case, we have $dprange[t, t_e] \geq 0$ and $dprange[prev_s[t], t_e] < 0$ (see eq. (1) and (2)). Hence, a PMI will start from t. The longer the $score\_range[t]$, i.e. the longer the distance between $prev\_prefix[t]$ and $prev\_prefix[prev_s[t]]$, the more likely it is that a future time-point t_e will arrive with $dp[t_e] \in score\_range[t]$, and thus that t will be the starting point of a PMI. Consequently, oPIEC^b stores in the support set the elements with the longer score range.

Algorithm 3 presents the support set maintenance algorithm for a support set of size m and k candidate elements. oPIEC^b gathers every element of the support set and the candidate tuples in set S. The goal is to compute the elements of S with the shortest score ranges, in order to cache the remaining elements in the new support set.

Algorithm 3 support_set_maintenance($support\_set$, new_tuples)

```
m ← support_set.length, k ← new_tuples.length
S ← ∅, S ⟸add support_set, S ⟸add new_tuples
counter ← 1, temp_array ← ∅
for (t, prev_prefix[t]) ∈ S do
    score_range_size ← prev_prefix[prev_s[t]]−prev_prefix[t]
    if counter ≤ k then
        temp_array ⟸add (t, prev_prefix[t], score_range_size)
        if counter == k then
            longest_elem ← find_longest_range(temp_array)
        end if
    else
        if score_range_size < longest_elem.score_range then
            temp_array.delete(longest_elem)
            temp_array ⟸add (t, prev_prefix[t], score_range_size)
            longest_elem ← find_longest_range(temp_array)
        end if
    end if
    counter += 1
end for
for elem ∈ temp_array do
    S.delete(elem)
end for
support_set ← S
return support_set
```

Table 4: oPIEC^b in action.

Time	1	2	3	4	5
In	0	0.3	0.3	0.6	0.9
L	-0.5	-0.2	-0.2	0.1	0.4
prefix	-0.5	-0.7	-0.9	-0.8	-0.4
prev_prefix	0	-0.5	-0.7	-0.9	-0.8
dp	-0.5	-0.7	-0.8	-0.8	-0.4

Example 5 Consider the dataset $In[1..5]$ presented in Table 4. With a threshold value $\mathcal{T}$ of 0.5, this dataset has a single PMI: $[2, 5]$. Assume that the data arrive in two batches: $In[1..4]$ and $In[5]$. Therefore, given an unbounded support set, oPIEC would have cached time-points 1, 2, 3 and 4 into the support set.

Assume now that the limit of the support set is set to two elements. oPIEC^b processes $In[1..4]$ to detect the time-points that may be used as starting points of PMIs, i.e. those satisfying condition (3). These are time-points 1, 2, 3 and 4. Due to the memory limitations, oPIEC^b has to compute the score ranges:

- $score\_range[1]$ is set to $+\infty$ since $t = 1$ has no predecessor in the support set.
- $score\_range[2] = [prev\_prefix[2], prev\_prefix[1]) = [-0.5, 0)$.
- $score\_range[3] = [prev\_prefix[3], prev\_prefix[2]) = = [-0.7, -0.5)$.

- $score\_range[4] = [prev\_prefix[4], prev\_prefix[3]) = [-0.9, -0.7)$.

Given these $score\_range$ values, oPIEC^b caches the tuples $(1, 0)$ and $(2, -0.5)$ in the support set, since these are the elements with the longest score range. oPIEC^b chooses time-point $t = 2$ e.g. over time-points $t = 3$ and $t = 4$ for the support set, because it is more likely that a future time-point t' will have a $dp[t']$ value within $score\_range[2]$ than within $score\_range[3]$ or $score\_range[4]$.

With such a support set, oPIEC^b is able to perform correct LTA recognition, i.e. compute PMI $[2, 5]$, upon the arrival of the second data batch $In[5]$. Note that $dprange[2, 5] = 0.1 \geq 0$ and $t = 5$ is the last time-point of the data stream so far. Also, for the other time-point of the support set, $t = 1$, we have $dprange[1, 5] = -0.4 < 0$, and hence a PMI cannot start from $t = 1$.

