Online Event Recognition over Noisy Data Streams

Periklis Mantenoglou[b,a,*], Alexander Artikis[c,a], Georgios Paliouras[a]

[a]*Institute of Informatics and Telecommunications, NCSR Demokritos, Athens, Greece*
[b]*Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece*
[c]*Department of Maritime Studies, University of Piraeus, Greece*

Abstract

Composite event recognition (CER) systems process streams of sensor data and infer composite events of interest by means of pattern matching. Data uncertainty is frequent in CER applications and results in erroneous detection. To support streaming applications, we present oPIEC$^{\text{bd}}$, an extension of oPIEC with a bounded memory, leveraging interval duration statistics to resolve memory conflicts. oPIEC$^{\text{bd}}$ may achieve comparable predictive accuracy to batch reasoning, avoiding the prohibitive cost of such reasoning. Furthermore, the use of interval duration statistics allows oPIEC$^{\text{bd}}$ to outperform significantly earlier versions of bounded oPIEC. The empirical evaluation demonstrates the efficacy of oPIEC$^{\text{bd}}$ on a benchmark activity recognition dataset, as well as real data streams from the field of maritime situational awareness.

Keywords: Event Calculus, temporal pattern matching, probabilistic logic programming, uncertainty, human activity recognition, maritime situational awareness

1. Introduction

'Composite event recognition' (CER) is the process of detecting composite activities/events of interest based on streams of time-stamped 'simple, derived events' (SDEs) [25], i.e., events detected on sensor data. A CER system identifies 'composite events' (CEs), i.e., patterns of SDE combinations subject to temporal and, possibly, atemporal constraints [10]. The distinction between SDEs and CEs imposes an event hierarchy which is innate in many domains of interest, such as human activity recognition [27] and maritime situational awareness [44]. For example, in the latter domain, a 'rendez-vous' of vessels is a suspicious activity during which two proximate vessels have low speed while being far from any port. 'rendez-vous' is a CE comprising vessel activities like 'moving at low speed' or 'having stopped', which may themselves be CEs with patterns based on SDEs like the velocity change of a vessel [44]. Recognising the critical events of a domain and modelling their dependencies is essential for CER. Such dependencies are often captured by logic-based languages, due to their expressivity and the direct routes to explainability [11]. We utilize the Event Calculus [32], a logic programming language for the representation

*Corresponding author
Email addresses: `periklismant@di.uoa.gr` (Periklis Mantenoglou), `a.artikis@unipi.gr` (Alexander Artikis), `paliourg@iit.demokritos.gr` (Georgios Paliouras)

of SDEs and the construction of rules for CE patterns. These patterns may be given by domain experts [51] or learned from data [28, 40].

Uncertainty is inherent in CER applications; consider, for instance, erroneous or incomplete SDEs which may lead to errors in CE recognition. For example, gaps in sensor data are common in human activity recognition as a result of object occlusion in videos [48]. Similarly, in maritime situational awareness, the malfunction of a signal transmitter may lead to a communication gap concerning the whereabouts of a vessel [44]. A catalogue of the sources of such uncertainty/noise may be found in [3].

A typical approach for tackling noise in CER is associating a probability value to incoming SDEs, serving as a confidence estimate (as, e.g., in ProbLog [31] or PITA [12]). In this setting, an incoming stream consists of a sequence of probabilistic facts expressed as $p :: SDE$, where p corresponds to the probability value of a SDE. A probabilistic CER system may then consume such streams to derive the probability of a CE at each point in time [3]. We employ Prob-EC [50] for point-based recognition; Prob-EC is a probabilistic extension of the Event Calculus based on the probabilistic logic programming framework ProbLog [23].

Point-based CER often compromises predictive accuracy. For example, noise may lead to temporary fluctuations in CE recognition probability, while instantaneous recognition probability may be changing too slowly to allow the accurate recognition of the start or the end of a CE [8]. To tackle these issues, we adopt an interval-based approach that consumes instantaneous CE probabilities, such as those derived by Prob-EC, in order to compute the temporal intervals of CE occurrences, paving the way for more robust CER [8]. For interval-based CER, we use PIEC (Probabilistic Interval-based Event Calculus) [8], an algorithm which computes the maximal intervals during which a CE is said to take place, with probability above a given threshold.

In [38], we introduced 'online PIEC' (oPIEC), an extension of PIEC designed to perform reasoning in data batches. oPIEC employs the 'support set', a memory structure with the minimal set of time-points, to guarantee correct interval computation. To support streaming applications, we present oPIEC^{bd}, an extension of oPIEC with a bounded support set, leveraging interval duration statistics to resolve memory conflicts. oPIEC^{bd} may achieve comparable predictive accuracy to batch reasoning, avoiding the prohibitive cost of such reasoning. Moreover, the use of interval duration statistics allows oPIEC^{bd} to outperform significantly earlier versions of bounded oPIEC.

The contributions of this paper are summarised as follows:

- We present a formal analysis of oPIEC (Section 4). First, we prove the correctness of oPIEC, i.e., we show that oPIEC computes exactly the same intervals as PIEC, and then present its complexity, which is significantly lower than that of PIEC.
- We introduce oPIEC^{bd}, a bounded-memory version of oPIEC which takes into account statistics regarding CE interval duration (Section 5.2). Furthermore, we present the complexity of bounded oPIEC (Section 5.3). Our analysis shows that the memory management algorithm of oPIEC^{bd} does not impose any performance delays.
- We present a thorough empirical evaluation on a benchmark dataset for human activity recognition, as well as real data streams from the field of maritime situational awareness (Section 6). In the former use case, we detect composite human activities based on symbolic simple events annotated by experts on video frames. For maritime situational awareness, we process streams of vessel position signals to recognise composite maritime activities, such

as a ‘rendez-vous’ of vessels or a tugging operation. To allow for reproducibility, the code of oPIEC as well as the datasets of the aforementioned applications are publicly available[1]. Our experimental evaluation demonstrates that oPIEC^{bd} is the preferred option for online CER, striking the best balance between predictive accuracy and computational efficiency.

To facilitate understanding, we have also extended the presentation of earlier results [38] with new examples and illustrations.

The remainder of the paper is structured as follows. Section 2 presents the background of this work, i.e., the Event Calculus, Prob-EC for point-based CER and PIEC for batch, interval-based CER. Then, in Section 3 we describe oPIEC, while in Section 4 we prove its correctness and present its complexity. Section 5 presents bounded oPIEC and the corresponding complexity analyses. Subsequently, Section 6 presents our empirical evaluation. Finally, Section 7 compares our work against related research, and Section 8 summarises our work and outlines further research directions.

2. Background

2.1. Event Calculus

The Event Calculus is a formalism for representing and reasoning about events and their effects [32]. Since its inception, many dialects have been put forward, including formulations in (variants of) first-order logic and as logic programs [43, 15, 56]. The ontology of Event Calculus dialects comprises time-points, events and ‘fluents’, i.e., properties that may have different values at different points in time. Event occurrences may change the value of fluents. Hence, the Event Calculus represents the effects of events via fluents. Given a fluent F, the term $F = V$ denotes that F has value V. Boolean fluents are a special case in which the possible values are true and false.

We employ a simple logic programming implementation of the Event Calculus in which time is linear, consisting of non-negative integer time-points [9]. Variables start with an upper-case letter, whereas constants and predicates start with a lower-case letter. $\mathsf{happensAt}(E,\ T)$ denotes that an event of type E occurs at time-point T. In CER, $\mathsf{happensAt}$ is typically used to express the incoming simple, derived events (SDEs). CEs are defined by means of the $\mathsf{initiatedAt}$ and $\mathsf{terminatedAt}$ predicates. These predicates denote, respectively, that a time period during which a fluent F has the value V continuously is initiated/terminated at time-point T. $\mathsf{initially}(F = V)$ expresses that fluent F has the value V at time-point 0, while $\mathsf{holdsAt}(F = V,\ T)$ states that fluent F has value V at time-point T. Table 1 summarises the main predicates of this Event Calculus dialect.

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 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. Consider the following *domain-independent* axiomatisation:

[1] `https://github.com/periklismant/oPIEC`

Table 1: Main Predicates of the Event Calculus.

Predicate	**Meaning**
$\mathsf{happensAt}(E,\ T)$	Event E occurs at time T
$\mathsf{initially}(F = V)$	The value of fluent F is V at time $\mathit{0}$
$\mathsf{holdsAt}(F = V,\ T)$	The value of fluent F is V at time T
$\mathsf{initiatedAt}(F = V,\ T)$	At time T, a period of time for which $F = V$ is initiated
$\mathsf{terminatedAt}(F = V,\ T)$	At time T, a period of time for which $F = V$ is terminated

$$\begin{aligned}&\mathsf{holdsAt}(F = V,\ T) \leftarrow\\ &\quad \mathsf{initially}(F = V),\\ &\quad \mathsf{not\ broken}(F = V,\ \mathit{0},\ T).\end{aligned} \tag{1}$$

$$\begin{aligned}&\mathsf{holdsAt}(F = V,\ T) \leftarrow\\ &\quad \mathsf{initiatedAt}(F = V,\ T_s),\ T_s < T,\\ &\quad \mathsf{not\ broken}(F = V,\ T_s,\ T).\end{aligned} \tag{2}$$

$$\begin{aligned}&\mathsf{broken}(F = V,\ T_s,\ T) \leftarrow\\ &\quad \mathsf{terminatedAt}(F = V,\ T_f),\\ &\quad T_s < T_f < T.\end{aligned} \tag{3}$$

$$\begin{aligned}&\mathsf{broken}(F = V,\ T_s,\ T) \leftarrow\\ &\quad \mathsf{initiatedAt}(F = V',\ T_f),\ V \neq V',\\ &\quad T_s < T_f < T.\end{aligned} \tag{4}$$

'not' expresses negation-by-failure [16], while $\mathsf{broken}(F = V,\ T_s,\ T)$ is an auxiliary predicate that checks whether $F = V$ is terminated, or F is initiated with a value other than V, in (T_s, T). F, V, T_s and T are ground at the time when broken is called. Rules (1) and (2) imply that $F = V$ holds at some time-point T if it held initially, i.e., at time 0, or it has been initiated by an event at a time-point before T, and has not been 'broken' in the meantime. $F = V$ is broken in (T_s, T) if, at an intermediate time-point T_f, $F = V$ is terminated (rule (3)) or $F = V'$ is initiated, for some $V' \neq V$ (rule (4)).

$\mathsf{initiatedAt}(F = V,\ T)$ and $\mathsf{terminatedAt}(F = V,\ T)$ are defined by means of *domain-dependent* rules. Consider the example below from activity recognition on symbolic representations of video content:

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{moving}(P_1, P_2) = \mathsf{true},\ T) \leftarrow\\ &\quad \mathsf{happensAt}(\mathit{walking}(P_1),\ T),\\ &\quad \mathsf{happensAt}(\mathit{walking}(P_2),\ T),\\ &\quad \mathsf{holdsAt}(\mathit{close}(P_1, P_2) = \mathsf{true},\ T),\\ &\quad \mathsf{holdsAt}(\mathit{similarOrientation}(P_1, P_2) = \mathsf{true},\ T).\end{aligned} \tag{5}$$

$$\begin{aligned}&\mathsf{terminatedAt}(\mathit{moving}(P_1, P_2) = \mathsf{true},\ T) \leftarrow\\ &\quad \mathsf{happensAt}(\mathit{walking}(P_1),\ T),\\ &\quad \mathsf{holdsAt}(\mathit{close}(P_1, P_2) = \mathsf{false},\ T).\end{aligned} \tag{6}$$

$$\begin{aligned}&\mathsf{terminatedAt}(\mathit{moving}(P_1, P_2) = \mathsf{true},\ T) \leftarrow\\ &\quad \mathsf{happensAt}(\mathit{active}(P_1),\ T),\\ &\quad \mathsf{happensAt}(\mathit{active}(P_2),\ T).\end{aligned} \tag{7}$$

The above rules specify the conditions for initiating and terminating a time period during which two people P_1 and P_2 are said to be moving together. Rule (5) suggests that if P_1 and P_2 are walking close to each other with a similar orientation, then P_1 and P_2 are moving together. Rules (6)–(7) express some termination conditions of the moving CE. For example, rule (6) states that

$moving$ is terminated when the two people in question walk away from each other. To compute whether 'moving' holds at a given time-point, the domain-independent axiomatisation of inertia, i.e., rules (1)-(4), must be combined with the domain-specific definition of 'moving', i.e., rules (5)-(7). Note that the $moving$ CE may have multiple initiations: two people may be interacting for several video frames. Similarly, $moving$ may have multiple terminations. In this formulation of the Event Calculus, $\mathsf{initiatedAt}(F = V,\ T)$ does not necessarily imply that $F \neq V$ at T. Similarly, $\mathsf{terminatedAt}(F = V,\ T)$ does not necessarily imply that $F = V$ at T. Suppose that $F = V$ is initiated at time-points 50 and 65 and terminated at time-points 75 and 86 (and at no other time-points). In that case, $F = V$ holds at all T such that $50 < T \leq 75$.

2.2. *Probabilistic Event Calculus*

The Event Calculus has been expressed in the probabilistic logic programming framework ProbLog [31, 23] in order to handle noisy input data. We summarise this probabilistic Event Calculus, i.e., Prob-EC, following [50]. Prob-EC processes uncertain indications of SDEs, i.e., a collection of $\mathsf{happensAt}(E,\ T)$ predicates with attached probability values, and derives the probability of CE occurrences by means of Event Calculus rules. In the language of ProbLog, a probabilistic fact is denoted as $p :: f$, indicating that fact f holds as true with probability p in each of its groundings. All such facts are treated as independent variables. To compute the probability of a query q which matches the head of some rules, ProbLog computes the probability of the conjunction of the facts in the body of each rule. Then, the probability of the query is computed as the probability of the disjunction of these rules:

$$P(q) = P(\bigvee_{e \in Proofs(q)} \bigwedge_{f_i \in e} f_i)$$

As an example, suppose that the query q is $\mathsf{terminatedAt}(moving(p_1, p_2) = \mathsf{true},\ t)$, which expresses that the $moving$ CE for persons p_1 and p_2 is terminated at time-point t. This query matches the head of rules (6)–(7) when applying the substitutions: $P_1 \mapsto p_1$, $P_2 \mapsto p_2$ and $T \mapsto t$. The success probability of query q, i.e., the probability that $\mathsf{terminatedAt}(moving(p_1, p_2) = \mathsf{true},\ t)$ holds in a ProbLog program, is computed as follows:

$$\begin{aligned}
&P(\mathsf{terminatedAt}(moving(p_1, p_2) = \mathsf{true},\ t)) = \\
&\quad P((\mathsf{happensAt}(walking(p_1),\ t) \wedge \mathsf{holdsAt}(close(p_1, p_2) = \mathsf{false},\ t)) \\
&\qquad \vee (\mathsf{happensAt}(active(p_1),\ t) \wedge \mathsf{happensAt}(active(p_2),\ t)))
\end{aligned}$$

According to the above expression, the probability of $\mathsf{terminatedAt}(moving(p_1, p_2) = \mathsf{true},\ t)$ is computed as the probability of the disjunction of rules (6)–(7). Each rule succeeds with the same probability as the conjunction of its body literals.

To derive CE occurrences, i.e., $\mathsf{holdsAt}(CE = \mathsf{true},\ T)$, Prob-EC integrates the law of inertia (axioms (1)–(4)) to its probabilistic framework. The probability of $\mathsf{holdsAt}(CE = \mathsf{true},\ T)$ is equal to the probability of the disjunction of the initiation conditions of $CE = \mathsf{true}$ before T, assuming that $CE = \mathsf{true}$ has not been 'broken' in the meantime. This probability is calculated as follows:

$$P(\mathsf{holdsAt}(CE = \mathsf{true},\ t)) = P(\vee_{\forall t_s < t}(\mathsf{initiatedAt}(CE = \mathsf{true},\ t_s) \wedge (\neg\ \mathsf{broken}(CE = \mathsf{true},\ t_s,\ t)))) \tag{8}$$

Therefore, multiple initiations of $CE = \mathsf{true}$ increase its probability. Moreover, if $CE = \mathsf{true}$ is 'broken' with probability p_b, then the probability of $CE = \mathsf{true}$ becomes equal to the product of the probability of the disjunction of the initiations and $1 - p_b$ (see expression (8)). The higher the probability p_b, the more significant the decrease in the probability of $CE = \mathsf{true}$. Consecutive terminations decrease further the probability of $CE = \mathsf{true}$.

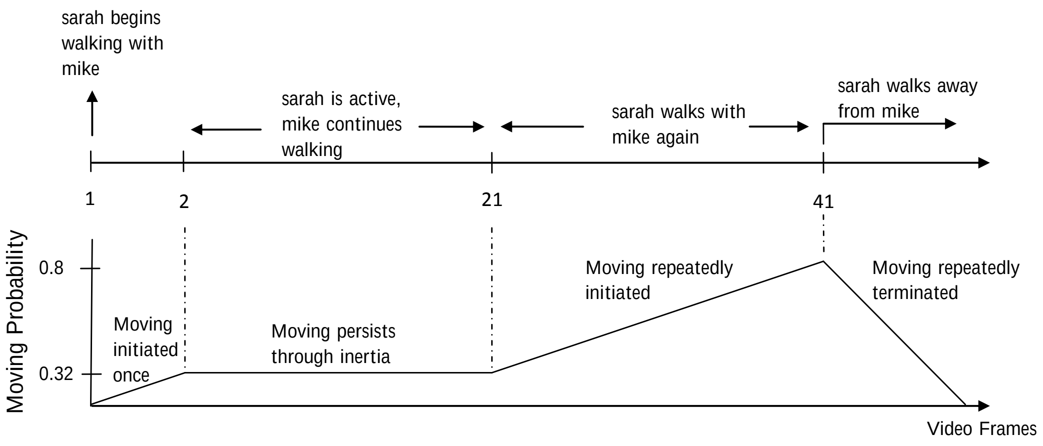

Figure 1: Probabilistic activity recognition with Prob-EC (after [50]).

Figure 1 illustrates the inference mechanism of Prob-EC when the task is to compute the probability of the *moving* CE. In this example, *moving* is initiated at time-point *1*, while no other initiations/terminations of *moving* take place until time-point *21*. Then, *moving* is being initiated repeatedly until time-point *41*. From then on, we have consecutive terminations of *moving* until the end of the video. The probability of the CE increases (decreases) after each initiation (termination), while its probability does not change as long as no initiations/terminations occur. The increase/decrease of the probability is proportional to the probability values of the corresponding input events. For instance, the probability of *moving* at time-point *22* is equal to the probability of the disjunction of the initiations of the CE at time-points 1 and 21, which is calculated as follows:

$$\begin{aligned} P(holds_{22}) = P(init_1 \lor init_{21}) &= P(init_1) + P(init_{21}) - P(init_1 \land init_{21}) \\ &= 0.32 + 0.1 - 0.32 \times 0.1 = 0.388 \end{aligned}$$

where $holds_t$ and $init_t$ are shorthands for $\mathsf{holdsAt}(moving(mike,\ sarah) = \mathsf{true},\ t)$ and $\mathsf{initiatedAt}(moving(mike,\ sarah) = \mathsf{true},\ t)$, respectively, and the probabilities that both people are walking close to each other, with a similar orientation at time-points *1* and *21* are *0.32* and *0.1*, respectively.

