Utility-Maximizing Event Stream Suppression [*]

Di Wang[◇], Yeye He[†], Elke Rundensteiner[◇], Jeffrey F. Naughton[‡]
[◇]Department of Computer Science, Worcester Polytechnic Institute
[†,‡] Department of Computer Sciences, University of Wisconsin-Madison
wangdi@cs.wpi.edu, heyeye@cs.wisc.edu, rundenst@cs.wpi.edu, naughton@cs.wisc.edu

ABSTRACT

Complex Event Processing (CEP) has emerged as a technology for monitoring event streams in search of user specified event patterns. When a CEP system is deployed in sensitive environments the user may wish to mitigate leaks of private information while ensuring that useful nonsensitive patterns are still reported. In this paper we consider how to suppress events in a stream to reduce the disclosure of sensitive patterns while maximizing the detection of nonsensitive patterns. We first formally define the problem of utility-maximizing event suppression with privacy preferences, and analyze its computational hardness. We then design a suite of real-time solutions to solve this problem. Our first solution optimally solves the problem at the event-type level. The second solution, at the event-instance level, further optimizes the event-type level solution by exploiting runtime event distributions using advanced pattern match cardinality estimation techniques. Our user study and experimental evaluation over both real-world and synthetic event streams show that our algorithms are effective in maximizing utility yet still efficient enough to offer near real-time system responsiveness.

Categories and Subject Descriptors

H.2 [**Information Systems**]: Database Management

Keywords

Complex Event Processing, Utility, Privacy

1. INTRODUCTION

Complex Event Processing (CEP) has gained increasing popularity for real-time pattern matching over event streams, and has received significant attention from both the research community [5, 22, 29] and industry [1, 3]. However, to date, the privacy implications of CEP applications have been overlooked. The privacy problems we study in this paper were motivated by real-world problems we encountered while developing and deploying CEP solutions for hospital environments. While our privacy problem is abstracted from this specific application domain, our problem formulation is sufficiently generic that it can be applied to general CEP applications. For concreteness, we begin with our motivating example.

Motivating Applications. Consider a CEP-powered health care system called HyReminder [27]. HyReminder, currently deployed at University of Massachusetts Memorial Hospital, is a hospital infection control system that aims to track, monitor and remind health-care workers with respect to hygiene compliance. In the hospital, each doctor wears an RFID badge that can be read by sensors installed throughout the hospital. As doctors move around in the hospital, sensor readings are triggered, which are then abstracted as events and transmitted to a CEP engine. Using CEP query processing techniques, event patterns that reveal hygiene compliance or violations can then be detected and reported in accordance with US hygiene regulations [7].

As an example of such a pattern, a doctor who exits a patient room (represented by an `Exit-patient-room` event) should perform hand sanitization (indicated by a `Sanitize` event) within a short period of time. We represent such a hygiene compliance pattern using Q1 below. Similarly, a doctor should wash hands (a `Wash` event) before entering an office (an `Enter-psychiatrist-office` event) to prevent spreading contagious diseases, as expressed by Q2 below. These regulations are commonly known as "wash-in, wash-out" in hospital jargon [7].

Q1: `SEQ(Exit-patient-room, Sanitize)`
`WITHIN 2 min`
Q2: `SEQ(Wash, Enter-psychiatrist-office)`
`WITHIN 2 min`

While the benefit of such CEP applications is apparent, some event patterns may be sensitive in that they reveal an individual's private information. For example, an observation that a doctor leaves a patient's room and then immediately enters a psychiatrist's office might serve as an indication that this patient is experiencing psychiatric problems. This event sequence, when expressed as a "private" pattern, can be written as:

P1: `SEQ(Exit-patient-room, Enter-psychiatrist-office)`
`WITHIN 5 min`

The issue is that when HyReminder monitors doctors' behaviors (the intended use of this application), the events being reported may disclose private information about individual patients as a side-effect. We observe that the simplistic approach of not directly reporting private patterns does not prevent their disclosure — the occurrence of several hygiene compliance patterns that HyReminder does report may be used by an adversary to infer the existence of private pattern matches.

As an example, in Figure 1, there is a stream of four events asso-

[*]This work is supported in part by NSF grant IIS-1018443, NIGMS grant R01LM011028-02, and UMMS-WPI CCTS Collaborative grant.
† Author's current affiliation: Microsoft Research.

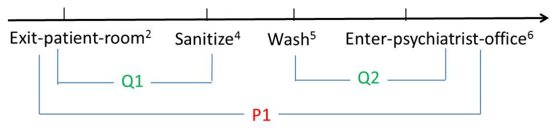


Figure 1: An adversarial attack using public pattern matches

ciated with the same doctor observed in HyReminder. Each event has a superscript indicating its timestamp in minutes. Over this particular event stream, there exists one match for Q1, (`Exit-patient-room`^{2}, `Sanitize`^{4}), and one match for Q2: (`Wash`^{5}, `Enter-psychiatrist-office`^{6}). Now by concatenating the matches of Q1 and Q2, an adversary can infer that a match for private pattern P1, namely (`Exit-patient-room`^{2}, `Enter-psychiatrist-office`^{6}) also exists. This example illustrates that the existence of private pattern matches can still be inadvertently revealed even when only legitimate queries are reported. It underlines the importance of taking private preferences into account when deploying CEP systems in sensitive environments like a hospital.

Of course, this hospital scenario is only one motivating example, that by no means cover the full spectrum of privacy threats with CEP systems – any time a CEP is deployed in a sensitive environment the possibility of a privacy breach exists. Furthermore, while one-off actions may be possible to mitigate privacy concerns within a particular application, our goal is broader — we wish to develop a general model for privacy and utility in CEP systems, and initial solutions to the problems that arise in that model.

At a high level, the problem we address can be described as follows. We consider two kinds of CEP patterns: those that should be detected and reported to data users (which we term *"public" patterns*), and those that users prefer not to reveal because of privacy concerns (which we term *"private" patterns*).

In this work we focus on the mechanism of *event suppression*, that is, deleting some events from the stream in order to "destroy" possible private pattern matches. For example, in Figure 1, dropping any event in the stream ensures that either a match of Q1 or Q2 would not be reported, thus preventing P1 from being inferred. In general, there exist numerous ways in which events could be suppressed. Note that dropping or keeping an event naturally represents a *trade-off* between reporting more useful public query matches and disclosing some undesirable private query matches. We thus provide a *utility-maximizing framework* that allows users to quantify their preferences by setting relative weights of public and private patterns, so that events can be suppressed in such a way that maximizes the utility of pattern matches that are preserved.

To the best of our knowledge, the work closest to ours is [15], which first identifies the privacy concern in CEP systems. However the work in [15] focuses on the theoretical hardness of a particular problem variant. Furthermore, no practical algorithm is designed nor empirical studies have been conducted. In this paper, we analyze possible variants of the more general version of the problem, develop practical algorithms that are useful in a real-time streaming environment, and experimentally evaluate our algorithms using both real-world and synthetic data.

Specifically, we make the following contributions in this work.

- We formally define the problem of utility-maximizing event suppression with privacy preferences. Our problem formulation enables users to quantify the relative importance of eliminating undesirable "private" matches versus producing useful "public" matches. Furthermore, we analyze problem variants (Section 3), and study their computational hardness (Section 4). We show that the problem in general is not only NP-hard, but also hard to approximate.
- Our second contribution is to develop a suite of real-time utility maximizing solutions. We first propose an approach based on linear programming that optimally solves the problem at the event-type level (which suppresses all events of the same type). For the computationally intractable instance-level problem (which suppresses individual events), we observe that our solution at the event-type level provides a useful basis for further optimization. Specifically, we introduce a *Hybrid solution* to tackle the instance-level problem by combining the solution at the event-type level with optimization heuristics based on run-time *pattern match cardinality estimation*. We further develop two techniques to address the subproblem of pattern match cardinality estimation, one based on event arrival rates and the other based on periodicity using the state-of-the-art periodicity mining algorithms [11, 20]. To the best of our knowledge, the problem of event stream cardinality estimation has not been explored. Just like in relational databases, we think our cardinality estimation may be useful for resource allocation and query optimization in event stream systems. (Section 5).
- To better understand the effectiveness and performance of our proposed solutions, we conducted a user study and performed extensive experiments on both real-world and synthetic event streams. We show that our Hybrid solutions preserve significantly more utility than alternative approaches, while still efficient enough to ensure real time system responsiveness. In addition, we demonstrate in both our user study and experiments that in at least one real world scenario, periodicity information is needed to produce accurate cardinality estimation. The advantage of periodicity-based estimation is then reaffirmed using synthetic streams (Section 6).

2. PRELIMINARIES

2.1 Event Data Model

The input to a CEP system is a potentially infinite **event stream** composed of **event instances**. Each event instance, denoted by a lowercase e_i, corresponds to an instantaneous and atomic occurrence of interest [29]. Each event instance e_i is associated with a timestamp from a discrete time domain, denoted by $e_i.ts$. Event instances are conceptually grouped into **event types**. Let Σ be the set of all possible event types. Then each event type, denoted by an uppercase $E_i \in \Sigma$, is distinguished by its event type name (e.g., `Exit-patient-room`). Each event type has associated attributes as defined by the schema of the event type.