Following a somewhat naive maintenance strategy, i.e. deleting the oldest element of the support set to make room for a new one, as opposed to oPIEC^b's strategy based on $score\_range$, would have generated, after processing $In[1..4]$, the following support set: $[(3, -0.7), (4, -0.9)]$. Consequently, upon the arrival of $In[5]$, the interval $[3, 5]$ would have been computed, which is not a PMI. $\square$

6 Experimental Evaluation

6.1 Datasets

To evaluate oPIEC^b we used CAVIAR[3], a benchmark activity recognition dataset. CAVIAR includes 28 videos with 26,419 video frames in total. The videos are staged, i.e. actors walk around, sit down, meet one another, fight, etc. Each video has been manually annotated by the CAVIAR team in order to provide the ground truth for both short-term activities (STA)s, taking place on individual video frames, as well as long-term activities (LTA)s. The input to the activity recognition system consists of the STAs 'inactive', i.e. standing still, 'active', i.e. non-abrupt body movement in the same position, 'walking' and 'running', together with their time-stamps, i.e. the video frame in which the STA took place. The dataset also includes the coordinates of the tracked people and objects as pixel positions at each time-point, as well as their orientation. Given such input, the task is to recognise LTAs such as two people having a meeting or fighting.

The CAVIAR dataset includes inconsistencies, as the members of the CAVIAR team that provided the annotation did not always agree with each other [23, 40]. To allow for a more demanding evaluation of activity recognition systems, Skarlatidis et al. [40] injected additional types of noise into CAVIAR, producing the following datasets:

- *Smooth noise*: a subset of the STAs have probabilities attached, generated by a Gamma distribution with a varying mean. All other STAs have no probabilities attached, as in the original dataset.
- *Strong noise*: probabilities have been additionally attached to contextual information (coordinates and orientation) using the same Gamma distribution, and spurious STAs that do not belong to the original dataset have been added using a uniform distribution.

In the analysis that follows, we use the original CAVIAR dataset as well as the 'smooth noise' and 'strong noise' versions. The target LTAs are 'meeting together' and 'fighting'. All versions of CAVIAR, along with the definitions of these LTAs in the Event Calculus, are publicly available[4].

[3] Section "Clips From INRIA" of `http://groups.inf.ed.ac.uk/vision/CAVIAR/CAVIARDATA1/`

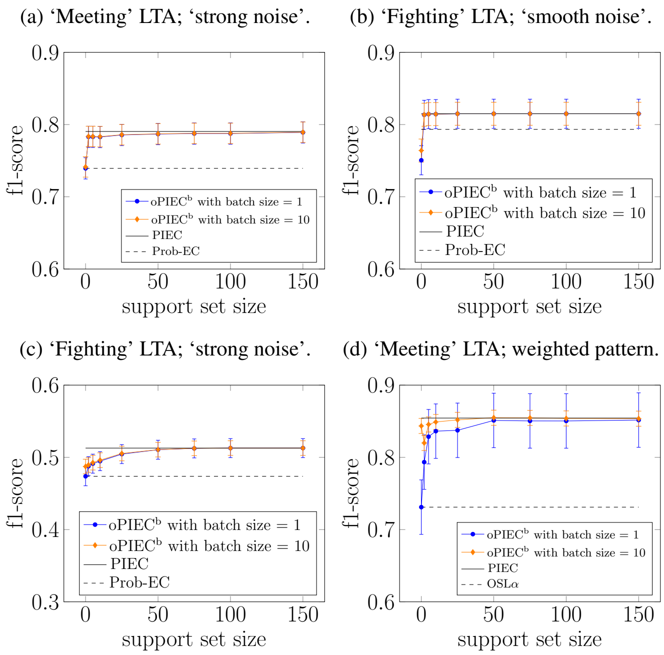
[4] `https://anskarl.github.io/publications/TPLP15/`

6.2 Predictive Accuracy

The aim of the first set of experiments was to compare the predictive accuracy of oPIEC^b against that of PIEC. We focused our comparison on four cases in which PIEC has a noticeably better performance than the underlying system performing point-based LTA recognition. In these experiments, we use two such systems, Prob-EC [40] and OSLα [26, 25]. Prob-EC is an implementation of the Event Calculus in ProbLog [21], designed to handle data uncertainty. OSLα is a supervised learning framework employing the Event Calculus in Markov Logic [41], to guide the search for weighted LTA definitions. Our comparison concerns the following cases:

- Prob-EC recognising the 'meeting' LTA when operating on the 'strong noise' dataset.
- Prob-EC recognising the 'fighting' LTA when operating on the 'smooth noise' and the 'strong noise' datasets. The LTA definitions used by Prob-EC were manually constructed and do not have weights attached [40].
- OSLα recognising the 'meeting' LTA when operating on the original CAVIAR dataset. Prior to recognition, OSLα was trained to construct the LTA definition in the form of weighted Event Calculus rules, given the annotation provided by the CAVIAR team (see [26] for the setup of the training process).

In each case, PIEC and oPIEC^b consumed the output of point-based LTA recognition.



(a) 'Meeting' LTA; 'strong noise'. (b) 'Fighting' LTA; 'smooth noise'. (c) 'Fighting' LTA; 'strong noise'. (d) 'Meeting' LTA; weighted pattern.

Figure 1: Predictive accuracy.

Figure 1 shows the experimental results in the terms of the f1-score, which was calculated using the ground truth of CAVIAR. Each of the diagrams of Figure 1 shows the performance of point-based LTA recognition (Prob-EC in Figures 1(a)–(c) and OSLα in Figure 1(d)), PIEC, oPIEC^b operating on data batches of 1 time-point, i.e. performing reasoning at every time-point and then discarding it, unless cached in the support set (see 'batch size=1' in Figure 1), and oPIEC^b operating on batches of 10 time-points. We used the threshold value leading to the best performance for each system. For recognising 'meeting' and 'fighting' under 'strong noise', we set $\mathcal{T} = 0.5$, while in the recognition of 'meeting' with the weighted definitions of OSLα, we set $\mathcal{T}=0.7$. In these cases where the same threshold value is used for point-based and interval-based recognition, the performance of oPIEC^b when operating on batches of 1 time-point with an empty support set, amounts to the performance of point-based recognition. See Figures 1(a), (c) and (d). In the case of 'fighting' under 'smooth noise', the best performance for Prob-EC is achieved for a different threshold value than that leading to the best performance for PIEC. Thus, Prob-EC operated with $\mathcal{T}=0.5$, while PIEC and oPIEC^b operated with $\mathcal{T}=0.9$.

Figure 1 shows that oPIEC^b reaches the performance of PIEC, even with a small support set, and with the common option for streaming applications of the smallest batch size. In other words, oPIEC^b outperforms point-based recognition, requiring only a small subset of the data.

(a) 'Meeting' LTA; 'strong noise'. (b) 'Fighting' LTA; 'strong noise'.

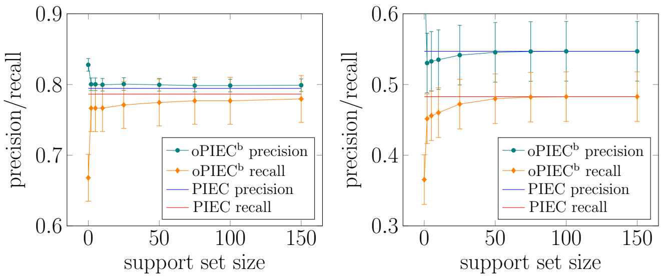


Figure 2: Precision and recall. For oPIEC^b, batch size = 1.

Figure 2 shows the precision and recall scores of oPIEC^b operating on batches of 1 time-point and PIEC, when recognising the 'meeting' and 'fighting' LTAs in the 'strong' noise dataset. In both cases, Prob-EC provides the instantaneous probabilities. In the case of 'fighting', both precision and recall increase for oPIEC^b as the support set increases, eventually reaching the precision and recall of PIEC. In the case of 'meeting', oPIEC^b's precision is initially higher than PIEC's precision, and drops, as the support set increases, close to PIEC's precision. Remember that, due the bounded support set, oPIEC^b may detect shorter intervals and/or less intervals than PIEC. In this case, this leads to correcting some of PIEC's errors, i.e. reducing its false positives.

(a) 'Meeting' LTA; 'strong noise'. (b) 'Fighting' LTA; 'strong noise'.

