Interval-based Queries over Lossy IoT Event Streams

NIMROD BUSANY, Tel Aviv University, Israel
HAN VAN DER AA, Humboldt-Universität zu Berlin, Germany
ARIK SENDEROVICH, University of Toronto, Canada
AVIGDOR GAL, Technion, Israel
MATTHIAS WEIDLICH, Humboldt-Universität zu Berlin, Germany

Recognising patterns that correlate multiple events over time becomes increasingly important in applications that exploit the Internet of Things, reaching from urban transportation, through surveillance monitoring to business workflows. In many real-world scenarios, however, timestamps of events may be erroneously recorded and events may be dropped from a stream due to network failures or load shedding policies. In this work, we present SimpMatch, a novel simplex-based algorithm for probabilistic evaluation of event queries using constraints over event orderings in a stream. Our approach avoids learning probability distributions for time-points or occurrence intervals. Instead, we employ the abstraction of segmented intervals and compute the probability of a sequence of such segments using the notion of order statistics. The algorithm runs in linear time to the number of lost events, and shows high accuracy, yielding exact results if event generation is based on a Poisson process and providing a good approximation otherwise. We demonstrate empirically that SimpMatch enables efficient and effective reasoning over event streams, outperforming state-of-the-art methods for probabilistic evaluation of event queries by up to two orders of magnitude.

CCS Concepts: • **Applied computing** → *Event-driven architectures.*

Additional Key Words and Phrases: Stream Queries, Event Data, Interval Semantics, Probabilistic Reasoning

ACM Reference Format:
Nimrod Busany, Han van der Aa, Arik Senderovich, Avigdor Gal, and Matthias Weidlich. 2020. Interval-based Queries over Lossy IoT Event Streams. *ACM Trans. Data Sci.* 00, 0, Article 00 (2020), 27 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

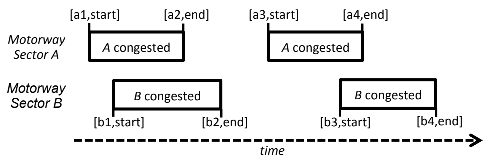
With the increased adoption of the Internet of Things (IoT) paradigm and cyber-physical systems, the importance of event streams in contemporary applications ranging from urban transportation, through surveillance monitoring to business workflows has been growing rapidly. Systems for Complex Event Processing (CEP) provide the backbone for applications in these areas by filtering, correlating, aggregating, and transforming events to derive actionable insights. For example, applications for intelligent urban transport rely on movement events for vehicles, cf., the Dublin Smart City initiative [2] or the Linear Road Benchmark [11] to detect complex situations, such as the build-up of traffic congestion as well as tardiness of public transport [13, 24].

Authors' addresses: Nimrod Busany, nimrodbusany@post.tau.ac.il, Tel Aviv University, Tel Aviv, Israel; Han van der Aa, han.van.der.aa@hu-berlin.de, Humboldt-Universität zu Berlin, Berlin, Germany; Arik Senderovich, sariks@mie.utoronto.ca, University of Toronto, Toronto, Canada; Avigdor Gal, avigal@technion.ac.il, Technion, Haifa, Israel; Matthias Weidlich, matthias.weidlich@hu-berlin.de, Humboldt-Universität zu Berlin, Berlin, Germany.

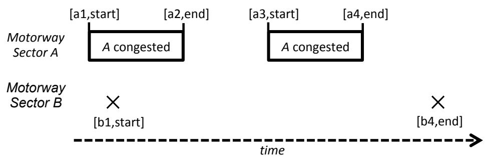
2577-3224/2020/0-ART00 $15.00
https://doi.org/10.1145/1122445.1122456

Interval-based queries. Interval-based queries identify patterns involving the correlation of events that have a duration over time. Consider, for example, a system for traffic monitoring that exploits sensor data, either emitted directly by transponders in vehicles (using On-Board-Units as deployed for toll collection, *e.g.*, in Germany) or captured by street sensors. Then, the input for the system are vehicle movement events indicating time, location, and speed. These events give rise to situations such as `congested` motorway sectors, defined by the number of tracked cars and their average speed. Such a scenario, with two motorway sectors, is illustrated in Fig. 1(a). The figure depicts two periods of congestions per sector, each with clearly defined start and end points.

An interval-based query may identify *severe* traffic congestions in a city centre by testing whether consecutive (or nearby) motorway sectors are `congested`. This occurs when the respective intervals of congestion *overlap*. For instance, given the sectors A and B in Fig. 1(a), an interval-based query would detect two occurrences of severe traffic congestions, since the `congested` intervals of the two sectors overlap twice.



(a) Without lost events



(b) With lost events in Sector B

Fig. 1. Congested intervals in a road traffic example.

Lost events. The events used to reason about interval-based queries are susceptible to various errors [21, 51], stemming from corrupted sensing mechanism at the source, loss of data over the network, load shedding policies, temporary lack of connections, *etc.* Hence, when processing interval-based queries, appropriate mechanisms for handling lost events should be in place to ensure query correctness. Consider again the intervals in Fig. 1. It may happen that the events denoting the end of the first interval and the beginning of the second interval for the congestion of sector B are lost, as shown in Fig. 1(b). Then, the identifiers of the received events may clearly indicate the number of events that have been emitted (four events), as well as which of them got lost (two events between $b1$ and $b4$). However, depending on the timestamps of these lost events and, hence, their order with the events received for congestion of sector A, different scenarios need to be considered when detecting an overlap of intervals. For instance, if event $b3$ is suspected to have a timestamp larger than the one of event $a4$, the second congestion intervals would no longer overlap and, thus, not be reported as results to a respective query.

Applications. While we considered an example from urban transportation for illustration, interval-based queries are applicable in diverse use-cases. For instance, they are important for surveillance applications, where movement events stem from automatic tagging of a video stream, cf., the CAVIAR project [1]. Here, interval-based queries relate to situations such as people walking together or package picking, see [12, 33, 39] for example queries. In the same vein, conformance analysis of business process execution [45] is grounded in a comparison of activity execution intervals against a process model to detect violations of activity order or resource utilisation. An example would be a claim handling business process, where recurring overlapping work on a particular claim by different roles can indicate non-conforming operations.

Challenges. The evaluation of interval-based queries over lossy event streams induces challenges along three dimensions, namely, C1) a *lack of abstractions for queries*, C2) dependency on *prior knowledge* on lost events, and C3) the *computational complexity.*

C1) Lack of abstractions for queries. Many applications require the detection of recurring behaviour in an event stream. In the above example, a few overlaps of congested intervals of two motorway sectors are likely to be observed by coincidence. A large number of such overlaps, in turn, may suggest a situation that should be detected. Most of the existing engines for the detection of sequence patterns, such as Cayuga [18], SASE+ [5], CEDR [14], and ZStream [29] can support such queries by tracing them back to point-based semantics [8] and using aggregations for the matches identified by atomic patterns (*i.e.*, single overlaps of intervals). Nevertheless, such an approach comes with a severe downside: Any dependencies between the atomic patterns are not part of the event processing model and, thus, cannot be exploited for query optimization.

C2) Dependency on prior knowledge. Probabilistic reasoning over events is often driven by prior knowledge. That is, a probability distribution is assigned to the occurrence time of an event either explicitly [39] or by adapting the model of probabilistic databases to streams [35], or to the occurrence interval of an event [51]. Learning such distributions induces overhead and postulates that the event stream shows regularity that is sufficient to build a model. In cold start-up scenarios or applications where event generation does not follow well-defined patterns (*e.g.*, unexpected congestions), learning may not be possible.

C3) Computational complexity. Most existing work on probabilistic reasoning over event streams assumes a discrete model of time, *e.g.*, [39, 40, 51]. Then, semantics of a query are defined based on an enumeration of all possible worlds in terms of possible timestamps of lost events. This results in a search space that is exponential in the size of the temporal distribution, which is intractable for real-world applications. Even though optimised reasoning techniques avoid the actual enumeration of all possible worlds, cf., [51], the complexity of probabilistic event pattern matching in this model inherently depends on the granularity of the discrete time domain.

Contributions. To address the outlined challenges, we propose a novel approach to answer interval-based queries in a probabilistic manner. We cater for the query abstraction challenge (C1) by presenting a model of *segmented intervals* that captures recurrent interval events such as the `congested` event mentioned above. Then, we show how segmented intervals provide the basis for solid probability assessment of the timestamps of lost events in the absence of prior knowledge (C2), except for the number of lost events. We also provide a method for answering, in probabilistic terms, interval-based queries using order statistics [36]. As opposed to other works that answer queries over interval-based events, we use a continuous temporal model. As such, our solution is independent of the granularity of a discrete time domain and only depends on the number of lost events, which is typically much smaller than the number of possible discrete time points (C3).

Specifically, our contributions, as well as the remainder of this paper, are summarised as follows:

- *A model for segmented interval events* (Section 2) in which time-point events are linked via segments, which in turn are connected via intervals. This model allows for the definition of interval-based queries that identify patterns while maintaining information regarding event correlations that stem from an interval hierarchy. To this end, we lift Allen's algebra to segmented interval events. We then define the problem of computing a probabilistic answer to interval-based queries specified in this model in the presence of lost events, i.e., events for which the timestamps are not known.
- *An analytical reasoning approach* (Section 3) is provided by SimpMatch, an algorithm to answer interval-based queries using probabilistic pattern matching over segmented interval events. Such queries involve analysing an exponential space of possible worlds, each representing a

specific order of time-point events that is matched separately by existing methods. However, the proposed algorithm avoids construction of all possible worlds and, therefore, improves over methods that solely match sequences of events. For a specific possible world, SimpMatch computes the probability of a pattern match in linear time to the number of lost events. We achieve that by grounding our approach in the order statistics over the timestamps of lost events. We show that this yields exact results when event generation follows a Poisson process, and provides an accurate approximation otherwise.

- *A comprehensive empirical analysis* (Section 4) evaluates our method using synthetic and benchmark datasets. The results indicate that our method scales well, achieving a throughput of up-to 2,000,000 events per second. If event generation follows a Poisson process, accuracy of pattern detection based on the probabilistic answer is approximately 90%, if 10% of the events are lost, gracefully reducing for higher levels of uncertainty.
  We also compare our approach to the reasoning technique of SASE+ [5, 51], a state-of-the-art engine for probabilistic matching of event patterns. The results show that our method yields higher detection accuracy for all levels of uncertainty and improves throughput by up to two orders of magnitude for highly uncertain event streams.
- *A demonstration of a Smart City application* (Section 5) that highlights the benefits of the proposed model and algorithm for a real-world case in the city of Dublin. Based on GPS data emitted by buses, we show how to effectively answer queries on congested bus stops.

We close the paper with a discussion of related work (Section 6) and conclusions (Section 7).

2 MODEL AND PROBLEM STATEMENT

We first introduce an interval-based event model (Section 2.1) and queries over intervals (Section 2.2). We then turn to the uncertainty induced by lost events (Section 2.3) and define the problem of probabilistic detection of interval relations (Section 2.4).

2.1 Interval Model

A data stream contains a sequence of *events*, each referred to as e. Events of similar structure and meaning belong to the same event type, denoted by $e.type$. Often, events are defined with a *point-based semantics* (also known as *point events*), such that an event e has a timestamp t, denoted by $e.t$, defining when the event has occurred.

Continuous interval events. In this work, we are particularly interested in events with *interval-based semantics.* Following earlier works, *e.g.*, [26, 44], we define interval-based semantics of an event using a sequence of events that adhere to point-based semantics. Therefore, a *continuous interval event* is a pair of point events $\varepsilon = \langle e_1, e_2 \rangle$. Here, e_1 denotes the start of an interval and e_2 its end. Therefore, ε has a start time of $e_1.t$, an end time of $e_2.t$, as well as a duration of $e_2.t - e_1.t$.

Segmented interval events. While continuous interval events enable the modelling of durative phenomena, a more expressive model is needed if several such intervals are semantically related. In that case, only the joint occurrence of several continuous intervals is considered to be an event from an application point of view. For instance, in the transportation example in Fig. 1, the event of interest is a motorway being `congested`, which is materialised as a sequence of `congested` intervals. Based on this abstraction, it becomes easier to model complex situations such as recurring congestion that spans over multiple motorway sectors.