2.2 Event Query Model

In this work, we focus on the core CEP query functionality, the `SEQ` query operator [22, 29]. A SEQ query looks for a sequence of events in a specified temporal order within a given time window. A **SEQ pattern query** Q is of the form $Q = SEQ\ (E_1, E_2, ...\ E_n)$, where $E_k \in \Sigma$ are event types. For each query Q there is a specified time-window $Window(Q) \in \mathbb{R}^+$ over which Q will be evaluated. While multiple pattern match semantics exist (see [5] for a classification), in this work we use *skip till any match* semantic, defined as follows. A **pattern match** of Q over an event stream S is produced if there exists a temporally ordered subsequence $S' = (e_1, e_2, ...\ e_n)$, that is $e_i.ts < e_j.ts$ for all $1 \leq i < j \leq n$, such that for all $k \in [1, n]$, e_k is of type E_k, and $e_n.ts - e_1.ts \leq Window(Q)$. We say the sequence S' is a match of Q. As is common in current CEP engines [5, 22, 29], we assume the output of a pattern match is the concatenation of participating event instances. In this paper, we focus on queries with positive event types, while the problem of supporting CEP queries with negation is an interesting area for future work.

Individual Match vs. Aggregate Statistics. Many CEP applications are interested in individual query matches. For example, in our hospital setting, it is important to know which doctor enters which ICU after exiting a potentially-contagious patient's room without following sanitization procedures. There are also scenarios where aggregate statistics of CEP outputs are useful. For instance, the hospital leadership team is interested in knowing the aggregate hygiene-compliance-ratio in the hospital by month over the last 12 months, which is also a real output produced by our HyReminder.

While both of these scenarios are important, in this paper we focus on mitigating privacy risks when individual CEP matches are reported, which to our knowledge has not been systematically studied. Ensuring privacy for aggregate CEP statistics is also important, although for that problem existing techniques such as differential-privacy may be appropriate. The inapplicability of differential-privacy to reporting individual matches is discussed in Related Work (Section 7).

2.3 The Complication: Private Patterns

The notion of **private query patterns** sets our problem apart from the conventional CEP literature. In terms of syntax, private query patterns are just like SEQ queries, i.e., they consist of a sequence of events and an associated time window. The fundamental difference, however, is that while as many SEQ pattern matches as possible should be reported, private pattern matches would better be suppressed. In the remainder of this work we will refer to these two types of patterns as **private query patterns** and **public query patterns**, respectively. Q1 and Q2 in the motivating application, for example, are public query patterns, while P1 is a private query pattern. We denote the set of public query patterns by $\mathcal{Q}$ and the set of private query patterns by $\mathcal{P}$.

2.3.1 Suppressing Private Pattern Matches

A natural way to suppress a private pattern match is to suppress events that participate in the match. For example, in Figure 1, dropping any event in the stream ensures that one match of Q1 or Q2 would not be reported, thus preventing P1 from being inferred.

One important advantage of using event suppression is that the output is a subset of the original stream, so it remains an event stream with the same set of event-types as the original stream. Thus any stream-based application that can process the original stream can process the filtered stream without modification.

In addition, event suppression ensures the desirable *query-result subset property* when CEP queries with positive events are used. Namely we only report a *subset* of the original CEP query matches, which ensures that no spurious matches that do not exist in reality will ever be produced. This is desirable because the opposite is troubling — it would be disturbing if a hospital hygiene compliance system reported false hygiene compliance/violations because they were generated by the privacy protection system.

While we focus on event suppression in this work, other possible approaches also exist. In the classical database anonymization literature, a frequently used technique is *generalization* [25, 21], in which a specific value is "generalized" so that its presence reveals less information. Similarly one could adopt "event generalization" for our problem by generalizing detailed events into generic events. Although event generalization may mitigate some privacy concerns, it can cause trouble in producing meaningful query matches. For example, in Figure 1, if we only observe a generalized `Exit-room` event instead of the `Exit-patient-room`, we are not sure if matches of Q1 can be produced. Even if a probabilistic interpretation is acceptable, query results would violate the *subset property* that event suppression can offer.

Another possible data manipulation approach is to add noise to the timestamps of events. This approach is also problematic, because it risks introducing spurious query pattern matches, thus violating the *subset property* outlined above.

Since event suppression is a simple yet effective mechanism for addressing privacy preference in the context of CEP, in this paper we will focus on using event suppression. Alternative approaches are interesting directions for future work.

3. PROBLEM STATEMENT & TAXONOMY

In this section, we first formally define the problem of utility-maximizing stream suppression with privacy preferences. We further propose a problem taxonomy to better understand its variants.

3.1 Problem statement

Motivation of problem formulation. Since there are multiple ways in which events could be suppressed to preserve privacy, the question arises is which events should be dropped over others. An intuitive answer is to suppress events such that more "important" public pattern matches are kept. It practice, some pattern matches are naturally more important than others. For example, in the hospital where HyReminder is deployed, another public pattern, henceforth referred to as Q3, may be needed to produce a query match if a doctor exits a highly-contagious patient room, does not sanitize his hands, and immediately enters the ICU. A match for Q3 represents a grave violation of hygiene regulations, and is considered more "useful" than Q1 and Q2 by the user.

In order to capture this intuition, we define a positive **utility weight** $w(Q_i)$ for each public query Q_i, that quantifies the usefulness of reporting one match for Q_i. Q3 mentioned above, for example, should be assigned a higher utility weight than a match for query Q1 or Q2. Assigning appropriate weights to different queries in real-world applications requires domain expertise. In this work we assume that the utility weights for queries are provided by domain experts as part of the input to the system.

We then define *utility gain* as a weighted sum of all public pattern matches as below.

DEFINITION 1. *Let $w(Q_i) \in \mathbb{R}^+$ be the utility gain weight of public query $Q_i \in \mathcal{Q}$, let $\mathcal{C}(Q_i, S)$ be the number of matches for Q_i over stream S. The* ***utility gain*** *generated for query Q_i is:*

$$U(Q_i, S) = w(Q_i) \cdot \mathcal{C}(Q_i, S) \tag{1}$$

The utility gain generated over the entire query set $\mathcal{Q}$ on S is:

$$\mathcal{U}_{\mathcal{Q}} = \sum_{Q_i \in \mathcal{Q}} U(Q_i, S) \tag{2}$$

On the other hand, to model the fact that private pattern matches are undesirable, each such match is assigned a *negative* utility value, or a **utility penalty weight**. The resulting aggregate *utility loss* is the side-effect caused by private patterns. We then formally define utility loss as follows.

DEFINITION 2. *Let $w(P_j) \in \mathbb{R}^-$ be the utility penalty weight of private pattern $P_j \in \mathcal{P}$, and $\mathcal{C}(P_j, S)$ be the number of matches for P_j over event stream S. The* ***utility penalty*** *or* ***utility loss*** *associated with P_j is:*

$$U(P_j, S) = w(P_j) \cdot \mathcal{C}(P_j, S) \tag{3}$$

Since our goal is to balance the need to maximize utility gain and minimize utility loss, our objective function is simply defined as the sum of the two.

DEFINITION 3. *The overall utility generated by $\mathcal{Q} = \{Q_i\}$ and $\mathcal{P} = \{P_j\}$ on S is:*

$$\mathcal{U}_{\mathcal{Q}+\mathcal{P}} = \sum_{Q_i \in \mathcal{Q}} U(Q_i, S) + \sum_{P_j \in \mathcal{P}} U(P_j, S). \tag{4}$$

This objective function is an instance of *aggregate objective function* that is commonly used in the multi-objective optimization literature [24].

With these, we formally define the utility-maximizing stream suppression problem as follows.

PROBLEM 1. *Given an event stream S, a set of public queries $\mathcal{Q}$, a set of private queries $\mathcal{P}$, and a utility weight function $w(\cdot) \in \mathbb{R}$, find a subset S' of S such that the total utility $\mathcal{U}_{\mathcal{Q}+\mathcal{P}}$ is maximized for the chosen S' over all possible subsets of S.*

3.2 A Problem Taxonomy

3.2.1 Hard-constraint vs. Soft-constraint

In certain scenarios, users may demand that the adversary cannot infer even a single private pattern match. In other words, the constraints imposed by the private patterns are "hard", assuring no private pattern match is ever disclosed. We term such private pattern a **hard-constraint**. If every private pattern is a hard-constraint, the resulting problem is a hard-constraint utility maximization problem. In our formulation defined in Definition 2, utility penalty weights for private patterns can be set to *negative infinity* to enforce the hard-constraint.

Although the hard-constraint privacy is needed in some environments, there are cases in which reporting a public query match is of such paramount importance that it may override certain privacy concerns. As a concrete example, consider the public pattern Q3 in HyReminder discussed before, where a doctor enters an ICU without proper sanitization, which corresponds to a serious hygiene violation with dire consequences. On the other hand, there may also exist private patterns representing fairly innocuous privacy violations (e.g., a patient has his blood and urine tested). In such a scenario, it may be desirable to report the public pattern match Q3, even at the potential cost of leaking not-so-harmful private information.

Since such trade-offs are best decided using domain expertise, we want a problem formulation flexible enough to allow users to specify queries with overriding utilities that can trump privacy concerns if necessary. In the **soft-constraints** variant, not all private patterns are strictly prohibited. Instead, each such match is penalized by a utility loss, as defined in our utility framework in Definition 3. This allows *flexible trade-offs* between reporting useful public pattern matches and disclosing undesirable private pattern matches. Furthermore, it can degenerate into the Hard-Constraint model by using negative infinity query weights, and thus represents a more general problem formulation.