2.3. Probabilistic Interval-based Event Calculus

The output of Prob-EC is a sequence of '$p :: \mathsf{holdsAt}(CE = \mathsf{true},\ T)$' indications which denote the evolution of the probability p of CE across time T. Some of these indications may be the result of a sensor's (temporary) malfunction. As an example, Figure 2 displays the probability of a CE as time progresses. Notice that there is an abrupt probability drop between the two peaks; in this example, the drop is the result of a sensor's malfunction. Prob-EC predicts that the target CE has a

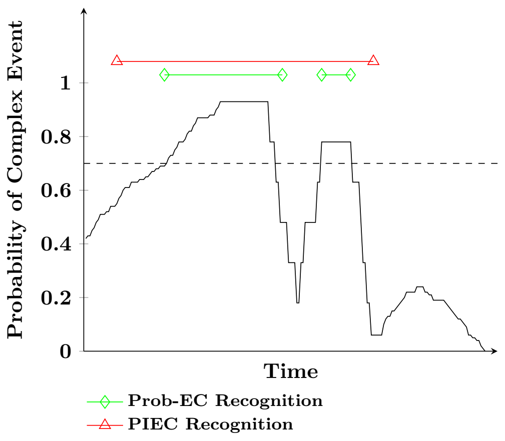

Figure 2: Probabilistic recognition over noisy data streams. The solid black line represents instantaneous CE probabilities, as computed by Prob-EC, while the dotted line designates the chosen threshold $\mathcal{T}$ for recognition. The outcome of Prob-EC, given this threshold value, is presented by the green straight lines. The red line expresses the recognition of PIEC.

low probability of occurrence during this time period. In order to make sense of such indications, we employ a threshold that allows us to distinguish between positive and negative CE instances. In the example of Figure 2, the threshold is equal to 0.7, and the recognition of Prob-EC, i.e., the time-points at which the CE holds with a probability greater than the threshold, is depicted in the form of intervals with the green straight lines. Since during the sensor's malfunction Prob-EC computes probability values lower than the threshold for the CE, there is a gap in the recognition of Prob-EC. All time-points in this gap constitute false negatives, i.e., the CE took place during that gap but was not detected.

To address such issues, i.e., abrupt probability fluctuations which do not adhere to reality, the Probabilistic Interval-based Event Calculus (PIEC) [8] consumes the output of such a point-based recognition, in order to compute the 'probabilistic maximal intervals' of a CE, i.e., the maximal intervals during which a CE is said to take place, with a probability above a given threshold. This way, PIEC is robust to short-term system failures. Below, we define the probability of an interval and probabilistic maximal intervals following [8], and then present the way PIEC detects such intervals.

Definition 1. *The* ***probability of interval*** $I_{CE} = [s, e]$ *of a CE, with* $length(I_{CE}) = e - s + 1$ *time-points, is*

$$P(I_{CE}) = \frac{\sum_{t=s}^{e} P(\mathit{holdsAt}(CE = \mathit{true},\ t))}{length(I_{CE})}\ ,$$

where $P(\mathit{holdsAt}(CE = \mathit{true},\ t))$ *is the probability of occurrence of the CE at time-point* t.

In other words, the probability of an interval of a CE is equal to the average of the instantaneous CE probabilities at the time-points that the interval contains. Note that the instantaneous CE probabilities, i.e., $P(\mathsf{holdsAt}(CE = \mathsf{true},\ t))$, are not independent, since the computation of holdsAt is based on the common-sense law of inertia (see Section 2.2). Computing the average instantaneous probability allows us to smooth out abrupt probability drops caused by sensor malfunction, such as that presented in Figure 2. In contrast, interval probability definitions that employ instantaneous probability multiplication are sensitive to abrupt probability drops.

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

Probabilistic maximal intervals (PMIs) may be overlapping. To choose an interval among overlapping PMIs of the same CE, PIEC computes the *credibility* of each such PMI [8].

By computing PMIs, PIEC addresses the problems of point-based recognition in the presence of noisy instantaneous CE probability fluctuations, and in the common case of non-abrupt probability change. See [8] for an empirical analysis supporting these claims. As an example, Figure 2 shows PIEC operating on top of the probability values computed by Prob-EC. The PMI of PIEC spans over the region of noisy abrupt probability decay, leading to more robust recognition as compared to Prob-EC.

Given a dataset of n instantaneous CE probabilities $In[1..n]$ and a threshold $\mathcal{T}$, PIEC infers all PMIs of the CE in linear time, by translating the problem of PMI computation to the problem of 'maximal non-negative sum interval computation' [5]. To calculate the PMIs, PIEC constructs:

- The $L[1..n]$ list containing each element of In minus the given threshold $\mathcal{T}$, i.e.,
$$\forall\ i \in [1, n], L[i] = In[i] - \mathcal{T} \tag{9}$$
It follows that:
$$\sum_{i=s}^{e} L[i] \geq 0 \overset{Eq.(9)}{\underset{Def.1}{\Longleftrightarrow}} P(I_{CE} = [s, e]) \geq \mathcal{T} \tag{10}$$
Expression (10) states that the probability of an interval I_{CE} is above the given threshold $\mathcal{T}$ iff the sum of the L values of all the time-points i that I_{CE} contains is non-negative.
- The $prefix[1..n]$ list containing the cumulative or prefix sums of list L, i.e.,
$$\forall\ i \in [1, n], prefix[i] = \sum_{j=1}^{i} L[j] \tag{11}$$
- The $dp[1..n]$ list, where
$$\forall\ i \in [1, n], dp[i] = \max_{i \leq j \leq n}(prefix[j]) \tag{12}$$
The elements of the dp list are calculated by traversing the $prefix$ list in reverse order.

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

PIEC additionally uses the following variable:
$$dprange[s, e] = \begin{cases} dp[e] - prefix[s-1] & \text{if } s > 1 \\ dp[e] & \text{if } s = 1 \end{cases} \tag{13}$$
Starting from Eq. (13), and substituting $prefix$ and dp with their respective definitions (Eq. (11) and (12)), we derive that $dprange[s, e]$ expresses the maximum sum that may be achieved by

Table 2: PIEC with threshold $\mathcal{T} = 0.5$ (after [8]).

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

adding the elements of list L from s to some $e^* \geq e$, i.e.:

$$dprange[s, e] = \max_{e \leq e^* \leq n} (L[s] + \cdots + L[e^*]). \tag{14}$$

The following equivalence is a corollary of Eq. (14):

$$dprange[s, e] \geq 0 \Leftrightarrow \exists e^* : e^* \geq e, \sum_{s \leq i \leq e^*} L[i] \geq 0. \tag{15}$$

which, according to equivalence (10), entails that:

$$dprange[s, e] \geq 0 \Leftrightarrow \exists e^* : e^* \geq e, P([s, e^*]) \geq \mathcal{T}. \tag{16}$$

This means that $[s, e^*]$ is a potential PMI.

PIEC processes a dataset of instantaneous CE probabilities sequentially using two pointers directed towards the starting point s and the ending point e of a potential PMI. According to expression (16), if $dprange[s, e]$ is non-negative, then $[s, e^*]$ is a potential PMI, for some $e^* > e$. In that 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 2 and a threshold $\mathcal{T} = 0.5$. Initially, $s = 1$, $e = 1$ 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 $dprange[1, 6] = -0.4$. PIEC produces the PMI $[1, 5]$ and increments s. Afterwards, PIEC calculates that $dprange[2, 6] = 0.1$ and thus increments e, i.e., $e = 7$, while s remains equal to 2. PIEC proceeds by re-calculating $dprange$, i.e., $dprange[2, 7] = -0.4 < 0$, and then produces the PMI $[2, 6]$ and increments s, i.e., $s = 3$. The condition $dprange[s, 7] < 0$ holds $\forall s \in [3, 7]$. Hence, PIEC increments s until $s = 8$ when $dprange[8, 7] = 0$. Note that $\forall t$, $dprange[t + 1, t] = dp[t] - prefix[t] \geq 0$; see the definition of dp (Eq. (12)). Hence, PIEC avoids such erroneous pointer values, i.e., $s > e$, by incrementing e. Here, e increases as long as $dprange[8, e] \geq 0$. This holds for every subsequent time-point of the dataset. Finally, PIEC produces the PMI $[8, 10]$ as $P([8, 10]) \geq \mathcal{T}$ and there is no subsequent time-point to add. Summarising, PIEC computes all PMIs of the dataset $In[1..10]$. $\square$

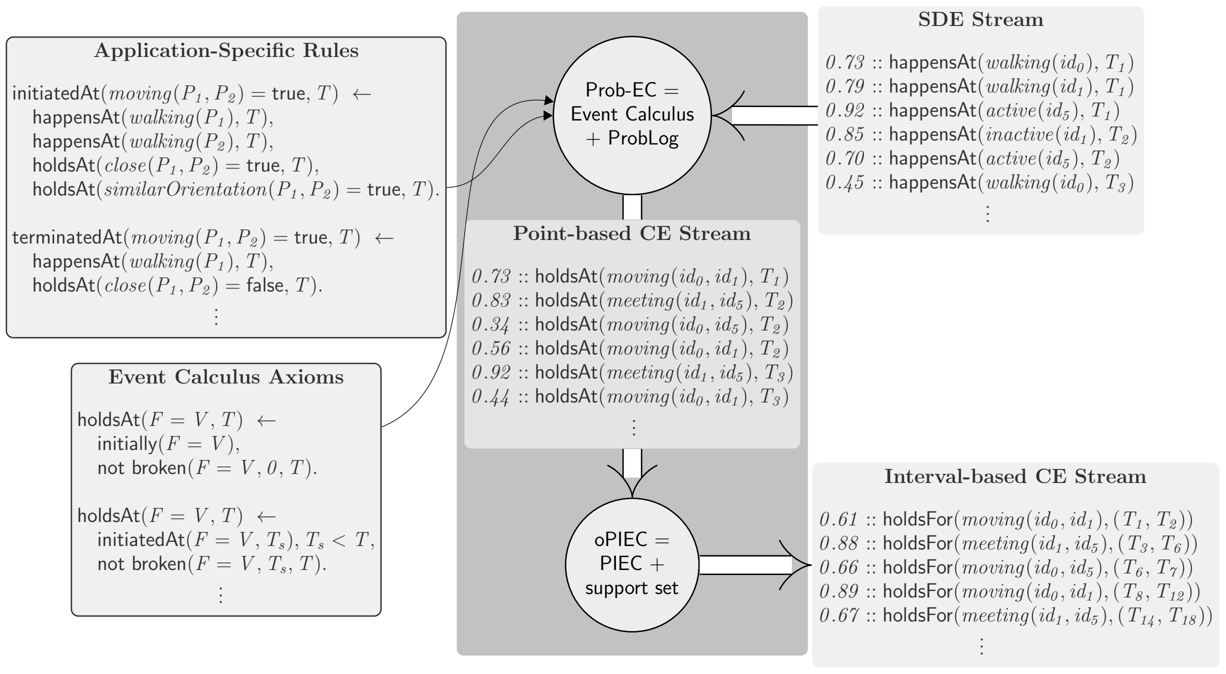
Application-Specific Rules
initiatedAt($moving(P_1, P_2)$ = true, T) ←
happensAt($walking(P_1)$, T),
happensAt($walking(P_2)$, T),
holdsAt($close(P_1, P_2)$ = true, T),
holdsAt($similarOrientation(P_1, P_2)$ = true, T).
terminatedAt($moving(P_1, P_2)$ = true, T) ←
happensAt($walking(P_1)$, T),
holdsAt($close(P_1, P_2)$ = false, T).
⋮
Event Calculus Axioms
holdsAt($F = V, T$) ←
initially($F = V$),
not broken($F = V, 0, T$).
holdsAt($F = V, T$) ←
initiatedAt($F = V, T_s$), $T_s < T$,
not broken($F = V, T_s, T$).
⋮
Prob-EC = Event Calculus + ProbLog
SDE Stream
0.73 :: happensAt($walking(id_0)$, T_1)
0.79 :: happensAt($walking(id_1)$, T_1)
0.92 :: happensAt($active(id_5)$, T_1)
0.85 :: happensAt($inactive(id_1)$, T_2)
0.70 :: happensAt($active(id_5)$, T_2)
0.45 :: happensAt($walking(id_0)$, T_3)
⋮
Point-based CE Stream
0.73 :: holdsAt($moving(id_0, id_1)$, T_1)
0.83 :: holdsAt($meeting(id_1, id_5)$, T_2)
0.34 :: holdsAt($moving(id_0, id_5)$, T_2)
0.56 :: holdsAt($moving(id_0, id_1)$, T_2)
0.92 :: holdsAt($meeting(id_1, id_5)$, T_3)
0.44 :: holdsAt($moving(id_0, id_1)$, T_3)
⋮
oPIEC = PIEC + support set
Interval-based CE Stream
0.61 :: holdsFor($moving(id_0, id_1), (T_1, T_2)$)
0.88 :: holdsFor($meeting(id_1, id_5), (T_3, T_6)$)
0.66 :: holdsFor($moving(id_0, id_5), (T_6, T_7)$)
0.89 :: holdsFor($moving(id_0, id_1), (T_8, T_{12})$)
0.67 :: holdsFor($meeting(id_1, id_5), (T_{14}, T_{18})$)
⋮

Figure 3: Online probabilistic CER. First, Prob-EC, equipped with the Event Calculus axioms (bottom left), and application-specific fluent initiation and termination rules (top left), reasons over a probabilistic data stream (top right) to derive the instantaneous probabilities of the target CEs ('Point-based CE stream' at the center of the picture). Second, oPIEC processes this stream of instantaneous CE probabilities to compute the probabilistic maximal intervals of the target CEs (bottom right).

3. Online Probabilistic Interval-based Event Calculus

PIEC was designed to operate in a batch mode, as opposed to an online setting where data stream into the recognition system. In the online setting, reasoning has to be performed in data batches and thus the predictive accuracy of PIEC may be compromised. To address this issue, we presented oPIEC [38], i.e., *online Probabilistic Interval-based Event Calculus*, an extension of PIEC which operates on data batches $In[i..j]$, where i and j are time-points with $i \leq j$. oPIEC processes each incoming data batch $In[i..j]$ and then discards it. Moreover, it identifies the minimal set of time-points that need to be cached in memory in order to guarantee correct PMI computation. 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. To make the paper self-contained, we summarise oPIEC following [38].

Figure 3 presents the pipeline of online probabilistic CER. Prob-EC reasons over the domain-independent Event Calculus axioms, the domain-dependent rules (e.g., defining when two people are said to be moving together in activity recognition), and a probabilistic data stream, in order to compute instantaneous CE probabilities. With the use of a caching technique, Prob-EC may operate in an online mode [50]. oPIEC consumes the instantaneous CE probabilities, as they stream out of Prob-EC, in order to calculate PMIs.

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, $\mathit{prefix}[i]$, is set to $\mathit{prefix}[i-1]+L[i]$. (For the first batch, $\mathit{prefix}[1]=L[1]$.) This way, the computation of the values of $\mathit{prefix}[i..j]$ 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 all PMIs starting from some time-point in the support set or the data batch $In[i..j]$ and ending in $In[i..j]$.
2. It identifies the minimal set of time-points of the data batch $In[i..j]$ that should be cached in the support set.

In what follows, we first present Step 2 and then move to Step 1.

3.1. Support Set

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

$$\mathit{prev\_prefix}[t] = \begin{cases} \mathit{prefix}[t-1] & \text{if } t > 1 \\ 0 & \text{if } t = 1 \end{cases} \tag{17}$$

With the use of $\mathit{prev\_prefix}[t]$, oPIEC is able to compute $\mathit{dprange}[t,t']$ 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 $dp[t'] \geq \mathit{prev\_prefix}[t]$ implies that $\mathit{dprange}[t,t'] \geq 0$, because

$$dp[t'] \geq \mathit{prev\_prefix}[t] \xrightarrow{Eq.(17)} dp[t'] \geq \mathit{prefix}[t-1] \xrightarrow{Eq.(13)} \mathit{dprange}[t,t'] \geq 0$$

Hence, t is the starting point of a PMI that may end either at t' or at a later time-point (see expression (16)).

oPIEC identifies the time-points of a data batch $In[i..j]$ that should be cached in the support set as follows. For each time-point $t \in [i,j]$, oPIEC checks whether $\mathit{prev\_prefix}[t]$ is less than the $\mathit{min\_prev\_prefix}$, i.e., the lowest $\mathit{prev\_prefix}$ value found so far. If $\mathit{prev\_prefix}[t]$ is less than $\mathit{min\_prev\_prefix}$, then $(t, \mathit{prev\_prefix}[t])$ is added to the support set and $\mathit{min\_prev\_prefix}$ is set to $\mathit{prev\_prefix}[t]$. Finally, the updated support set and the $\mathit{min\_prev\_prefix}$ value are passed to the next batch to ensure correct PMI computation. Example 2 illustrates this functionality of oPIEC.

Example 2. Consider the dataset in Table 2 arriving in three batches, $In[1..4]$, $In[5..8]$ and $In[9,10]$. The values of the prefix list are shown in Table 2—recall that oPIEC generates the same prefix list whether processing data batches or seeing all data in a single batch. oPIEC processes every time-point of each batch sequentially. For $t=1$, $\mathit{prev\_prefix}[1]=0$ is less than $\mathit{min\_prev\_prefix}$, since, initially, $\mathit{min\_prev\_prefix}=+\infty$. Thus, the tuple $(1, \mathit{prev\_prefix}[1]=0)$ is added to the support set and $\mathit{min\_prev\_prefix}$ is set to 0. Next, $t=2$ and $\mathit{prev\_prefix}[2]=-0.5$, which is less than $\mathit{min\_prev\_prefix}$. Therefore, oPIEC caches the tuple $(2,-0.5)$ and updates $\mathit{min\_prev\_prefix}$. The remaining time-points of the first batch are not added to the support set as the condition $\mathit{prev\_prefix} < \mathit{min\_prev\_prefix}$ is not satisfied by their $\mathit{prev\_prefix}$ value. By processing the remaining batches in a similar way, the support set evolves as follows:

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

This example illustrates that oPIEC augments the support set with the time-points having the smallest $\mathit{prev\_prefix}$ value, regarding the data seen so far, and no other time-points.

Theorem 1. *If $[t_s, t_e]$ is a PMI, then t_s satisfies the following condition:*

$$\forall t_{prev} \in [1, t_s),\ \mathit{prev\_prefix}[t_{prev}] > \mathit{prev\_prefix}[t_s] \tag{18}$$

Proof. See [38]. ■

Theorem 2. *oPIEC caches a time-point t_s in the support set iff t_s satisfies condition* (18).

Proof. See [38]. ■

According to Theorems 1 and 2, the starting point of a PMI has the smallest $\mathit{prev\_prefix}$ value up to that point, and oPIEC caches exclusively the time-points which have this property. As a result, oPIEC caches in the support set the *minimal* set of time-points that guarantees correct PMI computation, irrespective of the data that may arrive in the future.

Note that a time-point t may satisfy condition (18) and not be the starting point of a PMI in a given dataset. For instance, see time-point 9 in Example 2. 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 2 and assume that a fourth batch arrives with $In[11] = 0$. In this case, we have a new PMI: $[9, 11]$.

