Mining State Dependencies Between Multiple Sensor Data Sources

Vasile-Marian Scuturici[1] Marc Plantevit[2]
Céline Robardet[1]

March 26, 2013

[1]**Université de Lyon, CNRS**
INSA-Lyon, LIRIS, UMR5205, F-69621 Villeurbanne, France
E-mail: {marian.scuturici,celine.robardet}@insa-lyon.fr

[2]**Université de Lyon, CNRS**
Université Lyon 1, LIRIS, UMR5205, F-69622 Villeurbanne, France
E-mail: marc.plantevit@liris.cnrs.fr

Abstract

Pattern mining over data streams is critical to a variety of applications such as prediction and evolution of weather phenomena or anomaly detection in security applications. Most of the current techniques attempt to discover associations between events appearing on the same data stream but are not able to discover associations over multiple heterogeneous data streams. In this work, we aim to identify temporal dependencies between data streams. We represent event streams by state streams that are induced by the streams' events themselves. Each state has a duration, represented as a set of disjoint time intervals with respect to the events that occurred in the stream. Temporal relations between these interval sets infers dependencies between the corresponding datasources. Our interval-based approach is robust to the temporal variability of events that characterizes the time intervals during which the events are related. It links two types of events if the occurrence of one is often followed by the appearance of the other in a certain time interval. The proposed approach determines the most appropriate time intervals of a temporal dependency whose validity is assessed by a χ^2 test. As several intervals may redundantly describe the same dependency, the approach retrieves only the few most specific intervals with respect to a dominance relationship over temporal dependencies, and thus avoids the classical problem of pattern flooding in data mining. TEDDY algorithm, TEmporal Dependency DiscoverY, prunes the search space while certifying the discovery of all valid and significant temporal dependencies. We present empirical results on simulated data to show the scalability and the robustness

of our approach. We also report on case studies from smart real-world environments equipped with a number of cameras and motion sensors. These experiments demonstrate the efficiency and the effectiveness of our approach.

1 Introduction

In the last decade, the constant evolution in hardware and software technology has made it possible for companies to generate and store very large volumes of information from various data sources. This highly innovative context has been a fruitful source of motivation for the development of many data stream management and analysis techniques [1] and extending classical pattern mining techniques (e.g., frequent itemsets [16, 24], multidimensional data [20], sequences [9, 10, 32], multidimensional sequences [38] and graphs [2, 15]) to tackle the new challenges faced in this context [25]: handling infinite sequences of events occurring at steady a pace to provide actionable insights to end-users. Since the mining step has to be faster than the data acquisition process, it is not possible to store data streams in their entirety and then perform various scans on them. For this reason, data stream mining has been popularly recognized as an important research area with many applications, such as the prediction and evolution of weather phenomena, anomaly detection in security applications, or mining health monitoring streams, to mention just a few.

The recent breakthroughs in sensor technology have given users the ability to monitor many events in real time producing multiple heterogeneous data streams. As an example, smart environments equipped with various kinds of sensors (e.g., cameras, badge readers, motion sensors, automatic doors), that are so many data sources, generate simultaneously several data streams at high speed. In this paper we address the original problem of identifying temporal dependencies between data sources. It consists in discovering inter-stream relations that link two types of events if the occurrence of one is repeatedly followed by the appearance of the other in a certain time interval. Such dependencies between data streams may constitute key actionable insights for other timely challenges such as smart environment monitoring, partial failure detection in a smart environment, structural health monitoring, object tracking between various cameras, or data streams indexing.

By nature, events have no duration, but in practice, especially in a smart environment, they often describe actions or states which actually last for a period of time. Considering that the occurrence of an event changes the state of the stream, this makes it possible to design an interval-based approach that improves existing time-point based approaches with the following advantages: (1) it is suitable for handling events that are rare but which change the state of the stream for a long period of time, i.e., events with low frequency but long duration; (2) it is more robust to the temporal variability of events, (3) it is semantically rich and allows the discovery of sophisticated relations based on Allen's algebra [5], whereas time-point based approaches consider only simple

"before/after" relations between time points.

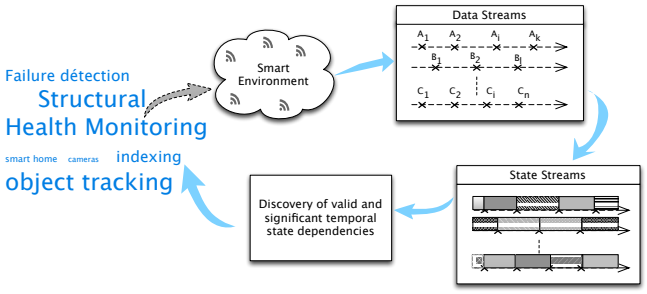


Figure 1: Overview of our approach.

Our interval-based approach not only identifies temporal dependencies between data sources, but also determines the most appropriate time-delay intervals that may exist between them. An overview of our approach is provided in Figure 1. Each data stream is first transformed into a *state stream*: the occurrences of the events change the state of the stream which over time goes from one state to another. Each state is described by its set of active time intervals. Then, valid and significant temporal dependencies are mined between distinct data sources. A temporal dependency between two states A and B denotes that the activation of state B relies on the activation of state A. The strength of the dependency is evaluated by the proportion of active time of A where B is also active. To take into account the fact that the activation of B can be delayed with respect to A, the active time intervals of B are shifted to maximize their intersection with the active period of A. Finally, to be robust to the inherent variability of the data, the active time intervals of B can be slightly extended so as to better coincide with A. The proposed approach determines the most appropriate time intervals of a temporal dependency whose validity is assessed by a χ^2 test. As several intervals may redundantly describe the same dependency, the approach retrieves only the few most specific intervals with respect to a dominance relationship over temporal dependencies, and thus avoids the classical problem of pattern flooding in data mining. Finally, the dependencies can be directly exploited by the end-users to monitor the environment or detect its failures, e.g. when a valid and significant dependency suddenly becomes unsatisfied, abnormal behavior or sensor failure can be suspected. Discovering all valid and significant temporal dependencies is challenging since, for every couple of states between two data sources, all possible time-delay intervals have to be taken into account. To overcome the complexity of this task, we propose an efficient algorithm TEDDY that benefits from different properties in order to prune the search space while certifying the discovery of all valid and significant temporal dependencies. We conduct an extensive experimental study on both synthetic and real-world data streams from smart environments equipped with various kinds of sensors (cameras, motion sensors, etc.). These experiments

show that the pruning techniques make it possible to discard a large part of the search space and speed up TEDDY running time by a factor that varies between 2 and 60. A qualitative analysis of the output shows that TEDDY produces a small set of non-redundant dependencies that well describe the phenomenon captured by the data.

To summarize, the main contributions of this paper are:

- The introduction of a novel problem: the discovery of dependencies between multi-sources data streams. We define the temporal state dependency as a suitable mathematical notion for the study of multiple heterogeneous data streams. We introduce the notion of confidence of a temporal state dependency and assess its validity based on a χ^2 test. We define a dominance relationship between temporal dependencies to control the intrinsic redundancy of the patterns.
- The design of an efficient algorithm that benefits from various properties.
- The evaluation of the efficiency of the algorithm on synthetic data and the illustration of its applicability on real data streams from two smart environments equipped with various cameras and motion sensors.

The remainder of this paper is organized as follows. Section 2 formally defines valid and significant temporal dependencies between multiple state streams. In Section 3 we present our approach to efficiently discovering temporal dependencies between data streams. Section 4 reports on an empirical study. Section 5 presents related work. Section 6 concludes this paper.

2 Temporal dependencies between multiple state streams

We consider a set of data streams $DS = \{DS_1, \cdots, DS_n\}$, where DS_k is a sequence of timestamped events produced by a data source: $DS_k = \{(t, e) \mid t \in \mathcal{T}, e \in E_k\}$. $\mathcal{T}$ is an infinite set of timestamps. We assume that a data source cannot produce more than one event at a time, and E_k is the event type set of the k^{th} data source. To process the infinite set of events of DS_k, $\mathcal{T}$ is partitioned in batches of events. Each batch is defined as the set of events contained in the interval $[t_{begin}, t_{end})$ and is indicated by $DS_k^{[t_{begin}, t_{end})}$, or simply DS_k if there is no ambiguity, as in the rest of the paper. The event that immediately occurs before the timestamp $t \in [t_{begin}, t_{end})$ is denoted $\mathbf{pred}(DS_k, t)$ whereas the event that occurs immediately after t or at t is indicated as $\mathbf{succ}(DS_k, t)$.