Choosing the Hard-constraint vs. the Soft-constraint model. Our hospital solution was developed in consultation with personnel in the hospital. Initially we favored the Hard-constraint problem formulation, for the simple reason that it offers precise privacy guarantees: anything that is labeled private will not appear in the result set.

While conceptually appealing, we later noticed that with this Hard-constraint approach, "utility" was significantly reduced. In particular, occasionally very important matches to public patterns, which must be reported in the hospital setting, were instead suppressed due to minor privacy concerns. For example, a query match that reports a grave hygiene violation (e.g., Q3 discussed earlier) must trigger alerts to the doctor as well as to the lead nurse on duty. However, in our initial experiments using the Hard-constraint approach, some such important matches were suppressed.

We were told this was unsatisfactory, because while it is important to mitigate privacy risks, ensuring hygiene compliance and controlling serious in-hospital infections are sometimes given higher priority. The system, after all, is used *internally* by medical workers; it is not an externally-facing system deployed on the Internet. We believe that this is a key distinction that makes our problem different from traditional privacy research.

Σ	set of all event types
$\mathcal{Q}$	set of all public patterns
$\mathcal{P}$	set of all private patterns
$w(Q_j)$	utility weight of pattern Q_j
λ_i	arrival rate of events of type E_i
$N_T(Q_j)$	expected number of matches of Q_j in a time span T

Table 1: Summary of symbols

This motivated us to study the more general Soft-constraint problem formulation, where private patterns are given negative weights, so that important public pattern matches may trump minor private matches, but at the same time serious private pattern matches can still be guaranteed to be hidden by setting their weights to negative infinity.

We realize that, intuitively, the Hard-constraint approach is more acceptable when talking about privacy. In this paper we chose to study the Soft-constraint problem not only because it is the one favored in our hospital scenario, but also because it is more generic: it subsumes the Hard-constraint variant because we can always simulate the Hard-constraint model by setting weights on private patters to negative infinity. The algorithms considered in this paper are applicable to both the Hard-constraint model and the Soft-Constraint model.

3.2.2 Type-level vs. Instance-level

Mirroring the classical taxonomy proposed for relational data privacy [17], our problem can be classified into "type-level" (corresponding to the "global recoding" in relational k-anonymity), or "instance-level" (corresponding to "local recoding").

More specifically, the **type-level** problem makes simplified suppression decisions at the *event type* level by either suppressing or preserving all events of the same type. That is, events of the same type are either all suppressed or all preserved, irrespective of when they occur in the stream.

On the other hand, the **instance-level** problem treats *each event differently* based on its run-time context, i.e., the previously arrived events in the active windows of queries, and the expectation of future events. An instance-level solution can suppress one event of a certain type while preserving another event of the same type. It thus allows flexible event suppression decisions that offer more opportunities for utility optimization, but as a consequence it presents a harder optimization problem.

3.2.3 Offline vs. Online

Orthogonal to the two dimensions above, our problem can be further classified into offline and online variants.

The **offline** variant produces decisions after the whole stream has arrived. Although this is not a practical assumption, studying the offline variant does allow us to eliminate the hardness arising from the randomness of the online problem and focus on optimal event suppression in a deterministic setting.

In the **online** variant, decisions have to be made in real-time for the currently arriving event without complete knowledge of future events. While this problem variant is more fitting for CEP applications, which typically demand real-time responsiveness, it is also intuitively more difficult. Challenges arise because (1) We must estimate future events and pattern matches that are probabilistic in nature, and (2) Even if future estimates are accurate, smart suppression decisions are still needed to maximize utility, which is in essence similar to the offline variant.

In the remainder of this paper we will focus on the online event suppression problem with soft-constraint, at both type-level and instance-level.

4. HARDNESS RESULTS

In this section, we study the hardness of the utility-maximizing event suppression problem. The proofs of our theorems are presented in Appendix. We first show in Theorem 1 that the instance-level offline variant is NP-hard.

THEOREM 1. *The problem of instance-level utility-maximizing event suppression with soft-constraint is NP-hard in the total number of events. It remains NP-hard even if each query contains exactly two event types.*

Moreover, we show that the instance-level problem is unlikely to be approximable in polynomial time.

THEOREM 2. *Let n be the total number of events in the stream. There is a fixed constant $\epsilon \geq 0$ such that if there is n^{ϵ} factor approximation algorithm for instance-level utility-maximizing event suppression with soft-constraint, then RP = NP.*

The hardness and inapproximability results illustrate the unfortunate fact that unless we are dealing with streams that have a small number of events, utility-maximizing event suppression at instance-level is unlikely to be even approximated efficiently. The problem is not much simpler even if we focus on very simple query constructs (e.g., two event types per query). Given that stream systems typically need to handle potentially a large number of event instances in real-time, efficient optimal solutions with a good quality guarantee are unlikely to exist.

Even for the less ambitious type-level variant, we show the surprising result that this variant exhibits a similar hardness as follows.

THEOREM 3. *The problem of type-level utility-maximizing event suppression with soft-constraint is NP-hard in the total number of event types.*

Despite the hardness result, we show in the following a *fixed-parameter-tractable* special case that under some natural assumptions, the type-level variant can be solved optimally.

PROPOSITION 1. *Suppose the expected number of matches for each query $Q \in \mathcal{Q}$ and $P \in \mathcal{P}$ over some standard time unit is known. Suppose the total number of public and private queries, $|\mathcal{P}| + |\mathcal{Q}|$, is some fixed constant. Then the problem of utility-maximizing online type-level event suppression can be solved in polynomial time to obtain the optimal solution in expectation.*

This proposition follows from a constructive algorithm to be described in Section 5.1. We would like to point out here that in comparison to the offline type-level variant, in the online version the solution is only optimal *in expectation*. The reason is because in the online version, only expectations of query matches are known and events can still arrive in an uncertain manner that deviates from the expectation. As such, one type-level solution may not be the optimal for all event stream instances. Rather it is optimal in the sense that in the long term, when the number of query matches converges to the expectation, the solution is optimal in expectation.

5. ONLINE SUPPRESSION ALGORITHMS

In this section, we devise two real-time event suppression strategies that maximize the overall utility. The first algorithm offers optimal suppression decisions at the event-type level (Sec. 5.1). Due to the fluctuations in the event stream, the optimal type-level decisions may not be the best decision for each event instance. This leads us to design the instance-level algorithm that tweaks type-level decisions based on run-time context, henceforth called *Hybrid* algorithm (Sec. 5.2). The key idea is that while the type-level approach provides a pretty good long-term decision, the instance-level decisions can fine-tune the type-level solution based on the events in the local windows.

Both of our algorithms need to estimate the cardinality of pattern matches as part of their decision making processes. Therefore, we propose two *cardinality estimation* approaches (Sec. 5.3). The first light-weight approach treats each type of event independent of each other, while the second sophisticated approach takes advantage of periodicities in the stream to produce more accurate estimations.

5.1 Optimal Type-Level Algorithm

Inspired by Proposition 1, we first explore the type-level variant. We propose to model the type-level suppression problem as an *integer linear program*. Specifically, let $\Sigma = \{E_i\}$ be the set of all event types. Let the integer $x_i \in \{0, 1\}$ be the decision variables to drop/keep all events of type E_i in order to ensure privacy, with $x_i = 1$ denoting to keep E_i, and $x_i = 0$ to drop E_i. Further let the integer $y_j \in \{0, 1\}$ denote whether or not public pattern $Q_j \in \mathcal{Q}$ will be output, and $z_k \in \{0, 1\}$ denote whether private pattern $P_k \in \mathcal{P}$ will be output.

Let $N_T(Q_j)$ and $N_T(P_k)$ be the expected number of pattern matches for Q_j and P_k produced over a standard time period T respectively. Using historical data, we can obtain statistics about the stream, like the average event arrival rate. These statistics then allow us to get a rough estimate of $N_T(Q_j)$ and $N_T(P_k)$ (we defer a detailed discussion on the orthogonal issue of cardinality estimation to Section 5.3). Now assume that $N_T(Q_j)$ and $N_T(P_k)$ are computed and treated as known constant values. The objective utility function then becomes:

$$\mathcal{U} = \sum_{Q_j \in \mathcal{Q}} w(Q_j) N_T(Q_j) y_j + \sum_{P_k \in \mathcal{P}} w(P_k) N_T(P_k) z_k \quad (5)$$

Recall that $w(\cdot)$ represents the utility weight function. We note that whether or not query Q_j can be reported (y_j is 0 or 1) depends on the values of the x_i's. Let $\sigma(Q_j)$ be the multi-set of all event types in Q_j. If one constituent event type E_i of a query pattern Q_j is dropped ($x_i = 0$, for $E_i \in \sigma(Q_j)$), then the query pattern can never be revealed ($y_j = 0$). We express the dependence using the following linear constraint. For all $Q_j \in \mathcal{Q}$, we have

$$0 \leq y_j \leq \frac{1}{|Q_j|} \sum_{E_i \in \sigma(Q_j)} x_i \quad (6)$$

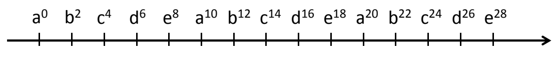
Intuitively, this says that in a type-level solution, Q_j can be reported ($y_j = 1$) if and only if none of its participating event types are dropped.

Similarly, the variable $z_k \in \{0, 1\}$ is subject to the constraint:

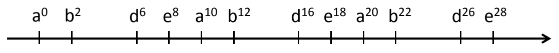
$$1 \geq z_k \geq \frac{1}{|P_k|} - 1 + \frac{1}{|P_k|} \sum_{E_i \in \sigma(P_k)} x_i \quad (7)$$

This captures the fact that the private pattern will be reported ($z_k = 1$) when all its participating event types are preserved. When at least one constituent event type is absent, z_k gets a value of 0, which will avoid the utility penalty.