3.2. Interval Computation

We now describe how oPIEC computes PMIs using the support set. Algorithm 1 outlines this process. oPIEC uses a pointer pt_{ss} to traverse the support set; moreover, oPIEC uses two pointers to traverse the prefix and dp lists of the data batch $In[i..j]$, and indicate the starting point s and ending point e of a potential PMI. Note that the elements of all lists are temporally sorted. The task of Algorithm 1 is to compute all PMIs which end in data batch $In[i..j]$, starting from either a time-point in the support set (lines 3–8) or in the current batch $In[i..j]$ (lines 9–15). While processing time-points in the support set, oPIEC calculates $\mathit{dprange}$ for the elements which correspond to the pt_{ss} and e pointers. If the value of $\mathit{dprange}$ is non-negative, then there is a PMI starting at the examined element of the support set and e is incremented in order to find the ending point of the PMI. Otherwise, if the value of $\mathit{dprange}$ is negative, then there is no PMI starting from the given element of the support set. In this case, oPIEC checks whether the ending point of a PMI was identified in the previous iteration, which is indicated by a Boolean flag variable. If this is the case, the interval $[\mathit{support\_set}[pt_{ss}].\mathit{timepoint}, e-1]$ is added to the list of computed PMIs, where $\mathit{support\_set}[pt_{ss}].\mathit{timepoint}$ expresses the time-point of the tuple in the support set indicated by pt_{ss}. At this point, oPIEC increments pt_{ss} in order to check the next potential starting point in the support set.

Once the support set has been processed, oPIEC computes PMIs starting in $In[i..j]$. Again, $\mathit{dprange}$ is computed and, depending on its value, the starting and ending points of PMIs are selected, as explained above. Finally, if after processing $In[i..j]$ there is a pending interval, i.e., an interval whose starting point has been computed but its ending point may be found in future data, then, given the data seen so far, this interval is a PMI which ends at the last time-point of $In[i..j]$, and thus is added to the output of oPIEC.

Algorithm 1 intervalComputation(i, j, $prefix$, dp, $prev\_prefix[i]$, $support\_set$)

Input: Indices i and j indicating the starting and ending point of the current data batch, lists $prefix[i..j]$ and $dp[i..j]$, $prev\_prefix[i]$, i.e., the last value of the $prefix$ list of the previous data batch, and the support set after processing the previous data batch.

Output: A temporally sorted list of PMIs, given the data up to the current data batch $In[i..j]$.

```
1: $s \leftarrow i,\ e \leftarrow i,\ pt_{ss} \leftarrow 1,\ flag \leftarrow false$
2: while $s \leq j$ and $e \leq j$ do
3:     if $pt_{ss} \leq length(support\_set)$ then          ▷ look for starting points in the support set
4:         $dprange \leftarrow dp[e] - support\_set[pt_{ss}].prev\_prefix$
5:         if $dprange \geq 0$ then $flag \leftarrow true,\ e{+}{=}1$
6:         else
7:             if $flag == true$ then $intervals.append((support\_set[pt_{ss}].timepoint, e-1))$
8:             $flag \leftarrow false,\ pt_{ss}{+}{=}1$
9:     else                                             ▷ look for starting points in the current data batch
10:        if $s == i$ then $dprange \leftarrow dp[e] - prev\_prefix[i]$
11:        else $dprange \leftarrow dp[e] - prefix[s-1]$
12:        if $dprange \geq 0$ then $flag \leftarrow true,\ e{+}{=}1$
13:        else
14:            if $s < e$ and $flag == true$ then $intervals.append((s, e-1))$
15:            $flag \leftarrow false,\ s{+}{=}1$
16: if $flag == true$ and $pt_{ss} \leq length(support\_set)$ then        ▷ append pending interval
17:     $intervals.append((support\_set[pt_{ss}].timepoint, e-1))$
18: else if $flag == true$ then $intervals.append((s, e-1))$
19: return $intervals$
```

Example 3. We complete Example 2 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 contents of lists In, L, $prefix$, $prev\_prefix$ and dp for each data batch. The first three lists are the same as in Example 2. Please recall that the consistency of $prefix$ in the presence of data batches is achieved by transferring its last element to the next data batch. The last two lines of Table 3 present the elements added to the support set and the newly computed PMIs after processing each data batch. Upon the arrival of the first batch $In[1..4]$, the support set is empty. Therefore, Algorithm 1 only considers intervals starting in $In[1..4]$ and computes the interval $[1, 4]$, which is a PMI, given the data seen so far. When the second batch $In[5..8]$ arrives, the support set is $[(1, 0), (2, -0.5)]$ (see Example 2). Hence, Algorithm 1 initializes pointer pt_{ss} to 1 and pointer e to 5. Since $dp[5] \geq prev\_prefix[1]$, the $flag$ becomes $true$ and e is incremented (lines 4–5 of Algorithm 1). In the following iteration, $dp[6] < prev\_prefix[1]$ and thus Algorithm 1 produces the PMI $[1, 5]$, which replaces the interval $[1, 4]$. Next, pt_{ss} is set to 2. Because $dp[6] > prev\_prefix[2]$, Algorithm 1 decides that there is a PMI starting at $t = 2$. However, it fails to extend it in the following iteration. Therefore, Algorithm 1 produces the PMI $[2, 6]$ and terminates for this data batch, since $dprange$ is negative for all values of pointers s and e. When the third batch $In[9..10]$ ar-

Table 3: oPIEC operating on data batches.

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
$prev\_prefix$	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
$support\ set$ (new tuples)	$[(1,0),(2,-0.5)]$				$[(8,-0.9)]$				$[(9,-1.4)]$	
$PMIs$ (ending in batch)	$[[1,4]]$				$[[1,5],[2,6]]$				$[[8,10]]$	

rives, the support set is $[(1,0),(2,-0.5),(8,-0.9)]$. Since $dp[9]=-0.9$ is less than any of the $prev\_prefix$ values of the first two elements of the support set, Algorithm 1 skips these elements, and sets $pt_{ss}=3$, for which $support\_set[3]=(8,-0.9)$. Finally, Algorithm 1 increments e as long as $dprange \geq 0$ and eventually computes the PMI $[8,10]$. $\square$

In this example, oPIEC computes all PMIs of the dataset. In contrast, PIEC cannot compute any of the PMIs because, in the data partitioning of this example, the interval starting points have been discarded at the time when their ending points arrive at the system.

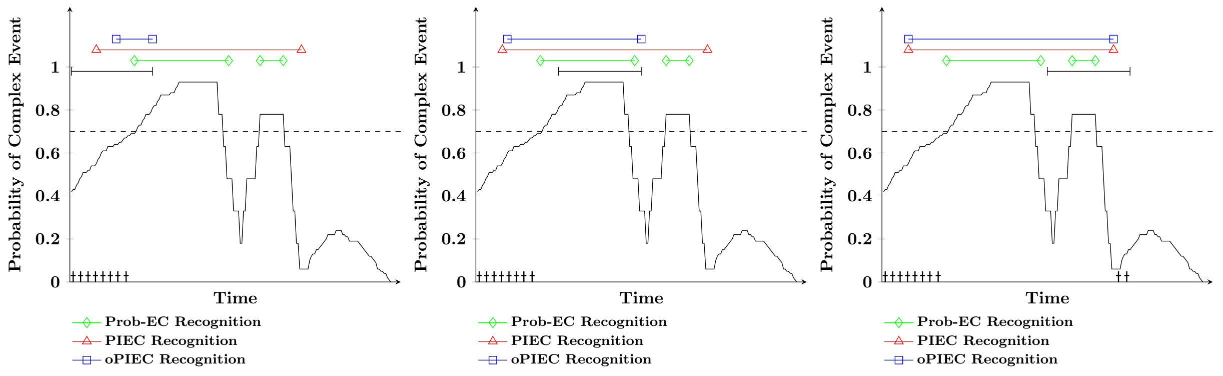

Figure 4: Probabilistic recognition over noisy data streams (continued from Figure 2). The solid black straight line denotes the current data batch. The black crosses on top of the horizontal axis denote the time-points stored in the support set, while the blue straight line denotes the recognition of oPIEC, given the data seen so far.

As another example, Figure 4 shows how oPIEC processes the instantaneous CE probabilities displayed in Figure 2 in three data batches. Figure 4 features three diagrams; each diagram shows the PMIs and the support set of oPIEC after processing each data batch. In the first batch (left diagram), the instantaneous probability of the CE is initially below the threshold $\mathcal{T}$, and it gradually increases, overcoming the threshold at some time-point t, i.e., $In[t] \geq \mathcal{T}$. oPIEC computes a

PMI I_1 including all time-points $t' : t \leq t' \leq j$, where j indicates the end of the data batch, and as many of the previous time-points as possible, while maintaining that $P(I_1) \geq \mathcal{T}$. Next, oPIEC caches the potential starting points of future PMIs based on their *prev_prefix* values. All time-points in $[0,\ t]$ satisfy condition (18) and thus, according to Theorem 2, are cached in the support set.

In the second data batch (middle diagram), oPIEC extends I_1 to a longer PMI I_2 ending in this batch, because, given the newly arrived probability values, I_1 is not a PMI as it is a sub-interval of I_2 and $P(I_2) \geq \mathcal{T}$. Note that I_2 starts earlier than I_1 as the starting point of I_2 was cached in the support set. Moreover, the instantaneous probabilities in this data batch are either above the given threshold or temporally close to those that are above the threshold, and thus the support set does not have to be extended. In the third data batch (right diagram), oPIEC can extend I_2 further to PMI I_3, since $P(I_3) \geq \mathcal{T}$, and thus match the output of PIEC. This batch includes low instantaneous probabilities which decrease the values of the *prev_prefix* list. As a result, $\mathit{prev\_prefix}[t] < \mathit{min\_prev\_prefix}$ holds for some time-points t near the end of the data batch. All such time-points are cached in the support set because they may become starting points of PMIs if high probability values appear later in the stream.

4. Formal Analysis of oPIEC

We extend previous work by proving the correctness of oPIEC, i.e., we demonstrate that every interval computed by oPIEC is a PMI, and every PMI is computed by oPIEC, given the data seen so far. Moreover, we present the complexity analysis of the algorithms for updating the support set and computing PMIs.

4.1. Correctness

Algorithm 1 uses *dprange* to identify PMIs. We associate *dprange* with PMIs using the following lemma.

Lemma 1. *If $I = [s, e]$ is a PMI, then $\mathit{dprange}[s, e]$ is non-negative. Moreover, for every interval $I' = [k, l]$, such that I is a sub-inteval of I', we have $\mathit{dprange}[k, l] < 0$.*

Proof. Since I is a PMI, then $P(I) \geq \mathcal{T}$, i.e., the probability of I is greater or equal to the threshold $\mathcal{T}$. By taking advantage of the definitions of the lists of oPIEC (see Section 2.3), we have the following:

$$P(I) \geq \mathcal{T} \overset{Eq.(10)}{\Longleftrightarrow} \sum_{s \leq m \leq e} L[m] \geq 0 \overset{Eq.(11)}{\Longleftrightarrow} \mathit{prefix}[e] - \mathit{prefix}[s-1] \geq 0.$$

We proceed as follows. According to Eq. (12), we have that $\mathit{dp}[e] = \max_{e \leq m \leq n} \mathit{prefix}[m]$, where n is the last time-point seen by oPIEC, and therefore $\mathit{dp}[e] \geq \mathit{prefix}[e]$. Consequently, it holds that:

$$\mathit{prefix}[e] - \mathit{prefix}[s-1] \geq 0 \Longrightarrow \max_{e \leq m \leq n} \mathit{prefix}[m] - \mathit{prefix}[s-1] \geq 0 \overset{Eq.(12)}{\Longleftrightarrow}$$

$$\mathit{dp}[e] - \mathit{prefix}[s-1] \geq 0 \overset{Eq.(13)}{\Longleftrightarrow} \mathit{dprange}[s, e] \geq 0$$

Therefore, we conclude that if $I = [s, e]$ is a PMI, then $dprange[s, e] \geq 0$.

We will now prove the second part of this lemma. Let $k \leq s, l \geq e, I' = [k, l], I' \neq I$, i.e., I is a sub-interval of I', and $dprange[k, l] \geq 0$. Then, we have:

$$dprange[k, l] \geq 0 \overset{Eq.(13)}{\Longleftrightarrow} dp[l] \geq prefix[k-1] \overset{Eq.(12)}{\Longleftrightarrow}$$

$$\max_{l \leq m \leq n} prefix[m] \geq prefix[k-1] \Longleftrightarrow \exists m : l \leq m \leq n, prefix[m] \geq prefix[k-1] \overset{Eq.(11)}{\Longleftrightarrow}$$

$$\exists m : l \leq m \leq n, \sum_{k \leq m' \leq m} L[m'] \geq 0 \overset{Eq.(10)}{\Longleftrightarrow} \exists m : l \leq m \leq n, P([k, m]) \geq \mathcal{T}$$

The probability of $I'' = [k, m]$, where $m \geq l$, is greater or equal to the threshold and $I \subset I' \subseteq I''$. Therefore, I is not a PMI. We reached a contradiction and thus $dprange[k, l] < 0$. ■

We proceed by presenting the proofs of soundness and completeness of oPIEC, focusing on PMIs starting with an element of the support set. The proofs for PMIs starting with an element of the current data batch are similar and thus omitted.

In the analysis that follows, it is also useful to define the state of Algorithm 1 of oPIEC as a triple $\{pt_{ss}, e, flag\}$, in order to refer to the values of variables pt_{ss}, e and $flag$ at a specific iteration of the *while* loop of Algorithm 1. Recall that pt_{ss} and e are pointers traversing, respectively, the support set and the current data batch, indicating the starting and the ending point of a potential PMI, while $flag$ is a Boolean variable declaring whether $dprange$ was non-negative in the previous iteration. For every PMI in a dataset, oPIEC is guaranteed to reach the state corresponding to the starting and ending point of the PMI, as shown below in Lemma 2.

Lemma 2. *Suppose that $In[i..j]$ is the current data batch, $[sst_1..sst_m]$ is the list of time-points in the support set, and the interval $I = [sst_k, l]$, where $1 \leq k \leq m$ and $i \leq l \leq j$, is a PMI. The execution of Algorithm 1 will eventually reach the state $\{k, l, true\}$ or the state $\{k, l, false\}$.*

Proof. The *while* loop of Algorithm 1 stops searching for PMIs starting with a time-point in the support set when either pt_{ss} has iterated over all elements in the support set, including sst_k, the starting point of I, or e has iterated over all elements of the current data batch, including l, the ending point of I. As a result, Algorithm 1 reaches a state for which either $pt_{ss} = k$ or $e = l$. There are two possibilities for the first encounter of such a state:

1. $\{k', l, true/false\}$, where $k' < k$. Since $I = [sst_k, l]$ is a PMI and a sub-interval of $[sst_{k'}, l]$, we have $dprange[sst_{k'}, l] < 0$ (see Lemma 1). Consequently, the condition of the *if* statement in line 5 of Algorithm 1 fails when its state is $\{k', l, true/false\}$, and pt_{ss} is subsequently incremented. The same holds for all the following states $\{k'', l, true/false\}$, where $k' < k'' < k$, because I is a sub-interval of $[sst_{k''}, l]$. Therefore, pt_{ss} will continue to increase until the algorithm reaches the state $\{k, l, false\}$; the $flag$ is set to $false$ because pt_{ss} is increased in the last iteration before state $\{k, l, false\}$ (see line 8 of Algorithm 1).

2. $\{k, l', true/false\}$, where $l' < l$. Since $I = [sst_k, l]$ is a PMI, we have $dprange[sst_k, l] \geq 0$ (Lemma 1). Equivalently, we have $dp[l] \geq prefix[sst_k - 1]$ (Eq. (13)). As a corollary of the

definition of dp (Eq. (12)), for $t_1 < t_2$, we have:

$$\max_{t_1 \leq t \leq n} (\mathit{prefix}[t]) \geq \max_{t_2 \leq t \leq n} (\mathit{prefix}[t]) \xRightarrow{Eq.(12)} dp[t_1] \geq dp[t_2].$$

In other words, dp is a decreasing sequence and thus $dp[l''] \geq \mathit{prefix}[sst_k - 1]$, when $l'' < l$. So, for every state $\{k, l'', \mathit{true/false}\}$, where $l' \leq l'' < l$, we have that $\mathit{dprange}[sst_k, l''] \geq 0$. The *if* condition in line 5 of Algorithm 1 holds for all these states and e increases continuously until Algorithm 1 reaches the state $\{k, l, \mathit{true}\}$. The *flag* is set to true, because e is increased in the last iteration before state $\{k, l, \mathit{true}\}$ (see line 5 of Algorithm 1).

Hence, given that $I = [sst_k, l]$ is a PMI, the execution of Algorithm 1 reaches either the state $\{k, l, \mathit{true}\}$ or the state $\{k, l, \mathit{false}\}$. ■

Theorem 3 (Soundness of Interval Computation). *Every interval computed by oPIEC is a PMI of the data seen so far.*

Proof. Suppose that $In[i..j]$ is the current data batch and $[sst_1..sst_m]$ is the list of time-points in the support set. Moreover, $I = [sst_k, l]$, where $1 \leq k \leq m, i \leq l \leq j$, is an interval computed by Algorithm 1. We will prove that I is a PMI.

In order for Algorithm 1 to reach line 7 and add the interval $I = [sst_k, l]$ to its results, the state must be $\{k, l+1, \mathit{true}\}$. Recall that the value of *flag* is true iff the *if* condition in line 5 was satisfied in the previous iteration, i.e., $\mathit{dprange}[sst_k, l] \geq 0$. However, since the execution reaches line 7, the *if* condition in line 5 fails in the current iteration, which means that $\mathit{dprange}[sst_k, l+1] < 0$. Hence, we have the following:

$$\mathit{dprange}[sst_k, l] \geq 0 \overset{Eq.(13)}{\Longleftrightarrow} dp[l] \geq \mathit{prefix}[sst_k - 1] \overset{Eq.(12)}{\Longleftrightarrow}$$

$$\max_{l \leq l' \leq j} (\mathit{prefix}[l']) \geq \mathit{prefix}[sst_k - 1] \overset{(a)}{\Longrightarrow} \mathit{prefix}[l] \geq \mathit{prefix}[sst_k - 1] \overset{Eq.(11)}{\Longleftrightarrow}$$

$$\sum_{sst_k \leq t \leq l} L[t] \geq 0 \overset{Eq.(10)}{\Longleftrightarrow} P(I = [sst_k, l]) \geq \mathcal{T}$$

Implication (a) holds because if there existed a $l' > l$ for which $\mathit{prefix}[l'] \geq \mathit{prefix}[l]$, then we would have $\mathit{dprange}[sst_k, l+1] \geq 0$, contradicting the fact that line 7 has been reached. Thus, we conclude that the probability of interval I is greater or equal to the threshold. Next, we will show that I is not a sub-interval of a PMI.

Suppose that I is a sub-interval of the PMI $I' = [sst_{k'}, l']$, i.e., $k' \leq k, l' \geq l, I \neq I'$ and $P(I') \geq \mathcal{T}$. Based on Lemma 2, Algorithm 1 will reach the state $\{k', l', \mathit{true/false}\}$. Also, since I' is a PMI, we have $\mathit{dprange}[sst_{k'}, l'] \geq 0$ (Lemma 1). So, the *if* condition in line 7 of Algorithm 1 is satisfied, e is incremented and the *flag* is set to true, thus reaching the state $\{k', l'+1, \mathit{true}\}$.