Against this background, we extend the definition of interval-based semantics to non-continuous, *i.e.,* segmented, A *segmented interval event* $\varepsilon = \langle \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n \rangle$ consists of n interval events with each being a *segment* of event ε. It is worth noting that this definition is recursive in nature, since any segment may be, by itself, constructed from segments. However, interval-based queries need to be expressed for a specific level of detail, which means that event data needs to be abstracted to a single, suitable level of recursion. In what follows, we shall therefore use a single level of recursion only, where for all $1 \leq i \leq n$, ε_i is a continuous interval event.

We now formally define a valid segmented interval event, to be used in the rest of this work. Given a segmented interval event $\varepsilon = \langle \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n \rangle$, we denote by e^i_j the j-th point event ($j = 1, 2$) of segment i.

Definition 1 (Valid segmented interval event). *A valid segmented interval event $\varepsilon = \langle \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n \rangle$ is a segmented interval event that satisfies:*

- 1-level: $\forall\ 1 \leq i \leq n$, *ε_i is a continuous interval event.*
- Start event: $e^1_1.role = start$.
- End event: $e^n_2.role = end$.
- Suspend events: $\forall\ 1 \leq i < n$, $e^i_2.role = suspend$.
- Resume events: $\forall\ 1 < i \leq n$, $e^i_1.role = resume$.
- Strict non-overlap: $\forall\ 1 \leq i < n$, $e^i_2.t < e^{i+1}_1.t$.

A valid segmented interval event is not merely a set of interval events. Rather, Def. 1 highlights the semantic relation between the intervals: A first point event, e^1_1, signals the start of the segmented interval event. Then, multiple pairs of suspend and resume events indicate that the interval event has been interrupted and recontinued. Finally, event e^n_2 signals that the interval event has ended. Notation-wise, $e_<$ and $e_>$ denote start and end events, and $e_\downarrow$ and $e_\uparrow$ suspend and resume events, respectively.

Following Def. 1, a segmented interval event $\varepsilon = \langle \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n \rangle$ can be represented as a sequence of fully-ordered point events. Given ε, its *flat representation* is a sequence $flat(\varepsilon) = \langle e^1_1, e^1_2, e^2_1, e^2_2, \ldots, e^n_1, e^n_2 \rangle$, where $e \in \varepsilon$ denotes the presence of an event e in $flat(\varepsilon)$. Also, we shall use hereafter a single-index notation for events in $flat(\varepsilon)$, as follows: $flat(\varepsilon) = \langle e_1, e_2, \ldots, e_p \rangle$.

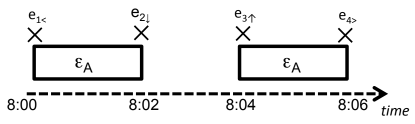


Fig. 2. Valid segmented interval event.

Example 1 (Valid segmented interval event). *Fig. 2 illustrates a valid segmented interval event ε_A, which corresponds to one motorway sector of our earlier example showing two periods of congestion. The segmented interval event ε_A is composed of four point events, yielding the flat representation $flat(\varepsilon_A) = \langle e_{1<}, e_{2\downarrow}, e_{3\uparrow}, e_{4>} \rangle$, such that $e_{1<}.t$ = 8:00, $e_{2\downarrow}.t$ = 08:02, $e_{3\uparrow}.t$ = 08:04, and $e_{4>}.t$ = 08:06.*

The definition of these segmented interval events allows us to define queries over their relations, as discussed next.

2.2 Queries based on Interval Relations

To reason about interval semantics, we rely on a query model based on Allen's algebra [8]. This algebra defines 13 distinct relations between intervals, seven of which can be seen as base cases that, except for *equals*, have a mirrored counterpart. Semantics for a relation is formally defined in terms of interval end points. For instance, two continuous interval events $\varepsilon_A = \langle e_<, e_> \rangle$ and $\varepsilon_B = \langle e'_<, e'_> \rangle$ satisfy the *is finished by* relation, if $e_<.t < e'_<.t$ and $e_>.t = e'_>.t$.

Fig. 3 provides notation and graphically illustrates the semantics of all 13 relations. Given two interval events ε_A and ε_B and an Allen relation C, we denote by $\varepsilon_A - C - \varepsilon_B$ the satisfaction of the C

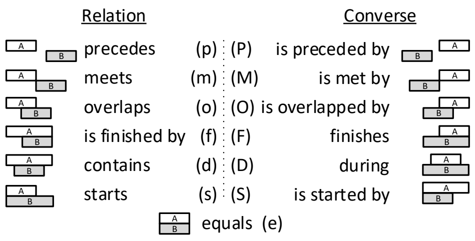


Fig. 3. Overview of Allen's relations, adapted from [9].

relation by the two events. Composite relations are constructed from subsets of the basic relations of Allen's algebra. Examples are the following relations:
Starts Before: $SB = \{p \vee m \vee o \vee d \vee f\}$.
Ends After: $EA = \{P \vee M \vee O \vee d \vee S\}$.
Intersects: $I = \{m \vee M \vee o \vee O \vee F \vee f \vee d \vee D \vee s \vee S \vee e\}$.

In its basic form, Allen's algebra relates to continuous interval events that are characterized by a start and an end event. We lift these relations to segmented interval events by applying them to event segments. Then, a relation is required to hold *for all*, *some k*, or *one* of the event segments of one interval event and *for all*, *some k*, or *one* segment of another event. Overloading set notation, given an interval relation for two interval events, we apply a universal ($\forall$), absolute ($\exists_k$), or existence ($\exists$) quantifier to either event type.

Table 1. $\varepsilon_A - C - \varepsilon_B$ segment extension.

	$\forall\ \varepsilon_B$	$\exists_{k'}\ \varepsilon_B$
$\forall\ \varepsilon_A$	$\forall\ \varepsilon \in \varepsilon_A$ $\forall\ \varepsilon' \in \varepsilon_B$: $\varepsilon - C - \varepsilon'$	$\forall\ \varepsilon \in \varepsilon_A$ $\exists\ K' = \{\varepsilon_1, \ldots, \varepsilon_{k'}\} \subseteq \varepsilon_B\ \forall\ \varepsilon' \in K'$: $\varepsilon - C - \varepsilon'$
$\exists_k\ \varepsilon_A$	$\exists\ K = \{\varepsilon_1, \ldots, \varepsilon_k\} \subseteq \varepsilon_A\ \forall\ \varepsilon \in K$ $\forall\ \varepsilon' \in \varepsilon_B$ $\varepsilon - C - \varepsilon'$	$\exists\ K = \{\varepsilon_1, \ldots, \varepsilon_k\} \subseteq \varepsilon_A\ \forall\ \varepsilon \in K$ $\exists\ K' = \{\varepsilon_1, \ldots, \varepsilon_{k'}\} \subseteq \varepsilon_B\ \forall\ \varepsilon' \in K'$: $\varepsilon - C - \varepsilon'$

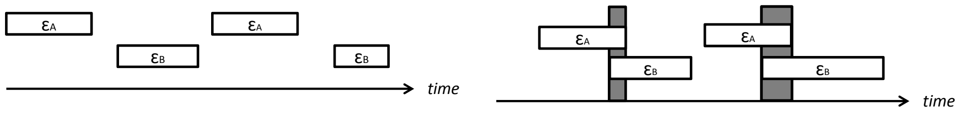
Table 1 summarises the extended semantics, for all combinations of universal and absolute quantifiers. The existential quantifier is derived directly from the absolute one, by interpreting $\exists$ as $\exists_1$. As an example, relation $\forall\ \varepsilon_A - p - \forall\ \varepsilon_B$ requires all segments of ε_A to precede all segments of ε_B.

The application of Allen's relations over segmented interval events naturally extends the existing algebra over continuous interval events, as indicated by the following property.

Property 1 (Segmented interval relations). *Let ϵ_A and ϵ_B be two continuous interval events and C an Allen relation. Then, it holds that $\varepsilon_A - C - \varepsilon_B \Leftrightarrow Q\varepsilon_A - C - Q'\varepsilon_B$ for any pair of quantifiers $Q, Q' \in \{\forall, \exists, \exists_k\}$.*

Property 1 follows directly from the semantics of the extended algebra, so that we omit the proof. To illustrate interval-based queries, consider the following two example queries capturing *k-sharing* and *containment* relations.

Example 2 (k-Sharing query). *Taking up our running example, we aim at detecting motorway sectors that are often jointly congested. In general, we query for this pattern using the* k-sharing *relation that, for two segmented interval events $\varepsilon_A, \varepsilon_B$, implies that for at least k segments of ε_A* (`congested` *intervals), there are intersecting segments in ε_B. An absolute quantifier enables fine tuning*



(a) k-Sharing is not satisfied for any k (b) k-Sharing is satisfied for $k \in \{1, 2\}$

Fig. 4. k-Sharing relation examples.

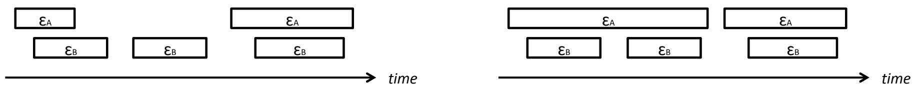
the strictness of the query. Using the intersection *relation I as defined above, we express* k-sharing *as* $\exists_k\ \varepsilon_A - I - \exists\ \varepsilon_B$*, where a higher k means that more intersections are required. Note that for* $k \geq 2$*, k-sharing is asymmetrical.*

Fig. 4(a) shows a case that does not satisfy the weakest form of k-sharing ($k = 1$*, at least one overlap). In Fig. 4(b), the query is satisfied for* $k = 1$ *and* $k = 2$*, since there are two segments of* ε_A *that intersect with segments of* ε_B*, as indicated by the shaded areas.*

To further illustrate interval-based queries, we demonstrate the use of the universal quantifier ∀ in the *containment* query, described in Example 3.

Example 3 (Containment query). *As a second example using the same interval events, one can aim to detect whether congestion at one motorway sector always occurs when a different sector is also congested. We query for this pattern using the* containment *relation that, for two segmented interval events* $\varepsilon_A, \varepsilon_B$*, implies that each segment of* ε_B *occurs during a segment of* ε_A*, i.e., for each segment of* $\epsilon_B \in \varepsilon_B$*, there is a segment* $\epsilon_A \in \varepsilon_A$ *such that* ϵ_A *starts before and finishes after* ϵ_B*. Using the* during *relation D as shown in Fig. 3, we express* containment *as* $\forall\ \varepsilon_B - D - \exists\ \varepsilon_A$*.*

Fig. 5(a) shows a case that does not satisfy the containment relation for two reasons. Namely, the first segment of ε_B *ends after the segment of* ε_A*, whereas the second segment of* ε_B *does not coincide with any segment of* ε_A*. In Fig. 5(b), the query is satisfied: each segment of* ε_B *is contained by a segment of* ε_A*. Here, it does not matter that two segments of* ε_B *are contained by the same segment of* ε_A*.*



(a) Containment relation not satisfied (b) ε_A contains ε_B

Fig. 5. Containment relation examples.

The example queries also allow us to illustrate the difference between our approach, based on queries for segmented intervals, and existing work on matching sequence patterns [5, 14, 18, 29]. For instance, the SASE+ engine, which supports probabilistic pattern matching [51], translates the k-sharing query into an exponential number of queries, since it requires to enumerate all possible sequences that satisfy the interval-based query. For the given example of two segmented interval events $\varepsilon_A, \varepsilon_B$, the following query encodes the case shown in Fig. 4(b).

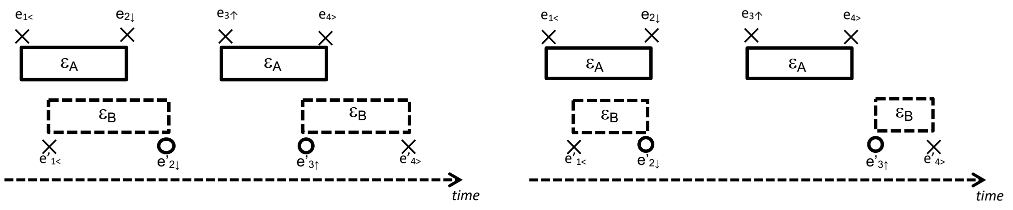
```
PATTERN
SEQ(A e1, B e1', A e2, B e2', A e3, B e3', A e4, B e4')
WHERE
e1.type  = "start" AND e1'.type = "start" AND
e2.type  = "suspend" AND e2'.type = "suspend" AND
e3.type  = "resume" AND e3'.type = "resume" AND
e4.type  = "end" AND e4'.type = "end"
WITHIN 5 days
```

The k-sharing relation for $k \in \{1, 2\}$ translates into 33 of these sequence patterns, i.e., into 33 different queries, even if the possibility of having events with equal timestamps is neglected. When considering larger k values, this number will increase in an exponential manner.