The problem of maximizing $\mathcal{U}$ is then an integer linear programming problem subject to the constraints in Equations (6) and (7). Putting the issue of estimating $N_T(Q_j)$ and $N_T(P_k)$ aside, this online type-level variant is really no different from the offline type-level variant. From Proposition 1, we know that it is solvable if the total number of event types or queries is limited.



(a) Original event stream



(b) With type-level suppression

Figure 2: Type-level LP-based algorithm

This is useful, for although the number of event instances may be unbounded in a stream system, the number of queries or event types tends to be limited (HyReminder, for example, has 16 event types and 14 queries and the corresponding LP can be solved in 20 ms). Furthermore, even when we are dealing with problems that have a large number of queries and events, since the type-level decisions would stay the same for every new arriving event, the LP only needs to be solved once, until new statistics arrive, at which point the decisions need to be re-computed. This approach thus provides a practical way to solve the type-level event suppression problem. We use Example 1 to illustrate this algorithm.

EXAMPLE 1. *Suppose we have five event types,* $\Sigma = \{A, B, C, D, E\}$. *A sample event stream is depicted in Figure 2a. The superscript of an event denotes its timestamp. As shown in the figure, events with different event types arrive with the same arrival rate, namely* 1 *event per* 10 *time units, in a simple and recurring pattern* (a, b, c, d, e). *Assume there are three public queries, namely*

$Q_1 = SEQ(B, C, D)$, $Window(Q_1) = 10$, $w(Q_1) = 5$;
$Q_2 = SEQ(A, B)$, $Window(Q_2) = 10$, $w(Q_2) = 20$;
$Q_3 = SEQ(D, E)$, $Window(Q_3) = 10$, $w(Q_3) = 20$;
and one private pattern,
$P_1 = SEQ(A, C, E)$, $Window(P_1) = 10$, $w(P_1) = -10$.

Let $[x_1, x_2, ..x_5]$ *be the decision variables of whether events of type* A, B, C, D, E *can be preserved,* $[y_1, y_2, y_3]$ *be the variables that indicate whether* Q_1, Q_2, Q_3 *can be reported, and* $[z_1]$ *be the variable indicating whether* P_1 *will be produced. Using Equations* (5), (6) *and* (7), *we obtain a linear program with the objective function:*

$$U = \frac{1}{10} \cdot 5 \cdot y_1 + \frac{1}{10} \cdot 20 \cdot y_2 + \frac{1}{10} \cdot 20 \cdot y_3 - \frac{1}{10} \cdot 10 \cdot z_1$$

where the four $\frac{1}{10}$*s are essentially* $N_T(Q_j)$ *and* $N_T(P_k)$, *i.e., the expected number of pattern matches over a time period T, subject to the constraints that can be instantiated from Equation* (6) *and* (7).

Solving the linear program gives us the optimal type-level solution, $[x_1, x_2, ...x_5] = [1, 1, 0, 1, 1]$. *Namely, events of type C will all be suppressed. This solution is intuitive. Dropping event type C ensures that no matches for* P_1 *and* Q_1 *will be produced. Since the arrival rate of* Q_1 *and* P_1 *are both* $\frac{1}{10}$, *and the weight* $w(Q_1) = 5$ *while* $w(P_1) = -10$, *it is profitable to drop event type C.*

Once the type-level decisions are computed, at execution time, all events of type C will simply be suppressed, resulting in the event stream in Figure 2b.

Discussion of LP-based algorithm. While this LP-based algorithm is practical for the type-level problem variant, it would be impractical for the instance-level variant. Recall that the instance-level variant makes suppression decisions at the granularity of event instances. Hence for an instance-level solution, a decision variable x_i is needed for *each event instance*, and each *pattern match* will be represented as a constraint. Note that both of these two can quickly become impractically large, resulting in too big a linear program to be solved efficiently. In particular, in our small-scale experiment involving 50 event instances, the total time needed to solve the corresponding LP is around 16 minutes. In light of this prohibitive cost, in the next section we propose a hybrid approach that combines the type-level solution with instance-level heuristics.

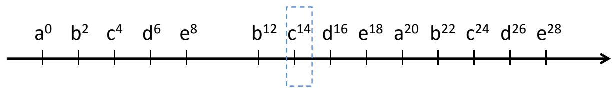


Figure 3: Motivation: suppressing c^{14} can be sub-optimal

5.2 Hybrid Instance-Level Algorithm

We now explore solutions to online instance-level event suppression, which can produce better utility than a type-level solution for the following reason. Recall that in the type-level solution, the expected number of pattern matches, $N_T(Q_j)$, is an average statistic produced over a long term. It is possible that over a short period of time event frequencies may deviate from their long term averages. Therefore, if the type-level solution is used as an instance-level solution, it may be sub-optimal in the short-term local windows. This observation is illustrated in the following example.

EXAMPLE 2. *We consider the same query patterns as in Example 1, now with a different event stream as shown in Figure 3. Recall that the type-level solution was to suppress event type C.*

Here, no a^{10} *arrives as expected, which is different from what the long-term statistics predict. Then when* c^{14} *arrives, there is no event of type A prior to* c^{14} *within* $Window(P_1) = 10$. *Hence, at time 14, no match for* $P_1 = SEQ(A, C, E)$ *can be produced even if* c^{14} *is kept. In that case, keeping* c^{14} *is essentially "free" (without any utility penalty). An optimal solution should keep* c^{14} *since it bears the potential of producing a match for* $Q_1 = SEQ(B, C, D)$. *The type-level solution in this example fails to make this optimal decision.*

The reasoning in the type-level solution, however, is not without merit. Over the long term, we expect one event of type A prior to an event of type C within $Window(P_1)$ *on average, which leads to one match of* P_1. *We also expect one match of* Q_1 *in which the same event of type C participates. So the type-level solution decides to suppress the event of type C, because the expected utility penalty of keeping it is* $w(P_1) = -10$, *which outweighs the expected utility gain* $w(Q_1) = 5$. *The particular case of* c^{14} *is different, because there is no previous event of type A in* c^{14}*'s active window. This deviation from the long term statistics renders the type-level solution sub-optimal.*

This example illustrates that due to fluctuations in event arrivals, the optimal type-level solution may *not* be the best decision. Instead it would be beneficial to deviate from the type-level decisions. This leads us to design an enhanced algorithm that extends type-level solution by instance-level heuristics that *"tweaks" type-level decisions based on run-time context*.

We describe our Hybrid algorithm in Algorithm 1. The overall flow of Algorithm 1 is summarized below. First it computes the expected utility gain of keeping the newly arrived event, e_i, by estimating the expected number of public query matches e_i could participate in. Then it computes the expected utility penalty if e_i is kept by estimating the expected number of private pattern matches e_i could be part of. It then decides to either suppress or keep e_i by comparing the expected utility gain and penalty.

We now describe Algorithm 1 in detail. First, the type-level decision for each event type is computed once in **Solve_LP**. Then, as each new event e_i arrives, we look up those events that have previously arrived for *partial matches* that e_i can participate in. Since

Algorithm 1 Hybrid Algorithm for Online Instance-level Suppression

```
Suppress_Hybrid_Instance_Level ($history$):
$Suppress\_T[]$ = Solve_LP($history$)
while $new\_event$ arrives do
    $utl\_gain$ = 0
    for each $partial\_match$ of $Q_j$ that $new\_event$ participates do
        $exp\_match \leftarrow$ Est_Match ($partial\_match, Suppress\_T[]$)
        $utl\_gain$ += $exp\_match * w(Q_j)$
    end for
    $utl\_penalty$ = 0
    for each $partial\_match$ of $P_k$ that $new\_event$ participates do
        $exp\_match \leftarrow$ Est_Match ($partial\_match, Suppress\_T[]$)
        $utl\_penalty$ += $exp\_match * w(P_k)$
    end for
    if $utl\_gain + utl\_penalty < 0$ then
        suppress $new\_event$
    else
        preserve $new\_event$
    end if
end while
```

each such partial match is only a *"prefix"* of a full match, we then estimate in the subroutine **Est_Match** the number of matches for the *"suffix"* pattern from events that are expected to arrive in the future.

To produce an accurate estimate, two challenges arise: (1) to estimate the expected number of future matches of the "suffix" pattern, and (2) to estimate the event suppression decisions for the future events. We defer the first issue to Section 5.3. The second issue is also hard, because the suppression decision of a future event could affect the decisions about other future events thereafter. Our intuition, however, is that the type-level solution should provide guidance of what events tend to be suppressed in general. After all, instance-level decisions deviate from the type-level solution only when the local event distribution differs substantially from the long term average. It can be expected that suppression decisions should in general *converge to* the type-level solution. Thus we estimate the number of matches for the "suffix" of the partial match by making the assumption that events in the future will be suppressed in the same manner that the type-level solution dictates.

With that estimate in **Est_Match** we can calculate an expected utility gain for each partial match. Summing all the utility expectations gives us an estimation of the benefit of keeping the newly arrived event. We can perform a similar estimation of the total utility penalty that we expect to suffer if the new event is kept. In the end, by a simple comparison of total utility gain and utility penalty we can determine, based on the events that have arrived, whether to suppress the newly arrived event or to keep it. We use the running example in Example 3 to illustrate Algorithm 1.