Since $I = [sst_k, l]$ is computed by oPIEC, the state $\{k, l+1, \mathit{true}\}$ is also reached. Moreover, since $k' \leq k$, the following transition between states has to take place, possibly in multiple steps:

$$\{k', l'+1, \mathit{true}\} \rightarrow \ldots \rightarrow \{k, l+1, \mathit{true}\}$$

We distinguish between two cases depending on the value of l'. If $l' > l$, the above state transition is infeasible since e does not decrease in Algorithm 1. If $l' = l$, the state transition can only be achieved by increasing pt_{ss} up to the value k while e remains constant. However, incrementing pt_{ss} is accompanied by setting the *flag* to *false* (see line 8 of Algorithm 1). The *flag* can only be reset when incrementing e (in line 5). Therefore, eventually either $e = l + 1$ and $flag = false$, or $e > l + 1$ and $flag = true$. In either case, the aforementioned transition cannot take place.

Consequently, oPIEC cannot reach the state $\{k, l + 1, true\}$ and, as a result, cannot compute I. By contradiction, there is no PMI I' such that I is a sub-interval of I'.

Therefore, since $P(I) \geq \mathcal{T}$ and I is not the sub-interval of a PMI, I is a PMI. Thus, every interval computed by oPIEC is a PMI. ■

Theorem 4 (Completeness of Interval Computation). *oPIEC computes all PMIs of the data seen so far.*

Proof. Suppose that $In[i..j]$ is the current data batch, $[sst_1..sst_m]$ is the list of time-points in the support set, and $I = [sst_k, l]$ is a PMI, where $1 \leq k \leq m$ and $i \leq l \leq j$. Based on Lemma 2, oPIEC will eventually reach a state $\{k, l, true/false\}$ where $pt_{ss} = k$ and $e = l$. Also, since I is a PMI, from Lemma 1, we have that $dprange[sst_k, l] \geq 0$. Therefore, the *if* condition in line 5 of Algorithm 1 will be satisfied, e incremented and the *flag* set to $true$. In the following iteration, the state will become $\{k, l + 1, true\}$. Since I is a PMI and a sub-interval of $[sst_k, l + 1]$, based on Lemma 1, we have that $dprange[sst_k, l + 1] < 0$. Therefore, the *if* condition will not be satisfied and, since the *flag* will be set to $true$, I will be computed in line 7 of Algorithm 1. Hence, every PMI is computed by Algorithm 1. ■

Theorems 3 and 4 show that, given the data seen so far, oPIEC computes every PMI and every interval computed by oPIEC is a PMI. Therefore, oPIEC is correct with respect to the data seen so far.

4.2. Complexity

Before presenting the complexity of oPIEC, we revisit the complexity of PIEC [8]. In PIEC, the computation of lists L, *prefix* and *dp* takes linear time $\Theta(n)$, where n is the number of the input time-points. With respect to PMI computation, in the worst case, the two pointers looking for interval starting and ending points have to traverse the input list of size n. Therefore, this step requires $\mathcal{O}(2n)$. Consequently, the complexity of PIEC may be summarised by the following proposition:

Proposition 1 (PIEC complexity). *The cost of computing the PMIs in a dataset of n time-points with PIEC is $\mathcal{O}(n)$.*

To update the support set, oPIEC iterates over the time-points of the current data batch $In[i..j]$. Therefore, this step is $\mathcal{O}(n_w)$, where n_w is the size of $In[i..j]$. The process of interval computation of oPIEC (Algorithm 1) iterates over the elements of the support set and time-points of $In[i..j]$. In the worst case, Algorithm 1 traverses the support set once and $In[i..j]$ twice. Hence, the complexity of interval computation is $\mathcal{O}(m + 2n_w)$, where m is the size of the support set. After simplifications, the complexity bound of Proposition 2 is reached.

Proposition 2 (oPIEC complexity). *The cost of computing the PMIs of a data batch and updating the support set with oPIEC is*

$$\mathcal{O}(m+n_w), \tag{19}$$

where m is the size of the support set and n_w is the size of the data batch.

The data batch size n_w can be constant, while the support set size m may increase over time. In rare cases, the support set may include a significant portion of the stream seen so far. Consider, e.g., the case where every probability value of the input stream is below the user-defined threshold. Then, the prefix list is strictly decreasing and oPIEC caches in the support set every time-point of the stream. In practice, $m + n_w$ is only a small fragment of the data seen so far, i.e., n. In other words, oPIEC is more efficient than PIEC, since the complexity of the latter increases much faster (with n) than the complexity of the former (which increases with m). In any case, streaming applications require constant complexity. To achieve this, the support set of oPIEC needs to be bounded. In the following section, we present ways in which this can be achieved.

5. Bounded Support Set

In streaming applications, memory and performance requirements demand a bounded support set. In [38], we introduced oPIEC^b, that is, a version of oPIEC in which the support set has a bounded size. oPIEC^b is equipped with a support set maintenance algorithm which decides which elements, if any, should be deleted from the support set, in order to make room for new ones. Consequently, compared to oPIEC which computes all the PMIs of a dataset, oPIEC^b may detect, for instance, fewer or shorter intervals.

Since our previous work [38], we have observed that the support set of oPIEC^b often includes elements with time-stamps much prior to the current time. In most applications, each CE has a typical *duration* and it is not useful to anticipate an instance of a CE which exceeds that duration significantly. As an example, in human activity recognition, it is improbable that two people are fighting for more than five minutes, and thus support set elements with temporal distance greater than five minutes from the current time are very unlikely to be useful for computing PMIs. To take advantage of this property, we propose an extension of the support set maintenance algorithm of oPIEC^b in order to consider CE-specific statistics concerning the duration of PMIs. In the following, we first outline the support set maintenance algorithm of oPIEC^b, and then present *oPIEC^{bd}*, the variant of oPIEC^b which considers the expected PMI duration.

5.1. Support Set Maintenance with oPIEC^b

The support set maintenance algorithm of oPIEC^b works as follows. When a time-point t satisfying condition (18) arrives, i.e., t may be the starting point of a PMI, oPIEC^b considers caching it in the support set. If the designated support set limit is exceeded, then oPIEC^b decides whether to cache t, replacing some older time-point of the support set, by computing the 'score range', i.e., an interval of real numbers defined as follows:

$$score\_range[t] = [prev\_prefix[t], prev\_prefix[prev_s[t]]) \tag{20}$$

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. The elements in S are tuples of the form $(t, prev\_prefix[t])$, which are sorted in ascending time-point order and, since they satisfy condition (18), these tuples are also sorted 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 in set S will indeed become the starting point of a PMI. 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 t will be the starting point of a PMI. Therefore, oPIEC^b stores in the support set the elements with the longer score range.

Table 4: Support set maintenance of oPIEC^b (after [38]).

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 4. 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 $In[5]$. 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. These are time-points 1, 2, 3 and 4. In order to respect the support set limit, oPIEC^b computes the score ranges:

- $score\_range[1]$ is set to $[0, +\infty)$ 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 ranges.

With such a support set, oPIEC^b is able to perform correct CER, i.e., compute PMI $[2, 5]$, upon the arrival of the second data batch $In[5]$. Note that we have $dprange[2, 5] = 0.1 \geq 0$ 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 a negative $dprange$, i.e., $dprange[1, 5] = -0.4 < 0$, and hence a PMI cannot start from $t = 1$. □

Figure 5 describes how oPIEC^b processes the instantaneous CE probabilities displayed in Figure 2 in three data batches, using a support set which may hold at most three elements. After

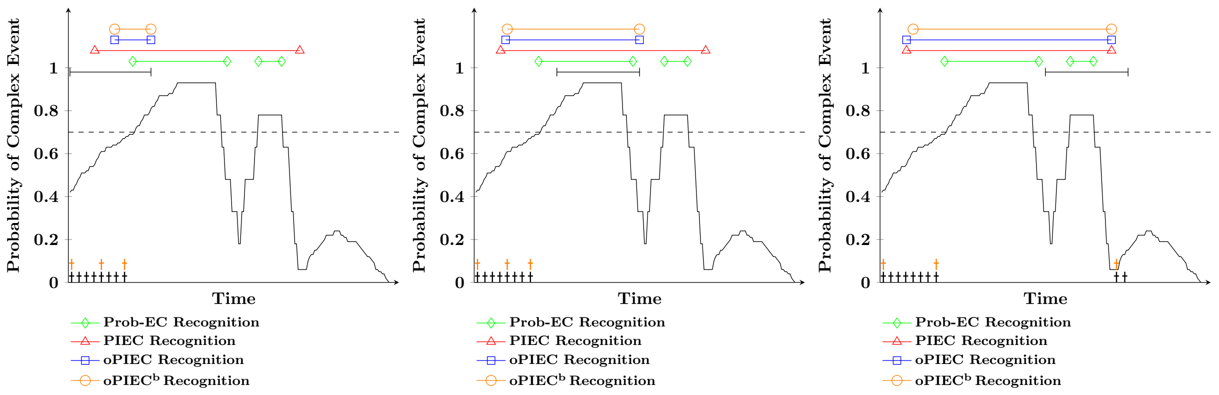

Figure 5: Probabilistic recognition over noisy data streams (continued from Figure 4). The orange straight line expresses the recognition of oPIEC^b after processing the current data batch. Black crosses denote the potential starting points of future PMIs computed by oPIEC/oPIEC^b and cached by oPIEC. Orange crosses depict the ones cached in the bounded support set of oPIEC^b after processing the current data batch.

processing the first data batch (left diagram), oPIEC^b computes the PMI I_1. Then, it identifies the same potential starting points of PMIs as oPIEC, but caches in the support set only three of them. These are the three starting points with the longest $score\_range$.

After processing the second data batch (middle diagram), oPIEC^b extends I_1. There are no potential starting points of PMIs in this data batch, and thus the support set of oPIEC^b is not updated. When oPIEC^b processes the third data batch (right diagram), it extends further I_1 to I_3^b. I_3^b, however, is not a PMI—compare it against the interval computed by PIEC and oPIEC. The starting point of the PMI was not cached in the support set after processing the first data batch, and oPIEC^b could only compute a slightly shorter interval. oPIEC^b updates the support set with a new element, because the low probability values near the end of this data batch have caused a large decrease in the values of $prefix$. As a result, the first candidate element of this batch has a longer $score\_range$ than one of the time-points previously cached in the support set, and thus takes its place.

5.2. Support Set Maintenance with oPIEC^{bd}

It is often the case that oPIEC^b maintains in the support set elements whose time-points are much earlier than the current data batch, while the duration of a CE is usually within some bounds. See, for example, the first cached time-point, depicted as an orange cross, in the right diagram of Figure 5—it is unlikely that this time-point will become the starting point of a PMI in a future data batch. To address this issue, we extend our previous work by introducing oPIEC^{bd}, a variant of oPIEC^b with a support set maintenance algorithm taking advantage of CE-specific knowledge regarding PMI *duration*, which may be obtained by observing historical data. We assume that the duration of PMIs is normally distributed and oPIEC^{bd} is given the mean μ and the standard deviation σ of the observed values. During online execution, when support set conflicts arise, oPIEC^{bd} utilises this information for support set maintenance.

The support set maintenance algorithm of oPIEC^{bd} works as follows. First, oPIEC^{bd} constructs the set S which contains every element of the support set and the candidate elements. When the

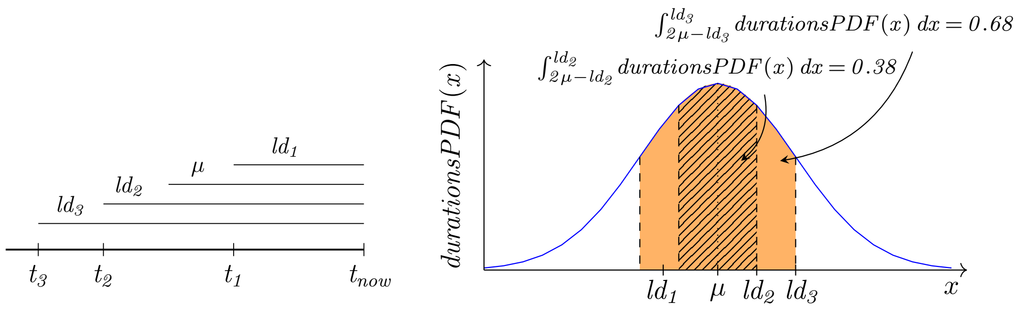
$\int_{2\mu-ld_3}^{ld_3} durationsPDF(x)\,dx = 0.68$
$\int_{2\mu-ld_2}^{ld_2} durationsPDF(x)\,dx = 0.38$
ld_1
μ
ld_2
ld_3
t_3
t_2
t_1
t_{now}
$durationsPDF(x)$
ld_1
μ
ld_2
ld_3
x

Figure 6: The deletion probabilities of the elements of set S whose time-points are t_1, t_2 and t_3. The left diagram shows the temporal positions of these time-points in relation to the last time-point of the current batch t_{now} and their least durations ld_1, ld_2 and ld_3 in relation to the mean duration μ of the observed values. The right diagram visualises the process of calculating the deletion probabilities of these time-points. For time-point t_1, $delPr(t_1) = 0$ because $ld_1 < \mu$. For time-points t_2 and t_3, whose least durations are greater than μ, $delPr$ is equal to the area under $durationsPDF$ defined by the integral of equation (21). As an example, $delPr(t_2)$ is equal to the area under $durationsPDF$ between $2\mu - ld_2$ and ld_2, i.e., $delPr(t_2) = \int_{2\mu-ld_2}^{ld_2} durationsPDF(x)\,dx$.

size of S is greater than the support set limit m_b, oPIEC^{bd} computes, for each element t_i in S, its least duration $ld_i = t_{now} - t_i + 1$, where t_{now} expresses the last time-point of the current batch. ld_i denotes the minimum duration of a PMI starting from t_i and ending in some future data batch. (Upon each incoming data batch, support set maintenance takes place after PMI computation. At this stage, therefore, oPIEC^{bd} has computed the PMIs ending in the current data batch.) Then, oPIEC^{bd} calculates the deletion probability of t_i, defined as follows:

$$delPr(t_i) = \begin{cases} 0 & ld_i \leq \mu \\ \int_{2\mu-ld_i}^{ld_i} durationsPDF(x)\,dx & ld_i > \mu \end{cases} \tag{21}$$

$durationsPDF$ is the probability density function of the normal distribution $\mathcal{N}(\mu, \sigma^2)$ with mean μ and standard deviation σ of the observed duration values. Thus, $\int_{-\infty}^{y} durationsPDF(x)\,dx$ is equal to the probability that a sample generated from $\mathcal{N}(\mu, \sigma^2)$ is smaller than y. As an example, $\int_{-\infty}^{\mu} durationsPDF(x)\,dx = 0.5$ denotes that a sample generated from $\mathcal{N}(\mu, \sigma^2)$ is shorter than the observed mean value μ with a probability equal to 50%. According to expression (21), the deletion probability $delPr(ld_i)$ of a time-point t_i is 0 when its least duration ld_i is shorter or equal to μ. In such cases, it is possible for a future PMI starting from t_i to have a duration shorter or equal to the mean μ of the observed duration values, and therefore it is too early to delete t_i. Otherwise, when all possible future PMIs starting from t_i may have a duration greater than μ, i.e., $ld_i > \mu$, $delPr(ld_i)$ is positive and increases with the probability that a sample generated from the given probability distribution is smaller than ld_i, i.e., the area under the probability density function up to ld_i. The integral in expression (21), however, starts from $2\mu - ld_i$ instead of $-\infty$ in order to have $delPr(\mu) = 0$ and $delPr(\infty) = 1$, i.e., $delPr$ is a cumulative distribution function.

As an example, Figure 6 presents the deletion probabilities of three elements with time-points t_1, t_2 and t_3, given that the observed duration values follow the normal distribution $\mathcal{N}(\mu, \sigma^2)$. The left diagram of Figure 6 shows the temporal positions of these time-points in relation to t_{now}, as well as their least durations. The least durations of t_1, t_2 and t_3 are $ld_1 < \mu$, $ld_2 = \mu + \frac{1}{2}\sigma$ and

$ld_3 = \mu + \sigma$, respectively. The right diagram of Figure 6 shows that the deletion probability of t_1 is 0, because $ld_1 < 0$ (see expression (21)). Moreover, the deletion probabilities of t_2 and t_3, which are equal to the line-highlighted and the orange-coloured areas, respectively, are derived by computing the integral of expression (21). As an example, for time-point t_3, whose least duration is $ld_3 = \mu + \sigma$, we have:

$$delPr(t_3) = \int_{2\mu - ld_3}^{ld_3} durationsPDF(x)\,dx = \int_{\mu-\sigma}^{\mu+\sigma} durationsPDF(x)\,dx = 0.68$$

After each element is assigned a deletion probability, it is removed from set S if this probability is greater than a threshold generated randomly from the uniform distribution $\mathcal{U}(0, 1)$. Since this is a stochastic process, it is not guaranteed that the length of S will be equal to m_b after iterating over all of its elements. If the length of S remains greater than m_b, oPIEC^{bd} invokes the support set maintenance algorithm of oPIEC^{b} to resolve the remaining conflicts.

Algorithm 2 supportSetFiltering(S, m_b, μ, σ, t_{now})

Input: Set S containing the support set elements and the candidate elements of the current data batch, the maximum size of the bounded support set m_b, the mean and the standard deviation of the given duration values μ and σ, and the last time-point of the current data batch t_{now}.
Output: Set S containing m_b tuples which comprise the new support set.

```
1: $durationsPDF \leftarrow \mathcal{N}(\mu, \sigma^2)$
2: for each $(t_i, prev\_prefix[t_i]) \in S$ do
3:     $ld_i \leftarrow t_{now} - t_i + 1$
4:     if $length(S) > m_b$ and $ld_i > \mu$ then
5:         $delPr(ld_i) \leftarrow \int_{2\mu - ld_i}^{ld_i} durationsPDF(x)\,dx$
6:         $threshold \leftarrow random(\mathcal{U}(0, 1))$
7:         if $delPr(ld_i) > threshold$ then $S.delete((t_i, prev\_prefix[t_i]))$
8: if $length(S) > m_b$ then $S \leftarrow$ supportSetMaintenance($S$, $length(S) - m_b$)
9: return $S$
```

Algorithm 2 describes the support set maintenance algorithm of oPIEC^{bd}. First, oPIEC^{bd} generates $durationsPDF$ (line 1). Then, for each time-point t_i in set S, oPIEC^{bd} computes its least duration ld_i (line 3). If the support set limit is exceeded, Algorithm 2 checks whether ld_i is greater than μ (line 4). If it is, we compute the deletion probability of time-point t_i (line 5). Subsequently, Algorithm 2 generates a threshold based on the uniform distribution $\mathcal{U}(0, 1)$ (line 6) and deletes the element of S whose time-point is t_i if its deletion probability exceeds that threshold (line 7). Finally, if the support set limit is still violated after iterating over all elements of S, Algorithm 2 invokes the support set maintenance algorithm of oPIEC^{b} (line 8). After this step, the length of S is guaranteed to be equal to the maximum support set size, and thus Algorithm 2 returns S as the new support set (line 9).

The following example illustrates oPIEC^{bd} and compares it against oPIEC^{b}.