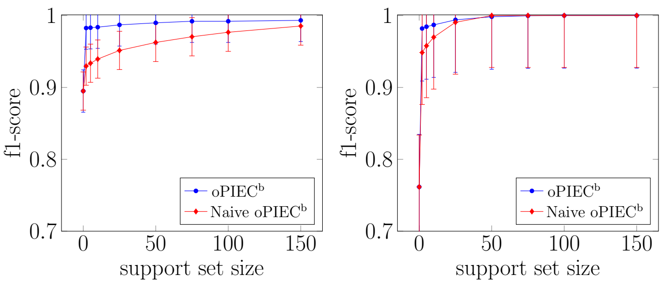


Figure 3: Support set maintenance. Batch size = 1.

The aim of the next set of experiments was to evaluate the effects of the support set maintenance strategy of oPIEC^b. Figure 3 compares the performance of oPIEC^b against that of a naive maintenance strategy, according to which the oldest element of the support set is deleted to make room for a new one. In these experiments, the ground truth was the output of PIEC. As shown in Figure 3, in most cases oPIEC^b's strategy based on the *score_range* (see Section 5) leads to a better approximation of PIEC's performance.

(a) 'Smooth noise'. (b) 'Strong noise'.

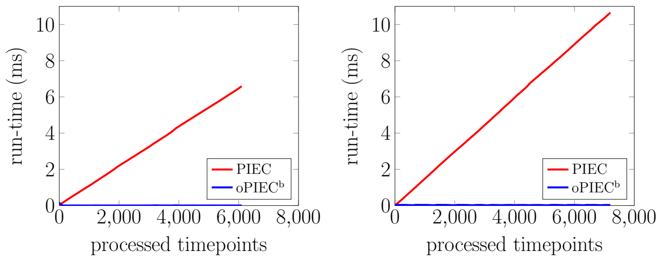


Figure 4: Recognition times for 'fighting': PIEC vs oPIEC^b with a support set limit of 60 elements.

6.3 Recognition Times

In these experiments, the aim was to compare the recognition times of oPIEC^b and PIEC in a streaming setting, i.e. when the recognition system must respond as soon as a data batch arrives. To achieve this, we instructed Prob-EC to recognise the 'fighting' LTA in the videos of the 'smooth' and 'strong noise' datasets with instances of this LTA. Then, we provided the output of Prob-EC in batches of 1 time-point to oPIEC^b and PIEC, for interval-based recognition. Upon the arrival of a data batch, PIEC was instructed to reason over all data collected so far, as this is the only way to guarantee correct PMI computation. oPIEC^b was instructed to operate on a support set limited to 60 elements, as this is sufficient for reaching the accuracy of PIEC. Figure 4 shows the experimental results. Note that the 'strong noise' dataset is larger due to the injection spurious STAs (see Section 6.1). As shown in Figure 4, oPIEC^b has a constant (low) cost, as opposed to the increasing cost of PIEC as data stream in to the recognition system. The comparison of recognition times under different configurations (LTAs, datasets, underlying point-based recognition system) yields similar results, and is not shown here to save space.

7 Summary and Further Research

We presented oPIEC^b, an algorithm for online activity recognition under uncertainty. oPIEC^b identifies the minimal set of data points that need to be cached in memory, in order to guarantee correct activity recognition in a streaming setting. Moreover, oPIEC^b adopts a method for further reducing the cached data points, paving the way for highly efficient recognition, while at the same time minimising the effects on correctness. Our empirical evaluation on a benchmark activity recognition dataset showed that oPIEC^b achieves a higher predictive accuracy than point-based recognition models that have been manually constructed (Prob-EC) or optimised by means of relational learning (OSLα). Moreover, oPIEC^b reaches the predictive accuracy of batch interval-based activity recognition (PIEC), with a small support set, thus supporting streaming applications.

For future work, we aim to develop support set maintenance techniques reducing further the errors of PIEC.

Acknowledgements

This work was supported by the INFORE project, which has received funding from the European Union's Horizon 2020 research and innovation programme, under grant agreement No 825070.

REFERENCES

[1] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, 'Efficient pattern matching over event streams', in *SIGMOD*, pp. 147–160, (2008).