EXAMPLE 3. *In Figure 4, we revisit the event stream in Example 2 while considering the same patterns. In Figure 4, event* a^0 *arrives first, which participates in* $Q_2 = SEQ(A, B)$. *According to the type-level decision, we expect events of type B will be kept in the future, with which* a^0 *is likely to contribute to matches of* Q_2. *Then the utility gain of keeping* a^0 *is expected to be* $w(Q_2) = 20$. *But* a^0 *also forms a partial match for* $P_1 = (A, C, E)$. *Suppose in the* ***Est_Match*** *procedure one match for the remaining suffix* (C, E) *is expected to arrive. However we reason that events of type C will most likely be suppressed, according to the type-level solution. As a result, the expected utility penalty of keeping* a^0 *is 0. Hence* a^0 *is preserved.*

Following the same logic, c^4 *is suppressed and* d^6, e^8 *and* b^{12} *will be preserved. So far all the suppression decisions have been the same as the type-level solution.*

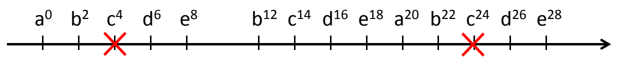


Figure 4: Running example of hybrid algorithm

Next we consider c^{14}. *First, for the partial match* (b^{12}, c^{14}) *of* Q_1 *that it participates in, suppose* ***Est_Match*** *again predicts that one event of type D will arrive in the remaining window, which amounts to a utility gain of 5. Then for* P_1, *in which* c^{14} *could participate, we find that from time 4 to 14, no event of type A exists, namely no partial match of* P_1 *that* c^{14} *could join with. Hence, keeping* c^{14} *will produce a net utility gain with no utility penalty. In this case we will keep it, which is a different decision from the type-level solution. This example manifests how the Hybrid algorithm can exploit local optimization opportunities to maximize utility.*

The upshot of this example is that the decisions converge to the type-level solution, which is an intuitive justification for our assumption that events which arrive in the future will most likely be suppressed as predicted by the type-level solution.

5.3 Pattern Match Cardinality Estimation

So far we have treated the computation of $N_T(Q_j)$, i.e., the number of matches of Q_j over a time span T in the type-level solution, and the cardinality estimation of **Est_Match** in the Hybrid solution as black boxes. Recall that the reason we need an estimate using $N_T(Q_j)$ in the type-level solution is to measure the expected utility that Q_j yields on average. The need of **Est_Match** is to weigh the utility gain and penalty of keeping the current event based on existing events in the active windows. The key issues behind both operations are essentially the same, which is to estimate the number of pattern matches over a time range T in the future.

In many applications the continuously arriving events tend to exhibit certain regular patterns [11, 20]. The problem then is to use historical data to estimate the cardinality of future pattern matches. Formally, given a pattern query $Q_j = SEQ(E_{j_1}, E_{j_2}, ..)$, and a time variable T, we want to estimate, using event arrival statistics, the number of matches for Q_j produced within T, henceforth referred to as $N_T(Q_j)$.

The time range T is a variable because it depends on how far apart in time the first event and the last event in the partial match are. For example, in Figure 4 of Example 3, when c^4 arrives, we need to estimate the number of events of type D that may arrive, which along with the existing partial match (b^2, c^4), can form full matches for query $Q_1 = SEQ(B, C, D)$. Because $Window(Q_1) = 10$, and the first event in the partial match is b^2, at current time 4, events of type D that arrive in the next $10-4+2 = 8$ time range can contribute to this particular match. This, then, is the value of T that we need to compute. However if the first event in the partial match is, say b^1 instead of b^2, then the window over which event of type D should be estimated is $10 - 4 + 1 = 7$, instead of 8. In other words the window of the partial suffix match is not fixed but rather varies. This makes the problem of cardinality estimation challenging (if the time range for cardinality estimation is always some fixed value then the problem can be solved by simply counting the historical data, with no computation involved).

In order for **Est_Match** to estimate the number of matches in the Hybrid solution, we can compute the corresponding value using the function $N_T(Q_j)$. Let $e_f.ts$ be the timestamp of the first event in the partial match, $e_c.ts$ be the timestamp of the current event under consideration, then the active window T can be computed as $Window(Q_j) - e_c.ts + e_f.ts$. Let Q_S be the suffix pattern that remains to be matched. Then **Est_Match** can be expressed as $N_{(Window(Q_j)-e_c.ts+e_f.ts)}(Q_S)$.

5.3.1 Estimation by Arrival Rate

Since Poisson process is a common approach to model arrivals in queuing theory and stream processing literature [6, 16], our first "baseline" estimation assumes that event arrivals follow the independent Poisson process. In the context of CEP, the Poisson process dictates that each type of event occurs continuously and independently of each other. Each type of event, E_i, arrives with an arrival rate λ_i, and the number of events that arrive in a fixed time period follows a Poisson distribution. In practice, when events arrive in a Poisson process, λ_i can be estimated by sampling the arriving events. Considering the fact that events may follow different distributions as time evolves, a moving sample of recently arrived events can be used to estimate the arrival rate [9].

Given the arrival rate λ_i for event type E_i, we describe how $N_T(Q_j)$ can be estimated. We first compute for each event type in Q_j the expected number of event occurrences in time range T, denoted by l_i. This can be computed as $l_i = \lambda_i T$. Further we denote by $\Gamma(Q_j)$ the multi-set of event types in Q_j, $\sigma(Q_j)$ the set of event types in Q_j, and $|Q_j|$ the pattern length of Q_j (for example, query $Q = SEQ(A, A, B)$ has $\Gamma(Q) = \{A, A, B\}$, $\sigma(Q) = \{A, B\}$, and $|Q| = 3$). Denote by $L_j = \sum_{E_i \in \sigma(Q_j)} l_i$ the expected number of occurrences of events relevant to Q_j in T. We can then estimate the expected number of query matches for Q_j in T as:

$$N_T(Q_j) = \binom{L_j}{|Q_j|} \prod_{E_i \in \Gamma(Q_j)} \frac{\lambda_i}{\sum_{E_k \in \sigma(Q_j)} \lambda_k} \tag{8}$$

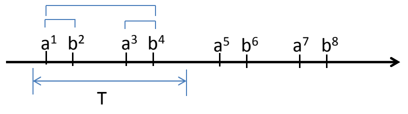
Equation (8) can be explained as follows. Given a total of L_j event occurrences in T, we need to pick $|Q_j|$ events to form one query match. Let us pick the first $|Q_j|$ events among the L_j events, and compute the probability that the first $|Q_j|$ events produce a match. Let the event at the first position of Q_j be of type E_{i_1}. The probability that the first event picked is actually of type E_{i_1} is $\frac{\lambda_{i_1}}{\sum_{E_k \in \sigma(Q_j)} \lambda_k}$, which is E_{i_1}'s arrival rate λ_{i_1} divided by the sum of arrival rates of all events in Q_j. For the second event in the sequence, the probability that it matches the event type at the second position of Q_j can be computed similarly. In the end the probability that the first $|Q_j|$ events all match the event types specified in Q_j and thus produces a query match is the cross product of individual terms, $\prod_{E_i \in \Gamma(Q_j)} \frac{\lambda_i}{\sum_{E_k \in \sigma(Q_j)} \lambda_k}$. Given that there are a total of $\binom{L_j}{|Q_j|}$ possible permutations out of L_j events and by symmetricity, the expected count of query matches can be expressed as the product of the two, thus Equation (8).

5.3.2 Estimation by Periodicity

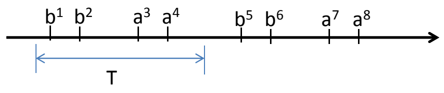
Estimation using the event arrival rate is generally applicable in the sense that it does not assume dependencies between events of interest. However, in practice, event streams may exhibit certain *periodic patterns* that are highly regular. For example, in the hospital setting, a nurse may sanitize and enter patient rooms at a regular time interval, say every hour, for routine patient check-ups. Such patterns, being periodic and regular, can be leveraged to produce even more accurate pattern match cardinality estimation. We use Example 4 to illustrate this observation.

EXAMPLE 4. *Suppose we have two event types, A and B, as in Figure 5. From their arrival rates we know there will be two instances of A and B in a time period T. Knowing only this information, the best we can do is to use Equation* (8) *to produce an average estimate of the number of matches for pattern SEQ(A, B).*

If we can ascertain that events of type A and B follow certain periodic patterns, then we can do a significantly better job. Suppose A and B have the periodic pattern "A, B, A, B" as illustrated in Figure 5a. Then starting at 0 *over a period of time* T *there are a total of three matches, namely* $\{(a^1, b^2), (a^1, b^4), (a^3, b^4)\}$. *On the other hand, if the periodic pattern is of the form "B, B, A, A" as shown in Figure 5b, then even if the arrival rate of A and B stays the same over T, no match for SEQ(A, B) can be produced.*



(a) Periodic pattern 1



(b) Periodic pattern 2

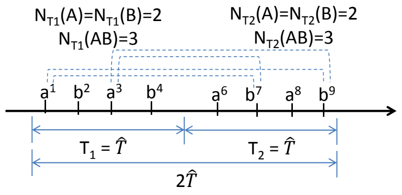
Figure 5: Estimating matches of SEQ(A,B) using periodic pattern

This illustrates the importance of discovering periodic patterns embedded in the event data, for if such patterns can be revealed and utilized, a much better pattern match estimate can be produced. This leads us to discover such periodic patterns in the input stream. The problem of periodicity mining has been extensively studied. In this work we adapt the techniques in [11, 20] to discover the periodic patterns.