Example 5. Suppose that oPIEC^{b} and oPIEC^{bd} operate on the data batches $In[1..4]$, $In[5..8]$ and $In[9..10]$ of Example 3. The maximum size of the support set for both systems is two elements.

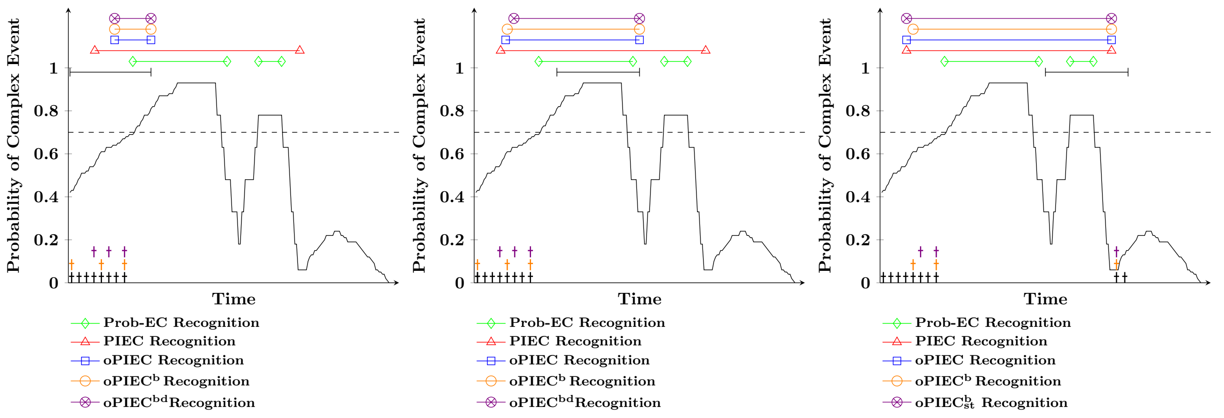

Figure 7: Probabilistic recognition over noisy data streams (continued from Figure 5). The violet straight line expresses the recognition of oPIEC^{bd} after processing the current data batch. Violet crosses depict the potential starting points of future PMIs cached by oPIEC^{bd} after processing the current data batch.

Recall that in Examples 2 and 3, given the same data stream, oPIEC computed the following support set and PMIs:

$$support\_set = [(1, 0), (2, -0.5), (8, -0.9), (9, -1.4)]$$
$$PMIs = [[1, 5], [2, 6], [8, 10]]$$

After processing $In[1..4]$, the support set of both oPIEC^b and oPIEC^{bd} is $[(1, 0), (2, -0.5)]$. Then, when $In[5..8]$ arrives, both systems compute the first two PMIs: $[1, 5]$ and $[2, 6]$. Subsequently, both systems attempt to add the tuple $(8, -0.9)$ to the support set, i.e., $S = [(1, 0), (2, -0.5), (8, -0.9)]$, and thus the respective maintenance algorithm is invoked, since the support set size is limited to two elements.

oPIEC^b computes the $score\_range$ of each element of S:

- $score\_range[1] = [0, +\infty)$.
- $score\_range[2] = [-0.5, 0)$.
- $score\_range[8] = [-0.9, -0.5)$.

Then, oPIEC^b removes element $(8, -0.9)$ because it has the shortest $score\_range$. Therefore, when data batch $In[9..10]$ arrives, oPIEC^b cannot compute the interval $[8, 10]$.

Suppose that the normal distribution fitting the duration data seen so far has $\mu = 4.3$ and $\sigma = 2.4$. Given $t_{now} = 8$, i.e., the last time-point of the current data batch $In[5..8]$, for the first element of S, where $t_i = 1$, we have $ld_i = t_{now} - t_i + 1 = 8$. This means that any future PMI starting from the first element of S will have a duration of at least 8 time-points. However, the PMI duration statistics indicate that PMIs very rarely exceed 8 time-points. When $ld_i > \mu$, oPIEC^{bd} calculates the deletion probability of each element of the set, which for $t_i = 1$ is $delPr(1) = 0.88$. Therefore, element $(1, 0)$ will very likely be deleted from S. If this happens, the support set of oPIEC^{bd} when $In[9..10]$ arrives will be $[(2, -0.5), (8, -0.9)]$ and the final PMI, i.e., $[8, 10]$, will be computed. If $t = 1$ is not deleted, $t = 2$ will be deleted with probability $delPr(2) = 0.74$, leading again to the computation of PMI $[8, 10]$. □

Figure 7 demonstrates how oPIEC^{bd} processes the instantaneous CE probabilities displayed in Figure 2 in three data batches, using a support set which may hold, at most, three elements. Similarly to oPIEC$^{\text{b}}$, after processing the first data batch (left diagram), oPIEC$^{\text{bd}}$ computes all the potential starting points of future PMIs, but caches only three of them. Algorithm 2, using duration statistics regarding the target CE, derives that some of the earlier time-points of the data batch are not likely to be starting points of future PMIs, because these statistics favour intervals with a shorter duration. As a result, the support set maintenance algorithm of oPIEC$^{\text{bd}}$ has cached more recent time-points, compared to oPIEC$^{\text{b}}$. Finally, after processing the third data batch (right diagram), oPIEC$^{\text{bd}}$ computes the correct interval, i.e., the PMI computed by PIEC and oPIEC, because the starting point of the PMI is in the support set.

5.3. Complexity of Support Set Maintenance

In oPIEC, the cost of PMI computation given a data batch with size n_w and a support set with size m is $\mathcal{O}(m{+}n_w)$ (see Section 4.2). In oPIEC$^{\text{b}}$ and oPIEC$^{\text{bd}}$, we have a bounded support set with, at most, m_b elements. The choice of the value of m_b constitutes a trade-off between accuracy and efficiency. For small values of m_b, it is more likely that a PMI is missed because its starting point is not present in the support set. Choosing a larger value for m_b, however, may hinder the efficiency of oPIEC.

Suppose that the support set is full (m_b elements) and there are n_w new candidate elements—in the worst case, there is one candidate element for each time-point in the current data batch $In[i..j]$. Both oPIEC$^{\text{b}}$ and oPIEC$^{\text{bd}}$ utilise set S, containing the elements of the support set and the candidate elements, i.e., $m_b{+}n_w$ elements in total. The support set maintenance algorithm of oPIEC$^{\text{b}}$ removes the n_w elements of S with the shortest score ranges. This is achieved with an iterative process which compares the score range of each element in S, with the largest score range among the n_w elements of S with the shortest score ranges found so far. This operation has a cost of $\mathcal{O}(m_b n_w)$. In total, the cost of oPIEC$^{\text{b}}$ amounts to that of interval computation ($\mathcal{O}(m_b + n_w)$), support set candidate element identification ($\mathcal{O}(n_w)$, because oPIEC checks the $prev\_prefix$ value of each time-point in the data batch) and support set maintenance ($\mathcal{O}(m_b n_w)$). After simplifications, the complexity of oPIEC$^{\text{b}}$ is presented in Proposition 3.

Proposition 3 (oPIEC$^{\text{b}}$ complexity). *The cost of computing the PMIs of a data batch and updating the support set with oPIEC^{b} is*

$$\mathcal{O}(m_b n_w), \tag{22}$$

where m_b is the size of the bounded support set and n_w is the size of the data batch.

In the case of oPIEC$^{\text{bd}}$, support set maintenance is performed by Algorithm 2. The filtering step of Algorithm 2 (lines 1–7) iterates over each element of S once, and removes it from S if its deletion probability exceeds a threshold. Therefore, the cost of this step is $\mathcal{O}(m_b{+}n_w)$. Afterwards, if the support set limit is still exceeded, oPIEC$^{\text{bd}}$ invokes the support set maintenance algorithm of oPIEC$^{\text{b}}$, whose complexity is $\mathcal{O}(m_b n_w)$. Therefore, the cost of support set maintenance in oPIEC$^{\text{bd}}$ is $\mathcal{O}(m_b{+}n_w{+}m_b n_w)$. After simplifications, the complexity of oPIEC$^{\text{bd}}$ is presented in Proposition 4.

Proposition 4 (oPIEC$^{\text{bd}}$ complexity). *The cost of computing the PMIs of a data batch and updating the support set with oPIEC^{bd} is*

$$\mathcal{O}(m_b n_w), \tag{23}$$

where m_b is the size of the bounded support set and n_w is the size of the data batch.

In other words, the support set maintenance algorithm of oPIEC$^{\text{bd}}$ (expression (23)) is of the same complexity order as the corresponding algorithm of oPIEC$^{\text{b}}$ (expression (22)). This is verified by the empirical comparison presented in the following section.

In contrast to oPIEC$^{\text{bd}}$ and oPIEC$^{\text{b}}$, the cost of oPIEC with an unbounded support set is $\mathcal{O}(m+n_w)$, where m is the size of the unbounded support set (see Proposition 2). m increases as the stream progresses, whereas m_b remains constant, and thus, in practice, $m_b \ll m$. Therefore, oPIEC with a bounded support set is the preferred choice for streaming applications.

6. Experimental Evaluation

We evaluated oPIEC$^{\text{bd}}$ on human activity recognition and maritime situational awareness. Our experiments were conducted on a single node of a desktop PC running Ubuntu 20.04 with Intel Core i7-4770 CPU 3.4GHz and 16GB RAM. All components for online probabilistic CER—see Figure 3—as well as the input data streams and execution instructions, are publicly available[1]. Therefore, our experimental analysis is *reproducible*.

6.1. Experimental Setup

We describe the setup of the experiments concerning human activity recognition and maritime situational awareness. We present the input data, i.e., the simple, derived events (SDEs) and accompanying contextual data, and the output composite events (CEs). In all experiments, the task of oPIEC$^{\text{bd}}$ is to compute the PMIs of CEs.

6.1.1. Human Activity Recognition

Data. To evaluate our work, we used CAVIAR[2], 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 SDEs (Simple Derived Events), taking place on individual video frames, as well as CEs (Composite Events). Table 5 describes the SDEs and the CEs of CAVIAR. The input to the activity recognition system consists of SDEs such as ‘inactive’, i.e., standing still, ‘active’, i.e., non-abrupt body movement in the same position, ‘walking’ and ‘running’, together with their time-stamps, that is, the video frame in which the SDE 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. Consider the following example:

$$\mathsf{happensAt}(\mathit{walking}(\mathit{id}_0),\ 680).$$
$$\mathsf{holdsAt}(\mathit{coord}(\mathit{id}_0) = (262, 285),\ 680).$$

[2]https://groups.inf.ed.ac.uk/vision/DATASETS/CAVIAR/CAVIARDATA1/

Table 5: The input SDEs (above the dashed line) and contextual data (below the dashed line), and the output CEs of human activity recognition.

	Entity	Description
SDEs & contextual data	$walking(P)$	Person P is walking.
	$running(P)$	Person P is running.
	$active(P)$	Person P performs non-abrupt body movements without changing position.
	$inactive(P)$	Person P is standing still.
	$abrupt(P)$	Person P performs abrupt body movements.
	$appear(P)$	Person P starts being tracked.
	$disappear(P)$	Person P stops being tracked.
	$coord(P) = (X, Y)$	Person P is at position (X, Y).
	$orientation(P) = \Theta$	Person P is facing at an angle of Θ degrees w.r.t. the x-axis in the two-dimensional video projection.
CEs	$moving(P_1, P_2)$	Persons P_1 and P_2 are moving together.
	$meeting(P_1, P_2)$	Persons P_1 and P_2 are having a meeting.
	$fighting(P_1, P_2)$	Persons P_1 and P_2 are fighting.

$$\mathsf{holdsAt}(orientation(id_0) = 0,\ 680).$$

According to this video frame annotation, id_0 is walking at video frame 680, located at $(262, 285)$ in the two dimensional projection of the video, and facing towards the horizontal axis of this projection.

CAVIAR includes inconsistencies, as the members of the CAVIAR team that provided the annotation for SDEs and CEs did not always agree with each other [36, 50]. Furthermore, to allow for a more demanding evaluation, Skarlatidis et al. [50] injected noise into CAVIAR, producing the following datasets:

- *Smooth noise*: SDEs have been attached with probability values, generated by a Gamma distribution with a varying mean, which signify the probability of their occurrence.
- *Strong noise*: Apart from SDEs, probabilities have also been attached to contextual information (coordinates and orientation) using the same Gamma distributions. Moreover, spurious SDEs that do not belong to the original dataset have been added using a uniform distribution.

Task. Given the SDEs and contextual data of each video frame, the task is to recognise the CEs 'moving', 'meeting' and 'fighting'. The patterns of these CEs are presented in Appendix A.1.

Evaluation. We evaluated the predictive accuracy of oPIEC^{bd} and oPIEC^{b} on the CAVIAR dataset for human activity recognition. The streams of instantaneous CE probabilities we used as input were generated either by Prob-EC or OSLα. Recall that Prob-EC is an implementation of the Event Calculus in ProbLog, designed to handle data uncertainty (see Section 2.2). OSLα translates the Event Calculus into Markov Logic [51], and uses supervised learning to generate weighted CE definitions [41, 40]. OSLα is not designed to handle probabilistic data, and thus was trained on the original CAVIAR dataset (see [41] for the setup of the training process). The CE definitions used by Prob-EC were constructed manually and are not weighted (see Appendix A.1).

The support set maintenance algorithm of oPIEC^{bd} requires CE duration statistics. For this reason, the presented experiments were performed using 5-fold cross-validation. We made sure that all folds were balanced with respect to the number of CE instances, and that no video was split between the training and test sets. In each fold, the durations of the PMIs identified by PIEC in the training set were used when testing oPIEC^{bd} in the corresponding test set. In other words, oPIEC^{bd} had to compute the PMIs in the test set given the mean and the standard deviation of the PMIs of the training set. The overall (micro) f1-score was derived by combining the results of all folds. A perfect f1-score implies that the intervals computed by oPIEC^{b}/oPIEC^{bd} match precisely the PMIs of PIEC. oPIEC^{bd} and oPIEC^{b} operated on data batches of one time-point, i.e., performing reasoning at every time-point and then discarding it, unless cached in the support set.

Table 6: The input SDEs (above the dashed line) and contextual data (below the dashed line), and the output CEs of maritime situational awareness.

	Entity	**Description**
SDEs & contextual data	$enteringArea(V, A)$	Vessel V enters area A.
	$leavesArea(V, A)$	Vessel V exits area A.
	$gapStart(V)$	Vessel V stops sending position signals.
	$gapEnd(V)$	Vessel V resumes sending position signals.
	$stopStart(V)$	Vessel V starts being idle.
	$stopEnd(V)$	Vessel V stops being idle.
	$slowMotionStart(V)$	Vessel V starts moving at a low speed.
	$slowMotionEnd(V)$	Vessel V stops moving at a low speed.
	$proximityStart(V_1, V_2)$	Vessels V_1 and V_2 start being close to each other.
	$proximityEnd(V_1, V_2)$	Vessels V_1 and V_2 stop being close to each other.
	$coord(V) = (Lat, Lon)$	Vessel V is at position (Lat, Lon).
	$velocity(V) = S$	Vessel V sails at speed S.
CEs	$rendezVous(V_1, V_2)$	Vessels V_1 and V_2 are nearby in the open sea, stopped or sailing at a low speed.
	$tugging(V_1, V_2)$	Vessels V_1 or V_2 is a tugboat and is pulling or towing the other vessel.

6.1.2. Maritime Situational Awareness

Data. To test further oPIEC^{bd}, we employed a publicly available dataset[3] which includes 18 million Automatic Identification System (AIS) position signals collected from 5 thousand ships sailing in the area of Brest, France, for a period of six months, between October 2015 and March 2016. AIS messages comprise dynamic information (position, speed, etc.), as well as static information (name, type, etc.) about vessels [13]. This dataset has been pre-processed by means of trajectory simplification [24] and spatial processing [45]. The former process annotated position signals of interest as 'critical', signifying major changes in a vessel's behaviour such as a stop, a turn, and a gap in signal transmission, while the latter process computed spatial relations between vessels, such as vessels being close to each other. Table 6 describes the SDEs and CEs of the Brest dataset. Consider the following example of input data:

$$\begin{array}{l}\mathsf{happensAt}(\mathit{leavesArea}(\mathit{id_2}, \mathit{fishing}),\ \mathit{492}).\\ \mathsf{holdsAt}(\mathit{coord}(\mathit{id_2}) = (\mathit{48.12}, -\mathit{4.35}),\ \mathit{492}).\\ \mathsf{holdsAt}(\mathit{velocity}(\mathit{id_2}) = \mathit{5.13},\ \mathit{492}).\end{array}$$

According to these records, vessel $\mathit{id_2}$ leaves some fishing area at time-point $\mathit{492}$; moreover, $\mathit{id_2}$ is located at $(\mathit{48.12}, -\mathit{4.35})$ and sailing with a speed of $\mathit{5.13}$ knots.

We injected noise into the dataset of Brest by assigning a probability value to every item of the dataset. We followed [62] and assumed that the confidence of an AIS signal decreases as the distance of the corresponding vessel from the nearest coastline base station increases, and annotated each signal of the dataset with a probability using the following function:

$$P_c(x) = \mathit{1} - \frac{x}{x + c}$$

x is the distance of the ship from the nearest base station and c is a distance threshold denoting our confidence concerning the veracity of signals. According to the function above, the probability of a position signal decreases as the value of c decreases. We constructed a 'smooth noise' and a 'strong noise' version of the Brest dataset by applying the noise functions P_{10000} and P_{5000} on the original dataset.

Task. Given the input presented in Table 6, the task is to recognise the CEs 'rendez-vous', i.e., a potential ship-to-ship transfer of goods in the open sea, and 'tugging', i.e., the activity of pulling a ship into or out of a port. These CEs were defined following expert opinion [44] and their patterns are presented in Appendix A.2.