[2] Massimiliano Albanese, Rama Chellappa, Naresh Cuntoor, Vincenzo Moscato, Antonio Picariello, V. S. Subrahmanian, and Octavian Udrea, 'PADS: A Probabilistic Activity Detection Framework for Video Data', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **32**(12), 2246–2261, (2010).

[3] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras, 'Probabilistic complex event recognition: A survey', *ACM Comput. Surv.*, **50**(5), 71:1–71:31, (2017).

[4] James F. Allen, 'Maintaining knowledge about temporal intervals', *Commun. ACM*, **26**(11), 832–843, (1983).

[5] Alexander Artikis, Evangelos Makris, and Georgios Paliouras, 'A probabilistic interval-based event calculus for activity recognition', *Annals of Mathematics and Artificial Intelligence*, (Aug 2019). `https://doi.org/10.1007/s10472-019-09664-4`.

[6] Alexander Artikis, Marek J. Sergot, and Georgios Paliouras, 'An event calculus for event recognition', *IEEE Trans. Knowl. Data Eng.*, **27**(4), 895–908, (2015).

[7] William Brendel, Alan Fern, and Sinisa Todorovic, 'Probabilistic event logic for interval-based event recognition', in *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 3329–3336, (2011).

[8] Nimrod Busany, Avigdor Gal, Arik Senderovich, and Matthias Weidlich, 'Interval-based queries over multiple streams with missing timestamps', (2017).

[9] I. Cervesato and A. Montanari, 'A calculus of macro-events: Progress report', in *TIME*, pp. 47–58, (2000).

[10] F. Chesani, P. Mello, M. Montali, and P. Torroni, 'A logic-based, reactive calculus of events', *Fundamenta Informaticae*, **105**(1-2), 135–161, (2010).

[11] L. Chittaro and A. Montanari, 'Efficient temporal reasoning in the cached event calculus', *Computational Intelligence*, **12**(3), 359–382, (1996).

[12] Gianpaolo Cugola and Alessandro Margara, 'Processing flows of information: From data stream to complex event processing', *ACM Comput. Surv.*, **44**(3), 15:1–15:62, (2012).

[13] Gianpaolo Cugola, Alessandro Margara, Matteo Matteucci, and Giordano Tamburrelli, 'Introducing uncertainty in complex event processing: model, implementation, and validation', *Computing*, **97**(2), 103–144, (2015).

[14] Fabio Aurelio D'Asaro, Antonis Bikakis, Luke Dickens, and Rob Miller, 'Foundations for a probabilistic event calculus', in *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR*, pp. 57–63, (2017).

[15] C. Dousson and P. Le Maigat, 'Chronicle recognition improvement using temporal focusing and hierarchisation', in *IJCAI*, pp. 324–329, (2007).

[16] Opher Etzion and Peter Niblett, *Event Processing in Action*, Manning Publications Company, 2010.

[17] R. Fagin, J. Halpern, and M. Vardi, 'What can machines know? On the properties of knowledge in distributed systems', *Journal of the ACM*, **39**(2), 328–376, (1992).

[18] Avigdor Gal and Nicolo Rivetti, 'Uncertainty in streams', in *Encyclopedia of Big Data Technologies*, (2019).

[19] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos Garofalakis, 'Complex event recognition in the big data era: A survey', *VLDB Journal*, (2019). `https://doi.org/10.1007/s00778-019-00557-w`.

[20] Matthew L. Ginsberg, 'Multivalued logics: a uniform approach to reasoning in artificial intelligence', *Computational Intelligence*, **4**, 265–316, (1988).

[21] A. Kimmig, B. Demoen, L. De Raedt, V. Santos Costa, and R. Rocha, 'On the implementation of the probabilistic logic programming language ProbLog', *Theory and Practice of Logic Programming*, **11**, 235–262, (2011).

[22] Robert A. Kowalski and Marek J. Sergot, 'A logic-based calculus of events', *New Generation Comput.*, **4**(1), 67–95, (1986).

[23] T. List, J. Bins, J. Vazquez, and R. B. Fisher, 'Performance evaluating the evaluator', in *Proceedings 2nd Joint IEEE Int. Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pp. 129–136, (2005).

[24] David C. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*, Wiley, 2011.