2.1 Converting a data stream into a state stream

Data streams are generally considered as sequences of time-point events, i.e., events that have no duration. However, when dealing with sensor data sources, we consider that the most meaningful temporal information is the time spent

between two consecutive events. Indeed, each event changes the internal state of the stream. For example, considering a data stream generated by a door, the events occurring are of two types: *opening* and *closing*. Therefore, the door is continually either in the state of being *open*, when the last event is *open*, or otherwise *close*. Let $\mathcal{S}(DS_k)$ be the set of all possible states of DS_k and $\mathbf{state} : E_k \to S(DS_k)$ be the mapping of events into states. DS_k is in state s at time t if $\mathbf{state}(\mathbf{pred}(DS_k, t)) = s$. Therefore, a state $s \in \mathcal{S}(DS_k)$ is defined by the period of time when it is active in DS_k. This time period is generally discontinuous, and it is thus represented by a set of maximal time intervals, called *active interval set*, denoted $I(s)$ and defined as follows:

$$I(s) = \{[t_i, t_j) \subseteq [t_{begin}, t_{end}) \quad | \quad \forall t \in [t_i, t_j),\ \mathbf{state}(\mathbf{pred}(DS_k, t)) = s \text{ and } \mathbf{state}(\mathbf{pred}(DS_k, t_i)) \neq s \text{ and } \mathbf{state}(\mathbf{succ}(DS_k, t_j)) \neq s\}$$

The significance of a stream state is evaluated by the sum of the lengths of its active intervals:

$$\mathbf{len}(s) = \sum_{[t_i, t_j) \in I(s)} (t_j - t_i)$$

2.2 Active interval sets matching

The dependency of two states, from distinct data sources, is evaluated on the basis of the intersection of their active interval sets. Let s_k and s_ℓ be two states from distinct data streams. The intersection of $I(s_k)$ and $I(s_\ell)$ is defined as the union of intersecting intervals: $I(s_k) \cap I(s_\ell) = \{[t_i, t_j) \cap [t'_i, t'_j) \mid [t_i, t_j) \in I(s_k),\ [t'_i, t'_j) \in I(s_\ell)\}$. Figure 2 provides an example of the intersection of two interval sets. However, two stream states s_k and s_ℓ may be in temporal dependency $s_k \to s_\ell$ whereas they are not active at the same time but s_ℓ is time-delayed with respect to s_k. To capture such dependencies the active interval set of s_ℓ may undergo some transformations so as to better coincide with the active interval set of s_k. Two types of transformations can be applied: (1) $I(s_\ell)$ can be shifted so as to maximize its intersection with $I(s_k)$, or (2) $I(s_\ell)$ can be slightly extended so as to make the temporal dependency measure more robust to the inherent variability of the data. Shifting an active interval set I with β time units consists in translating each time interval of I by β time units: $I_{[\beta,\beta]} = \bigcup_{[t_i,t_j)\in I}[t_i + \beta, t_j + \beta)$. To slightly extend the sets of I, the first bound of each interval of I is translating by only α time units, with $0 \leq \alpha \leq \beta$. It results in the following new interval set: $I_{[\alpha,\beta]} = \bigcup_{[t_i,t_j)\in\mathcal{I}}[t_i + \alpha, t_j + \beta)$. Notice that the intervals of $I_{[\alpha,\beta]}$ may intersect. In that case, intersecting intervals are merged. Figure 3 illustrates some active interval set shifts.

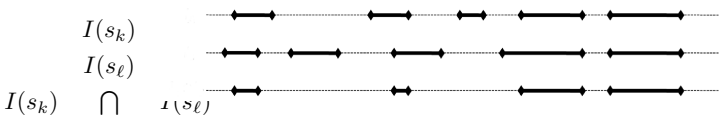


Figure 2: Example of active interval sets intersection.

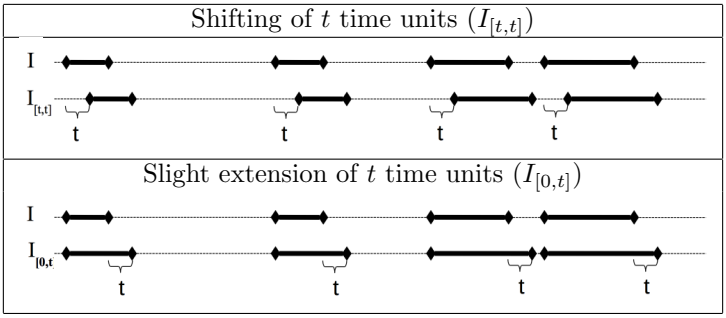


Figure 3: Example of active interval set shifts

2.3 Temporal dependency assessment

The temporal dependency of a state s_k over a state s_ℓ is evaluated by the proportion of time where the two states are active over the active time of s_k. This confidence measure is formally defined below.

Definition 1 (Confidence of state dependency) *Considering two data stream states s_k and s_ℓ from distinct data streams, as well as a shifting interval $[\alpha, \beta]$, the strength of a $[\alpha, \beta]$-temporal dependency between s_k and s_ℓ, denoted $s_k \xrightarrow{[\alpha,\beta]} s_\ell$, is evaluated by the following confidence measure:*

$$\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell) = \frac{\mathbf{len}(s_k \cap s_\ell^{[\alpha,\beta]})}{\mathbf{len}(s_k)}$$

where $s_k \cap s_\ell^{[\alpha,\beta]} = I(s_k) \cap I_{[\alpha,\beta]}(s_\ell)$.

We can observe that $\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell)$ is equal to 1 iff each interval of $I_{[\alpha,\beta]}(s_\ell)$ is included in an interval of $I(s_k)$.

For an effective search for temporal dependencies, the shifting intervals $[\alpha, \beta]$ are taken in $[t_{min}, t_{max}]$, where t_{min} and t_{max} are parameters set by the end-user. They specify the time range in which looking for shifting intervals. Some instantiations of $[\alpha, \beta]$ convey some specific semantics of the confidence measure. For instance, $s_k \xrightarrow{[0,0]} s_\ell$ highlights a *simultaneous* dependency between s_k and s_ℓ. $s_k \xrightarrow{[\beta,\beta]} s_\ell$ means that state s_k is in the relation *after exactly* β timestamps with state s_ℓ, and $s_k \xrightarrow{[0,\beta]} s_\ell$ means that s_k is in the relation *after at most* β

timestamps with state s_ℓ. On Figure 4, $\mathbf{conf}(s_k \xrightarrow{[0,0]} s_\ell) = \frac{2+3+5+3}{2+4+5+3} = \frac{13}{14}$ and $\mathbf{conf}(s_k \xrightarrow{[1,2]} s_\ell) = \frac{2+2+4+2}{2+4+5+3} = \frac{10}{14}$.

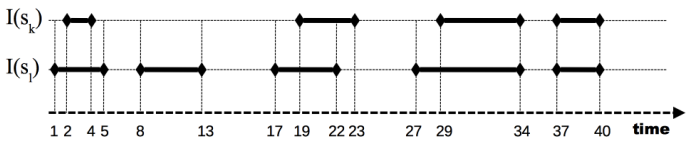


Figure 4: Example of two active interval sets.

To statistically assess the value of $\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell)$, we propose to perform a Pearson's chi-squared test of independence [36]. Considering the time interval $[t_{begin}, t_{end})$, that is to say the batch time window, the test determines whether the occurrences of s_k and $s_\ell^{[\alpha,\beta]}$ are statistically independent over $[t_{begin}, t_{end})$. At every time point of $[t_{begin}, t_{end})$, s_k and $s_\ell^{[\alpha,\beta]}$ are active or not. These two possible outcomes of a stream state s are denoted $active(s)$ and $\overline{active(s)}$. Table 1 is the contingency table O that crosses the observed outcomes of s_k and $s_\ell^{[\alpha,\beta]}$.

	$active(s_\ell^{[\alpha,\beta]})$	$\overline{active(s_\ell^{[\alpha,\beta]})}$
$active(s_k)$	$\mathbf{len}(s_k \cap s_\ell^{[\alpha,\beta]})$	$\mathbf{len}(s_k) - \mathbf{len}(s_k \cap s_\ell^{[\alpha,\beta]})$
$\overline{active(s_k)}$	$\mathbf{len}(s_\ell^{[\alpha,\beta]}) - \mathbf{len}(s_k \cap s_\ell^{[\alpha,\beta]})$	$T - \mathbf{len}(s_k) - \mathbf{len}(s_\ell^{[\alpha,\beta]}) + \mathbf{len}(s_k \cap s_\ell^{[\alpha,\beta]})$

where $T = t_{end} - t_{begin}$.

Table 1: Matrix O that contains the number of time points of $[t_{begin}, t_{end})$ for the four possible outcomes of s_k, s_ℓ.

The null hypothesis states that the occurrences of the outcomes $active(s_k)$ and $active(s_\ell^{[\alpha,\beta]})$ are statistically independent. If we suppose that $active(s_k)$ occurs uniformly over $[t_{begin}, t_{end})$, there are $\frac{\mathbf{len}(s_k)}{T}$ chances that state $s_\ell^{[\alpha,\beta]}$ is active at the same time. As $active(s_\ell^{[\alpha,\beta]})$ occurs during $\mathbf{len}(s_\ell^{[\alpha,\beta]})$ timestamps, the expected number that $s_\ell^{[\alpha,\beta]}$ occurs simultaneously with s_k under the null hypothesis is $\frac{\mathbf{len}(s_\ell^{[\alpha,\beta]}) \times \mathbf{len}(s_k)}{T}$. The three others outcomes under the null hypothesis are constructed on the same principle. All these expected outcomes E are given in table 2.