Evaluation. For our experiments on maritime situational awareness, we employed Prob-EC to produce probability values for the instantaneous occurrences of the target CEs, i.e., 'rendez-vous' and 'tugging'. Subsequently, we evaluated oPIEC^{b} and oPIEC^{bd} on the derived probability streams. We compared the intervals computed by oPIEC^{b} and oPIEC^{bd} when processing online the CE probabilities of Prob-EC against the PMIs computed by PIEC when processing the same data as a single batch. The duration statistics required by oPIEC^{bd} were derived by a cross-validation setting similar to that of the experiments on human activity recognition. We made sure that the folds were balanced in terms of CE instances, and that the set of vessels in the training set is disjoint from the set of vessels in the test set. The batch size of oPIEC^{b} and oPIEC^{bd} was set to

[3]`https://zenodo.org/record/1167595`

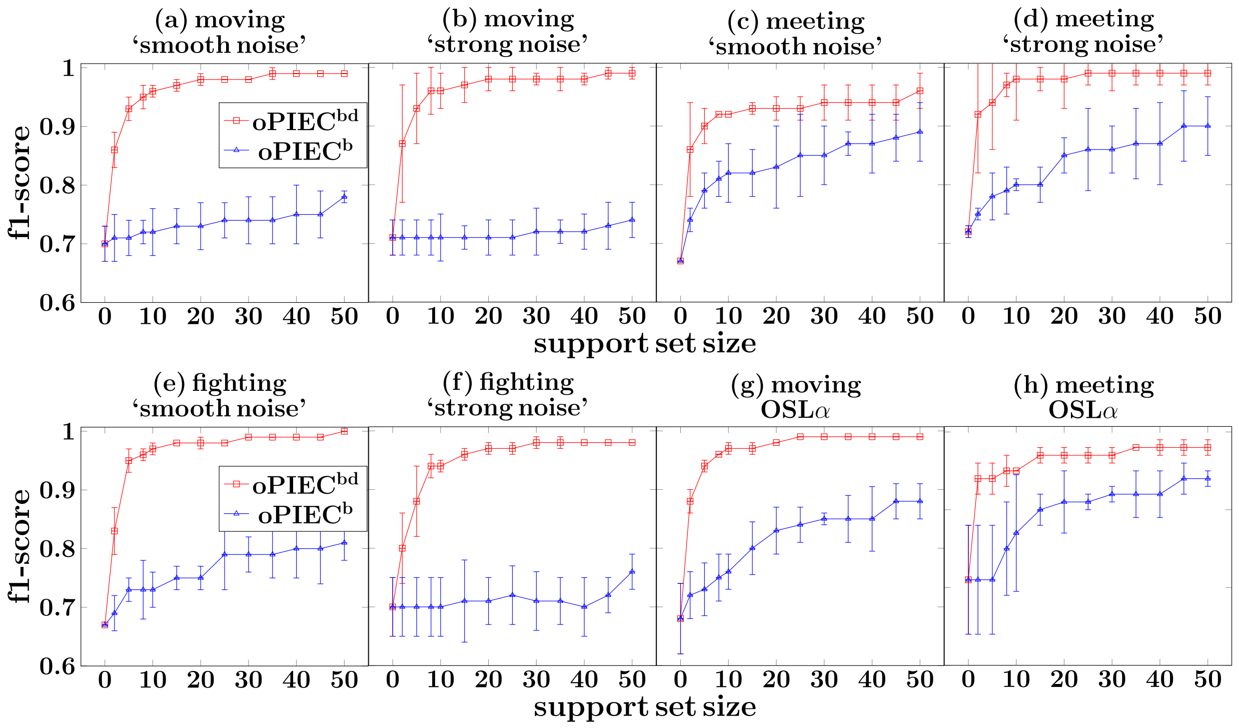

Figure 8: The predictive accuracy of oPIEC$^{\mathrm{bd}}$ and oPIEC$^{\mathrm{b}}$ on human activity recognition. The intervals computed by oPIEC$^{\mathrm{bd}}$ and oPIEC$^{\mathrm{b}}$ are compared to the PMIs computed by PIEC. The point-based systems providing the input probability streams are Prob-EC (diagrams (a)–(f)) and OSLα (diagrams (g)–(h)).

one, while the probabilistic threshold $\mathcal{T}$ was set to 50%.

6.2. Experimental Results

Human Activity Recognition. Figure 8 presents the predictive accuracy of oPIEC$^{\mathrm{bd}}$ and oPIEC$^{\mathrm{b}}$, operating with a probabilistic threshold $\mathcal{T} = 50\%$; using this value, PIEC outperforms point-based recognition. These results demonstrate that the use of oPIEC$^{\mathrm{bd}}$ leads to significant performance gains compared to oPIEC$^{\mathrm{b}}$. In all cases, oPIEC$^{\mathrm{bd}}$ outperforms oPIEC$^{\mathrm{b}}$, highlighting the importance of taking into consideration PMI duration statistics. Note that in the case of ‘meeting’ oPIEC$^{\mathrm{bd}}$ performs better under ‘strong noise’ than ‘smooth noise’, when Prob-EC provides the input stream (see Diagrams 8(c) and 8(d)). Some intervals which were PMIs under ‘smooth noise’ are not PMIs under ‘strong noise’ because their probabilities have dropped below the threshold. This exclusion of lower probability PMIs produces more reliable duration statistics for oPIEC$^{\mathrm{bd}}$.

In order to illustrate the performance of oPIEC$^{\mathrm{bd}}$, we analyse some typical snapshots of our experiments. Figure 9 displays two PMIs, I_1 and I_2, computed by PIEC for ‘meeting’ under ‘strong noise’, and the corresponding intervals computed by oPIEC$^{\mathrm{bd}}$ and oPIEC$^{\mathrm{b}}$. Table 7 displays the endpoints of PMIs I_1 and I_2, along with the support sets of oPIEC$^{\mathrm{b}}$ and oPIEC$^{\mathrm{bd}}$ at the time denoted by the ending point of the corresponding PMI. The support sets of oPIEC$^{\mathrm{b}}$ and oPIEC$^{\mathrm{bd}}$ may hold at most five elements, while the mean and the standard deviation of the duration values provided to oPIEC$^{\mathrm{bd}}$ are 411 and 22 time-points, respectively.

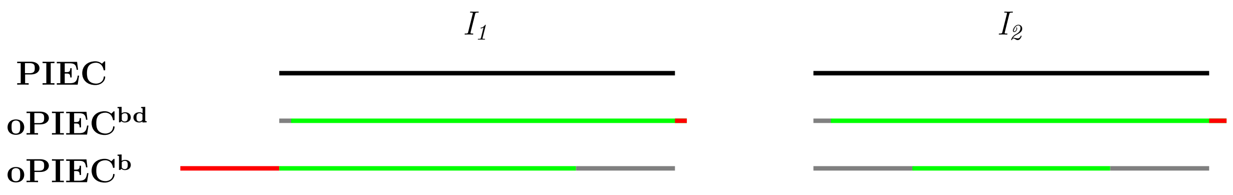

Figure 9: Indicative comparison of the intervals computed by PIEC (top) against those computed by oPIEC^{bd} (middle) and oPIEC^{b} (bottom). Green lines denote true positives, red lines denote false positives, while grey lines denote false negatives.

Table 7: Indicative interval computation comparison between PIEC, oPIEC^{bd} and oPIEC^{b}. The first column presents the PMIs computed by PIEC. The second and third columns present the support sets (lists of tuples in the form of $(t,\ prev\_prefix[t])$) of oPIEC^{bd} and oPIEC^{b}, respectively, at the time denoted by the ending point of the corresponding PMI.

Target PMI	oPIEC^{bd} support set	oPIEC^{b} support set
$I_1 = [17492, 18881]$	$[(17312, -7075), (17390, -7114), (17495, -7167), (17556, -7192), (17673, -7211)]$	$[(0, 0), (4858, -1096), (10391, -3628), (13025, -4936), (17133, -6986)]$
$I_2 = [21602, 22960]$	$[(21503, -8475), (21559, -8503), (21616, -8532), (21680, -8555), (21793, -8572)]$	$[(0, 0), (10391, -3628), (13025, -4936), (17133, -6986), (20715, -8081)]$

Consider oPIEC^{b} and oPIEC^{bd} attempting to compute PMI $I_1 = [17492, 18881]$ with the corresponding support sets displayed in Table 7. oPIEC^{b} computed the interval $[17133, 18522]$ starting from the time-point of the last support set element, i.e., $(17133, -6986)$. The time-point $t = 17492$, with $prev\_prefix[t] = -7164$, was not stored in the support set because its $score\_range$ was shorter than the $score\_range$ of all support set elements. In contrast, oPIEC^{bd} maintained in the support set the time-point 17495, and thus computes the interval $[17495, 18883]$, having only three false negatives and two false positives. In oPIEC^{bd}, support set elements which are 'outdated' with respect to duration statistics are eventually removed. Note that the intervals computed by oPIEC^{b} and oPIEC^{bd} are PMIs. PIEC chose I_1 over these intervals because I_1 has the highest 'credibility', i.e., among overlapping PMIs, PIEC chooses the one with the highest probability.

In the case of PMI $I_2 = [21602, 22960]$, all intervals starting from a time-point in the support set of oPIEC^{b} have a probability below the threshold. In such cases, the recognition of oPIEC^{b} follows point-based recognition, i.e., it returns the time periods during which instantaneous CE probability is greater or equal to the threshold, which resulted in the construction of a sub-interval of I_2, i.e., oPIEC^{b} could not compute a PMI. In contrast, oPIEC^{bd} maintained in its support set time-point 21616, which is the starting point of a PMI overlapping I_2.

Maritime Situational Awareness. Figure 10 presents the f1-scores of oPIEC^{bd} and oPIEC^{b}. oPIEC^{bd} performs at least as well as oPIEC^{b} in the case of 'rendez-vous' and clearly outperforms oPIEC^{b} in the case of 'tugging'. In the recognition of 'tugging', it is often the case that all intervals starting from a time-point in the support set of oPIEC^{b} and ending in the current data batch have a probability below the threshold. It such cases, the intervals computed by oPIEC^{b} are confined only to time-points with probability above the threshold, and thus oPIEC^{b} detects sub-intervals

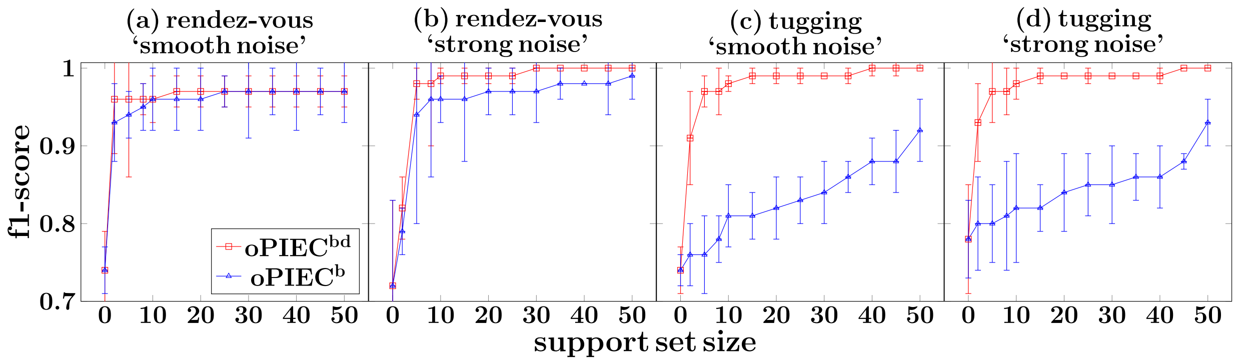

Figure 10: The predictive accuracy of oPIEC^{bd} and oPIEC^{b} on maritime situational awareness. The intervals computed by oPIEC^{bd} and oPIEC^{b} are compared to the PMIs computed by PIEC. The point-based system providing the input probability streams is Prob-EC.

Table 8: Run-times of oPIEC^{bd}, oPIEC^{b} and PIEC when computing the PMIs of 'rendez-vous' under 'strong noise'.

(a) Total run-times of oPIEC^{bd}, oPIEC^{b} and PIEC in seconds when processing data streams of increasing size.

total stream size (number of time-points)	**1K**	**2K**	**4K**	**8K**
PIEC	1.92 ± 0.32	7.53 ± 1.24	29.76 ± 4.9	134.63 ± 22
oPIEC^{b}	$\boldsymbol{0.09 \pm 0.02}$	$\boldsymbol{0.19 \pm 0.05}$	$\boldsymbol{0.38 \pm 0.1}$	$\boldsymbol{0.7 \pm 0.2}$
oPIEC^{bd}	$\boldsymbol{0.09 \pm 0.02}$	$\boldsymbol{0.19 \pm 0.05}$	0.39 ± 0.1	0.72 ± 0.23

(b) Total run-times of oPIEC^{bd} and oPIEC^{b} in seconds as the support set size increases.

support set size (number of elements)	**50**	**100**	**200**	**400**
oPIEC^{b}	$\boldsymbol{0.7 \pm 0.21}$	$\boldsymbol{1.27 \pm 0.5}$	$\boldsymbol{2.4 \pm 1.1}$	$\boldsymbol{4.79 \pm 2.41}$
oPIEC^{bd}	$\boldsymbol{0.7 \pm 0.23}$	$\boldsymbol{1.27 \pm 0.53}$	$\boldsymbol{2.4 \pm 1.1}$	$\boldsymbol{4.79 \pm 2.41}$

of PMIs, resulting in false negatives. In contrast, oPIEC^{bd} is more accurate than oPIEC^{b} because it promptly discards obsolete time-points from the support set. In the case of 'rendez-vous', the elements in the support set of oPIEC^{b} often coincide with those maintained after consulting the provided duration statistics. Thus, the performance of oPIEC^{b} is closer to that of oPIEC^{bd}.

The aim of the next set of experiments was to compare the run-times of oPIEC^{bd}, oPIEC^{b} and PIEC. We show the results of this comparison on 'rendez-vous' under 'strong noise'; the results for the other CEs in maritime situational awareness and human activity recognition are similar and thus not shown here. The support set size limit was set to 50 elements for both oPIEC^{bd} and oPIEC^{b}, as this value leads to near-perfect PMI computation (see Figure 10(b)). The data batch size was set to one time-point. Upon each new batch arrival, PIEC had to process every input probability value from the start of the stream until the latest data batch, in order to ensure correct PMI computation. Table 8(a) shows the performance of PIEC, oPIEC^{b} and oPIEC^{bd} for

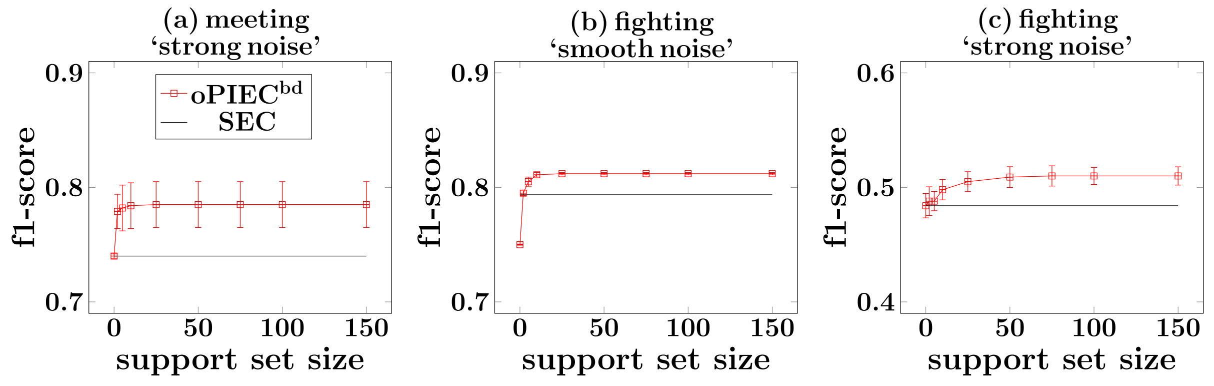

Figure 11: The predictive accuracy of oPIEC^{bd} and SEC against the ground truth provided by the CAVIAR team. The standard deviations in diagram (b) are small, and thus not visible. oPIEC^{bd} consumed the instantaneous CE probabilities computed by Prob-EC (which are the same as the instantaneous CE probabilities computed by SEC).

input data streams with size ranging from 1,000 to 8,000 time-points. oPIEC^{bd} and oPIEC^{b} have effectively the same performance; their run-times increase linearly with the input stream size, as the cost of processing an incoming data batch depends only on the size of the data batch and the size of the support set, which remain constant, and not on the entire history of the stream. Not surprisingly, the use of PIEC becomes prohibitively expensive as the stream progresses. These results are compatible with the complexity analyses of Sections 4.2 and 5.3.

In order to stress test oPIEC^{bd} and oPIEC^{b} further, we performed experiments for support set sizes ranging from 50 to 400 elements, while the stream size remains constant and equal to 8,000 time-points. Table 8(b) shows the results. Again, the performance of both systems is nearly identical. In other words, the additional operations of oPIEC^{bd} impose practically no overhead to the system (see Section 5.3).

6.3. Experimental Comparison

We compared oPIEC^{bd} with three state-of-the-art systems: the Simplified Event Calculus implemented in ProbLog (SEC), the Probabilistic Event Calculus (PEC) and OSLα. SEC is a probabilistic Event Calculus framework that supports uncertainty in both SDEs and CE definitions [39]. Similar to Prob-EC, SEC performs point-based recognition. We compared the predictive accuracy of oPIEC^{bd} and SEC using the CE annotation provided by the CAVIAR team as the ground truth. Following [38], we investigated the detection of the 'meeting' CE under 'strong noise' and the detection of the 'fighting' CE under 'smooth noise' and 'strong noise'. We used the threshold values leading to the best performance for each system. When recognising 'meeting' and 'fighting' under 'strong noise', we set $\mathcal{T} = 50\%$ for all systems. In the case of 'fighting' under 'smooth noise', the best performance for SEC is achieved with $\mathcal{T} = 90\%$, while oPIEC^{bd} reached its best performance with $\mathcal{T} = 50\%$. oPIEC^{bd} was evaluated using 5-fold cross-validation, while SEC does not require training. Figure 11 shows the f1-scores of oPIEC^{bd} and SEC, as the size of the support set used by oPIEC^{bd} increases from 0 to 150 elements. Our results show that the use of oPIEC^{bd} improves upon the point-based recognition of SEC, while requiring only a small subset of the data.

Figure 12 shows the reasoning times of oPIEC^{bd} and SEC when processing SDE streams of

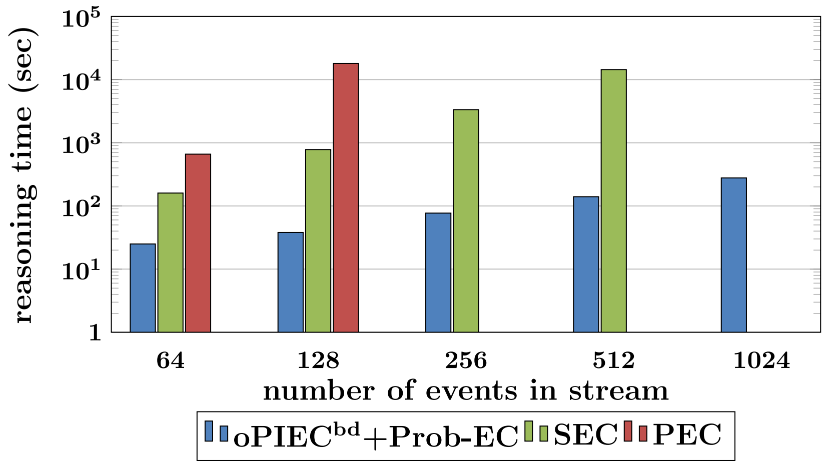

Figure 12: The reasoning times of oPIEC^{bd} and Prob-EC (oPIEC^{bd}+Prob-EC), SEC and PEC on the task of detecting the 'moving' CE under 'smooth noise' in logarithmic scale. PEC and SEC required more than 5 hours to process streams including at least 256 events and 1024 events, respectively, and thus we terminated their execution in these cases.

increasing size. The reasoning times presented for oPIEC^{bd} include the time that Prob-EC required to derive the instantaneous CE probabilities which are necessary for PMI computation. As a matter of fact, PMI computation takes up approx. 0.2% of the times shown in Figure 12. Our results show that oPIEC^{bd} scales much better than SEC. SEC computes the probability of a CE at each time-point from scratch, without taking into consideration the CE probabilities derived at previous time-points. In contrast, oPIEC^{bd} uses Prob-EC, which derives CE probabilities incrementally as time progress, avoiding redundant computations and maintaining in memory only the probabilities of CEs at the previous time-point.