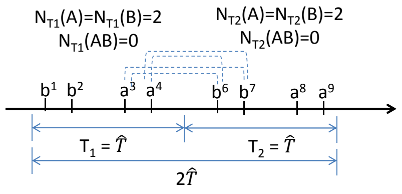
Once such periodic patterns are discovered, we can compute the number of matches for pattern Q_j in each periodic cycle $\hat{T}$, denoted by $N_{\hat{T}}(Q_j)$. The problem that remains to be solved is to extrapolate the expected pattern match cardinality of an arbitrarily long time period T_X. More specifically, the problem can be stated as given a periodic cycle $\hat{T}$ and $N_{\hat{T}}(Q_j)$, what is the number of pattern matches of Q_j in time period T_X, or $N_{T_X}(Q_j)$? Care must be taken in calculating $N_{T_X}(Q_j)$. While it is tempting to think that if $T_X = 2\hat{T}$, then $N_{2\hat{T}}(Q_j)$ can be simply computed as $N_{2\hat{T}}(Q_j) = 2N_{\hat{T}}(Q_j)$, the analysis presented in Example 5 shows the error in this thinking.

EXAMPLE 5. *We revisit Example 4 in Figure 6. Given the periodic pattern "A, B, A, B" of stream in Figure 6a. Suppose we know that in the periodicity* $\hat{T}$, *there are two instances of A and B, respectively, denoted by* $N_{\hat{T}}(A) = N_{\hat{T}}(B) = 2$, *and three matches of pattern SEQ(A,B), or* $N_{\hat{T}}(AB) = 3$. *Then what is the number of pattern matches for SEQ(A, B) in* $2\hat{T}$? *Note that* $N_{2\hat{T}}(AB) = 2N_{\hat{T}}(AB) = 6$ *is problematic, because when the time period is extended from* $\hat{T}$ *to* $2\hat{T}$, *A-s in* T_1 *and B-s in* T_2 *also produce matches in a cross-product manner. In this particular case,* $N_{2\hat{T}}(AB) = N_{T_1}(AB) + N_{T_2}(AB) + N_{T_1}(A)N_{T_2}(B) = 3 + 3 + 2 * 2 = 10$. *Similarly, in the second periodic pattern in Figure 6b, the number of pattern matches* $N_{2\hat{T}}(AB)$ *follows the same formula:* $N_{2\hat{T}}(AB) = N_{T_1}(AB) + N_{T_2}(AB) + N_{T_1}(A)N_{T_2}(B) = 0+0+2*2 = 4$. *This corresponds to* $\{(a^3, b^5), (a^3, b^6), (a^4, b^5), (a^4, b^6)\}$.

In general, to compute $N_{n\hat{T}}(Q_j)$ given $N_{\hat{T}}(Q_j)$, where $n \in \mathbb{Z}^+$, Equation (9) can be used following the logic derived in Example 5. First, define a *continuous i-segmentation* of Q_j as a segmentation that breaks the event sequence of Q_j into i non-empty segments (for example, 2-segmentations of $Q = SEQ(A, B, C, D)$ include (A/BCD), (AB/CD), and (ABC/D)). Each segment in an i-segmentation is referred to as a *chunk* (for example, A and BCD are two chunks in the segmentation (A/BCD)). We denote all continuous i-segmentations of Q_j by $M_i(Q_j)$. Let $m = (c_1, c_2, ...c_i)$ be one such segmentation in $M_i(Q_j)$, where c_k is the k-th chunk in m. Let $D(m) = \prod_{c_k \in m} N_{\hat{T}}(c_k)$ be the cross-product of the count statistics of chunk c_k. Then the estimation of $N_{n\hat{T}}(Q_j)$ can be written as:



(a) Periodic pattern 1



(b) Periodic pattern 2

Figure 6: Estimating via periodic pattern for time $2\hat{T}$

$$N_{n\hat{T}}(Q_j) = \sum_{i=1}^{n} \binom{n}{i} \sum_{m \in M_i(Q_j)} D(m) = \sum_{i=1}^{n} \binom{n}{i} \sum_{m \in M_i(Q)} \prod_{c_k \in m} N_{\hat{T}}(c_k) \tag{9}$$

So far we have only considered estimating pattern matches in a time period that is an exact multiple of a periodic cycle $\hat{T}$. Next we extend the logic to the general case. To estimate pattern match cardinality of a time period that is less than a full periodic cycle, we can detect the number of pattern matches within that time period from the known periodic pattern. To handle a time period that is longer than $\hat{T}$ but not an exact multiple of $\hat{T}$, we first estimate the utility within the time of a multiple of $\hat{T}$ using Equation (9), and then detect the number of matches within the remaining time period using the periodic pattern. Finally we combine the two parts of estimation to obtain a final estimate.

6. EXPERIMENTAL EVALUATION

We have implemented our proposed algorithms on the HP CHAOS stream engine [14], and used the OptimJ [2] module as the linear program solver. All experiments were conducted on a machine with an Intel Core 2 Duo 3.0GHz CPU and 3.2GB of RAM running Windows 7 and Java JRE 6.

Our experiments were conducted using both a real-world workload and a synthetic workload. The real-world workload was based on anonymized event streams collected from the University of Massachusetts Memorial Hospital where HyReminder is deployed.

In our real workload, public query patterns were created when HyReminder was developed from 2010 to 2012, following hygiene regulations established for US hospitals [7]. Examples of such queries include Q1, Q2, Q3 discussed earlier. As part of our new investigation, private patterns were constructed in consultation with domain experts familiar with the hospital application and privacy regulations. For example, P1 discussed earlier is a private query. A doctor checking-out some sensitive equipment (e.g. External Defibrillators) before entering a patient's room is another private query. We summarize the characteristics of these query patterns in Table 2.

In our experimental system, weights of query patterns were determined through a process in which we asked the domain experts the basic question "compared to a query match for Q_i, what is the relative importance of a match for Q_j (or P_k)." Suppose the weight of Q_i is initially set to w. For a private query P_k, if we determine that the benefit of reporting two matches of Q_i would negate the potential damage of revealing one match of P_k, we can then set weight of P_k to be $-2w$. Weights of public patterns can be determined similarly. This process is intuitive given the linearity of our objective function, and is easily comprehensible to non-technical users.

Parameter	Value range
Pattern length	2 to 5
Time-based window size	1 min to 90 min
Weight of public patterns	1, 2, 3
Weight of private patterns	-1, -2, -3

Table 2: Pattern query characteristics

In our approach an important question arises as to how to identify all possible private queries a priori. In our hospital application, where we only have a small number of *event-types*, we can afford to exhaustively enumerate possible event-type combination to ensure that no bad queries are overlooked. Although this type of simplicity is not isolated (we know a few other CEP applications used in practice that can encode the domain of interest using a small number of event-types), we think it is an interesting area for future work to understand how to scale this process to problems with a large number of event types, possibly through some automatic process.

The synthetic event stream was produced with a parameterized *degree of periodicity*, where periodicity 1 corresponds to a perfectly periodic stream, while periodicity 0 means a perfectly random stream. We produced the synthetic stream by first generating a perfectly periodic stream. We then added "noise" to this stream by replacing events in the stream using different events with random timestamps and event-types drawn from Possion processes. In that way, an event stream with degree of periodicity $\gamma\%$ will have $\gamma\%$ of events generated through the addition of noise.

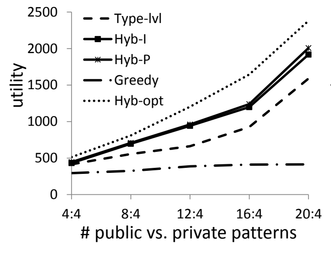
6.1 Algorithms Compared

We compare algorithms proposed in this work, namely the type-level LP-based solution (labeled as "Type-lvl"), the Hybrid solution with independent Poisson arrival estimation, ("Hybrid-I"), and the Hybrid solution with periodicity estimation ("Hybrid-P"). We also implemented two alternative event suppression solutions, namely the Hybrid-optimal algorithm ("Hybrid-opt") and the Greedy algorithm ("Greedy"), to compare against our proposed algorithms, in order to shed light on the performance of our algorithms.

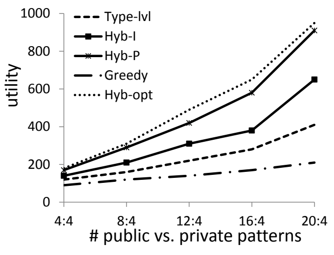
Hybrid optimal. The first baseline approach is the so-called Hybrid-optimal. Recall that one key problem that we address in this work is to estimate the cardinality of pattern matches in the future. In this algorithm, we assume the Hybrid-optimal algorithm has perfect statistics, that is, it magically knows the actual number of pattern matches in the future. This can be implemented by looking at the events in the future and counting the number of matches. Comparing our approach with such an "oracle" algorithm that essentially "cheats" helps us to understand the utility loss due to inaccurate match cardinality estimations.

Global optimal. Note that there is still a gap between Hybrid-Optimal and global optimal. As such, we are interested in finding the size of this gap to establish a real utility upper-bound. For that, we randomly sampled a small number of events and used an LP-solver to solve the corresponding instance-level LP. This turns out to be very expensive; a stream of 50 events took about 16 minutes to finish. Over randomly sampled small-scale streams, we found a very small utility gap (2-4%) between the Hybrid-Optimal and the LP-solved global optimal. Given this result we regard Hybrid-Optimal as our approximate utility upper-bound.