	$active(s_\ell^{[\alpha,\beta]})$	$\overline{active(s_\ell^{[\alpha,\beta]})}$
$active(s_k)$	$\frac{\mathbf{len}(s_\ell^{[\alpha,\beta]})\times\mathbf{len}(s_k)}{T}$	$\frac{\left(T-\mathbf{len}(s_\ell^{[\alpha,\beta]})\right)\times\mathbf{len}(s_k)}{T}$
$\overline{active(s_k)}$	$\frac{\mathbf{len}(s_\ell^{[\alpha,\beta]})\times(T-\mathbf{len}(s_k))}{T}$	$\frac{\left(T-\mathbf{len}(s_\ell^{[\alpha,\beta]})\right)\times(T-\mathbf{len}(s_k))}{T}$

Table 2: Matrix E that contains the number of time points of $[t_{begin}, t_{end})$ for the four possible outcomes of s_k, $s_\ell^{[\alpha,\beta]}$ under the null hypothesis.

The value of the test-statistic is

$$\begin{aligned} X^2 &= \sum_{i=1}^{2}\sum_{j=1}^{2}\frac{(O_{ij}-E_{ij})^2}{E_{ij}} \\ &= \frac{T\left(T\,\mathbf{len}\left(s_k\cap s_\ell^{[\alpha,\beta]}\right)-\mathbf{len}(s_k)\mathbf{len}(s_\ell^{[\alpha,\beta]})\right)^2}{\mathbf{len}(s_k)\mathbf{len}(s_\ell^{[\alpha,\beta]})(T-\mathbf{len}(s_k))(T-\mathbf{len}(s_\ell^{[\alpha,\beta]}))} \end{aligned} \quad (1)$$

The null distribution of the statistic is approximated by the χ^2 distribution with 1 degree of freedom, and for a significant level of 5%, the critical value is equal to $\chi^2_{0.05} = 3.84$. Consequently, X^2 has to be greater than 3.84 to established that the active interval set intersection is sufficiently large not to be due to chance. From equation (1) we derive the following quadratic equation in $\mathbf{len}\left(s_k\cap s_\ell^{[\alpha,\beta]}\right)$:

$$\begin{aligned} &\left(T\,\mathbf{len}\left(s_k\cap s_\ell^{[\alpha,\beta]}\right)-\mathbf{len}(s_k)\mathbf{len}(s_\ell^{[\alpha,\beta]})\right)^2 \geq \\ &\frac{3.84}{T}\mathbf{len}(s_k)\mathbf{len}(s_\ell^{[\alpha,\beta]})(T-\mathbf{len}(s_k))(T-\mathbf{len}(s_\ell^{[\alpha,\beta]})) \end{aligned}$$

which is satisfied iff $0 \leq \mathbf{len}\left(s_k\cap s_\ell^{[\alpha,\beta]}\right) \leq \cap_1$ or $T \geq \mathbf{len}\left(s_k\cap s_\ell^{[\alpha,\beta]}\right) \geq \cap_2$, $\cap_1$ and $\cap_2$ been the roots[1] of this equation.

Intersection values that range between 0 and $\cap_1$ are much smaller than the one expected under the null hypothesis. Such values can be used to detect anomalies, but, in the following we focus on the intersection values that are unexpectedly high. Therefore, we conclude that a temporal dependency $s_k \xrightarrow{[\alpha,\beta]} s_\ell$ is valid iff

$$\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell) \geq \frac{\cap_2}{\mathbf{len}(s_k)} \quad (2)$$

As the χ^2 test only works well when the dataset is large enough, we use the conventional rule of thumb [36] that enforces all the expected numbers (cells in Table 2) to be greater than 5.

[1] $\cap_i = \frac{\mathbf{len}(s_k)\mathbf{len}(s_\ell^{[\alpha,\beta]}) \overset{+}{-} \sqrt{\frac{3.84}{T}\mathbf{len}(s_k)\mathbf{len}(s_\ell^{[\alpha,\beta]})(T-\mathbf{len}(s_k))\left(T-\mathbf{len}(s_\ell^{[\alpha,\beta]})\right)}}{T}$.

2.4 Significant temporal dependencies selection

For two stream states in temporal dependency, a huge number of shifting intervals $[\alpha, \beta]$ may exist that result in valid temporal dependencies. These intervals may describe distinct temporal dependencies (e.g., different paths may exist between two motion captors producing as many temporal dependencies), but they can also be redundant, depicting the same phenomenon several times. Redundancy between shifting intervals mainly relies on the following property:

Property 1 (Confidence monotonicity) *Let s_k and s_ℓ be two data stream states and $[\alpha_1, \beta_1]$, $[\alpha_2, \beta_2]$ be two shifting intervals. If $[\alpha_1, \beta_1] \subseteq [\alpha_2, \beta_2]$, then* $\mathbf{conf}(s_k \xrightarrow{[\alpha_1,\beta_1]} s_\ell) \leq \mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)$.

Proof 1 *$[\alpha_1, \beta_1] \subseteq [\alpha_2, \beta_2]$ implies that $I_{[\alpha_1,\beta_1]}(s_\ell) \subseteq I_{[\alpha_2,\beta_2]}(s_\ell)$ and* $\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]} \cap s_k) \leq \mathbf{len}(s_\ell^{[\alpha_2,\beta_2]} \cap s_k)$. *As a result,* $\mathbf{conf}(s_k \xrightarrow{[\alpha_1,\beta_1]} s_\ell) \leq \mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)$.

□

To be useful in real data streams, we wish our mining process to automatically discover the shifting intervals that best describe the temporal dependencies of two given states while avoiding the pattern flooding that may result from the computation of all valid temporal dependencies. Among all the shifting intervals included in $[t_{min}, t_{max}]$ that lead to valid temporal dependencies, those that are of interest should have (1) a high confidence value and (2) be as specific as possible with respect to the inclusion relation. This leads to the following definition of the *dominance* relationship on the set of temporal dependencies.

Definition 2 (Dominance relationship) *Let $d_{[\alpha_1,\beta_1]} = s_k \xrightarrow{[\alpha_1,\beta_1]} s_\ell$ and $d_{[\alpha_2,\beta_2]} = s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell$ be two temporal dependencies between s_k and s_ℓ. We say that $d_{[\alpha_1,\beta_1]}$ dominates $d_{[\alpha_2,\beta_2]}$, $d_{[\alpha_1,\beta_1]} \preceq d_{[\alpha_2,\beta_2]}$, iff $[\alpha_1, \beta_1] \subseteq [\alpha_2, \beta_2]$ and*

$$1 - \frac{\mathbf{conf}(s_k \xrightarrow{[\alpha_1,\beta_1]} s_\ell)}{\mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)} < 1 - \frac{\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_\ell^{[\alpha_2,\beta_2]})} \tag{3}$$

The rationale behind this definition is that when $[\alpha_1, \beta_1]$ dominates $[\alpha_2, \beta_2]$, the loss of the confidence measure of $[\alpha_1, \beta_1]$ is less than the reduction of its active interval set length and thus $I_{[\alpha_2,\beta_2]\setminus[\alpha_1,\beta_1]}(s_\ell) \cap I(s_k)$ is almost empty. Indeed, if the reduction of the active interval set length of $s_\ell^{[\alpha,\beta]}$ is uniformly distributed over $[t_{min}, t_{max}]$, then the length of its intersection with $I(s_k)$ will be reduced in the same proportion:

$$\frac{\mathbf{len}(s_k \cap s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_k \cap s_\ell^{[\alpha_2,\beta_2]})} \approx \frac{\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_\ell^{[\alpha_2,\beta_2]})}$$

However, if the reduction of the active interval set length of $s_\ell^{[\alpha,\beta]}$ mainly occurs when s_k is not active, then the length of its intersection with $I(s_k)$ decreases less than its active interval set length:

$$1 - \frac{\mathbf{len}(s_k \cap s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_k \cap s_\ell^{[\alpha_2,\beta_2]})} < 1 - \frac{\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_\ell^{[\alpha_2,\beta_2]})}$$

This dominance relationship makes it possible to refine an interval while controlling the loss of the confidence measure. If an interval reduction leads to a significant loss, then the refinement process has to be stopped, since the portion of $I(s_k)$ non covered by the interval will not be subsequently either. Therefore, significant temporal dependencies are the most specific temporal dependencies that dominate all their supersets:

Definition 3 (Significant temporal dependencies) *For two states s_k and s_ℓ, let Σ be the set of temporal dependencies $d_{[\alpha,\beta]} = s_k \xrightarrow{[\alpha,\beta]} s_\ell$ such that (i) $d_{[\alpha,\beta]}$ dominates all of its supersets, and (ii) every superset of $d_{[\alpha,\beta]}$ dominates its supersets as well:*

$$\begin{aligned}\Sigma = \{d_{[\alpha_1,\beta_1]} \mid \quad & \forall \quad [\alpha_2,\beta_2] \text{ such that } [\alpha_1,\beta_1] \subseteq [\alpha_2,\beta_2],\ d_{[\alpha_1,\beta_1]} \preceq d_{[\alpha_2,\beta_2]} \\ \text{and} \quad & \forall \quad [\alpha_3,\beta_3] \text{ such that } [\alpha_2,\beta_2] \subseteq [\alpha_3,\beta_3],\ d_{[\alpha_2,\beta_2]} \preceq d_{[\alpha_3,\beta_3]}\}\end{aligned}$$

Temporal dependencies that belong to the positive border of $(\Sigma, \preceq)$ are said to be significant.