A closely related system is PEC, i.e., a probabilistic Event Calculus framework for point-based recognition, which is implemented in answer set programming (ASP) for evaluation with a state-of-the-art ASP solver [21, 20]. The ASP implementation of PEC has been used in a decision-making system monitoring the attention levels of children engaging in rehabilitation exercises [22]. We tried to compare the predictive accuracy of oPIEC^{bd} against that of PEC, but could not conduct a meaningful comparison due to the high reasoning times of PEC. Figure 12 shows that the reasoning time of PEC increased exponentially as we doubled the number of SDEs in the input stream. Moreover, given a stream with at least 256 SDEs, we had to terminate the execution of PEC because it ran for more than 5 hours.

OSLα is a supervised learning system optimising the structure and weights of CE definitions in an Event Calculus expressed in Markov Logic [41, 40]. OSLα has proven effective in the task of learning human activity definitions, often outperforming manually constructed rules. We compared the predictive accuracy of oPIEC^{bd} and OSLα with respect to detecting the 'meeting' CE against the ground truth provided by the CAVIAR team. OSLα is not designed to handle probabilistic data, and thus operated on the original CAVIAR dataset. OSLα included a training phase during which it learned a definition of 'meeting' in the form of weighted Event Calculus rules, given a training set containing a subset of the annotation provided by the CAVIAR team (see [41]

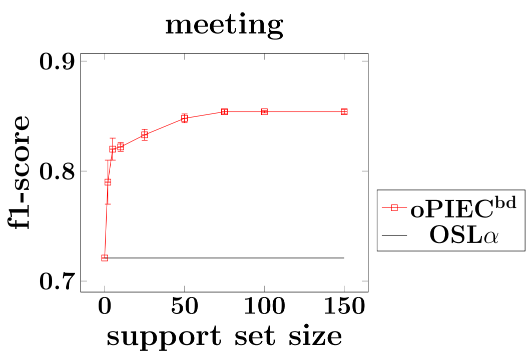

Figure 13: The predictive accuracy of oPIEC^{bd} and OSLα on the task of detecting the 'meeting' CE against the ground truth provided by the CAVIAR team.

for the setup of the training process). Afterwards, OSLα was evaluated in the remaining instances of 'meeting', using the learned pattern to compute the probability of 'meeting' at each time-point. oPIEC^{bd} processed the instantaneous probabilities derived by OSLα in order to compute the intervals of 'meeting' and was trained using 5-fold cross-validation. The threshold value for all systems was set to 70% because this choice maximises the predictive accuracy of each system.

Figure 13 presents our results, according to which the interval-based recognition of oPIEC^{bd} improves upon the recognition of OSLα. The instantaneous probabilities computed by OSLα include frequent fluctuations (similar to the one presented in Figure 2), which lead to false negatives. oPIEC^{bd} addressed this issue by computing PMIs and, as a result, outperformed the point-based recognition of OSLα.

7. Related Work

oPIEC^{bd} is a complex event recognition (CER) framework; CER refers to the identification of high-level, composite events (CE)s of interest based on streams of simple, derived events (SDE)s. These simple event streams may include various types of noise. For instance, in activity recognition, erroneous SDE indications are often the result of failures in the object tracker or object occlusion [48, 30], while maritime data streams may include empty fields and erroneous field values [13]. To address these issues, various frameworks have been proposed for probabilistic CER. See [3, 8] for two comprehensive surveys. Tables 9 and 10 compare probabilistic CER frameworks extending [3]. Table 9 compares the state-of-the-art in terms of the underlying formalism, temporal model, support for background knowledge, uncertainty model and learning capabilities. Table 10 reports the time complexity, empirical efficiency and predictive accuracy of these frameworks, as well as their code availability. In what follows, we review the probabilistic CER systems along the dimensions-columns of Tables 9 and 10.

7.1. Expressive Power

Probabilistic CER frameworks are commonly automata-based or logic-based [3]. Automata-based methods, which often extend the CER engine SASE+ [1], have been designed to handle uncertainty in event symbols and their time-stamps [60, 61]. The approach based on deterministic

Table 9: A comparison of probabilistic CER frameworks in terms of the underlying formalism, temporal model, support for background knowledge, uncertainty model and learning capabilities. B.K.: Background Knowledge, M.L.: Machine Learning.

Approach	Formalism	Temporal Model	B.K.	Uncertainty			M.L.
				Data	Pattern	Model	
oPIEC[bd]	Logic programming	Intervals, Explicit, Event Calculus.	✓	✓		Probabilistic logic programming	
SEC [39]	Logic programming	Points, Explicit, Event Calculus.	✓	✓	✓	Probabilistic logic programming	
DeepProbCEP [58]	Logic programming	Points, Explicit.	✓	✓		Neural probabilistic logic programming	✓
Apriceno et al. [6]	Logic programming	Intervals, Explicit.	✓	✓		Neural probabilistic logic programming	✓
CPT-L [54]	Disjunctive logic programming	Points, Implicit.	✓		✓	Probabilistic logic programming	✓
Qualitative time CP-Logic [26]	Disjunctive logic programming	Intervals, Explicit.	✓		✓	Dynamic Bayesian Networks	
WOLED [29]	Answer set programming	Points, Explicit, Event Calculus.	✓		✓	Derivation of answer set that maximises the sum of weights of satisfied rules	✓
PEC [21, 20, 19]	Action logic translated to answer set programming	Points, Implicit, Event Calculus.		✓	✓	Multiplication of independent probabilistic choices	
OnPad [2]	First-order logic	Intervals, Explicit.		✓		Probabilities assigned to predicates for object equality	
MLN-EC [51]	First-order logic	Points, Explicit, Event Calculus.	✓		✓	Markov Logic Networks	✓
MLN-Allen [42]	First-order logic	Intervals, Explicit, Allen's Algebra.	✓	✓	✓	Markov Logic Networks	
OSLα [41, 40]	First-order logic	Points, Explicit, Event Calculus.	✓		✓	Markov Logic Networks	✓
Apriceno et al. [7]	First-order logic	Intervals, Explicit.	✓	✓		Mixed integer linear programming	✓
PEL [14]	Event logic [49]	Intervals, Implicit, Allen's Algebra.	✓		✓	Weighted event logic formulas	
CEP2U [18]	TESLA [17]	Points, Explicit.		✓	✓	Bayesian Networks	
Partial-State MTL [35]	Metric temporal logic	Points, Implicit.		✓		Multiplication of state transitions probabilities	
ProbSTL [55]	Signal temporal logic	Points, Implicit.		✓		Bayesian filtering	
Neuroplex [59]	Finite state machines and logic programming	Points, Explicit.	✓		✓	Probabilistic logic programming approximated with a deep learning model	✓
SASE++ [60, 61]	Non-deterministic finite automata	Points, Implicit.		✓		Probability distribution on time attribute	
Sugiura and Ishikawa [52, 53]	Deterministic finite automata	Intervals, Implicit.		✓		Multiplication of probabilistic transition matrices	
				Data	Pattern	Model	
Approach	Formalism	Temporal Model	B.K.	Uncertainty			M.L.

Table 10: A comparison of probabilistic CER frameworks in terms of the time complexity, empirical efficiency, predictive accuracy and code availability. C.A.: Code Availability.

Approach	**Complexity**	**Efficiency**	**Accuracy**	**C.A.**
oPIEC^{bd}	$\mathcal{O}(m_b n_w)$, where m_b is the bounded memory size and n_w is the batch size.	Processes a stream of 8K probabilities in less than 1 second.	F1-score of at least 90% w.r.t. batch processing.	✓
SEC [39]	Not reported.	Not reported.	Not reported.	✓
DeepProbCEP [58]	Not reported.	Slower training and reasoning times than Neuroplex by more than an order of magnitude.	Outperforms state-of-the-art neural approaches when trained using sparse data.	✓
Apriceno et al. [6]	Not reported.	Not reported.	Improves upon fully neural benchmark.	
CPT-L [54]	Program transformed into a BDD in polynomial time. Probabilistic inference is linear to the size of the BDD. The size of the BDD may be exponential to the size of the program.	Inference over 200K nodes in the ground network in approx. 20 seconds.	Approx. 84% in the most challenging domain of their experiments.	
Qualitative time CP-Logic [26]	Not reported.	Not reported.	Not reported.	
WOLED [29]	Reduction to weighted MaxSat, which is NP-complete.	MAP inference over batches including approx. 30K ground atoms in less than 1 second.	Induced patterns for ‘moving’ and ‘meeting’ yield f1-scores of 82% and 89%, respectively.	✓
PEC [21, 20, 19]	Not reported.	Processes approx. 4K traces/sec.	Approximate solution computes probabilities within 95% confidence intervals w.r.t. exact probabilities by sampling 100 worlds.	✓
OnPad [2]	$\mathcal{O}(max(\|O\|, \|l\|, \|pl\|)^{\|af\|})$, where is $\|O\|$ is the number of objects in a video, $\|l\|$ is the number of ground boolean atoms, $\|pl\|$ is the number of probabilistic ground atoms and $\|af\|$ is the number of atoms in the formula of the activity of interest.	Processes approx. 4 video frames per second.	Divergence from human reviewer annotation ranges from 9% to 20%.	
Approach	**Complexity**	**Efficiency**	**Accuracy**	**C.A.**

Table 10: Continued.

Approach	**Complexity**	**Efficiency**	**Accuracy**	**C.A.**
MLN-EC [51]	Marginal inference with the approximate sampling algorithm MC-SAT. Approximate MAP inference through a transformation into Integer Linear Programming.	Marginal and MAP inference in less than 5 minutes.	Increased precision, slight decrease in recall w.r.t. deterministic solution.	✓
MLN-Allen [42]	Not reported.	Not reported.	F1-score $>$ 65%, even for small window sizes.	
OSLα [41, 40]	Not reported.	Training time for 'meeting' CE approx. 2 hours. Uses MLN-EC for inference.	Accuracy of learned CE definitions is similar and often better than the accuracy of manually curated rules.	✓
Apriceno et al. [7]	Not reported.	Not reported.	Improves upon fully neural benchmark.	
PEL [14]	Transforms the knowledge base in CNF in time linear to its size and applies an approximate stochastic local search approach for MAP inference.	Not reported.	On average, $>$75%.	
CEP2U [18]	Not reported.	$<$10ms/event after introducing data and pattern uncertainty.	$>$80% in all of their experiements.	
Partial-State MTL [35]	Not reported.	$<$3 minutes for formulas corresponding to automata with up to 15K states. Approx. 1.5 minutes with approximate reasoning.	Accuracy of formula probability approximation w.r.t. exact solution is controlled by a user-defined parameter.	✓
ProbSTL [55]	Proportional to the length of the formula defining the target complex event and the complexity of the domain-specific functions mapping stochastic signals to real values.	Not reported.	Probabilistic approach improves upon the recall metric, which is crucial for safety formulas where no instances should be missed.	
Neuroplex [59]	Not reported.	Not reported.	Outperforms corresponding neural solutions without human knowledge injected.	✓
SASE++ [60, 61]	Exponential to window size if Kleene closure is included in the language.	Throughput 300K–7M events/sec or 13.62 events/second/node w.r.t. 6 queries.	Not reported.	
Sugiura and Ishikawa [52, 53]	$\mathcal{O}(\sqrt{w}n^2+n^3)$, where w is the window size and n is the number of states in the deterministic finite automaton representation of a pattern.	Throughput of $>$1K events/sec for a deterministic finite automaton with $<$10 states.	Removing states with probability $<$0.0001 to improve efficiency introduces an error of approx. 7%.	
Approach	**Complexity**	**Efficiency**	**Accuracy**	**C.A.**

finite automata of Sugiura and Ishikawa [52, 53] employs adaptive sliding windows and optimisation techniques. Automata-based methods lack the explicit representation of time and the expressive power of logic-based frameworks, such as the Event Calculus, making the modelling of CE patterns with complex temporal constraints and background knowledge cumbersome, and often impossible. Logic-based approaches are typically based on (subsets of) first-order logic. Several probabilistic Event Calculus dialects, such as Prob-EC and SEC [39], model CE definitions through logic programming rules. Other approaches employ extensions of linear temporal logic [35, 55], action logics [21] or the 'event logic' [47, 49]. These approaches model state machines where time is specified implicitly as the index of a state in an execution sequence. Metric Temporal Logic (MTL) augments the expressive power of state machines by supporting temporal operators whose scope can be constrained by user-defined temporal intervals. de Leng et al. presented Partial-State MTL [35], an incremental reasoning algorithm for computing formula satisfaction under uncertainty in MTL. ProbSTL [55] employs an extension of MTL with dense-time semantics where logical statements represent continuous time signals. Because of their point-based, implicit time model, these approaches are designed for path checking, i.e., finding whether a possible state evaluation is a model for a given query, and do not support durative CE computation. Through the use of Prob-EC, oPIEC^{bd} is based on logic programming, and thus avoids the aforementioned representation issues.

There are additional point-based frameworks in the literature. Skarlatidis et al. [51] proposed MLN-EC, i.e., an Event Calculus expressed in Markov Logic and implemented using Markov Logic Networks (MLNs). CEP2U [18] extends the TESLA [17] event specification language with probabilistic modelling. CPT-Logic [54] is a temporal extension of Causal-Probabilistic Logic (CP-Logic) [57] using Markov processes. SEC and PEC [21] are probabilistic Event Calculus dialects based on probabilistic logic programming and an action logic, respectively. In all of these approaches, CE inference is performed at each individual time-point (e.g., video frame). It has been shown that point-based approaches are often insufficient for CER under uncertainty [8]. Noisy instantaneous CE probability fluctuations, and non-abrupt probability change affect the performance of point-based recognition. Therefore, it is preferrable to employ interval-based techniques for CER.

Towards this, Brendel et al. [14] integrated the interval-based Probabilistic Event Logic (PEL) into an activity recognition framework for detecting CEs from a set of noisy, durative SDEs. MLN-Allen [42] is an interval-based activity recognition framework that avoids the enumeration of all possible intervals of a CE. In [26], CP-logic was combined with Allen's interval relations [4] to perform interval-based CER under uncertainty. Contrary to oPIEC^{bd}, these frameworks cannot compute CE intervals with a single pass over the input stream. Complexity and performance are discussed in Section 7.3.

Some frameworks support CE patterns that rely on background knowledge. For example, in the maritime domain, it is often necessary to retrieve the type of a vessel and its dimensions, or the regulations governing vessel behaviour in a designated area. Table 9 shows that the automata-based frameworks, i.e., SASE++ and the work of Sugiura and Ishikawa, do not support background knowledge, while most logic-based frameworks do. oPIEC^{bd} is based on logic programming, and thus has inherent support for background knowledge.

A probabilistic CER framework may express data uncertainty in the input SDEs, and/or pattern uncertainty, i.e., uncertainty in the definitions of CEs. Moreover, a model is used to handle the uncertainty of SDE occurrences and/or CE definitions in reasoning. Table 9 compares the probabilistic CER frameworks along these dimensions. oPIEC^{bd} processes input SDEs using Prob-EC, which supports data uncertainty as SDEs are associated with probability values. Moreover, Prob-EC, like all probabilistic logic programming frameworks displayed in Table 9, is based on Sato's distribution semantics [46]. PEL models uncertainty by defining CEs as weighted event logic formulas [14]. OnPad supports probability annotations only on equality predicates and employs an ad-hoc algorithm to propagate their uncertainty to its output. Some state transition systems propagate uncertainty by multiplying the probabilities of all transitions in some sequence [52, 55], while other frameworks employ Bayesian Networks [18, 26]. PEC derives the instantaneous probability of a CE by computing all possible worlds in which the CE holds and adding their probabilities. As demonstrated in Section 6.3, this approach is not suitable for probabilistic CER.

7.2. Learning

Several probabilistic CER frameworks support learning; see the last column of Table 9. WOLED combines inductive logic programming techniques with answer set programming (ASP) in order to learn the structure and the weights of probabilistic Event Calculus rules [29]. For example, WOLED was able to learn a more accurate definition for 'meeting' than the related learner ILASP [33, 34], while being more efficient [29]. Our experimental evaluation included OSLα, a supervised framework for learning the structure of weighted Event Calculus rules in Markov Logic [41]. OSLα has been employed in a semi-supervised setting using an online supervision completion method adapted to first-order logic [40]. In this setting, OSLα was able to learn accurate definitions for the 'moving' and 'meeting' CEs, even in the presence of partially labelled data. Learning CE definitions is orthogonal to our work; the definitions used by oPIEC^{bd} may be learned or manually constructed.

Neuro-symbolic frameworks for CER typically employ a neural layer for deriving SDEs from multimedia data and a logic layer for defining patterns of CEs. Neuroplex [59] is a neuro-symbolic CER framework that translates logical rules expressing CE definitions in a 'neural reasoning layer', resulting in a fully differentiable neural architecture that may be trained end-to-end using CE annotations. DeepProbCEP [58] is a neuro-symbolic CER framework whose logic layer is based in DeepProbLog [37], incorporating an Event Calculus dialect inspired by Prob-EC. The use of DeepProbLog enables training with CE labels without translating CE definitions into a neural representation. The probabilistic Event Calculus implementation of DeepProbCEP, however, is a bottleneck for the training and reasoning efficiency of the system, suggesting the need for further optimisations. Moreover, DeepProbCEP and Neuroplex are point-based, and thus exhibit the drawbacks of point-based recognition [8].

Apriceno et al. [6] proposed a neuro-symbolic framework based on DeepProbLog which leverages background knowledge about the SDEs and the visual objects in CE patterns. In a more recent work [7], Apriceno et al. employed a mixed integer linear programming formulation instead of DeepProbLog. SDE durations were used as soft constraints in order to find the most probable sequence of SDEs. The logic layers of [6, 7] employ an Event Calculus dialect where SDEs and CEs are durative. Contrary to our work, this dialect is very minimal as it contains only a durative incarnation of the happensAt predicate. Moreover, CE recognition is limited to one CE per video clip, while the SDEs generated by the neural network are mutually exclusive, i.e., concurrent or overlapping SDEs are not considered.

7.3. Performance

Table 10 reports the time complexity, empirical efficiency and predictive accuracy of each system. Note that some frameworks do not provide a complexity analysis and/or empirical results. Futhermore, we identify the systems with available code. Regarding time complexity, OnPad computes the probability that a durative CE takes place, given a possibly incomplete video sequence, with a cost exponential to the length of the formula defining the CE. PEL resorts to approximate inference techniques to mitigate its exponential worst-case cost for the task of durative CE computation. Sugiura and Ishikawa translate the definition of the target CE into a deterministic finite automaton and use an algorithm with cost $\mathcal{O}(\sqrt{w}n^2 + n^3)$, where w is the window size and n is the number of states in the automaton. In our work, Prob-EC computes CE probabilities incrementally, maintaining in memory only the probabilities of CEs at the previous time step. Subsequently, oPIEC$^{\text{bd}}$ performs one pass over the derived instantaneous CE probabilities and computes PMIs with a cost of $\mathcal{O}(m_b n_w)$, where the bounded support set size m_b and the data batch size n_w are typically very small.

Table 10 describes the best reported empirical results of each system in terms of efficiency. OnPad processed approx. 4 video frames per second, indicating the need for further optimisations for supporting real-time processing. Sugiura and Ishikawa showed that their approach can process thousands of events per second, when the size of the automaton expressing the target CE pattern includes less than 10 states. Given a stream of 8K instantaneous probabilities, oPIEC$^{\text{bd}}$ was able to compute the PMIs of the 'rendez-vous' CE in less than 1 second (see Table 8(a)).