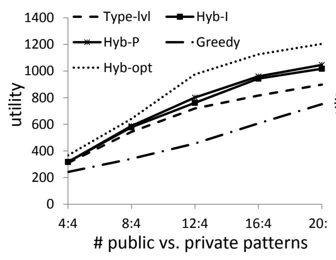
Greedy solution. In addition, we experimented with a greedy algorithm that detects and eliminates breaches in a *detecting-then-removing* fashion [28]. Specifically, for each event e_i, it calculates



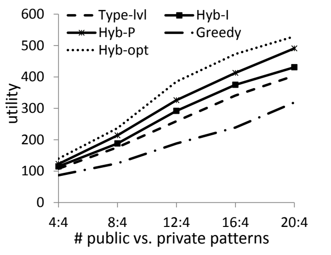
(a) Daytime stream, Soft-constraint



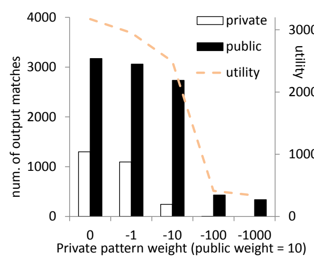
(b) Nighttime stream, Soft-constraint



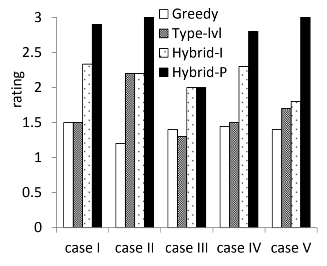
(c) Daytime stream, Hard-constraint



(d) Nighttime stream, Hard-constraint

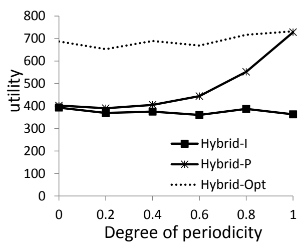


(e) Vary private query weights

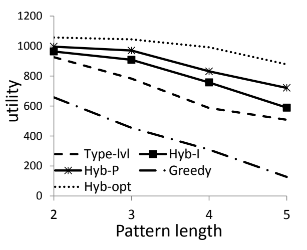


(f) User study results

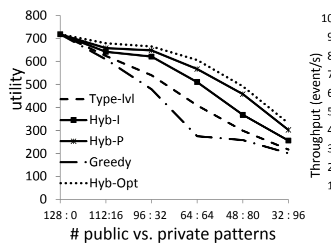
Figure 7: Performance on hospital workload



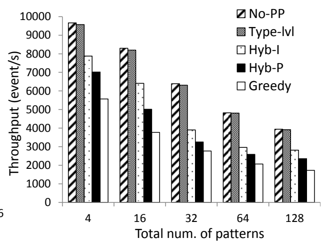
(a) Vary degree of periodicity



(b) Vary pattern length



(c) Vary public/private patterns



(d) Throughput

Figure 8: Performance on synthetic workload

the utility gain of public query matches that e_i triggers, as well as the utility loss of private query matches that e_i triggers over the stream of events output for public query matches. Both utility gain and loss are calculated using currently detected matches *without* estimating the effect of e_i on future matches. The greedy approach would suppress e_i if the utility loss outweighs the utility gain, and keeps e_i otherwise.

6.2 Experiments on Healthcare Workload

We observe that events of the health care workload during the day are dominated by random healthcare worker activities, which tend to be aperiodic. However, events during the night are more regular with a higher degree of periodicity, which may be due to the fact that at night nurses on duty typically only conduct routine check-ups in every room at regular intervals. We thus separate the stream into two sub-streams, one with day-time events and the other with night-time events.

Number of patterns vs. utility, using Soft-constraint. In this experiment, we have 4 private queries while varying the number of public queries from 4 to 20, over a stream of 20,000 events extracted from real stream trace collected during the daytime. Figure 7a shows the utility gain over the day-time stream. Clearly, Hybrid-opt has best utility gains. Our Hybrid-P approach comes second, slightly outperforming Hybrid-I. Both hybrid algorithms consistently outperform the Type-level algorithm by around 30%, and Hybrid-P outperforms the Greedy algorithm by up to 4x. This underlines the effectiveness of our Hybrid algorithms.

Figure 7b shows the utility of the same experiment using a stream of 10,000 events extracted from a night-time stream trace. In Figure 7b Hybrid-P produces 40% more utility than Hybrid-I. Note it has only 10% less utility than the Hybrid-opt. This is encouraging, because it shows that if the event stream exhibits periodic patterns, then our periodicity based cardinality estimation can capture these patterns and produce accurate estimations that ultimately lead to better utility-preserving decisions.

Number of patterns vs. utility, using Hard-constraint. We have mentioned that the algorithms considered in this paper are applicable to the Hard-constraint setting for strict privacy guarantees, by setting private weights to negative infinity and considering positive utility contribution of public queries in making suppression decisions. To explore this we conducted an additional set of experiments, where we used the same hospital data as in the previous experiment, but set all private pattern weights to minus infinity. This mimics a more strict environment where no private match is ever produced.

Figures 7c and 7d show similar trends for the Hard-constraint model — the Hybrid approaches also outperform alternative algorithms here. This suggests that Hybrid algorithms can be an attractive solution even when strict privacy guarantees are needed.

Pattern weight vs. number of output. In this experiment we vary weights of 4 private patterns from 0 to -1000, while fixing weights of 10 public pattern at 10. Results are reported using the same daytime stream used previously. Figure 7e shows the number of pattern matches produced by our Hybrid-P approach as well as the corresponding utility. As the weights become more negative, the number of private matches being output decreases. Observe that when the negative weight has significantly larger absolute value than the positive weight, no private match will be output at all. This also shows that our solutions can be configured with extremely low negative weights to solve the hard-constraint problem.

A user study on algorithm effectiveness. We invited 14 users, all familiar with HyReminder, to evaluate the output produced by the four algorithms considered in this work. We asked users to rate the effectiveness of algorithms by inspecting query matches produced by each algorithm, taking into account the perceived usefulness of public matches and potential damaging effects of private matches.

We created five test-cases for users to judge in the following manner. For each test-case, we randomly selected a subsequence of 100 events (all about a single healthcare worker) from the event trace. We also randomly picked 7 public patterns and 2 private patterns. We executed four algorithms using that workload, highlighted any remaining public and private query matches, and asked users to inspect the results. Users had their own intuitive ideas about the relative importance of the queries, but were not given the exact weights used by the system. The users were asked to rate the output of each algorithm as "bad" (score = 1), "fair" (2) or "good" (3), based on their subjective judgment of how well each algorithm makes the tradeoff between reporting useful public queries and hiding risky private queries.

Figure 7f summarizes the average score of each algorithm in five test-cases. Hybrid-P has an average score of 2.74 and ranks the highest across all test-cases. In test-case II and V, every user rated it as "good". Hybrid-I scores 2.12 on average, which is the second-best based on user ratings, while Greedy ranks the lowest with a score of 1.38. In summary, results of this user study are consistent with our utility-function based evaluation, confirming the effectiveness of our approach in capturing user's preferences, and affirming the usefulness of Hybrid algorithms.

6.3 Experiments on Synthetic Workload

Synthetic pattern queries were generated by randomly picking pattern lengths, event types, window sizes and weights using values listed in Table 2. For each experiment, numbers are reported using an averaged over 20 randomly generated workloads.

Periodicity degree vs. utility. Now we assess the relationship between the degree of periodicity of a stream and the resulting utility. Figure 8a compares algorithms with varying degrees of periodicity. We used 96 public patterns and 32 private patterns in this experiment, and executed algorithms over a stream of 10,000 events. With no periodicity Hybrid-P and Hybrid-I yield almost the same utility. As we move from a less periodic stream to a more periodic one, Hybrid-P achieves better utility preservation. The utility gain of Hybrid-P in the case of perfect periodicity (periodicity degree = 1) is almost 2x than that of Hybrid-I. It is also worth noting that even for streams with low degree of periodicity, say 0.2, there is a positive, albeit slight, utility advantage of using periodicity based cardinality estimation. For the remainder of experiments we fix the degree of periodicity at 0.4, a middle ground value.

Pattern length vs. utility. Next, we vary pattern complexity by changing pattern length from 2 to 5 and compare the utility of different algorithms. For each pattern length, we executed 96 public queries and 32 private queries over a stream of 10,000 events. As we can see in Figure 8b the utility drops as the pattern length increases. This is because the number of matches decreases when the pattern length increases. Our proposed Hybrid-P achieves only 12% less utility than the Hybrid-Opt, and outperforms all other approaches in this experiment.

Number of public/private patterns vs. utility. In Figure 8c we vary the ratio between the number of private patterns versus public patterns, while fixing the total number of patterns at 128, and report utility over a random stream with 10,000 events. As we can see, when there is no private pattern, each approach produces the same utility, as no event suppression is needed. As the number of private patterns increases, the proposed Hybrid-P and Hybrid-I outperform all other alternatives except for the oracle version Hybrid-Opt.

Number of patterns vs. throughput. Finally, we conduct a throughput test by increasing the total number of patterns. The ratio between the number of private patterns versus public patterns was fixed at 1:3. Figure 8d shows that the Hybrid-P algorithm attains at least 55% of the throughput of a baseline system where no privacy preservation is performed (labeled as "No-PP"). Its relatively efficiency and superior utility makes it an appealing approach. We also note that the Type-level approach achieves only slightly less throughput than No-PP. Considering the fact that Type-level significantly outperforms the Greedy approach in utility, it may also be an attractive alternative when system resources is a concern.