Property 2 (Σ-belonging monotonicity) *Let $[\alpha_1,\beta_1] \subseteq [\alpha_2,\beta_2]$. If $d_{[\alpha_1,\beta_1]}$ belongs to Σ then $d_{[\alpha_2,\beta_2]} \in \Sigma$.*

Proof 2 *It is derived from definition 3.*

□

Property 3 (Dominance transitivity) *Let $d_{[t_i,t_j]}$ designate the temporal dependency $s_k \xrightarrow{[t_i,t_j]} s_\ell$. For all intervals $[\alpha,\beta]$ such that $[\alpha_1,\beta_1] \subseteq [\alpha,\beta] \subseteq [\alpha_2,\beta_2]$, if $d_{[\alpha_1,\beta_1]} \preceq d_{[\alpha,\beta]}$ and $d_{[\alpha,\beta]} \preceq d_{[\alpha_2,\beta_2]}$ then $d_{[\alpha_1,\beta_1]} \preceq d_{[\alpha_2,\beta_2]}$.*

Proof 3

$$\left(\frac{\mathbf{conf}(s_k \xrightarrow{[\alpha_1,\beta_1]} s_\ell)}{\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell)} > \frac{\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_\ell^{[\alpha,\beta]})}\right) \frac{\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell)}{\mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)}$$

$$\Rightarrow \frac{\mathbf{conf}(s_k \xrightarrow{[\alpha_1,\beta_1]} s_\ell)}{\mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)} > \frac{\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_\ell^{[\alpha,\beta]})} \frac{\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell)}{\mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)}$$

and

$$\left(\frac{\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell)}{\mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)} > \frac{\mathbf{len}(s_\ell^{[\alpha,\beta]})}{\mathbf{len}(s_\ell^{[\alpha_2,\beta_2]})}\right) \frac{\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_\ell^{[\alpha,\beta]})}$$

$$\Rightarrow \frac{\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_\ell^{[\alpha,\beta]})} \frac{\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell)}{\mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)} > \frac{\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_\ell^{[\alpha_2,\beta_2]})}$$

Therefore, $\frac{\mathbf{conf}(s_k \xrightarrow{[\alpha_1,\beta_1]} s_\ell)}{\mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)} > \frac{\mathbf{len}(s_\ell^{[\alpha_1,\beta_1]})}{\mathbf{len}(s_\ell^{[\alpha_2,\beta_2]})}$.

□

3 Efficient Temporal Dependencies Discovery

Discovering temporal dependencies is time-consuming for large volumes of data. Indeed, a naive algorithm, that looks for dependencies between two stream states s_k and s_ℓ, will explore all possible time shift intervals included in $[t_{min}, t_{max}]$ (the number of such intervals is in $\Theta((t_{max} - t_{min})^2)$). For each interval, it will compute its confidence value, which can be done in $\Theta(\#I)$, where $\#I$ is the number of active intervals of s_k or s_ℓ. Such an algorithm has to be executed with a relatively high frequency over data stream batches. Therefore, it is a key issue to improve its efficiency to make it suitable for the context of multiple data streams. To do that we propose TEDDY, TEmporal Dependency DiscoverY, an algorithm that (1) takes advantage of the monotonic characteristic of the confidence measure, as stated in property 1, to avoid considering time shift intervals that are guaranteed not to be valid; (2) exploits an upper bound on the confidence measure, whose complexity is $\mathcal{O}(1)$, to reduce the computation required for the confidence evaluation; (3) explores the search space using a level-wise approach in order to discover significant temporal dependencies while computing the confidence value of each interval at the most once.

TEDDY is sketched in Algorithm 1. For every pair of states, it explores the temporal dependencies in a breadth-first approach. The inclusion operation over time shift intervals defines a semi-lattice, illustrated in Figure 5, where intervals at given depth d have the same length: $t_{max} - t_{min} - d$. This semi-lattice is traveled level by level. At each iteration of the loop *while*, Cand_d contains the d-depth shift interval candidates. Line 7, A_d is computed as the restriction of Cand_d to the dependencies whose confidence value is greater than a lower bound. If a temporal dependency from A_d dominates its two ancestors, then it is a promising dominant candidate and thus belongs to Σ_d (line 8). As such, it is added to the *Border* set whereas its ancestors are removed, as they are no longer the most specific intervals of *Border*. Line 9, $d+1$-depth candidates are generated if their d-depth ancestors belongs to Σ_d. Line 12 processes *Border* only to extract valid and significant temporal dependencies.

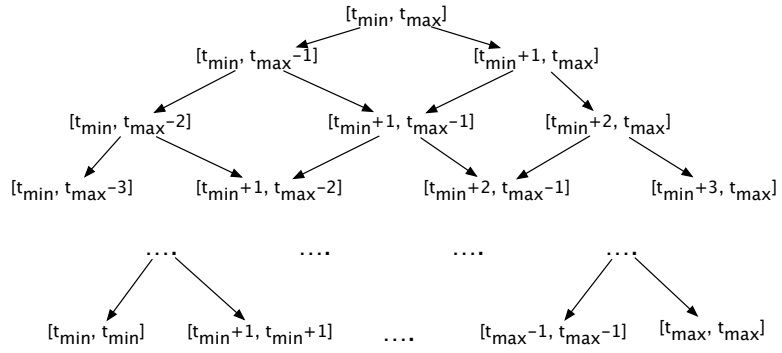


Figure 5: Interval shift join semi-lattice with respect to $\subseteq$.

The four most important steps of this algorithm, the candidate generation, the pruning based on confidence and dominance, and the identification of valid and significant dependencies, are detailed in the following subsections.

Algorithm 1 `TEDDY`

Require: A time interval $[t_{begin}, t_{end})$, $\mathcal{DS} = \{DS_1^{[t_{begin},t_{end})}, \ldots, DS_n^{[t_{begin},t_{end})}\}$ the data stream batch, t_{min}, t_{max}.
Ensure: All significant temporal dependencies over $\mathcal{DS}$.

```
for all s_i ∈ FS do
  for all s_j ∈ FS do
    Border ← ∅
    Cand_0 ← [t_min, t_max]
    d ← 0
    while Cand_d ≠ ∅ do
      A_d ←Pruning_based_on_confidence(Cand_d)
      [Σ_d, Border] ←Pruning_based_on_dominance(A_d, Border)
      Cand_{d+1} ←Candidate_generation(Σ_d)
      d ← d + 1
    end while
    Significant_{s_i,s_j} ←Compute_valid_and_significant_TD(Border)
  end for
end for
return ⋃_{s_i,s_j} Significant_{s_i,s_j}
```

3.1 Candidate time shifts generation

As stated in property 1, the confidence measure increases monotonically with time shift interval inclusion. In addition, property 2 stipulates that Σ-belonging is also a monotonic property. So, to prune the search space made of temporal

dependencies that are not valid or not significant, the interval semilattice is traversed from the largest interval down to the singletons. If a time shift interval is not valid or does not dominate one of its direct ancestors, then none of the intervals included in it can correspond to a valid significant temporal dependency. As each interval at depth $d+1$ is included in at most two intervals at depth d, we generate $d+1$-depth candidate by intersecting d-depth promising time shifts. Algorithm 2 presents the candidate generation procedure. The first test (line 2) checks that the bottom of the semilattice is not reached. It processes L, the list of promising d-depth intervals ordered by their first endpoint. If the first (resp. last) interval, lines 3-5 (resp. lines 15-17), is $[t_{min}, t_{max} - d]$ (resp. $[t_{min} + d, t_{max}]$), then $[t_{min}, t_{max} - (d+1)]$ (resp. $[t_{min} + (d+1), t_{max}]$) is a $d+1$-depth candidate as its only one ancestor belongs to L. The loop (lines 7-14) generates all other temporal dependencies whose two direct ancestors are in L by intersecting their time shift intervals.