Table 10 also reports empirical results on the predictive accuracy of probabilistic CER frameworks. For OnPad, PEL and MLN-Allen, the accuracy of CE recognition was evaluated against the CE annotations provided by experts. For the approach of Sugiura and Ishikawa, which supports approximate reasoning, predictive accuracy was reported as a comparison against the CE recognition derived by the corresponding exact algorithm. We demonstrated that the recognition of oPIEC$^{\text{bd}}$ is comparable to that of PIEC with a very small support set (see Figures 8 and 10). Moreover, we showed that oPIEC$^{\text{bd}}$ outperforms point-based recognition, using ground truth offered by experts (see Figures 11 and 13).

The last column of Table 10 marks the systems with publicly available code. As mentioned earlier, oPIEC$^{\text{bd}}$ is open-source and our experiments are reproducible[1].

8. Summary and Further Research

We presented a formal computational framework for online, interval-based composite event recognition (CER) under uncertainty. Our framework consumes the output of a point-based CER system to compute the most likely maximal intervals during which a composite activity is said to take place. oPIEC employs a 'support set', a memory structure with the minimal set of time-points, to guarantee correct interval computation. To support streaming applications, we presented oPIEC$^{\text{bd}}$, an extension of oPIEC with a bounded support set, leveraging interval duration statistics to resolve memory conflicts. oPIEC^{bd} achieves comparable predictive accuracy to batch reasoning, avoiding the prohibitive cost of such reasoning. Moreover, oPIEC^{bd} stands out from the state-of-the-art because it addresses the issues of point-based recognition and implicit interval representation, supports expressive activity patterns, such as those including background knowledge, and computes composite activity intervals with a single pass over the input data. The remaining contributions of the paper include the theoretical analysis of oPIEC, proving its correctness and presenting its complexity, the complexity analysis of bounded oPIEC, and an extensive, reproducible empirical evaluation demonstrating the benefits of oPIEC^{bd}. For future work, we aim to integrate our system into a neuro-symbolic framework for adaptive, interval-based CER under uncertainty.

Acknowledgements

This work was supported by the ENEXA project, which has received funding from the European Union's Horizon 2020 research and innovation programme, under grant agreement No 101070305, and by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 6011).

References

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

[2] Albanese, M., Chellappa, R., Cuntoor, N., Moscato, V., Picariello, A., Subrahmanian, V.S., Udrea, O., 2010. PADS: A Probabilistic Activity Detection Framework for Video Data. IEEE Trans. Pattern Anal. Mach. Intell. 32, 2246–2261.

[3] Alevizos, E., Skarlatidis, A., Artikis, A., Paliouras, G., 2017. Probabilistic complex event recognition: A survey. Commun. ACM 50, 71:1–71:31.

[4] Allen, J.F., 1983. Maintaining knowledge about temporal intervals. Commun. ACM 26, 832–843.

[5] Allison, L., 2003. Longest biased interval and longest non-negative sum interval. Bioinform. 19, 1294–1295.

[6] Apriceno, G., Passerini, A., Serafini, L., 2021. A neuro-symbolic approach to structured event recognition, in: TIME, pp. 11:1–11:14.

[7] Apriceno, G., Passerini, A., Serafini, L., 2022. A neuro-symbolic approach for real-world event recognition from weak supervision, in: TIME, pp. 12:1–12:19.

[8] Artikis, A., Makris, E., Paliouras, G., 2021. A probabilistic interval-based event calculus for activity recognition. Ann. Math. Artif. Intell. 89, 29–52.

[9] Artikis, A., Sergot, M.J., Paliouras, G., 2010. A logic programming approach to activity recognition, in: EIMM, ACM. pp. 3–8.

[10] Artikis, A., Sergot, M.J., Paliouras, G., 2015. An event calculus for event recognition. IEEE Trans. Knowl. Data Eng. 27, 895–908.

[11] Artikis, A., Skarlatidis, A., Portet, F., Paliouras, G., 2012. Logic-based event recognition. Knowl. Eng. Rev. 27, 469–506.

[12] Bellodi, E., Alberti, M., Riguzzi, F., Zese, R., 2020. Map inference for probabilistic logic programming. Theory Pract. Log. Program. 20, 641–655.

[13] Bereta, K., Chatzikokolakis, K., Zissis, D., 2021. Maritime reporting systems, in: Guide to Maritime Informatics, pp. 3–30.

[14] Brendel, W., Fern, A., Todorovic, S., 2011. Probabilistic event logic for interval-based event recognition, in: CVPR, pp. 3329–3336.

[15] Cervesato, I., Franceschet, M., Montanari, A., 2000. A guided tour through some extensions of the event calculus. Comput. Intell. 16, 307–347.

[16] Clark, K., 1978. Negation as failure, in: Logic and Databases, pp. 293–322.

[17] Cugola, G., Margara, A., 2010. TESLA: a formally defined event specification language, in: DEBS, ACM. pp. 50–61.

[18] Cugola, G., Margara, A., Matteucci, M., Tamburrelli, G., 2015. Introducing uncertainty in complex event processing: model, implementation, and validation. Computing 97, 103–144.

[19] D'Asaro, F., Bikakis, A., Dickens, L., Miller, R., 2020. Probabilistic reasoning about epistemic action narratives. Artif. Intell. 287, 103352.

[20] D'Asaro, F.A., 2019. Probabilistic epistemic reasoning about actions. Ph.D. thesis. University College London, UK.

[21] D'Asaro, F.A., Bikakis, A., Dickens, L., Miller, R., 2017. Foundations for a probabilistic event calculus, in: LPNMR, pp. 57–63.

[22] D'Asaro, F.A., Raggioli, L., Malek, S., Grazioso, M., Rossi, S., 2022. An application of a runtime epistemic probabilistic event calculus to decision-making in e-health systems. Theory and Practice of Logic Programming , 1–24.

[23] Fierens, D., Van Den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., De Raedt, L., 2014. Inference and learning in probabilistic logic programs using weighted boolean formulas. Theory Pract. Log. Program. 15, 358–401.

[24] Fikioris, G., Patroumpas, K., Artikis, A., Paliouras, G., Pitsikalis, M., 2020. Fine-tuned compressed representations of vessel trajectories, in: CIKM, pp. 2429–2436.

[25] Giatrakos, N., Alevizos, E., Artikis, A., Deligiannakis, A., Garofalakis, M., 2020. Complex event recognition in the big data era: a survey. VLDB J 20, 313–352.

[26] van der Heijden, M., Lucas, P.J.F., 2013. Describing disease processes using a probabilistic logic of qualitative time. Artif. Intell. Medicine 59, 143–155.

[27] Hongeng, S., Nevatia, R., Brémond, F., 2004. Video-based event recognition: activity representation and probabilistic recognition methods. Comput. Vis. Image Underst. 96, 129–162.

[28] Katzouris, N., Artikis, A., 2020. WOLED: A tool for online learning weighted answer set rules for temporal reasoning under uncertainty, in: KR, pp. 790–799.

[29] Katzouris, N., Paliouras, G., Artikis, A., 2023. Online learning probabilistic event calculus theories in answer set programming. Theory Pract. Log. Program. 23, 362–386.

[30] Khan, A., Serafini, L., Bozzato, L., Lazzerini, B., 2019. Event detection from video using answer set programing, in: CILC, pp. 48–58.

[31] Kimmig, A., Demoen, B., Raedt, L.D., Costa, V.S., Rocha, R., 2011. On the implementation of the probabilistic logic programming language ProbLog. Theory Pract. Log. Program. 11, 235–262.

[32] Kowalski, R.A., Sergot, M.J., 1986. A logic-based calculus of events. New Gener. Comput. 4, 67–95.

[33] Law, M., 2018. Inductive learning of answer set programs. Ph.D. thesis. Imperial College London, UK.

[34] Law, M., 2023. Conflict-driven inductive logic programming. Theory Pract. Log. Program. 23, 387–414.

[35] de Leng, D., Heintz, F., 2019. Approximate stream reasoning with metric temporal logic under uncertainty, in: AAAI, pp. 2760–2767.

[36] List, T., Bins, J., Vazquez, J., Fisher, R.B., 2005. Performance evaluating the evaluator, in: VS-PETS, pp. 129–136.

[37] Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., De Raedt, L., 2021. Neural probabilistic logic programming in deepproblog. Artif. Intell. 298, 103504.

[38] Mantenoglou, P., Artikis, A., Paliouras, G., 2020. Online probabilistic interval-based event calculus, in: ECAI, pp. 2624–2631.

[39] McAreavey, K., Bauters, K., Liu, W., Hong, J., 2017. The event calculus in probabilistic logic programming with annotated disjunctions, in: AAMAS, pp. 105–113.

[40] Michelioudakis, E., Artikis, A., Paliouras, G., 2019. Semi-supervised online structure learning for composite event recognition. Mach. Learn. 108, 1085–1110.

[41] Michelioudakis, E., Skarlatidis, A., Paliouras, G., Artikis, A., 2016. OSLα: Online structure learning using

background knowledge axiomatization, in: ECML-PKDD, pp. 232–247.

[42] Morariu, V.I., Davis, L.S., 2011. Multi-agent event recognition in structured scenarios, in: CVPR, pp. 3289–3296.

[43] Mueller, E.T., 2009. Automating commonsense reasoning using the event calculus. Commun. ACM 52, 113–117.

[44] Pitsikalis, M., Artikis, A., Dreo, R., Ray, C., Camossi, E., Jousselme, A., 2019. Composite event recognition for maritime monitoring, in: DEBS, pp. 163–174.

[45] Santipantakis, G.M., Vlachou, A., Doulkeridis, C., Artikis, A., Kontopoulos, I., Vouros, G.A., 2018. A stream reasoning system for maritime monitoring, in: TIME, pp. 20:1–20:17.

[46] Sato, T., 1995. A statistical learning method for logic programs with distribution semantics, in: ICLP, pp. 715–729.

[47] Selman, J., Amer, M.R., Fern, A., Todorovic, S., 2011. PEL-CNF: Probabilistic event logic conjunctive normal form for video interpretation, in: ICCV Workshop, pp. 680–687.

[48] Singh, T., Vishwakarma, D.K., 2019. Video benchmarks of human action datasets: a review. Artif. Intell. Rev. 52, 1107–1154.

[49] Siskind, J.M., 2001. Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic. J. Artif. Intell. Res. 15, 31–90.

[50] Skarlatidis, A., Artikis, A., Filipou, J., Paliouras, G., 2015a. A probabilistic logic programming event calculus. Theory Pract. Log. Program. 15, 213–245.

[51] Skarlatidis, A., Paliouras, G., Artikis, A., Vouros, G.A., 2015b. Probabilistic event calculus for event recognition. ACM Trans. Comput. Log. 16.

[52] Sugiura, K., Ishikawa, Y., 2019. Regular expression pattern matching with sliding windows over probabilistic event streams, in: IEEE BigComp, pp. 1–8.

[53] Sugiura, K., Ishikawa, Y., 2020. Multiple regular expression pattern monitoring over probabilistic event streams. IEICE Trans. Inf. Syst. 103-D, 982–991.

[54] Thon, I., Landwehr, N., De Raedt, L., 2011. Stochastic relational processes: Efficient inference and applications. Mach. Learn. 82, 239–272.

[55] Tiger, M., Heintz, F., 2020. Incremental reasoning in probabilistic signal temporal logic. Int. J. Approx. Reason. 119, 325 – 352.

[56] Tsilionis, E., Artikis, A., Paliouras, G., 2022. Incremental event calculus for run-time reasoning. J. Artif. Intell. Res. 73, 967–1023.

[57] Vennekens, J., Denecker, M., Bruynooghe, M., 2009. Cp-logic: A language of causal probabilistic events and its relation to logic programming. Theory Pract. Log. Program. 9, 245–308.

[58] Vilamala, M.R., Xing, T., Taylor, H., Garcia, L., Srivastava, M., Kaplan, L.M., Preece, A.D., Kimmig, A., Cerutti, F., 2023. Deepprobcep: A neuro-symbolic approach for complex event processing in adversarial settings. Expert Syst. Appl. 215, 119376.

[59] Xing, T., Garcia, L., Vilamala, M.R., Cerutti, F., Kaplan, L., Preece, A., Srivastava, M., 2020. Neuroplex: Learning to detect complex events in sensor networks through knowledge injection, in: SenSys, p. 489–502.

[60] Zhang, H., Diao, Y., Immerman, N., 2013. Recognizing patterns in streams with imprecise timestamps. Inf. Syst. 38, 1187–1211.

[61] Zhang, H., Diao, Y., Immerman, N., 2014. On complexity and optimization of expensive queries in complex event processing, in: SIGMOD, ACM. pp. 217–228.

[62] Zocholl, M., Iphar, C., Pitsikalis, M., Jousselme, A., Artikis, A., Ray, C., 2019. Evaluation of maritime event detection against missing data, in: QUATIC, pp. 275–288.

Appendix A. Complex Event Patterns

Appendix A.1. Human Activity Recognition

'moving' CE Definition:

initiatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($walking(P_1)$, T),
  $moveDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) =$ true, T),
  happensAt($walking(P_2)$, T),
  not happensAt($disappear(P_1)$, T),
  not happensAt($disappear(P_2)$, T),
  holdsAt($similarOrientation(P_1, P_2) =$ true, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($walking(P_1)$, T),
  $moveDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) =$ false, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($walking(P_2)$, T),
  $moveDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) =$ false, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($active(P_1)$, T),
  happensAt($active(P_2)$, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($active(P_1)$, T),
  happensAt($inactive(P_2)$, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($running(P_1)$, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($running(P_2)$, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($abrupt(P_1)$, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($abrupt(P_2)$, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($disappear(P_1)$, T).

terminatedAt($moving(P_1, P_2) =$ true, T) $\leftarrow$
  happensAt($disappear(P_2)$, T).

‘meeting’ CE Definition:

initiatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($active(P_1)$, T),
  happensAt($person(P_2)$, T),
  $interactDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) = $ true, T),
  not happensAt($abrupt(P_2)$, T),
  not happensAt($running(P_2)$, T),
  not happensAt($disappear(P_1)$, T),
  not happensAt($disappear(P_2)$, T).

initiatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($inactive(P_1)$, T),
  happensAt($person(P_1)$, T),
  happensAt($person(P_2)$, T),
  $interactDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) = $ true, T),
  not happensAt($running(P_2)$, T),
  not happensAt($active(P_2)$, T),
  not happensAt($abrupt(P_2)$, T),
  not happensAt($disappear(P_1)$, T),
  not happensAt($disappear(P_2)$, T).

terminatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($walking(P_1)$, T),
  $meetDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) = $ false, T).

terminatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($walking(P_2)$, T),
  $meetDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) = $ false, T).

terminatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($running(P_1)$, T).

terminatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($running(P_2)$, T).

terminatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($abrupt(P_1)$, T).

terminatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($abrupt(P_2)$, T).

terminatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($disappear(P_1)$, T).

terminatedAt($meeting(P_1, P_2) = $ true, T) ←
  happensAt($disappear(P_2)$, T).

terminatedAt($moving(P_1, P_2) = $ true, T) ←
  happensAt($active(P_1)$, T),
  happensAt($active(P_2)$, T).

‘fighting’ CE Definition:

initiatedAt($fighting(P_1, P_2) = $ true, T) ←
  happensAt($abrupt(P_1)$, T),
  holdsAt($person(P_2) = $ true, T),
  $fightDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) = $ true, T),
  not happensAt($inactive(P_2)$, T),
  not happensAt($disappear(P_1)$, T),
  not happensAt($disappear(P_2)$, T).

terminatedAt($fighting(P_1, P_2) = $ true, T) ←
  happensAt($walking(P_1)$, T),
  $fightDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) = $ false, T).

terminatedAt($fighting(P_1, P_2) = $ true, T) ←
  happensAt($walking(P_2)$, T),
  $fightDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) = $ false, T).

terminatedAt($fighting(P_1, P_2) = $ true, T) ←
  happensAt($running(P_1)$, T),
  $fightDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) = $ false, T).

terminatedAt($fighting(P_1, P_2) = $ true, T) ←
  happensAt($running(P_2)$, T),
  $fightDist(Dist)$,
  holdsAt($close(P_1, P_2, Dist) = $ false, T).

terminatedAt($fighting(P_1, P_2) = $ true, T) ←
  happensAt($disappear(P_1)$, T).

terminatedAt($fighting(P_1, P_2) = $ true, T) ←
  happensAt($disappear(P_2)$, T).

Appendix A.2. Maritime Situational Awareness

‘rendez-vous’ CE Definition:

initiatedAt($rendezVous(Vessel_1, Vessel_2) =$ true, T) $\leftarrow$
  happensAt($proximityStart(Vessel_1, Vessel_2)$, T),
  not $oneIsTug(Vessel_1, Vessel_2)$,
  not $oneIsPilot(Vessel_1, Vessel_2)$,
  (holdsAt($lowSpeed(Vessel_1) =$ true, T); holdsAt($stopped(Vessel_1) = farFromPorts$, T)),
  (holdsAt($lowSpeed(Vessel_2) =$ true, T); holdsAt($stopped(Vessel_2) = farFromPorts$, T)),
  not holdsAt($withinArea(Vessel_1, nearPorts) =$ true, T),
  not holdsAt($withinArea(Vessel_2, nearPorts) =$ true, T),
  not holdsAt($withinArea(Vessel_1, nearCoast) =$ true, T),
  not holdsAt($withinArea(Vessel_2, nearCoast) =$ true, T).

terminatedAt($rendezVous(Vessel_1, Vessel_2) =$ true, T) $\leftarrow$
  happensAt($proximityEnd(Vessel_1, Vessel_2)$, T),
  not $oneIsTug(Vessel_1, Vessel_2)$,
  not $oneIsPilot(Vessel_1, Vessel_2)$,
  (holdsAt($lowSpeed(Vessel_1) =$ true, T); holdsAt($stopped(Vessel_1) = farFromPorts$, T)),
  (holdsAt($lowSpeed(Vessel_2) =$ true, T); holdsAt($stopped(Vessel_2) = farFromPorts$, T)),
  not holdsAt($withinArea(Vessel_1, nearPorts) =$ true, T),
  not holdsAt($withinArea(Vessel_2, nearPorts) =$ true, T),
  not holdsAt($withinArea(Vessel_1, nearCoast) =$ true, T),
  not holdsAt($withinArea(Vessel_2, nearCoast) =$ true, T).

‘tugging’ CE Definition:

initiatedAt($tugging(Vessel_1, Vessel_2) =$ true, T) $\leftarrow$
  happensAt($proximityStart(Vessel_1, Vessel_2)$, T),
  $oneIsTug(Vessel_1, Vessel_2)$,
  not $oneIsPilot(Vessel_1, Vessel_2)$,
  holdsAt($tuggingSpeed(Vessel_1) =$ true, T),
  holdsAt($tuggingSpeed(Vessel_2) =$ true, T).

terminatedAt($tugging(Vessel_1, Vessel_2) =$ true, T) $\leftarrow$
  happensAt($proximityEnd(Vessel_1, Vessel_2)$, T),
  $oneIsTug(Vessel_1, Vessel_2)$,
  not $oneIsPilot(Vessel_1, Vessel_2)$,
  holdsAt($tuggingSpeed(Vessel_1) =$ true, T),
  holdsAt($tuggingSpeed(Vessel_2) =$ true, T).