2.3 Interval Uncertainty through Lost events

Event recordings are susceptible to various errors, such as data losses, load shedding policies, and synchronization issues [21, 51]. Due to such errors, not all point events of a segmented interval event may have been properly recorded, *i.e.,* events may have been lost. This gives a situation in which a segment may not be temporarily bounded by point events, leaving it open at one or both ends. Such as, for instance, depicted in Fig. 6(a), where both segments of ε_B have open ends.

It is important to recognize that the identifiers assigned to the observed events allow us to infer how many point and which type of point events were lost in a given interval. For instance, in Fig. 6(a), we know that two point events, one *suspend* and one *resume*, were lost. By including such inferred lost events, given a valid segmented interval event $\varepsilon = \langle \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n \rangle$, the timestamp $e.t$ becomes a random variable for each point event $e \in \varepsilon$. An event e that was recorded properly, *i.e.,* for which an occurrence time is known deterministically, $e.t = t$, is characterized by assigning a probability of 1 to time t. We use an indicator $R(e)$ that is set to *true* if e was recorded and *false* if e was lost. For a lost event e, such that $R(e) = \mathit{false}$, we assume a probability distribution over $e.t$, bounded by the last observed event preceding e, and the first observed event succeeding e. If all events preceding or succeeding e are lost, we assume that the earliest and latest possible occurrence times are known and can serve as bounds.



(a) Possible occurrence with two overlaps (b) Possible occurrence with single overlap

Fig. 6. Impact of lost events on segmented intervals.

Given a valid segmented interval event ε and two consecutive point events e_1, e_2 in *flat*(ε), we denote by $d^{e_1}_{e_2}$ a random variable representing the duration between e_1 and e_2, thus $d^{e_1}_{e_2} = e_2.t - e_1.t$. Given a set of point events E, which may be known either deterministically or stochastically, $f(d^{e_1}_{e_2}|E)$ represents a conditional distribution function, conditioned by some of the events in E.

Example 4 (Lost events). *Fig. 6(a) provides an illustration of a segmented interval event with lost point events.* ε_B *consists of two segments, such that* $\varepsilon_B = \langle e'_{1<}, e'_{2\downarrow}, e'_{3\uparrow}, e'_{4>} \rangle$. *Assume that* $e'_{1<}.t =$ *8:01 and* $e'_{4>}.t =$ *08:10, whereas the* $e'_{2\downarrow}$ *and* $e'_{3\uparrow}$ *point events were lost,* i.e., $R(e'_{2\downarrow}) = R(e'_{3\uparrow}) = \mathit{false}$, *as depicted in the figure by the circles. As a result of this, the timestamp of, for instance,* $e'_{3\uparrow}$ *is unknown, but bounded by the known timestamps of its closest recorded preceding and succeeding events,* i.e., $e'_{3\uparrow}.t$ *is a random variable bounded by* $e'_{1<}.t$ *and* $e'_{4>}.t$.

The presence of lost events has a considerable impact on interval-based queries, since the unknown length of segments caused by lost events can make it impossible to answer queries in a deterministic manner, such as illustrated by the following example.

Example 5. *Reconsider the case from Example 4, where ε_B has two unbounded segments due to lost events. When trying to answer, for instance, a k-sharing query for this case, one needs to consider the impact of lost events on this query. In particular, given that the starting point of the second segment* (i.e., *$e'_{3\uparrow}.t$) is unknown, one cannot be certain if this segment overlaps with a segment from ε_A. This is illustrated by Fig. 6, which depicts two possible situations that may have occurred. In the first case, $e'_{3\uparrow}$ occurred* before *$e_{4>}$, which results in an overlap between the segments of ε_A and ε_B. However, as depicted in Fig. 6(b), event $e'_{3\uparrow}$ may also have occurred only after the end of ε_A's segment, in which case there would be no overlap, thus leading to a different answer of the k-sharing query.*

To overcome these issues caused by lost events, we shall therefore approach interval-based querying in a probabilistic manner, as defined next.

2.4 Problem Statement

With the model of uncertain intervals and the extension of Allen's algebra to segmented interval events, we can now introduce the *probabilistic interval-based querying problem.*

Problem 1 (Probabilistic interval-based querying). *Given two valid segmented interval events $\varepsilon_1, \varepsilon_2$ and a query comprising an interval relation C and two quantifiers $Q, Q' \in \{\forall, \exists_k, \exists\}$, the* probabilistic interval-based querying problem *is the computation of* $\Pr\{Q\varepsilon_1 - C - Q'\varepsilon_2\}$, i.e., *the probability of the query to hold true for ε_1 and ε_2.*

Solving Problem 1 assists in answering interval-based queries such as the congestion query in urban transportation, which serves as our use-case example. With certain events missing, we can only provide a probabilistic response to the query, drawing on a solution to Problem 1 to quantify the extent to which our query response is valid.

3 PROBABILISTIC INTERVAL-BASED QUERY RESOLUTION

This section describes our approach for the resolution of probabilistic interval-based queries. To address this task, we first formulate probabilistic resolution as a *possible worlds* problem (Section 3.1), *i.e.,* we tackle query resolution based on an enumeration over different event orders. Then, we introduce a method for the computation of probabilities for such event orders (Section 3.2). We formally show that our method yields an exact solution if event arrival follows a Poisson process. Otherwise, it returns a heuristic solution that serves as the best stochastic approximation. Finally, we present the *SimpMatch* algorithm as an efficient way of resolving probabilistic interval-based queries (Section 3.3)

3.1 Possible World Formulation

To resolve probabilistic interval-based queries, we recognize that lost events, *i.e.*, events for which timestamps are not known, lead to different possible orders in which the point events of two segmented interval events have occurred, also referred to as *possible worlds.* To formalize these, we first need to introduce the concept of a *timestamp assignment.* Such an assignment represents an instantiation of timestamps for all point events in a segmented interval event ε, defined as follows.

Definition 2 (Timestamp assignment). *Let ε be a valid segmented interval event with flat(ε) = $\langle e_1, e_2, \ldots, e_n \rangle$. A* timestamp assignment *of ε is a set $T = \{t_1, t_2, \ldots, t_n\}$ such that $t_1 < t_2 < \ldots < t_n$ and for all $1 \leq i \leq n$, if $R(e_i)$ = true then* $\Pr\{e_i.t = t_i\} = 1$.

Note that, for a timestamp assignment T of an event ε with *flat*(ε) = $\langle e_1, e_2, \ldots, e_n \rangle$, we denote by $t_T(e_i) = t_i$ the timestamp assigned to event e_i under T. If T is clear from the context, we simply write $t(e_i)$. Based on the notion of timestamp assignments, we define a possible world as a total order of the point events of segmented intervals events ε_A and ε_B, given by Def. 3.

Definition 3 (Possible world). *Given two segmented interval events ε_A and ε_B with flat(ε_A) = $\langle e_1, \ldots, e_n \rangle$ and flat(ε_B) = $\langle e'_1, \ldots, e'_m \rangle$, respectively, a* possible world $w = \langle g_1, g_2, \ldots, g_{n+m} \rangle$ *is a sequence of events, such that*

- *each event stems from either ε_A or ε_B,* i.e., *g_i, $1 \leq i \leq n+m$, is an element of flat(ε_A) or flat(ε_B),*
- *each event is unique,* i.e., *$g_i = g_j$ for $1 \leq i, j \leq n+m$ implies $i = j$,*
- *there exist timestamp assignments T of ε_A and T' of ε_B that induce the order of w,* i.e., $t(g_1) \leq t(g_2) \leq \ldots \leq t(g_{n+m})$.

As an illustration, reconsider the example from Fig. 6, which depicts two possible worlds induced by the lost events, *i.e.*, $w_1 = \langle e_1, e'_1, e_2, e'_2, e_3, e'_3, e_4, e'_4 \rangle$ and $w_2 = \langle e_1, e'_1, e'_2, e_2, e_3, e_4, e'_3, e'_4 \rangle$. However, note that there are already a total of 10 possible worlds for this example.

Since possible worlds are independent, an interval-based query $Q\varepsilon_A - C - Q'\varepsilon_B$ can be resolved by summing up the probabilities of all possible worlds that satisfy the query. Using $w \models Q\varepsilon_A - C - Q'\varepsilon_B$ to denote this set, the resolution of a query is then given by:

$$\Pr\{Q\varepsilon_A - C - Q'\varepsilon_B\} = \sum_{w \models Q\varepsilon_A - C - Q'\varepsilon_B} \Pr\{w\}\,. \tag{1}$$

In the remainder of this section, we first consider how to compute the probability $Pr\{w\}$ of a possible world (Section 3.2), followed by our proposed algorithm for the efficient resolution of interval-based queries (Section 3.3).

3.2 Probability Computation

In this section, we introduce our method for the efficient probability computation of a possible world, *i.e.*, the probability that the point events of two segmented interval events followed a specific sequence. We show that our method yields an exact solution if event arrival follows a Poisson process. Otherwise, it returns a heuristic solution that serves as the best stochastic approximation, given that the only knowledge about the segmented interval events are the observed events.

Lost event sequence. We base the probability computation for a possible world on probabilities assigned to subsequences of lost events bounded by recorded events. We shall refer to such a subsequence as a *lost event sequence*, formally defined as follows:

Definition 4 (Lost event sequence). *Let ε_A be a segmented interval event with flat(ε_A) = $\langle e_1, e_2 \ldots, e_n \rangle$. A* lost event sequence $\langle e_k, e_{k+1}, \ldots, e_l \rangle$, *$1 < k \leq l < n$ is a subsequence of flat(ε_A), such that $R(e_i)$ = false for $k \leq i \leq l$, while $R(e_{k-1}) = R(e_{l+1})$ = true.*

Consider, for instance, a possible world $w_3 = \langle e_1, e'_1, e_2, e'_2, e'_3, e_3, e_4, e'_4 \rangle$, with e'_2 and e'_3 as lost events. The world w_3 has one lost event sequence, $\langle e'_2, e'_3 \rangle$, bounded by the observed events e_2 and e_3. For such a lost event sequence, assuming that lost events are uniformly distributed across bounded windows allows for efficient computation of the probability of their ordering. As we show later, this assumption holds true for event arrival according to a Poisson process. Moreover, this assumption is reasonable if the arrival rates of events (*i.e.*, their distributions) are not known at all.

Probability density function. We can compute the joint probability distribution of a lost event sequence using *order statistics* [36]. Given a lost event sequence $\langle e_k, e_{k+1}, \ldots, e_l \rangle$, we define a set of i.i.d. random variables $\mathcal{U} = \{U_k, U_{k+1}, \ldots, U_l\}$. These random variables are uniformly distributed over the window bounded by the deterministically known timestamps of the sequence's preceding and succeeding events, *i.e.*, over the bounded window $(e_{k-1}.t, e_{l+1}.t)$. Each $U_i \in \mathcal{U}$ represents the time that elapses from the window start, $e_{k-1}.t$, until the arrival of an event from the lost event sequence. This means that all U_i are defined relatively to the same starting point.