Algorithm 2 `Candidate_generation`

Require: L, the list of promising d-depth intervals, ordered by their first endpoint.
Ensure: Cand, the list of $d+1$-depth candidate intervals.

```
1:  Cand← ∅
2:  if t_max − t_min > d then
3:     if (L[0] = [t_min, t_max − d]) then
4:        Cand← Cand ∪ [t_min, t_max − (d + 1)]
5:     end if
6:     i ← 0
7:     while i < #L − 1 do
8:        [α, β] ← L[i]
9:        [γ, δ] ← L[i + 1]
10:       if (α = γ − 1) and (β = δ − 1) then
11:          Cand← Cand ∪ [γ, β]
12:       end if
13:       i ← i + 1
14:    end while
15:    if (L[#L − 1] = [t_min + d, t_max]) then
16:       Cand← Cand ∪ [t_min + d + 1, t_max]
17:    end if
18: end if
19: return Cand
```

3.2 Pruning-based on confidence measure

In order to avoid the computation of the confidence values of unpromising dependencies, we consider the following property, that bounds the difference of confidence between two time shift intervals:

Property 4 (Bounds on confidence) *Let s_k and s_ℓ be two data stream states, and $[\alpha_1, \beta_1]$ and $[\alpha_2, \beta_2]$ be two time intervals:*

$$|\mathbf{conf}(s_k \xrightarrow{[\alpha_1,\beta_1]} s_\ell) - \mathbf{conf}(s_k \xrightarrow{[\alpha_2,\beta_2]} s_\ell)| \leq \frac{(|\alpha_1 - \alpha_2| + |\beta_1 - \beta_2|) \times \#I(s_\ell)}{\mathbf{len}(s_k)}$$

where $\#I(s_\ell)$ represents the number of intervals in $I(s_\ell)$.

Proof 4 *By shifting an interval $[t_i+\alpha_1, t_j+\beta_1] \in I_{[\alpha_1,\beta_1]}(s_\ell)$ with $[\alpha_2-\alpha_1, \beta_2-\beta_1]$, the length of the resulting interval may win or lose a maximum of $(|\alpha_1 - \alpha_2|+|\beta_1-\beta_2|)$ time units. By multiplying this quantity by the number of intervals in $I(s_\ell)$, the result follows.*

□

Furthermore, as stated by equation (2) page 8, valid temporal dependencies have a confidence value greater than

$$\text{MinConfidence}\,(L(\alpha,\beta)) \equiv \frac{\lambda L(\alpha,\beta) + \sqrt{\frac{3.84}{T}\lambda(T-\lambda)L(\alpha,\beta)(T-L(\alpha,\beta))}}{\lambda T}$$

where $L(\alpha,\beta) = \mathbf{len}(s_\ell^{[\alpha,\beta]})$ and $\lambda = \mathbf{len}(s_k)$. Property 5 provides a lower bound on $\text{MinConfidence}\,(L(\alpha,\beta))$:

Property 5 (Lower bound on MinConfidence $(L(\alpha,\beta))$)

$$\textit{MinConfidence}\,(L(\alpha,\beta)) \geq \min\,(1, \textit{MinConfidence}\,(L(0,0)))$$

Proof 5 *$L(\alpha,\beta)\,(T - L(\alpha,\beta))$ is a quadratic function which vanishes at $L(\alpha,\beta) = 0$ and $L(\alpha,\beta) = T$. Therefore, MinConfidence $(L(\alpha,\beta))$ first increases and then decreases over $[0,T]$ with MinConfidence $(0) = 0$ and MinConfidence $(T) = 1$. Let $x_1 < T$ be such that MinConfidence $(x_1) = 1$. We can observe that MinConfidence (x) increases over $[0,x_1]$ (see Figure 6). As $L(\alpha,\beta) \geq L(\alpha,\alpha) = L(0,0)$, we have MinConfidence $(L(\alpha,\beta)) \geq \min\,(1, \textit{MinConfidence}\,(L(0,0)))$.*

□

$\mathbf{conf}(s_k \xrightarrow{[\alpha,\beta]} s_\ell)$ is upper bounded by 1, therefore if MinConfidence > 1, there is no valid temporal dependency. Algorithm 3 details the evaluation of the confidence measure. The confidence value of the first candidate is computed (line 4). Then, the confidence value of the following candidates is estimated based on Property 4 (line 7). If the upper-bound (`lastConf + maxGain`) of the confidence value of a candidate is lower than $\text{MinConfidence}\,(L(0,0))$ (`boundMinConfidence`, estimated thanks to property 5), then the candidate cannot be valid. Otherwise, its exact confidence is evaluated (line 10) and, if it is greater than `boundMinConfidence` (line 11), the candidate is considered as a promising valid temporal dependency. Notice that the confidence measure is stored for future needs (line 12). This confidence value is used as a new reference for further `maxGain` evaluations, since `maxGain` tends to decrease when evaluated on distant intervals in Cand.

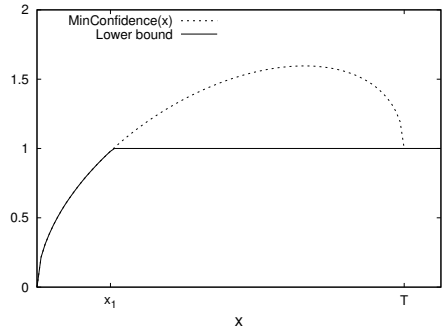


Figure 6: Illustration of MinConfidence function.

3.3 Pruning-based on dominance relationships

`Pruning_based_on_dominance` function consists simply in evaluating whether each promising candidate satisfies equation (3) for its direct ancestors. Actually, property 3 states that if a temporal dependency dominates its direct ancestors, then it also dominates all its ancestors, and thus belongs to Σ. It is also added to the *Border* set whereas its ancestors are removed, as they are no more the most specific temporal dependencies of Σ.

3.4 Identification of valid and significant dependencies

The last step of TEDDY is to consider the temporal dependencies of *Border* to ensure they are valid, that is, they truly satisfy equation (2). Algorithm 4 states that if a temporal dependency is valid (line 7) and more specific than any other dependencies of R (line 8), then it is added to R (line 9) and temporal dependencies that are more general are removed from R (line 10). If the dependency is not valid (line 12), its direct ancestors are recursively considered in lines 14 and 17.

If R is implemented as an interval tree, evaluating that a temporal dependency is the most specific among the n elements of R can be done in $O(\log(n))$. Finding all the dependencies of R that are more general than $d_{[\alpha,\beta]}$ can be done in $O(\min(n, k\log(n)))$ where k is the number of dependencies in the output list [12].

4 Experimental Study

In this section, we report experimental results to illustrate the interest of our approach. We start by describing four synthetic and two real-life datasets we use, as well as the questions we aim to answer. Then, we provide a performance study and give some qualitative results. All experiments were performed on a

Algorithm 3 `Pruning_based_on_confidence`

Require: Cand, an ordered list of candidate intervals, $\#I(s_j)$, the number of intervals in $I(s_j)$ and $\mathbf{len}(s_i)$.

Ensure: R, the set of promising valid temporal dependencies and their confidence value.

$R \leftarrow \emptyset$
$k \leftarrow 0$
$[\alpha, \beta] \leftarrow$Cand$[k]$
lastConf$\leftarrow \mathbf{conf}(s_i \xrightarrow{[\alpha,\beta]} s_j)$
while $k < \#Cand$ **do**
  $[\alpha_k, \beta_k] \leftarrow$Cand$[k]$
  maxGain$\leftarrow (|\alpha - \alpha_k| + |\beta - \beta_k|) \times \frac{\#I(s_j)}{\mathbf{len}(s_i)}$
  if (lastConf + maxGain $\geq$ boundMinConfidence **then**
    $[\alpha, \beta] \leftarrow$Cand$[k]$
    lastConf$\leftarrow \mathbf{conf}(s_i \xrightarrow{[\alpha,\beta]} s_j)$
    if (lastConf $\geq$ boundMinConfidence **then**
      Cand$[k]$.confidence$\leftarrow$ lastConf
      $R \leftarrow R \cup Cand[k]$
    end if
  end if
  $k \leftarrow k + 1$
end while
return R

8 GB RAM computer with a octo-core processor cadenced at 3 GHz, running Windows 7. TEDDY algorithm is implemented in standard C++.

4.1 Dataset description

Synthetic datasets

To generate the synthetic datasets, we built a simulator of a sensor surveillance network. It consists in the simulation of 8 video cameras that record what is going on in a rectangular space. Each camera captures the images of an elliptical area of this space (see Figure 7). The simulation consists in moving objects along eight predefined rectilinear paths. To control the number of events occurring per time unit, objects are generated according to a Poisson distribution. The area covered by each camera is divided into subareas that are as many data sources. In total, there are 216 data sources that produce events in our experiments. A data source produces an event *"object detected"* every time step in which an object is located in the associated subarea and the corresponding stream is in the state *"object detected"*. At the other time steps, the stream is in the state *"no object detected"*. In the following, we are interested only in dependencies between *"object detected"* states.