7. RELATED WORK

Complex Event Processing (CEP). CEP technologies exhibit sophisticated capabilities for pattern matching in high volume event streams [1, 5, 22, 29]. However, privacy-aware CEP is an emerging problem that has not been well studied. To the best of our knowledge the work closest to us is [15]. However the focus of [15] is on theoretical properties of the particular hard-constraint variant. It does not present solutions or empirical studies for the more general soft-constraint variant.

Privacy for data streams. Authors in [8, 31] consider k-anonymity in streaming data, where the aim is to generalize clusters of tuples into equivalence classes of size at least k. Differential privacy and its streaming variants (e.g., [10, 23]) have been proposed as a rigorous alternative. However, their applications are mostly limited to statistical aggregates (e.g., SUM query), and are not directly applicable to CEP pattern matching operation. Because many CEP applications are interested in individual pattern matches, and adding noise to individual matches (as opposed to aggregates) differentially privately can easily render the result useless.

In numerical streams, correlation attacks are studied in [18] and random noises are used to mitigate such attacks. Recently, authors in [13] consider the problem of filtering sensitive user location data in a stream to preserve privacy. These techniques, while useful for their respective purposes, are not applicable as they do not consider CEP pattern query semantics nor the utility optimization objective.

Hiding patterns in sequence databases. The problem of sequential pattern hiding was investigated in [4, 12] for static sequence databases. Their approaches operate on a set of sequences which are independent from each other. While in our CEP context, the single input stream contains potentially endless events that are temporally correlated. Therefore their approaches cannot be applied to solve our problem.

Combining global and local prediction. Lastly, the general idea of combining "global" (type-level) prediction and "local" (instance-level) prediction in our Hybrid algorithm has been applied to problems in completely different contexts, such as to predict the movement of mobile users [19], and to predict branches in the computer architecture literature [30].

8. CONCLUSIONS

In this paper we study the problem of utility-maximizing event stream suppression. We show that the Hybrid approach that combines LP-based solution and periodicity-based cardinality estimation is effective in maximizing utility while taking into account privacy penalties. in at least one real world scenario and many more synthetically generated datasets. Leveraging periodicity information for purposes beyond privacy (e.g., CEP query optimization) seems to be interesting areas for future research.

9. REFERENCES

[1] Microsoft StreamInsight: http://msdn.microsoft.com/en-us/sqlserver/ee476990.aspx.

[2] OptimJ: ateji.com/optimj.

[3] StreamBase: www.streambase.com.

[4] O. Abul, F. Bonchi, and F. Giannotti. Hiding sequential and spatiotemporal patterns. In *IEEE Trans. Knowl. Data Eng.*, volume 22, pages 1709–1723, 2010.

[5] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD*, 2008.

[6] B. Babcock, S. Babu, R. Motwani, and M. Datar. Chain: Operator scheduling for memory minimization in data stream systems. In *SIGMOD*, 2003.

[7] J. M. Boyce and D. Pittet. Guideline for hand hygiene in healthcare settings. *MMWR Recomm Rep.*, 2002.

[8] C.-Y. Chow, M. F. Mokbel, and T. He. A privacy-preserving location monitoring system for wireless sensor networks. *IEEE Trans. Mob. Comput.*, 10(1):94–107, 2011.

[9] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. In *PODS*, 2010.

[10] C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, and S. Yekhanin. Pan-private streaming algorithms. In *ICS*, 2010.

[11] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid. Stagger: Periodicity mining of data streams using expanding sliding windows. In *Proceedings of ICDM*, 2006.

[12] A. Gkoulalas-Divanis and G. Loukides. Revisiting sequential pattern hiding to enhance utility. In *ACM SIGKDD*, pages 1316–1324, 2011.

[13] M. Goetz, S. Nath, and J. Gehrke. Maskit: Privately releasing user context streams for personalized mobile applications. In *Proceedings of SIGMOD*, 2012.

[14] C. Gupta and S. W. et al. Chaos: A data stream analysis architecture for enterprise applications. In *CEC*, 2009.

[15] Y. He, S. Barman, D. Wang, and J. Naughton. On the complexity of privacy-preserving complex event processing. In *Proceedings of PODS*, 2011.

[16] Q. Jiang and S. Chakravarthy. Queueing analysis of relational operators for continuous data streams. In *Proceedings of CIKM*, 2003.

[17] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: efficient full-domain k-anonymity. In *Proceedings of SIGMOD*, 2005.

[18] F. Li, J. Sun, S. Papadimitriou, G. A. Mihaila, and I. Stanoi. Hiding in the crowd: Privacy preservation on evolving streams through correlation tracking. In *ICDE*, 2007.

[19] T. Liu, P. Bahl, and I. Chlamtac. Mobility modeling, location tracking, and trajectory prediction in wireless atm networks. *IEEE Journal on Selected Areas in Communications*, 1998.

[20] H. J. Loether and D. G. McTavish. *Descriptive and inferential statistics, an introduction*. 1993.

[21] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Proceedings of ICDE*, 2006.

[22] Y. Mei and S. Madden. Zstream: A cost-based query processor for adaptively detecting composite events. In *SIGMOD*, 2009.

[23] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. *SIGMOD*, 2010.

[24] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computations, and Application*. 1986.

[25] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10, 2002.

[26] J. Tan. Inapproximability of maximum weighted edge biclique and its applications. In *Proceedings of TAMC*, 2008.

[27] D. Wang, E. Rundensteiner, and H. Wang. Active complex event processing: Applications in realtime health care. In *Proceedings of VLDB*, 2010.

[28] T. Wang and L. Liu. Butterfly: Protecting output privacy in stream mining. In *Proceedings of ICDE*, 2008.

[29] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*, 2006.

[30] T.-Y. Yeh and Y. N. Patt. Two-level adaptive training branch prediction. In *Proceedings of MICRO*, 1991.

[31] B. Zhou, Y. Han, J. Pei, B. Jiang, Y. Tao, and Y. Jia. Continuous privacy preserving publishing of data streams. In *Proceedings of EDBT*, 2009.

APPENDIX

A. PROOFS FOR HARDNESS RESULTS

Proof of Theorem 1.

PROOF. We reduce the problem of *Maximum Weighted Edge Biclique (MWEB)* [26] to offline instance-level event suppression problem with soft-constraint. In the decision version of MWEB, given a bipartite graph $G = (V_1, V_2, H)$ where edges take on both positive and negative weights $W(h) \in \mathbb{R}, h \in H$, the problem is to determine if there exists a bipartite subgraph whose sum of edge weights is no less than a given number U. For each vertex $v \in V_1 \cup V_2$, construct an event type E_v and one event e_v of type E_v. The input event sequence S is a randomly ordered sequence that consists of one e_v for all $v \in V_1 \cup V_2$. For each edge $h = (u, v) \in H$, if the edge weight is positive $W(h) > 0$, we build two public queries $Q_h = SEQ(E_u, E_v)$ and $Q'_h = SEQ(E_v, E_u)$, both with utility weight $W(h)$, and infinite window size. Similarly if the edge weight is negative, we build two private queries $P_h = SEQ(E_u, E_v)$ and $P'_h = SEQ(E_v, E_u)$ also with utility weight $W(h)$ and infinite window size. We first show that if there is a solution to MWEB with edge weight no less than U, then there exists a solution to the event suppression problem that has utility no less than U. Let $G' = \{V'_1, V'_2, H'\}$ be the subgraph of G that has weight C, where $C \geq U$. We can preserve every event e_v if $v \in V'_1 \cup V'_2$. The set of query matches would correspond directly to the set of edges H' in G', thus producing a total utility of C. Since we know $C \geq U$, this proves the forward direction. Now we need to show that if there is a solution to the event suppression problem with utility at least U, there is a bi-clique in the original graph that has an edge weight at least U. Let $T = e_v$ be the set of events preserved in event suppression solution. The subgraph G_T of G induced by $V = \{v : e_v \in T\}$ has a one-to-one correspondence between an edge in G_T and a query match in the solution T. The edge weight of G_T is thus the same as the utility of the solution T, which is no less than U. This completes our proof. □

Proof of Theorem 2.

PROOF. It was shown in [26] that there exists a constant $\epsilon > 0$, such that unless RP = NP, the MWEB problem cannot be approximated within a factor of n^ϵ in polynomial time, where n is the number of vertices in the graph. We note that the reduction from MWEB above is value preserving. That is, if a MWEB problem instance has a weight value of U, then the corresponding event suppression problem we construct also has a utility value of U. We show that the soft-constraint variant cannot be approximated within n^ϵ by contradiction. Suppose there exists an polynomial time algorithm that approximates the offline variant within a factor of n^ϵ. Then we would have found an algorithm that approximates MWEB within n^ϵ. This contradicts with the inapproximability result in [26] under standard complexity assumptions, thus proves the inapproximability of the offline soft-constraint variant of the event suppression problem. □

Proof of Theorem 3.

PROOF. We show this problem variant is NP-hard using a proof similar to the proof in Theorem 1, which is for the offline instance-level variant. In proving the hardness of offline instance-level variant, each problem instance we construct for each instance of MWEB problem has exactly one event instance per event type. In this particular case, an instance-level solution is also a type-level solution, and vice versa. Thus, using a similar proof, we can also show the hardness of the offline type-level variant. □