Given $\mathcal{U} = \{U_k, U_{k+1} \ldots, U_l\}$, the *order statistics* of $\mathcal{U}$ define a sorted order of $\mathcal{U}$, given as a set of random variables $\{U_{(1)}, U_{(2)}, \ldots U_{(m)}\}$, with $m = l - k + 1$. The elements of the lost event sequence $\langle e_k, e_{k+1}, \ldots, e_l\rangle$ are ordered in a strictly increasing order of their timestamp. Therefore, we can uniquely assign a timestamp to each event e_i (for $k \leq i \leq l$), *i.e.,* $e_i.t = e_{k-1}.t + U_{(i-k+1)}$. Based thereon, the joint probability density function for the timestamps of the events in a lost event sequence follows the basic property of order statistics [20], as given by Lemma 1:

LEMMA 1 (JOINT PROBABILITY DENSITY FUNCTION). *Let $\langle e_k, e_{k+1}, \ldots, e_l\rangle$ be a lost event sequence and $\{U_{(1)}, U_{(2)} \ldots U_{(m)}\}$, $m = l - k + 1$, the respective order statistics. The joint probability density function of $e_k.t, e_{k+1}.t, \ldots, e_l.t$ is given as:*

$$\begin{aligned} f(e_k.t, e_{k+1}.t, \ldots, e_l.t) &= f(e_{k-1}.t + U_{(1)}, e_{k-1}.t + U_{(2)}, \ldots, e_{k-1}.t + U_{(l-k+1)}) \\ &= f(U_{(1)}, U_{(2)}, \ldots, U_{(m)}) \\ &= m! f(U_1) f(U_2) \ldots f(U_m) \\ &= \frac{m!}{(e_{l+1}.t - e_{k-1}.t)^m} \end{aligned}$$

Based on the density function of a single lost event sequence, we can compute the probability of a specific order of two such sequences, as discussed next.

Merged sequence probability. Consider two lost event sequences $\varepsilon = \langle e_k, e_{k+1}, \ldots, e_l\rangle$ and $\varepsilon' = \langle e'_q, e'_{q+1}, \ldots, e'_r\rangle$ of two segmented interval events ε_A and ε_B, respectively. Recall that the timestamps of the boundary events, $e_{k-1}, e_{l+1}, e'_{q-1}$, and e'_{r+1}, are all deterministically known. We would like to compute the probability of a timestamp ordering of all events in both lost sequences, within an interval set by these boundary events, *i.e.,* an interval $(\max(e_{k-1}.t, e'_{q-1}.t), \min(e_{l+1}.t, e'_{r+1}.t))$. To this end, we exploit the following theorem. It links the probability of a particular timestamp ordering to the density functions of the two lost event sequences that are merged.

THEOREM 1 (MERGED SEQUENCE PROBABILITY). *Let $\varepsilon = \langle e_k, e_{k+1}, \ldots, e_l\rangle$ and $\varepsilon' = \langle e'_q, e'_{q+1}, \ldots, e'_r\rangle$ be the lost event sequences of segmented interval events ε_A and ε_B, respectively, such that $|\varepsilon| = m$ and $|\varepsilon'| = n$. Let $e_{(0)}.t = \max(e_{k-1}.t, e'_{q-1}.t)$, $e_{(m+n+1)}.t = \min(e_{l+1}.t, e'_{r+1}.t)$, and $\langle e_{(1)}, e_{(2)}, \ldots, e_{(n+m)}\rangle$ be a (possibly interleaving) ordering of the sequences ε and ε'. Then,*

$$\Pr\left\{e_{(0)}.t < e_{(1)}.t < \ldots < e_{(n+m)}.t < e_{(m+n+1)}.t\right\} = \frac{(e_{(m+n+1)}.t - e_{(0)}.t)^{n+m}}{(n+m)!} \cdot \frac{m!}{(e_{l+1}.t - e_{k-1}.t)^m} \cdot \frac{n!}{(e_{r+1}.t - e_{q-1}.t)^n}.$$

PROOF. We integrate over the density functions of both lost event sequences (Lemma 1), exploiting the interleaving ordering for the integration:

$$\Pr\left\{e_{(0)}.t < e_{(1)}.t < \ldots < e_{(n+m)}.t < e_{(m+n+1)}.t\right\} = \int_0^{e_{(m+n+1)}.t - e_{(0)}.t} \int_0^{e_{(m+n)}.t} \ldots \int_0^{e_{(2)}.t} \frac{m!}{(e_{l+1}.t - e_{k-1}.t)^m} \cdot \frac{n!}{(e_{r+1}.t - e_{q-1}.t)^n} \, de_{(1)}.t, \ldots, de_{(m+n)}.t$$

The density functions of both lost sequences are independent of the integration variables, so that can rewrite the equation as follows:

$$\Pr\left\{e_{(0)}.t < e_{(1)}.t < \ldots < e_{(n+m)}.t < e_{(m+n+1)}.t\right\} = \frac{m!}{(e_{l+1}.t - e_{k-1}.t)^m} \cdot \frac{n!}{(e_{r+1}.t - e_{q-1}.t)^n} \int_0^{e_{(m+n+1)}.t - e_{(0)}.t} \int_0^{e_{(m+n)}.t} \ldots \int_0^{e_{(2)}.t} 1 \, de_{(1)}.t, \ldots, de_{(m+n)}.t$$

The above integration can be viewed as the volume of a simplex in an $n + m$-dimensional space. Hence, the probability is given as

$$\Pr\left\{e_{(0)}.t < e_{(1)}.t < \ldots < e_{(n+m)}.t < e_{(m+n+1)}.t\right\} = \frac{m!}{(e_{l+1}.t - e_{k-1}.t)^m} \cdot \frac{n!}{(e_{r+1}.t - e_{q-1}.t)^n} \cdot \frac{(e_{(m+n+1)}.t - e_{(0)}.t)^{n+m}}{(n+m)!}$$

which concludes the proof. □

An immediate corollary of Theorem 1 is that the probability depends on the sizes of the lost event sequences and the positioning of the bounding events on the time-line, but not on the ordering of events. This property becomes handy when optimising event detection, since the probability computed for a certain lost event sequence may be reused for different interleavings of the same lost events as part of different possible worlds.

Unbounded sequences. Theorem 1 holds for event sequences bounded by deterministically known timestamps. In extremes cases, either a start or an end event may be lost. If so, the probability is directly computed by integrating over the inter-arrival random variables.

Guarantee for Poisson event arrival. The arrival of events of an uncertain segmented interval event can be modelled as a stochastic process $N(t), t \geq 0, t \in \mathbb{N}$ being the time offset from t_s, the start of the process. Then, $N(t)$ is the number of events in the interval $[t_s, t_s + t]$.

The inter-arrival time distribution of Poisson processes is *memoryless.* Every time an event arrives, the process resets and the probability of event arrival depends only on the time of the last arrival, but not on the absolute time or the number of previous events. Further, the time between two event arrivals follows an exponential distribution.

Given a Poisson process, the distribution of a lost event sequence is according to the order statistics of the lost events over the bounding window [20]. Hence, Theorem 1 yields the analytic solution for the probability of the interleaving ordering of the sequences.

Possible world probability. A possible world may contain multiple lost event sequences. Given the probabilities computed for each of these lost event sequences, the probability of the world itself is determined as the product of the probabilities of its lost event sequences. The rationale here is that the lost event sequences are independent of each other.

3.3 Resolution Algorithm

We propose the SimpMatch algorithm for the probabilistic resolution of interval-based queries over lossy event streams. Given an interval-based query $Q\varepsilon_1 - C - Q'\varepsilon_2$ and two valid segmented interval events ε_A and ε_B, the algorithm iteratively constructs a *possible worlds tree* that captures the different sequences in which point events of ε_A and ε_B may occur, as well as the probabilities of each (sub)sequence in the tree. The SimpMatch algorithm is formalized in Algorithms 1 and 2 and described in detail in the remainder.

Initialisation. The SimpMatch algorithm (Alg. 1) starts by initializing necessary variables. The algorithm records the number of point events in ε_A and ε_B (lines 1–2), prior to establishing sentinel events for ε_A and ε_B (line 3), which are dummy events with infinitely large timestamps. Further, the algorithm initialises a possible worlds tree T, consisting of just a root node r, it also initializes a variable recording the current probability, pr. Afterwards, a first call to the ExpandNode function is made (line 7), setting the positions i and j to the first point events in ε_A and ε_B.

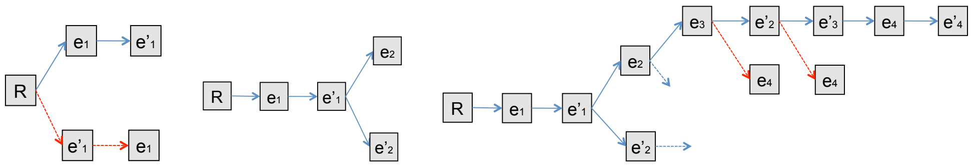
ExpandNode function. The expandNode function is called with a pointer to a current node r in the possible worlds tree and positions in the segmented interval events, i and j. Given the current

Algorithm 1: The SimpMatch algorithm

```
input  : valid segmented interval events $\varepsilon_A$ and $\varepsilon_B$,
         a query comprising an interval relation $C$ and quantifiers $Q$ and $Q'$, $Q\varepsilon_1 - C - Q'\varepsilon_2$
output : probability of the query to hold true, $\Pr\{Q(\varepsilon_A) - C - Q'(\varepsilon_B)\}$
1  $n := |\varepsilon_A|$;
2  $m := |\varepsilon_B|$;
3  $e_{n+1}.t := e'_{m+1} := \infty$;                                   /* set sentinels */
4  $T :=$ initTree();                                         /* create possible worlds tree $T$ */
5  $r :=$root($T$);                                  /* set the root of $T$ as the current node $r$ */
6  $pr := 1.0$;                                          /* initialise probability variable $pr$ */
7  expandNode($T, r, 1, 1, pr$)                                 /* first call to recursive function */
8  function expandNode($T, r, i, j, pr$)
9  if $i \leq n \vee j \leq m$ then                          /* check if end of intervals has been reached */
10     $e_i := \varepsilon_A$.get($i$);                                        /* Get $i$th event from $\varepsilon_A$ */
11     $e_j := \varepsilon_B$.get($i$);                                        /* Get $j$th event from $\varepsilon_B$ */
12     $pr_A :=$ FollowPathA($T, r, e_i, e_j$);     /* query resolution probability with next event from $\varepsilon_A$ */
13     $pr_B :=$ FollowPathB($T, r, e_i, e_j$);     /* query resolution probability with next event from $\varepsilon_B$ */
14     return $pr_A + pr_B$;                              /* Return the sum of the two probabilities */
15 return $pr$;                                      /* End reached, return current probability value */
```

node r, there are two possible ways to expand the tree: either point event e_i from ε_A is the next event to be considered or point event e_j from ε_B. expandNode calls functions for both possibilities (lines 12–13). These functions, respectively, return the probability that the query $Q\varepsilon_A - C - Q'\varepsilon_B$ matches if the next event to be considered is e_i or e_j, stored in pr_A and pr_B. Given that the two options are independent, the function returns their sum (line 14). As visualized in Fig. 7(a), for the first call of expandNode for the running example, these options involve appending e_1 or e'_1 to R.

Note that if expandNode is called with $i = n$ and $j = m$, it means that all point events of both ε_A and ε_B have been explored in the current path, *i.e.,* that the end of the path has been reached. In that case, the algorithm returns the current probability pr (line 15), which represent the probability of the entire possible world. This is, for instance, the case when e'_4 is reached for the possible world depicted in Fig. 7(c).



(a) First iteration (b) Lost event e'_2 (c) Complete possible world

Fig. 7. Progression of a possible worlds tree in SimpMatch for 2-sharing.

FollowPathA function. As denoted in Alg. 2, followPathA computes the probability of the interval-based query to hold when e_i is the next event, added to the current node r in the possible worlds tree. To determine this probability, the algorithm first checks if the proposed ordering, in which e_i precedes e_j is possible according to their timestamps (line 1). To do this, the checkOrder function considers the following cases, depending on whether or not e_i and e_j represent recorded events:

- **Both events recorded**, *i.e.*, $R(e_i) = true \wedge R(e_j) = true$. If both events have been recorded, then the possiblity of the ordering $\langle e_i, e_j\rangle$ can be directly checked by comparing their (known) timestamps. If $e_i.t \leq e_j.t$, the order is possible, otherwise not. This case happens at the first

Algorithm 2: FollowPathA function

```
input  : the next point events from the two segmented interval events, $e_i$ and $e_j$,
         the worlds tree $T$, the current node of focus $r$, and the current probability value $pr$
output : the probability of the interval-based query to hold with $e_i$ as the next event
1 if checkOrder($e_i, e_j$) then                                  /* Check if $e_i$ can occur before $e_j$ */
2     $(T', r') :=$ Add($T, r, e_i$);                 /* Add $e_i$ after node $r$ in the possible worlds tree */
3     if matchesQuery($T', r'$) then                          /* Check if current path still matches query */
4         if $R(e_i) = true$ then                                  /* Check if $e_i$ is a recorded event */
5             $\langle e_k, e_{k+1}, \ldots, e_l\rangle :=$ TraceBack($T', r'$);   /* Trace back tree to last recorded event */
6             $pr = pr \cdot$ computeProb($\langle e_k, e_{k+1}, \ldots, e_l\rangle$);  /* Compute sequence probability using Theorem 1 */
7         return expandNode($T', r', i+1, j, pr$) ;       /* Continue algorithm by expanding the new node $r'$ */
8 return 0.0;                        /* The current sequence is not possible or does not match the query */
```

iteration of the algorithm, as visualized in Fig. 7(a). Here, the order $\langle e_1, e_1'\rangle$ is feasible, since $e_1. < e_1'.t$, whereas $\langle e_1', e_1\rangle$ is not. Therefore, in the first iteration, only e_1 is added to R.