Algorithm 4 `Compute_valid_and_significant_TD`

Require: $Border$
Ensure: R the set of valid and significant TD.
1: $R \leftarrow \emptyset$
2: **for all** $[\alpha, \beta] \in Border$ **do**
3: `Confident_and_most_specific`$([\alpha, \beta], R)$
4: **end for**
5: **return** R

Function `Confident_and_most_specific`$([\alpha, \beta], R)$

6: **if** $depth([\alpha, \beta]) \geq 0$ **then**
7: **if** $([\alpha, \beta]$.confidence $\geq$ MinConfidence(α, β) **then**
8: **if** is_most_specific$([\alpha, \beta],\ R)$ **then**
9: insert$([\alpha, \beta], R)$
10: $R \leftarrow R \setminus$ set_of_most_general$([\alpha, \beta], R)$
11: **end if**
12: **else**
13: **if** $\alpha - 1 \geq t_{min}$ **then**
14: `Confident_and_most_specific`$([\alpha - 1, \beta], R)$
15: **end if**
16: **if** $\beta + 1 \leq t_{max}$ **then**
17: `Confident_and_most_specific`$([\alpha, \beta + 1], R)$
18: **end if**
19: **end if**
20: **end if**

We generate four different datasets, which differ from the average number of events produced per minute and per stream. Quantitative characteristics of these datasets are given in Table 3, lines SYNT02, SYNT04, SYNT08, SYNT16.

Real-World datasets

CASAS Smart Home project, a multi-disciplinary research project at Washington State University [11], uses a physical testbed (three bedrooms, one bathroom, a kitchen, a dinning room), depicting a smart home where sensors are used to monitor different daily activities. Various experiments performed in this physical testbed produced different datasets, called *Tokyo*, *Milan*, *Aruba* and so on. In the following, we focus on a single dataset called **Milan** for which the residents in the home were a woman and a dog. The woman's children visited on several occasions. We consider only the data produced by the motion sensors and by the contact switch sensors located on the doors. For each room, there are between 20 to 70 data streams producing a total of 433 665 events over 3 months. Data streams are split into batches of one day (24 hours). Figure 8 (left) reports on the distribution on these events throughout the week.

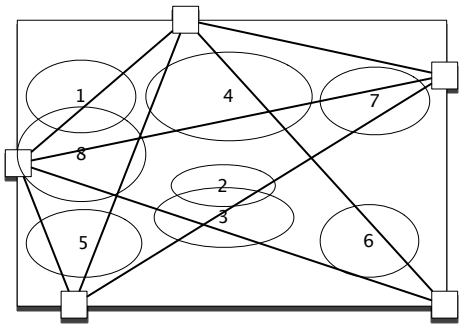


Figure 7: Synthetic testbed: a rectangular space is equipped with 8 video cameras. Objects are moving along 8 rectilinear trajectories.

Dataset	# Data sources	# Events	Duration	Avg events
SYNT02	216	85,806	3 hours	2
SYNT04	216	173,645	3 hours	4
SYNT08	216	352,553	3 hours	8
SYNT16	216	696,677	3 hours	18
Milan	31	433,665	three months	0.1
Foxstream	1604	33,278,036	one week	2.1

Table 3: Dataset characteristics. The last column shows the average number of events per minute and per stream.

Foxstream is a real-world dataset obtained from the company Foxstream. It depicts one week of activity of a smart environment: two video-cameras, two thermographic cameras and four motion sensors are used for outdoor surveillance of a building, whereas another motion sensor is located inside it. The area captured by each camera is partitioned into 400 rectangular subareas that correspond to as many data sources. The movement detection is carried out as follows. For each subarea, we compute a background image by averaging the last 50 frames. Every second, the new image is compared with the background. A pixel is considered to have changed if its difference from the background corresponding pixel is greater as a given threshold (15 in our case). If the subarea has at least 75% of its pixels that have changed, an event *"motion detected"* is produced. We emphasize that motion detection in video is still a challenging problem. Therefore, this pre-processing may produce erroneous data: for instance, spurious motions can be detected due to changes in weather or lighting (rain, shadow). During the one-week period, the cameras generated about 33 million events, whereas the motion sensors produce around 38 000 events (see Table 3). Data streams are split into batches of one hour. Figure 8 (right) reports the distribution on these events through the different batches of this week.

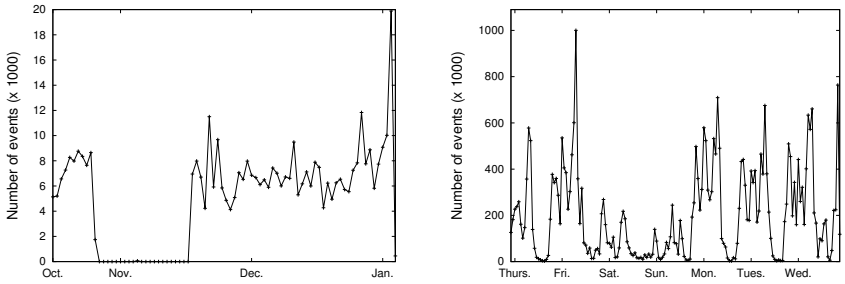


Figure 8: Number of events per batch: **Milan** (left) (Batch_size=24h) and **Foxstream** (right) (Batch_size=1h) datasets.

4.2 Aims

We analyze the experimental results with regard to the following questions:

- *What is the efficiency of TEDDY with regard to dataset characteristics that may affect its execution time?* At the beginning of section 3, we mention that a baseline algorithm, that explores all possible time intervals included in $[t_{min}, t_{max}]$, would have a time complexity function that is quadratic with regard to $t_{max} - t_{min}$ and linear with the number of intervals in the active interval sets. Therefore, it is interesting to know whether TEDDY outperforms the baseline algorithm and to compare their empirical complexities. Notice that both algorithms have the same complexity in the worth case, but TEDDY uses pruning techniques that should increase its efficiency in practice. As the number of intervals in active interval sets increases with batch size, in the following, we study the behavior of TEDDY with regard to t_{max} and batch size.

- *How effective are TEDDY's pruning properties?* We carry out a detailed study of the impact of each pruning technique on TEDDY's performance.

- *Does TEDDY scale?* We want to investigate the scalability property of TEDDY's execution time.

- *Is TEDDY robust when data are noisy?* We analyze TEDDY's ability to discover temporal dependencies in noisy data.

- *What about TEDDY's temporal dependencies?* Last but not least, we examine the ability of TEDDY's temporal dependencies to account for the phenomena recorded in the data. To that end, we construct a graph whose nodes are the data stream states and whose arcs are the extracted temporal dependencies (i.e., TEDDY's output). We demonstrate that such a graph is a useful tool to describe a system made of heterogeneous datasources. Beside experiments on real-world data to establish that the discovered temporal dependencies make sense, we aim to highlight how

temporal dependencies can be used to detect abnormal measurements or sensor failures.

The experimental study we report aims at answering all the above questions.

4.3 Quantitative experiment results

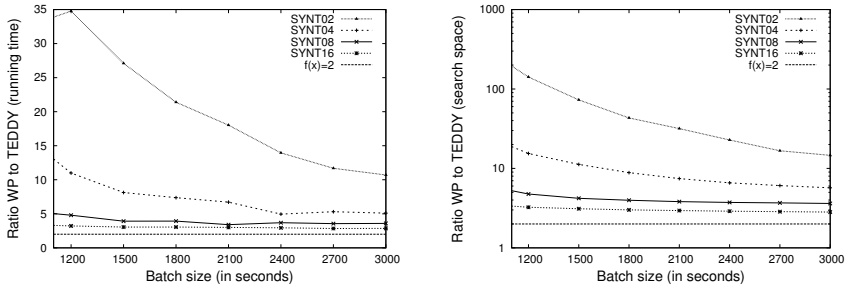
In this section, we study the behavior of TEDDY with respect to various parameters: the frequency of events, the size of the batches (Batch_size) and t_{max}. In all the experiments, t_{min} is set to 0. Besides, we examine the impact of the constraints that define valid and significant temporal dependencies (the χ^2 assessment of the confidence measure and the non-dominated constraint) on the search space size as well as on the execution time of TEDDY. To this end, the four following configurations of algorithm 1 are studied:

1. **WP** (without pruning): lines 7 and 8 are removed and all possible temporal dependencies are considered.
2. **Chi2** (χ^2-based pruning): line 8 is removed and only the constraint on the confidence measure is pushed aside to reduce the search space.
3. **Gradient** (dominance-based pruning): line 7 is removed and only the dominance constraint makes it possible to discard unpromising dependencies.
4. **TEDDY**: both pruning constraints are fully exploited as presented in algorithm 1.