[25] Evangelos Michelioudakis, Alexander Artikis, and Georgios Paliouras, 'Semi-supervised online structure learning for composite event recognition', *Machine Learning*, **108**(7), 1085–1110, (2019).

[26] Evangelos Michelioudakis, Anastasios Skarlatidis, Georgios Paliouras, and Alexander Artikis, 'Oslα: Online structure learning using background knowledge axiomatization', in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*, pp. 232–247, (2016).

[27] M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. van der Aalst, 'Monitoring business constraints with the Event Calculus', *ACM TIST*, **5**(1), (2014).

[28] Vlad I. Morariu and Larry S. Davis, 'Multi-agent event recognition in structured scenarios', in *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 3289–3296, (2011).

[29] E. Mueller, 'Event calculus and temporal action logics compared', *Artificial Intelligence*, **170**(11), 1017–1029, (2006).

[30] Erik T. Mueller, *Commonsense Reasoning*, Morgan Kaufmann, 2006.

[31] A. Paschke, 'ECA-RuleML: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics', Technical Report 11, Technische Universität München, (2005).

[32] A. Paschke and M. Bichler, 'Knowledge representation concepts for automated SLA management', *Decision Support Systems*, **46**(1), 187–205, (2008).

[33] Manolis Pitsikalis, Alexander Artikis, Richard Dreo, Cyril Ray, Elena Camossi, and Anne-Laure Jousselme, 'Composite event recognition for maritime monitoring', in *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems, DEBS*, pp. 163–174, (2019).

[34] M. Richardson and P. Domingos, 'Markov logic networks', *Machine Learning*, **62**(1–2), 107–136, (2006).

[35] Adam Sadilek and Henry A. Kautz, 'Location-Based Reasoning about Complex Multi-Agent Behavior', *Journal of Artificial Intelligence Research (JAIR)*, **43**, 87–133, (2012).

[36] Joseph Selman, Mohamed R. Amer, Alan Fern, and Sinisa Todorovic, 'PEL-CNF: Probabilistic event logic conjunctive normal form for video interpretation', in *ICCVW*, pp. 680–687. IEEE, (2011).

[37] Zhitao Shen, Hideyuki Kawashima, and Hiroyuki Kitagawa, 'Probabilistic event stream processing with lineage', in *Proc. of Data Engineering Workshop*, (2008).

[38] Vinay D. Shet, Jan Neumann, Visvanathan Ramesh, and Larry S. Davis, 'Bilattice-based Logical Reasoning for Human Detection', in *(CVPR)*, pp. 1–8. IEEE Computer Society, (2007).

[39] Jeffrey Mark Siskind, 'Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic', *JAIR*, **15**, 31–90, (2001).

[40] Anastasios Skarlatidis, Alexander Artikis, Jason Filipou, and Georgios Paliouras, 'A probabilistic logic programming event calculus', *TPLP*, **15**(2), 213–245, (2015).

[41] Anastasios Skarlatidis, Georgios Paliouras, Alexander Artikis, and George A. Vouros, 'Probabilistic event calculus for event recognition', *ACM Trans. Comput. Logic*, **16**(2), (2015).

[42] Efthimis Tsilionis, Nikolaos Koutroumanis, Panagiotis Nikitopoulos, Christos Doulkeridis, and Alexander Artikis, 'Online event recognition from moving vehicles: Application paper', *TPLP*, **19**(5-6), 841–856, (2019).

[43] Jue Wang and Pedro Domingos, 'Hybrid Markov Logic Networks', in *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pp. 1106–1111. AAAI Press, (2008).

[44] Y. H. Wang, K. Cao, and X. M. Zhang, 'Complex event processing over distributed probabilistic event streams', *Computers & Mathematics with Applications*, **66**(10), 1808–1821, (2013).

[45] Segev Wasserkrug, Avigdor Gal, Opher Etzion, and Yulia Turchin, 'Efficient processing of uncertain events in rule-based systems', *IEEE Trans. Knowl. Data Eng.*, **24**(1), 45–58, (2012).

[46] Haopeng Zhang, Yanlei Diao, and Neil Immerman, 'Recognizing patterns in streams with imprecise timestamps', *VLDB*, **3**(1-2), 244–255, (2010).