- **$\mathrm{e_i}$ not recorded**, *i.e.,* $R(e_i) = false$. If the candidate next event, e_i, was not recorded, then we do not know its time of occurrence. Therefore, the ordering $\langle e_i, e_j\rangle$ is possible, independent of whether e_j was recorded or its timestamp.
- **Only $\mathrm{e_i}$ recorded**, *i.e.*, $R(e_i) = true \wedge R(e_j) = false$. If e_i was recorded but not e_j, we consider if e_i occurred before the next known timestamp at which an event of the other interval (ε_B) was recorded, which we shall denote as $\textsc{next}(\varepsilon_B).t$. If $e_i \leq \textsc{next}(\varepsilon_B).t$ then there may be a timestamp assignment in which $e_i.t \leq e_j.t$, *i.e.,* in which $\langle e_i, e_j\rangle$ is possible. However, if $e_i > \textsc{next}(\varepsilon_B).t$, it means that e_i already occurred after the event succeeding e_j. Since it must hold that $e_j.t < \textsc{next}(\varepsilon_B).t$, e_i can only have occurred after e_j.
  This situation is depicted in Fig. 7(b), where both the observed event e_2 and the lost event e_2' may follow event $e1'$, since e_4', the next observed event in ε_B, occurred after e_2.

If the ordering $\langle e_i, e_j\rangle$ is possible, *i.e.,* $\textsc{checkOrder}(e_i, e_j) = true$, the algorithm continues by adding e_i as a new node, r', to the possible worlds tree (line 2). Given the updated tree T' and the node r', the algorithm then determines if the current branch of the possible worlds tree can still satisfy or if it will certainly violate the interval-based query (line 3), as described next.

MatchesQuery predicate. The size of the possible worlds tree is exponential in the number of lost point events in ε_A and ε_B. However, using the segmented interval event model, constraints over the orderings of events enable elimination of significant parts of the tree. Specifically, the SimpMatch algorithm, using the MatchesQuery function in Alg. 2, performs early detection of tree branches that are known to violate the queried relation and can thus be pruned.

In the example of Fig. 7(c), this pruning step occurs when exploring the addition of e_4 after e_3. If e_4 is added to the current path, then the 2-sharing query will not be able to match any longer, given that there is currently only a single overlapping segment and that e_4 is the *end* event of ε_A. As a result, MatchesQuery shall return false in this case meaning that this path will not be explored further and a probability of 0.0 is returned. By contrast, the option of adding e_2' after e_3 does leave open the possibility of the query to match, which means that this path shall be continued.

ComputeProb function. If the order is possible and there is no query violation, the algorithm subsequently considers if e_i was recorded or not (line 4). If this is the case, then e_i can serve as a boundary event for the computation of a probability for a lost event sequence, as described in Section 3.2. To do this, the algorithm first retrieves the lost event sequence (line 5) by tracing back in the possible worlds tree to the last recorded event in the current path. The sequence $\langle e_k, e_{k+1}, \ldots, e_l\rangle$, identified in this manner, represents a lost event sequence and its boundary events. Using Theorem 1, we can, therefore compute the probability of the current subsequence (line 6). Afterwards, the algorithm continues by calling the EXPANDNODE function for the next pair of events (at positions $i + 1$ and j, line 7).

As mentioned in Section 3.2, the probability computation for a lost event sequence depends solely on its length and the positioning of the bounding events, but not on the ordering of events in the sequence. As a consequence, caching the computed probabilities (line 6) helps to improve the runtime of the algorithm. the probability of one lost event sequence may be reused for all interleavings of the same events.

Note that if either of the CHECKORDER or MATCHESQUERY checks fail, a probability of 0.0 is returned (line 8), since the path being explored will not be able to satisfy the query. Furthermore, we omit the FOLLOWPATHB function, for brevity, since it is identical to FOLLOWPATHA except that it considers the order $\langle e_j, e_i \rangle$ instead.

Streaming environments. Finally, we note that the SimpMatch algorithm can be applied in streaming environments. It can be activated in a step-wise manner, each time a new event is observed. The algorithm terminates when the last events of both intervals are observed or once a timeout has been reached. Events that are processed after the timeout are considered to be lost.

4 EXPERIMENTAL EVALUATION

This section presents an experimental evaluation of the SimpMatch algorithm in terms of effectiveness and efficiency. For this evaluation, we employ a real-world benchmark dataset, as well as synthetic data. The results of our experiments indicate that the algorithm allows for solving the probabilistic interval detection problem with a throughput of up to 2,000,000 events per second and an accuracy of around 90% when 10% of the timestamps are missing. We also compare SimpMatch with two discrete baselines that do not employ probabilistic reasoning as well as with the reasoning technique of a state-of-the-art engine. The results demonstrate that our method achieves consistently higher accuracy. In comparison to the existing reasoning technique, it increases the runtime performance by up to two orders of magnitude.

Below, we present the used datasets (Section 4.1) and applied experimental setup (Section 4.2), before reporting our results (Section 4.3).

4.1 Datasets

We employ the following benchmark and synthetic datasets for our evaluation experiments.

Linear Road Benchmark dataset. The Linear Road Benchmark (LRB) [11] comprises movement events of vehicles on toll motorways. Toll depends on the level of congestion of motorway sectors, which is derived from the average speed of vehicles. We used 2 hours worth of events for 200 motorway sectors to identify segmented interval events, representing the congestion of a particular sector in one direction. Following the specification of the LRB, congestion is identified when the average speed of vehicles falls below a threshold. We used two such threshold values, 15 and 20 km/h. We refer to the datasets resulting from these congestion thresholds, respectively, as *benchmark (15)* and *benchmark (20)*.

In general, we observe large variability in terms of congestion levels throughout the data: For the benchmark (20) data, 138 sectors are never congested, 12 sectors have one congestion interval segment and, in the extreme case, a sector has up to 224 congestion interval segments. In the benchmark (15) data, 169 sectors are never congested. Still, the maximum number of congestion interval segments is high, reaching up to 102 segments for a sector.

In our experiments, we focused on the detection of correlations of congestion in different motorway sectors. By pairing each motorway sector to all other, we derived 19,900 event streams, each comprising a different pair of motorway sectors.

Table 2. Query selectivity statistics.

k	1	2	3	4	5	6	7	8	9	10	11	12
Synthetic	100.0%	100.0%	100.0%	100.0%	99.9%	99.7%	92.6%	72.1%	40.2%	10.8%	0.0%	0.0%
Benchmark (15)	39.5%	26.0%	22.1%	18.4%	15.6%	14.0%	11.3%	10.6%	8.7%	6.9%	5.9%	5.5%
Benchmark (20)	30.1%	24.5%	22.1%	20.6%	19.2%	18.1%	17.7%	17.1%	16.7%	16.0%	15.8%	15.6%

Synthetic dataset. To further test our method in a controlled setting, we created a synthetic event log of 500 event streams, each containing a pair of segmented interval events of 20 segments *i.e.*, 80 events. Each segmented interval event was generated by sampling the independent random variables for the durations between events from an exponential distribution with $\lambda = 1/5$. Consequently, for this dataset, arrival of events adheres to a Poisson process. We denote this dataset as *synthetic*.

4.2 Experimental Setup

Below, we discuss the queries used in our evaluation, the controlled parameters, evaluation measures, and methods for comparison.

Event queries. In our experiments, we detect variations of the *k-sharing* temporal pattern. In particular, we test for the existence of an intersection of any interval segments, which corresponds to the k-sharing query with k set to one. In addition, we test for the k-sharing pattern with an absolute parameter that sets a minimal number (k) of required intersections. These queries are representative for many real-world workloads in IoT applications that determine the joint occurrence of particular phenomena, such as congestion management in public transportation (*e.g.*, intersection of traffic flow peaks with bus delays, correlation of congestions at different areas) [13] and the evaluation of surveillance video streams (*e.g.*, intersections of persons or objects being present at particular locations, intersections of location information and activity information) [12, 33]. The *intersection* query is defined as $\exists\ \varepsilon_A - I - \exists\ \varepsilon_B$, while the *k-sharing* query is defined as $\exists_k\ \varepsilon_A - I - \exists\ \varepsilon_B$ (see also Example 2).

Table 2 illustrates the frequencies of query matches, *i.e.*, for how many pairs of segmented interval events the respective queries match. For instance, the *intersection* query ($k = 1$) matches in all cases in the synthetic dataset, but only in 39.5% and 30.1% of the cases in the benchmark datasets, respectively. Considering the *k-sharing* query and increasing the value for parameter k decreases the amount of matches. For the synthetic dataset, no matches are obtained for k larger than 10. For the benchmark datasets, a small number of cases still shows the respective number of intersecting interval segments. Note that these query matches are obtained on data without lost events and, therefore, represent the gold standard against which we shall evaluate our probabilistic approach.

Parameters. The *probabilistic interval detection problem* (Section 2.4) is tuned with two main parameters that relate to the data and the pattern recognition, as follows:

Noise level (E): A feature of the data, representing the fraction of lost events. We consider noise levels for E in the range [0%, 40%], in increments of 5%. For a given noise level E, each event has a probability of E to be considered as a lost event. However, we keep the end events with a known timestamp.

Detection threshold (T): The probability that defines pattern recognition by a query. For example, $T > 0.5$ means that a pattern is recognized whenever the computed occurrence probability is larger than 50%. We vary T over the range $[0.0, 1.0]$ in increments of 0.1.

Measures. We, respectively, evaluated the efficiency and effectiveness of our approach in terms of *throughput* and *accuracy*.

We measure efficiency based on throughput, *i.e.*, the average number of processed events per second. Throughput is measured on a desktop machine (Intel Core i5, 2.5GHz). Accuracy records the share of correctly detected (or undetected) relations. With TP as the set of true positives (correctly detected relations), FP the set of false positives (incorrectly detected relations), TN the set of true negatives (correctly undetected relations), and FN the set of false negatives (incorrectly undetected relations), accuracy is defined as follows:

$$Accuracy = \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|}.$$

Comparison with discrete baseline techniques. We compared the SimpMatch algorithm against two discrete baseline techniques that do not involve any probabilistic reasoning. They answer the query through a discretization, i.e., they deterministically construct a certain segmented interval event from the uncertain one by considering the semantics of the observed point events:

- *BaselineIgnore* simply ignores lost events. To obtain a sound segmented interval event when ignoring the lost events, algorithmically, all point events that have the same semantics as the preceding event are also ignored.
- *BaselineStatic* reconstructs some lost events based on the average duration of segments. For each point event that is observed, but denotes the beginning (start and resume events) of a segment that is not terminated by a corresponding suspend or end event (i.e., this event is lost), the latter is reconstructed. The timestamp of this reconstructed event is determined by taking the minimum of (i) the time obtained by adding the mean duration over all observed segments to the timestamp of the event that denotes the beginning and (ii) the timestamp of the next observed event. The mirrored construction is applied to suspend and end events that are not matched by resume and start events, respectively.