4.3.1 Evaluation of the pruning efficiency

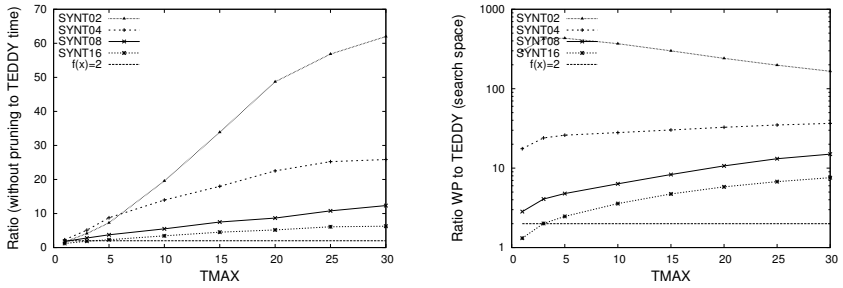
There is no other algorithm that computes temporal state dependencies using the same constraints as in our proposal. Therefore, we first study the performance of TEDDY in comparison with the baseline algorithm. This algorithm considers all possible temporal dependencies and removes, in a post-treatment, the non valid or non significant dependencies. For these experiments, we do not take into account the execution time required by this post-processing step.

Figures 9 and 10 depict the behavior of TEDDY when Batch_size and t_{max} vary. In each figure, the running time and search space size ratios of WP to TEDDY are evaluated on the synthetic datasets. Each value is averaged over all the batches of the same size. In most cases, TEDDY is at least twice as fast as WP. The ratio of the execution time increases with t_{max} since the number of possible intervals is quadratic in $t_{max} - t_{min}$ and TEDDY is able to prune a large part of them early on. On the contrary, when Batch_size increases, the ratio tends to decrease since the number of active intervals $\#I$ of each state tends to increase and TEDDY is not able to prune the search space as much. Indeed, maxGain increases linearly with $\#I$ and the condition at line 8 of algorithm 3 tends to be always true which implies that the time interval cannot be pruned. Furthermore, from Figures 9 and 10, we can also notice that the



(a) (b)

Figure 9: Comparison of TEDDY and WP w.r.t. Batch_size ($t_{min} = 0$, $t_{max} = 10$): execution time ratio (a), search space size ratio (b).



(a) (b)

Figure 10: Comparison of TEDDY and WP w.r.t. t_{max} (Batch_size=900, $t_{min} = 0$): execution time ratio (a), search space size ratio (b).

denser the datasets, the lower the ratios are. Actually, the number of extracted dependencies increases with the dataset density as well as the portion of the search space considered by TEDDY, as shown in the following figures.
Figures 11 and 12 show the proportion of the search space explored by TEDDY. Among the pruned candidates, we make a distinction between those removed thanks to the chi2-based constraint and those discarded by the gradient-based constraint. These quantities are evaluated with respect to Batch_size (Figure 11) and t_{max} (Figure 12). A first observation is that the number of candidates avoided thanks to the two constraints is much higher than the number of dependencies considered by TEDDY, except when the dataset is very dense and t_{max} very small. The gradient constraint is even more efficient when the dataset density increases or the values of Batch_size and t_{max} grow. Indeed, while the batch size increases, the number of candidates avoided with gradient-based constraint increases or remains stable whatever the dataset density. This pruning criterion becomes even more effective when t_{max} increases. The larger the length of a pruned interval, the greater the size of the pruned search space. Indeed, if an interval $[\alpha, \beta]$ does not dominate one of its direct ancestors, it is pruned

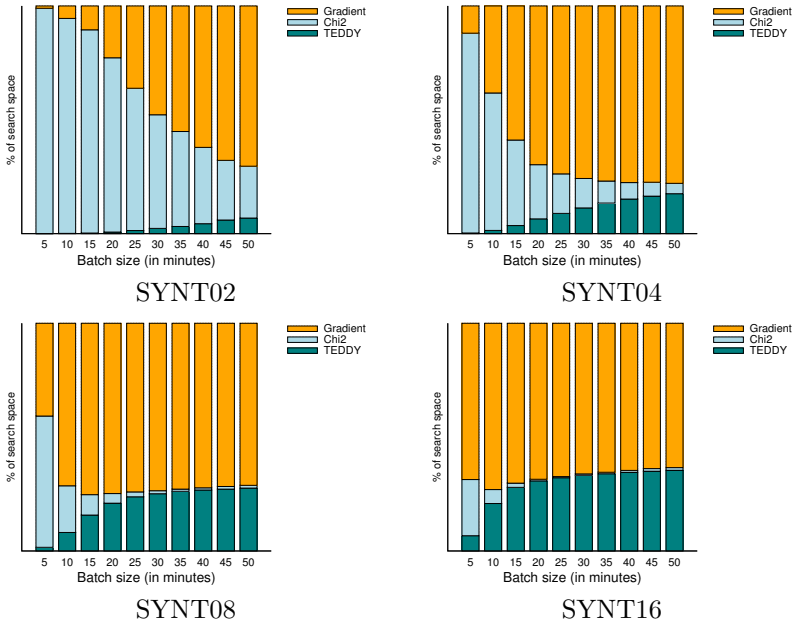


Figure 11: Impact of each constraint on search space size with regard to Batch_size: number of candidates pruned by Chi2 and Gradient, and number of candidates considered by TEDDY.

by the gradient-based constraint as well as $\frac{(\beta-\alpha)\times(\beta-\alpha+1)}{2} - 1$ other candidates, that is to say all the dependencies that are below $[\alpha, \beta]$ in the semi-lattice. Beside, chi2-based pruning tends to be less efficient when t_{max} and/or Batch_size increase. As explained above, this is due to maxGain that increases with the time interval length and the number of active intervals of the stream state.

Figure 13 reports the execution time of TEDDY and the average number of dependencies discovered per state pair and batch when Batch_size varies. Figure 14 also studies the performance TEDDY but according to the value of t_{max}. In a manner consistent with what has been observed from Figures 11 and 12, TEDDY benefits from the two pruning techniques in obtaining very good time performance. Notice that, for each dataset the execution time is always much lower than the batch size (at least 200 times). Therefore the temporal state dependencies computation is faster than the data acquisition process, which is a prerequisite to data stream mining techniques. For dense datasets (SYNT08 and SYNT16), the number of extracted dependencies increases with Batch_size and t_{max}. For SYNT02 and SYNT04, the number of dependencies decreases with Batch_size: some values may become statistically insignificant on a large batch of data, when the density of events is low. The number of dependencies also decreases slightly with t_{max}. This is due to the gradient-based constraint: a dependency satisfies this constraint if it dominates all its ancestors. Thus,

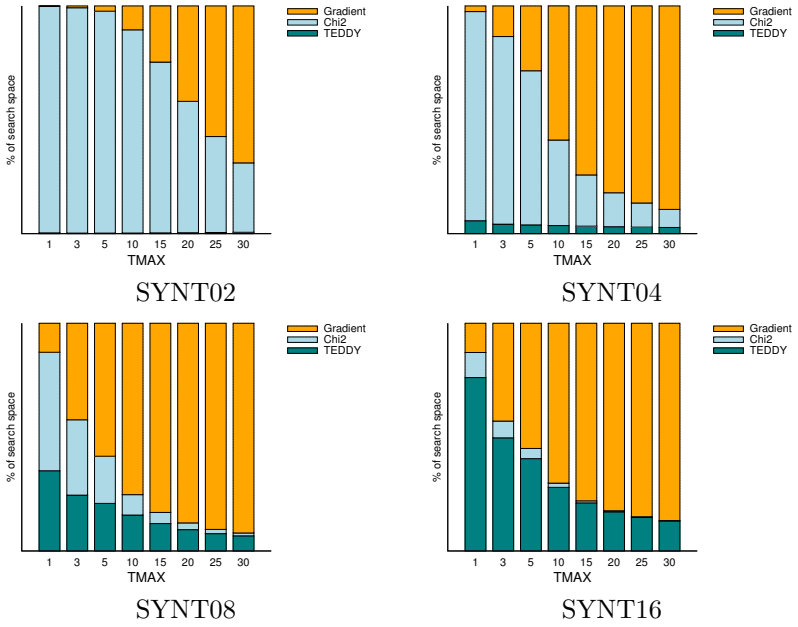


Figure 12: Impact of each constraint on search space size with regard to t_{max}: number of candidates pruned by Chi2 and Gradient, and number of candidates considered by TEDDY .

as the number of ancestors increases with t_{max}, the constraint may become unsatisfied, especially when the dataset density is small.

These first series of experiments are very conclusive and we can argue that our approach scales up well with regard to t_{max} and Batch_size parameters.