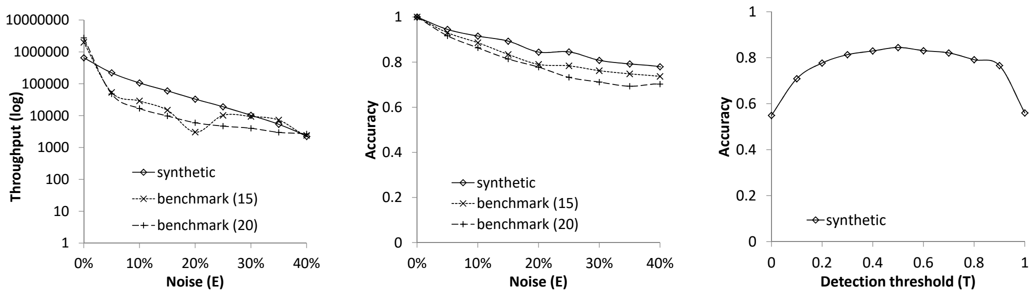
As an example, consider a segmented interval event ε with $flat(\varepsilon) = \langle e_{1<}, e_{2\downarrow}, e_{3\uparrow}, e_{4\downarrow}, e_{5\uparrow}, e_{6>} \rangle$, such that $e_{1<}.t$ = 8:00, $e_{2\downarrow}.t$ = 08:02, $e_{3\uparrow}.t$ = 08:04, $e_{4\downarrow}.t$ = 08:12, $e_{5\uparrow}.t$ = 08:20, and $e_{6>}.t$ = 08:24. Assume that events $e_{2\downarrow}$ and $e_{6>}$ are lost.

Based on these lost events, the *BaselineIgnore* technique ignores events $e_{3\uparrow}$ and $e_{5\uparrow}$, which lack a counterpart according to the semantics of the recorded (i.e., not lost) point events. As a result, the baseline effectively merges the first two segments, while discarding the third one. The resulting segmented interval event ε_I is characterised as $flat(\varepsilon_I) = \langle e_{1<}, e_{4\downarrow} \rangle$.

The *BaselineStatic* technique, in turn, would reconstruct the two lost events based on the mean duration of the observed segments, i.e., eight time units ($e_{3\uparrow}.t - e_{4\downarrow}.t$) in the example. The reconstructed events would be $\hat{e}_{2\downarrow}$ and $\hat{e}_{6>}$ with $\hat{e}_{2\downarrow}.t$ = 08:08 and $\hat{e}_{6>}.t$ = 08:28. The resulting segmented interval event ε_S is therefore characterised as $flat(\varepsilon_S) = \langle e_{1<}, \hat{e}_{2\downarrow}, e_{3\uparrow}, e_{4\downarrow}, e_{5\uparrow}, \hat{e}_{6>} \rangle$.

Comparison with the state-of-the-art. We further considered the probabilistic reasoning technique of SASE+ [51]. The SASE+ engine represents the state-of-the-art in terms of complex event detection under uncertainty. It includes a scheme for the computation of the probability of event detection for a given event sequence with missing timestamps, which can be seen as an alternative to the proposed method for probability computation described in Section 3.2.

To incorporate SASE+'s computation scheme into our context, we replaced the implementation of the ComputeProbability function in Alg. 2 with a re-implementation of the respective algorithm from [51]. Since the SASE+ engine works with a discrete time domain, in our experiments, we define an interval of discrete timestamps for each event e_t for which the timestamp is missing. This domain is symmetrical and defined based on a *timestamp domain size* parameter δ, *i.e.*, we consider



(a) Throughput vs. noise (b) Accuracy vs. noise (c) Accuracy vs. threshold

Fig. 8. Efficiency and effectiveness of our approach to probabilistic detection of interval relations.

the interval $[e_t - \delta, e_t + \delta]$ with $10 \leq \delta \leq 80$. Further details on the algorithm can be found in Appendix A.

4.3 Experimental Results

Efficiency. We evaluated the efficiency of our approach to probabilistic detection of interval relations as follows. For the benchmark datasets, we focused on the *intersection* query, since there is a relatively low number of segment intersections. For the synthetic dataset, we used the *k-sharing* query and report on the results obtained for the most challenging query selectivity when varying k from 1 to 12. That is, we ran the query with all values of k and selected the configuration that led to the lowest detection accuracy. As such, our results are obtained under a worst case assumption and show the lower bound of the results for *k-sharing*. We later explore the sensitivity of the results regarding the choice of parameter k in a separate experiment.

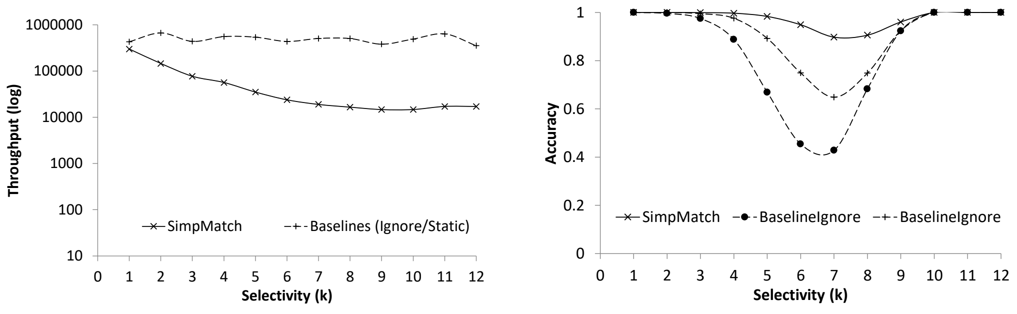
Fig. 8(a) shows the results in terms of throughput (log scale) when varying the noise level ($E \in [0\%, 40\%]$ in steps of 5%) with a detection threshold of $T = 0.5$. In the absence of noise, throughput is high for all datasets, reaching 2,000,000 events per second (e/s) for the benchmark dataset and 650,000 e/s for the synthetic dataset. Increasing the noise level reduces the throughput. For instance, if $E = 25\%$, we observe throughput values between 5,000 and 33,000 e/s, depending on the dataset.

In sum, for small and medium noise levels, we observe an overall good performance of the SimpMatch algorithm. However, throughput drops considerably as uncertainty increases, given the number of possible worlds that need to be considered by SimpMatch.

Effectiveness. We evaluated the effectiveness of our approach using the aforementioned datasets and queries. Fig. 8(b) shows the results in terms of accuracy when varying the noise level ($E \in [0\%, 40\%]$ in steps of 5%) and with $T = 0.5$. In general, accuracy is high, *e.g.*, around 0.92 at a noise level of 10%. As expected, increasing the noise level decreases accuracy, a trend which is stronger for the benchmark datasets. However, even under high noise levels, accuracy is around 0.7.

The detection threshold defines the probability based on which patterns are recognised. Using the *synthetic* dataset and averaging over all noise levels, Fig. 8(c) suggests that accuracy peaks at $T = 0.5$ when varying the detection threshold in steps of 0.1. For this threshold, the approach averages an accuracy between 84% to 94%. We conclude that the SimpMatch algorithm achieves high accuracy that degrades gracefully with increasing noise levels.

Sensitivity to query selectivity. To assess the sensitivity of our approach to the query selectivity, Fig. 9 reports on experiments with the *synthetic* dataset and *k-sharing* query ($E = 20\%$). As illustrated in Table 2, increasing parameter k step-wise changes the selectivity from 100% to 0%. Fig. 9(a) shows



(a) Throughput vs. query selectivity (b) Accuracy vs. query selectivity

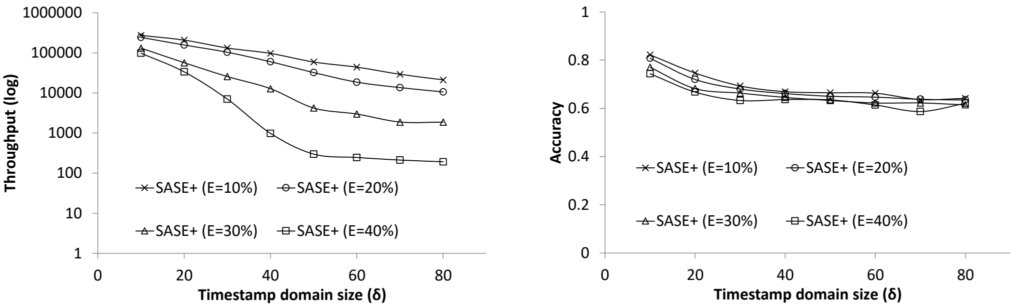
Fig. 9. Analysis of the sensitivity of our approach to the query selectivity.

that lowering the selectivity leads to a reduction of the throughput achieved with SimpMatch. This is expected since there are fewer opportunities to prune the possible worlds tree (which occurs when more than k intersecting intervals have already been predicted for a possible world). As a result, the throughput drops by around an order of magnitude in comparison to the baseline techniques that do not employ any reasoning (and hence, their throughput is not affected by query selectivity).

Yet, Fig. 9(b) shows that the baseline techniques perform much worse in terms of accuracy. Answering the query is most challenging for medium selectivities ($5 \leq k \leq 8$) and simply ignoring lost events (*BaselineIgnore*) leads to very low overall accuracy. While a static reconstruction of some lost events (*BaselineStatic*), our probabilistic reasoning approach leads to the best results, by far. Note, again, that all results for the k-sharing query in the other experiments are shown for $k = 7$, i.e., the worst case in terms of query selectivity.

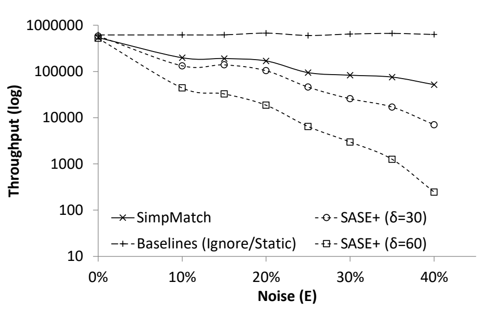
Comparison with baseline techniques and the state-of-the-art. As detailed in Section 4.2, we also compared our approach to the reasoning technique of the SASE+ engine [51]. Again, we used the *synthetic* dataset and *k-sharing* query with the selectivity that corresponds to the worst case.

Using the reasoning technique of SASE+, we have to determine the timestamp domain size δ. Unlike our approach, SASE+ assumes a discrete timestamp domain. In a first step, we therefore evaluate the choice of δ on the efficiency and effectiveness of this approach in Fig. 10, when varying the considered noise level. It turns out that increasing δ, lowers the throughput significantly, while

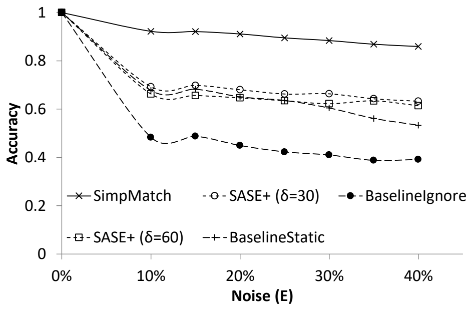


(a) Throughput sensitivity (b) Accuracy sensitivity

Fig. 10. Analysis of the sensitivity of the reasoning approach based on SASE+.



(a) Throughput vs. noise



(b) Accuracy vs. noise

Fig. 11. Comprehensive comparison of the efficiency and effectiveness.

increased noise levels amplify this trend. The reasoning being the growth of the search space, which is exponential in the size of the temporal distribution employed for each of the lost events.

The accuracy, in turn, is slightly higher for small timestamp domains, which stems from the fact that the uncertainty interval induced by δ is then close to the mean inter-arrival time of point events, which happens to be the best case for the reasoning. Apart from that special case, the accuracy is relatively stable for $\delta > 30$. Since the mean inter-arrival time of point events is not known in the general case, we consider the configurations of $\delta = 30$ and $\delta = 60$ as representative choices in the remaining experiments.

Next, we compare all approaches in terms of their efficiency when exploring fine-granular changes in the noise level ($E \in [0\%, 40\%]$). According Fig. 11(a), we observe that SimpMatch achieves higher throughput than SASE+, with the difference becoming larger with increased noise levels and increased sizes of the timestamp domain. In the extreme case ($\delta = 80$, $E = 40\%$), SimpMatch outperforms SASE+ by two orders of magnitude in throughput. This highlights the importance of reasoning methods that, as SimpMatch, are independent of the granularity of the temporal model. However, the results also illustrate the overhead of probabilistic reasoning, as the baseline technique achieve a throughput that is still an order of magnitude higher.

Turning to the effectiveness of the methods, Fig. 11(b) shows the accuracy under various noise levels ($E \in [0\%, 40\%]$). For a given noise level, SimpMatch achieves consistently the highest accuracy. For example, for $E = 10\%$, the SASE+ routine achieves an accuracy value of 0.69 for $\delta = 30$ (0.66 for $\delta = 60$), the baseline techniques yield accuracy values of 0.48 and 0.68, respectively, whereas SimpMatch yields an accuracy value of 0.91. We conclude that SimpMatch is more effective for the reasoning tasks addressed in this paper, consistently leading to higher detection accuracy.

5 SMART CITY APPLICATION WITH REAL-WORLD DATA

To further illustrate the benefits of the proposed model and algorithm in the context of IoT infrastructures, this section presents a Smart City application, employed as a solution in the VaVeL European project [3]. The application was developed for real-world GPS data streams, emitted by buses in the city of Dublin. Every bus transmits its timestamped location at a 20-seconds resolution, which also includes information on the nearest bus stop. The readings are processed in real-time, thus allowing for online analysis, such as anomaly detection and travelling time prediction [22, 32].

Our model enables us to effectively use the recorded data to answer complex queries, without the need to assume ‘continuous tracking’ (no lost events) [32], or a bias by imputation of timestamps of lost events [22]. Analysis of the transportation network in Dublin is based on a *stop congestion* query, testing the co-location of buses at specific stops. Stop congestions causes road disturbances, since buses cannot properly enter a stop occupied by other buses. Answering this query enables operational improvement and optimisation of traffic flow, *e.g.*, by adapting dispatching policies.

Below, we demonstrate the use of SimpMatch in answering stop congestion queries. To this end, we first present the available data and its casting into our model of segmented interval events. Subsequently, we show performance results when applying our approach to answer stop congestion queries on real-world GPS bus data.

The Dublin bus data. Buses follow a *journey* that is modelled as a segmented interval event $\epsilon = \langle \varepsilon_1, \ldots, \varepsilon_n \rangle$, with $\varepsilon_1, \varepsilon_n$ being the start and end events, respectively. Intermediate events ε_i, $2 \leq i \leq n-1$, are associated with $n-2$ stops visited by the bus during a single journey. The number of stops is known in advance. Then, e_2^i, $i < n$ corresponds to a suspend event, representing a bus halting at stop i, while e_1^i, $i > 1$ is a resume event, indicating when a bus leaves a stop.

As explained above, Dublin buses do not emit events when entering or leaving a bus stop, but rather create a continuous stream of spatial-temporal events. As such, the recorded data is a (flat) sequence of events $\xi = \langle \xi_1, \ldots, \xi_m \rangle$, such that

- $\xi_i.l$ is the GPS location,
- $\xi_i.t$ is a timestamp, inducing a total order on ξ, and,
- $\xi_i.s$ is the closest bus stop associated with the event.

As the available events are not directly linked to the intervals in ϵ, in essence, all suspend and resume events in ϵ are missing. Thus, the entire journey can be seen as an uncertainty interval, with a $2 \times (n-2)$ lost events. However, the attribute indicating the closest bus stop ($\xi_i.s$) in the recordings along with the bus schedule enable us to set boundaries on the uncertainty window of lost events.

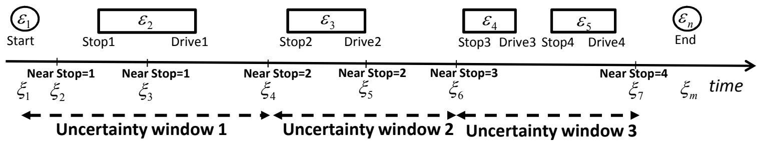


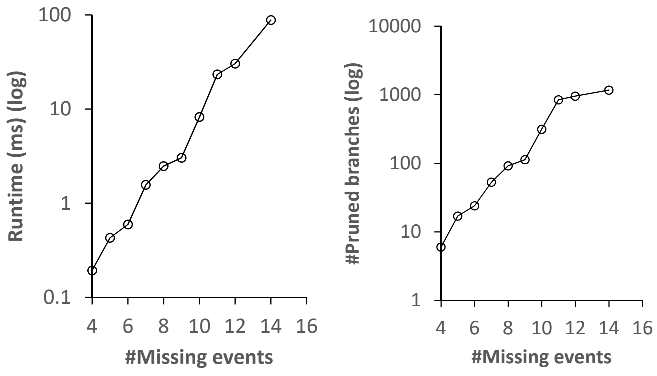
Fig. 12. Transforming the data into segmented interval events.

We illustrate the application model in Fig. 12, where rectangles represent true, unknown intervals of a bus being at a stop. Recorded events are marked on the timeline below.

A single journey is partitioned into W time windows, with every window having $2 \times K_w \leq 2 \times (n-2)$, $w \in \{1, \ldots, W\}$ lost events (dashed lines in Fig. 12). Specifically, we scan pairs of consecutive events, ξ_i and ξ_{i+1}, for which one of the following cases holds: (1) $\xi_i.s \neq \xi_{i+1}.s$, the closest bus stop changed; or (2) $\xi_i.s = \xi_{i+1}.s$, the stop remained the same. The first case alternately opens and closes an uncertainty window, whereas the second case extends the current window. For each window determined by this procedure, we count the number of missing stops based on the bus schedule. This procedure is applied for every journey in the data.

Application results. Equipped with the uncertainty windows, we use the SimpMatch algorithm to answer the stop congestion query (overlap of segments at the same stop) for every pair of journeys. Here, we summarise the results in terms of computational performance. The algorithm was executed (Intel Core i7, 12GB RAM, SSD HD) for real-world data from September 2014, which comprises over 4,000 bus journeys.

Fig. 13(a) shows the average runtime observed for SimpMatch as a function of the number of lost events per time window. Since in every window, at least two events are missing per journey, SimpMatch has to cope with at least four lost events when reasoning over two journeys. However, we also observe much larger degrees of lost events, at the worst case reaching 14 lost events



(a) Runtime vs. # missing (b) Pruning vs. # missing

Fig. 13. Runtime of SimpMatch for the Smart City application.

per window. Nevertheless, the comparatively low absolute runtimes illustrate the feasibility of our approach for this Smart City application. As shown in Fig. 13(b), this is achieved by massive pruning of the possible worlds tree. An increased number of missing events implies more pruning opportunities, which leads to an increased relative runtime improvement.

6 RELATED WORK

Complex Event Processing. A variety of systems for Complex Event Processing (CEP) that filter, correlate, aggregate, and transform events have been presented over the decade, including SQL-TS [37], Streams [10], Cayuga [18], SASE(+) [5, 49], CEDR [14], ZStream [29], and XSeq [30]. Zhang *et al.* [52] pointed out that most of these systems employ an event query language built of a few core concepts, such as sequencing and negation. However, these concepts are typically defined for point events, not interval events as addressed in this work. Detection of interval-based querying of event streams has been addressed by Li *et al.* [26], who presented *ISEQ*, a temporal operator based on Allen's algebra. Our model can be viewed as an extension of *ISEQ* in terms of expressiveness and the added support for reasoning over events with missing timestamps.

Uncertain event detection. The roots of uncertainty in event-based systems has been discussed by Etzion and Niblett [21]. A recent survey of models and approaches for uncertain management has been presented by Alevizos et al. [7].

The possible world semantics for event detection adopted in our work goes back to [47], which provides a probabilistic framework for reasoning on events. Yet, this framework is grounded in causal dependencies between events, lacking a query models as proposed in this paper based on interval-relations. Closest to our work is the uncertainty management implemented in the SASE+ engine [5, 49, 51]. SASE+ handles missing timestamps of events based on a discrete distribution of timestamps [51]. Our work, in contrast, is based on a continuous time distribution for the duration between point events of the same segmented interval event. Consequently, unlike the SASE+ engine, our SimpMatch algorithm is independent of the time distribution and only depends linearly on the length of lost event sequences. Another difference is the use of segmented interval events in our approach, so that correlations of point events are exploited to reduce the number of possible orderings and prune illegal orderings during probability computation.

Although feasible, our experimental results on the comparison with the reasoning technique of the SASE+ engine indicate that an approach that traces back the problem of probabilistic interval relation detection to sequence patterns comes with reduced accuracy and a drastic reduction of throughput by up to two orders of magnitude. However, we note that our work is orthogonal to the SASE+ approach, since reasoning in SASE+ is tailored towards processing of events that can satisfy multiple patterns simultaneously, which is not part of our model. Hence, it may be beneficial to include the computation approach proposed in this work in the SASE+ engine.

Methods for top-k pattern matching over probabilistic data streams using sliding windows were proposed by Li *et al.* [27] and Sugiura and Ishikawa [42, 43]. Interval-based reasoning over uncertain events was discussed in the context of pattern recognition [39, 40] based on the Event Calculus. To cope with uncertainty in Event Calculus, Skarlatidis et al. [40] use Markov Logic Networks (MLN) [19], combining first-order logic with probabilistic graphical models. Filipou at al. [39] rely on Problog, a probabilistic logic programming language. These approaches focus on timepoint probability computation and assume discrete time. Our approach, in turn, computes interval probabilities and its complexity is independent of the granularity of the temporal model.

Our model defines that the existence of lost events is known, so that uncertainty is related to their timestamp and, thus, ordering. As such, one may also consider to first impute or repair timestamps before answering a query, *e.g.*, based on domain constraints [41], or reconstruct their ordering based on contextual information [17]. However, if the goal is query answering, such imputation creates unnecessary overhead, since, as shown in this paper, queries may be answered probabilistically in a direct manner.

Finally, assuming that uncertainty (*i.e.*, lost events) are rare, some of the ideas developed for anomaly detection over event streams [53] or, more general, data streams [15] are related. While the adopted techniques are similar to those for uncertain event processing, the actually addressed problem is different, since abnormal events are always characterised relative to some context defined by other events (whereas the definition of a lost event is not context dependent).

Optimisations of event detection. Uncertain event detection typically comes with high processing costs and our evaluation exemplified the impact of increased noise levels on run-time performance. Various angles have been taken for the optimisation of event detection, among them query rewriting [28, 38, 48], re-use of intermediate query results [25, 34, 50], load shedding [23], and query distribution [6]. However, these techniques are not directly applicable to probabilistic event detection, and therefore we integrate optimisations in terms of pruning and caching directly into the SimpMatch algorithm.

7 CONCLUSIONS

In this paper, we introduced SimpMatch, an algorithm for answering interval-based queries over lossy event streams. We introduced segmented interval events to model complex patterns and proposed a query model based on Allen's algebra. Using order statistics, we further introduced an efficient method for computing the probability of ordering a set of lost events, *i.e.*, events that are known to exist, but for which no timestamp information is available. The computation is linear in the number of lost events and independent of the probability distribution size. While query answering is exponential in the number of lost events, using order statistics, we showed how to optimise the algorithm by pruning and caching.

Our evaluation shows that our method is more efficient than the state-of-the-art as the number of lost events grows, while yielding more accurate results. We further demonstrated the benefits of the proposed model and algorithm for IoT scenarios through a Smart City application in the city of Dublin. In future work, we plan to adapt our reasoning techniques for further types of queries, *e.g.*, those correlating events based on trends over their attribute values.

ACKNOWLEDGEMENTS

This project received funding from the EU Horizon 2020 Programme, grant agreement 688380, the German Research Foundation (DFG), grant agreement 246594964, and the Alexander von Humboldt Foundation. We thank Yossi Dahari for support in the experiments.

REFERENCES

[1] [n. d.]. CAVIAR project, http://homepages.inf.ed.ac.uk/rbf/CAVIAR/.

[2] [n. d.]. Dublinked, http://dublinked.ie/.

[3] [n. d.]. VaVeL European project, http://www.vavel-project.eu/.

[4] 2007. *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*. www.cidrdb.org.

[5] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. 2008. Efficient pattern matching over event streams, See [46], 147–160.

[6] Mert Akdere, Ugur Çetintemel, and Nesime Tatbul. 2008. Plan-based complex event detection across distributed sources. *PVLDB* 1, 1 (2008), 66–77.

[7] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. 2017. Probabilistic Complex Event Recognition: A Survey. *ACM Comput. Surv.* 50, 5 (2017), 71:1–71:31. https://doi.org/10.1145/3117809

[8] James F. Allen and Patrick J. Hayes. 1985. A Common-Sense Theory of Time. In *IJCAI*, Aravind K. Joshi (Ed.). Morgan Kaufmann, 528–531.

[9] Thomas A. Alspaugh. 2005. *Software support for calculations in Allen's Interval Algebra.* Technical Report.

[10] Arvind Arasu, Shivnath Babu, and Jennifer Widom. 2006. The CQL continuous query language: semantic foundations and query execution. *VLDB J.* 15, 2 (2006), 121–142.

[11] Arvind Arasu, Mitch Cherniack, Eduardo F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. 2004. Linear Road: A Stream Data Management Benchmark. In *VLDB*, Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer (Eds.). Morgan Kaufmann, 480–491.

[12] Alexander Artikis, Marek Sergot, and Georgios Paliouras. 2010. A logic programming approach to activity recognition. In *Proceedings of the 2nd ACM international workshop on Events in multimedia (EiMM '10)*. ACM, New York, NY, USA, 3–8. https://doi.org/10.1145/1877937.1877941

[13] Alexander Artikis, Matthias Weidlich, François Schnitzler, Ioannis Boutsis, Thomas Liebig, Nico Piatkowski, Christian Bockermann, Katharina Morik, Vana Kalogeraki, Jakub Marecek, Avigdor Gal, Shie Mannor, Dimitrios Gunopulos, and Dermot Kinane. 2014. Heterogeneous Stream Processing and Crowdsourcing for Urban Traffic Management. In *EDBT*, Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy (Eds.). OpenProceedings.org, 712–723.

[14] Roger S. Barga, Jonathan Goldstein, Mohamed H. Ali, and Mingsheng Hong. 2007. Consistent Streaming Through Time: A Vision for Event Stream Processing, See [4], 363–374.

[15] Lei Cao, Jiayuan Wang, and Elke A. Rundensteiner. 2016. Sharing-Aware Outlier Analytics over High-Volume Data Streams, See [31], 527–540. https://doi.org/10.1145/2882903.2882920

[16] Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis (Eds.). 2006. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006.* ACM.

[17] Raffaele Conforti, Marcello La Rosa, and A ter Hofstede. 2018. Timestamp Repair for Business Process Event Logs. (2018). Preprint available at https://minerva-access.unimelb.edu.au/handle/11343/209011.

[18] Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. 2007. Cayuga: A General Purpose Event Monitoring System, See [4], 412–422.

[19] Pedro Domingos and Daniel Lowd. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence* (1st ed.). Morgan and Claypool Publishers.

[20] R. Durrett. 1998. *Essentials of Stochastic Processes.* Springer-Verlag, Chapter 2, 126–127.

[21] Opher Etzion and Peter Niblett. 2010. *Event Processing in Action.* Manning Publications Company. I–XXIV, 1–360 pages.

[22] Avigdor Gal, Avishai Mandelbaum, François Schnitzler, Arik Senderovich, and Matthias Weidlich. 2017. Traveling time prediction in scheduled transportation with journey segments. *Information Systems* 64 (2017), 266–280.

[23] Yeye He, Siddharth Barman, and Jeffrey F. Naughton. 2014. On Load Shedding in Complex Event Processing. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy (Eds.). OpenProceedings.org, 213–224. https://doi.org/10.5441/002/icdt.2014.23

[24] Pekka Kaarela, Mika Varjola, Lucas P.J.J. Noldus, and Alexander Artikis. 2011. PRONTO: support for real-time decision making. In *Proceedings of the 5th ACM international conference on Distributed event-based system (DEBS '11).* ACM,

New York, NY, USA, 11–14. https://doi.org/10.1145/2002259.2002262

[25] Sailesh Krishnamurthy, Chung Wu, and Michael J. Franklin. 2006. On-the-fly sharing for streamed aggregation, See [16], 623–634.

[26] Ming Li, Murali Mani, Elke A. Rundensteiner, and Tao Lin. 2011. Complex event pattern detection over streams with interval-based temporal semantics. In *DEBS*, David M. Eyers, Opher Etzion, Avigdor Gal, Stanley B. Zdonik, and Paul Vincent (Eds.). ACM, 291–302.

[27] Zheng Li, Tingjian Ge, and Cindy X. Chen. 2013. ϵ-Matching: event processing over noisy sequences in real time. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias (Eds.). ACM, 601–612. https://doi.org/10.1145/2463676.2463715

[28] Mo Liu, Elke A. Rundensteiner, Daniel J. Dougherty, Chetan Gupta, Song Wang, Ismail Ari, and Abhay Mehta. 2011. High-performance nested CEP query processing over event streams. In *ICDE*, Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan (Eds.). IEEE Computer Society, 123–134.

[29] Yuan Mei and Samuel Madden. 2009. ZStream: a cost-based query processor for adaptively detecting composite events. In *SIGMOD Conference*, Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul (Eds.). ACM, 193–206.

[30] Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. 2012. High-performance complex event processing over XML streams. In *SIGMOD Conference*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 253–264.

[31] Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). 2016. *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. ACM. https://doi.org/10.1145/2882903

[32] Nikolaos Panagiotou, Nikolas Zygouras, Ioannis Katakis, Dimitrios Gunopulos, Nikos Zacheilas, Ioannis Boutsis, Vana Kalogeraki, Stephen Lynch, and Brendan O'Brien. 2016. Intelligent urban data monitoring for smart cities. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 177–192.

[33] Kostas Patroumpas, Elias Alevizos, Alexander Artikis, Marios Vodas, Nikos Pelekis, and Yannis Theodoridis. 2017. Online event recognition from moving vessel trajectories. *GeoInformatica* 21, 2 (2017), 389–427. https://doi.org/10.1007/s10707-016-0266-x

[34] Medhabi Ray, Chuan Lei, and Elke A. Rundensteiner. 2016. Scalable Pattern Sharing on Event Streams, See [31], 495–510. https://doi.org/10.1145/2882903.2882947

[35] Christopher Ré, Julie Letchner, Magdalena Balazinska, and Dan Suciu. 2008. Event queries on correlated probabilistic streams, See [46], 715–728.

[36] S. Ross. 1997. *A First Course in Probability*. Prentice Hall.

[37] Reza Sadri, Carlo Zaniolo, Amir M. Zarkesh, and Jafar Adibi. 2004. Expressing and optimizing sequence queries in database systems. *ACM Trans. Database Syst.* 29, 2 (2004), 282–318.

[38] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter R. Pietzuch. 2009. Distributed complex event processing with query rewriting. In *DEBS*, Aniruddha S. Gokhale and Douglas C. Schmidt (Eds.). ACM.

[39] Anastasios Skarlatidis, Alexander Artikis, Jason Filipou, and Georgios Paliouras. 2015. A probabilistic logic programming event calculus. *TPLP* 15, 2 (2015), 213–245. https://doi.org/10.1017/S1471068413000690

[40] Anastasios Skarlatidis, Georgios Paliouras, Alexander Artikis, and George A. Vouros. 2015. Probabilistic Event Calculus for Event Recognition. *ACM Transactions on Computational Logic* 16, 2 (2015).

[41] Shaoxu Song, Yue Cao, and Jianmin Wang. 2016. Cleaning Timestamps with Temporal Constraints. *PVLDB* 9, 10 (2016), 708–719. https://doi.org/10.14778/2977797.2977798

[42] Kento Sugiura and Yoshiharu Ishikawa. 2017. Top-k Pattern Matching Using an Information-Theoretic Criterion over Probabilistic Data Streams. In *Web and Big Data - First International Joint Conference, APWeb-WAIM 2017, Beijing, China, July 7-9, 2017, Proceedings, Part I (Lecture Notes in Computer Science)*, Lei Chen, Christian S. Jensen, Cyrus Shahabi, Xiaochun Yang, and Xiang Lian (Eds.), Vol. 10366. Springer, 511–526. https://doi.org/10.1007/978-3-319-63579-8_39

[43] Kento Sugiura and Yoshiharu Ishikawa. 2019. Regular Expression Pattern Matching with Sliding Windows cover Probabilistic Event Streams. In *IEEE International Conference on Big Data and Smart Computing, BigComp 2019, Kyoto, Japan, February 27 - March 2, 2019*. IEEE, 1–8. https://doi.org/10.1109/BIGCOMP.2019.8679331

[44] David Toman. 1996. Point vs. Interval-based Query Languages for Temporal Databases. In *PODS*, Richard Hull (Ed.). ACM Press, 58–67.

[45] Wil M. P. van der Aalst. 2011. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer. I–XVI, 1–352 pages.

[46] Jason Tsong-Li Wang (Ed.). 2008. *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*. ACM.

[47] Segev Wasserkrug, Avigdor Gal, Opher Etzion, and Yulia Turchin. 2008. Complex event processing over uncertain data. In *Proceedings of the Second International Conference on Distributed Event-Based Systems, DEBS 2008, Rome, Italy, July 1-4, 2008 (ACM International Conference Proceeding Series)*, Roberto Baldoni (Ed.), Vol. 332. ACM, 253–264. https://doi.org/10.1145/1385989.1386022

[48] Matthias Weidlich, Holger Ziekow, Avigdor Gal, Jan Mendling, and Mathias Weske. 2014. Optimizing Event Pattern Matching Using Business Process Models. *IEEE Trans. Knowl. Data Eng.* 26, 11 (2014), 2759–2773. https://doi.org/10.1109/TKDE.2014.2302306

[49] Eugene Wu, Yanlei Diao, and Shariq Rizvi. 2006. High-performance complex event processing over streams, See [16], 407–418.

[50] Di Yang, Elke A. Rundensteiner, and Matthew O. Ward. 2012. Shared execution strategy for neighbor-based pattern mining requests over streaming windows. *ACM Trans. Database Syst.* 37, 1 (2012), 5.

[51] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2013. Recognizing patterns in streams with imprecise timestamps. *Inf. Syst.* 38, 8 (2013), 1187–1211.

[52] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2014. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD Conference*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 217–228.

[53] Haopeng Zhang, Yanlei Diao, and Alexandra Meliou. 2017. EXstream: Explaining Anomalies in Event Stream Monitoring. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017.*, Volker Markl, Salvatore Orlando, Bernhard Mitschang, Periklis Andritsos, Kai-Uwe Sattler, and Sebastian Breß (Eds.). OpenProceedings.org, 156–167. https://doi.org/10.5441/002/edbt.2017.15

A SASE+ REASONING TECHNIQUE

For a comparative evaluation, we replaced the ComputeProb function in Alg. 2 with the reasoning technique of the SASE+ engine, i.e., Alg. 5 in [51]. While the SASE+ algorithm is triggered each time an event with a missing timestamp is received, our TraceBack and ComputeProb functions (Alg. 2) evaluate complete lost event sequences. Thus, we adapted the algorithm by adding a loop to traverse several events at once, as detailed in Alg. 3. The algorithm assumes that each event has a possible timestamp domain, denoted by $pos(e)$ for point event e.

The idea of Alg. 3 is to enumerate all valid orderings and sum up the respective probabilities. The algorithm starts by placing each valid timestamp of the first event as a start of a path. Then, in a nested loop, each valid timestamp of the current event is set to extend a partial path of the preceding event. A timestamp is considered valid, if it falls into the bounds defined by the observed events of the current window.

Algorithm 3: COMPUTE function based on Alg. 5 in [51]

```
input  : lost event sequence $\varepsilon = \langle e_k, e_{k+1}, \ldots, e_l \rangle$,
         range $pos(e_i)$ of possible timestamps of $e_i$, for $i \in \{k, \ldots, l\}$
output : probability $pr$ for occurrence of $\varepsilon$

1  $pr := 0$;
2  $lost := l - k$;                                        /* number of lost events */
3  $dens := 1/\sum_{k \leq i \leq l} |pos(e_i)|$;          /* probability per value */
4  for $t \in pos(e_{k+1})$ do
5      if $t >= e_k.t \wedge t <= e_l.t$ then
6          Create path for $t$ starting at $\min(pos(e_{k+1}))$;
7          $pr := pr + dens$;
8  if $lost = 1$ then return $pr$ ;
9  $pr := 0$;
10 for $i := 0, \ldots, lost - 2$ do
11     for $t_i \in pos(e_{k+i})$ do
12         if $t_i >= e_k.t \wedge t_i <= e_l.t$ then
13             for $t_j \in pos(e_{k+i+1})$ do
14                 if $t_j >= t_i \wedge t_j <= e_l.t$ then
15                     for path $path_i$ ending at $t_i$ do
16                         Create $path_j := (path_i, t_j)$;
17                         if $i = (lost - 1)$ then
18                             $pr := pr + dens^{lost}$;
19 return $pr$;
```