4.3.2 Robustness to noise

This empirical study also aims to investigate TEDDY's robustness against noise. We assume that synthetic datasets described in Table 3 are noiseless, having been generated following a specific scenario in a testbed. We introduce a uniform random noise in each dataset by adding x% of spurious events, $x \in \{1, 3, 5, 10, 20, 30\}$. It consists in adding p *"object detected"* events and q *"no object detected"* ones, where p is the proportion of object detected events in the stream times x% and q is the proportion of "no object detected" events in the stream times x%. Therefore, the probability of occurrence of each event type is maintained throughout the process. Based on our previous assumption, we suppose that the set of valid and significant dependencies of the original dataset are those expected and constitute the relevant dependencies. TEDDY's robustness is thus evaluated by computing the recall and the precision of this set of dependencies. To declare whether two temporal dependencies $s_k^1 \xrightarrow{[\alpha^1,\beta^1]} s_\ell^1$

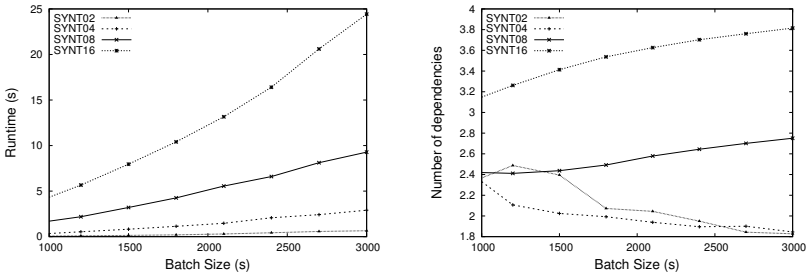


Figure 13: Runtime (left) and average number of discovered dependencies per state pair and batch (right) with respect to Batch_size ($t_{min} = 0$, $t_{max} = 10$).

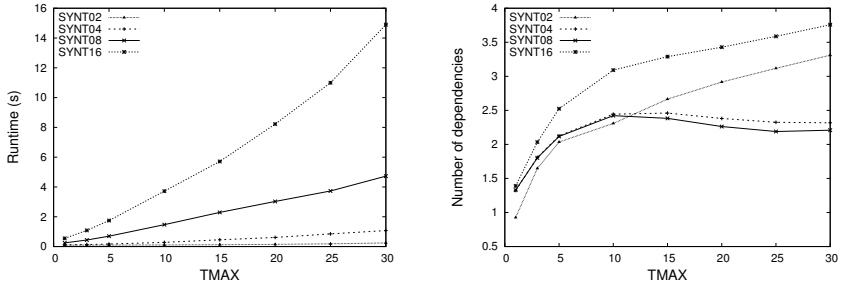


Figure 14: Runtime (left) and number of discovered dependencies (right) with respect to t_{max} (Batch_size=900, $t_{min} = 0$).

and $s_k^2 \xrightarrow{[\alpha^2,\beta^2]} s_\ell^2$ depict the same phenomenon, we consider the two following cases:

- the exact matching: all the temporal dependency parameters are equal ($s_k^1 = s_k^2$, $s_\ell^1 = s_\ell^2$, $\alpha^1 = \alpha^2$, $\beta^1 = \beta^2$);
- the relaxed matching: the two temporal dependencies are between the same data states ($s_k^1 = s_k^2$, $s_\ell^1 = s_\ell^2$).

Finally, we report the F_1 score, that is a trade-off between precision and the recall score which reaches its best value at 1 and worst score at 0:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Figure 15 shows the F1 score computed for the exact matching (left) and the relaxed matching (right) for the different datasets. It demonstrates that TEDDY is rather robust to noise as the harmonic mean of precision and recall remains satisfactory even when the percentage of noise is high. Two things are worth noticing. Firstly, exact matching gives worse results. In fact, the shifting intervals are less well identified by TEDDY when the percentage of noise

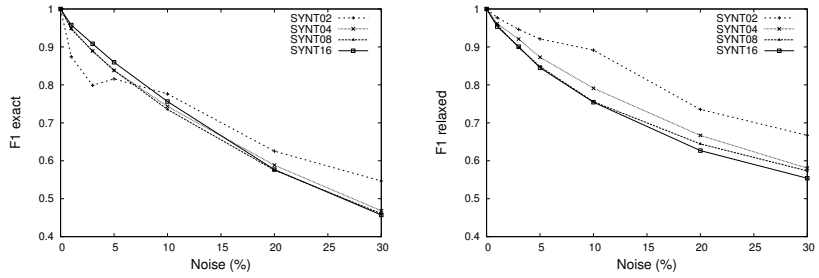


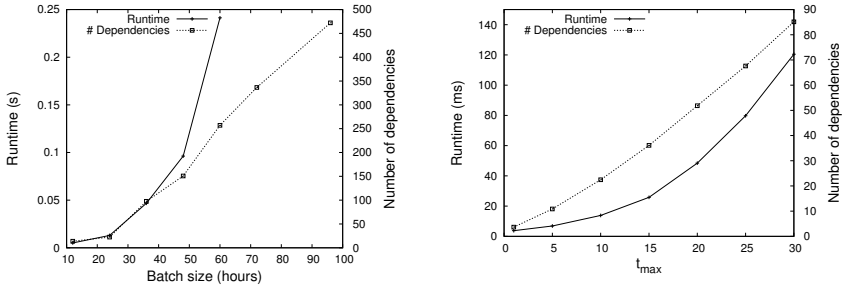
Figure 15: Robustness of TEDDY on synthetic dataset with respect to percentage of noise (Batch_size=900; $t_{min} = 0$, $t_{max} = 5$).

increases. Secondly, the higher the number of events per minute, the less robust against noise TEDDY becomes. If the proportion of "object detected" events is small, then the added spurious events are too sparse to produce additional temporal dependencies and the F1 score remains high.

4.4 Qualitative experiment results

Milan dataset

Figure 16 reports the running time of TEDDY with respect to the batch size (a) and t_{max} (b). These results confirm the observations made in the previous section. The execution times are negligible compared to the duration of the batches.



(a) (b)

Figure 16: Runtime of TEDDY on **Milan** with respect to the size of the batches (a) and t_{max} (b) (default values Batch_size=24 hours, $t_{min} = 0s$ and $t_{max} = 10s$).

Looking for temporal dependencies between smart home sensors, our aim is to describe the daily behavior of the persons living in the **Milan** smart home, as well as identifying unusual events. It is important to notice that, even if the data owner has some partial knowledge of the dependencies that may exist between

the smart environment sensors, the effective temporal dependencies depend on what really happens within such a smart environment.

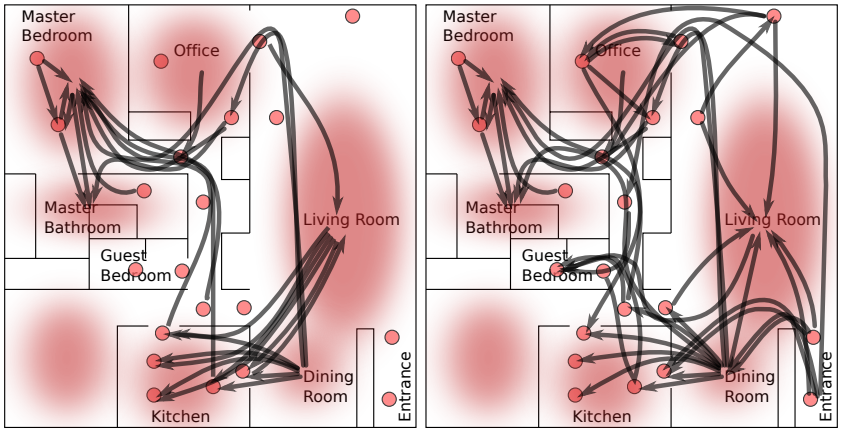
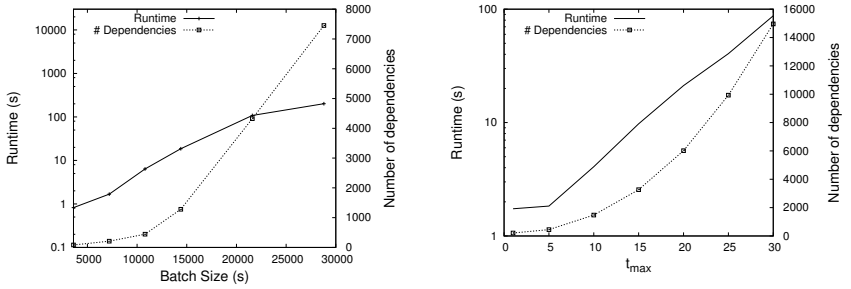


Figure 17: Two dependency graphs from the **Milan** dataset: temporal dependencies that occur the 26th October (left) and the 1st January (right).

The whole set of temporal dependencies discovered by TEDDY can be represented as a graph whose nodes are the data states and the arcs the temporal dependencies. Figure 17 illustrates two different dependency graphs extracted respectively from the batch related to October 26th and the batch of January 1st with $t_{min} = 5$ and $t_{max} = 10$ seconds. The background of the figure represents the plan of the apartment. Circles and color stains stand for motion sensors. On the graph corresponding to October 26th, we can observe that the temporal dependencies mainly involve sensors from the living room and the kitchen, as well as the sensors from the kitchen and the master bedroom and bathroom. The graph obtained from the batch related to January 1st is denser. There are edges that involve entrance sensors with kitchen or living/dining room sensors, indicating that the sensors detected many such movements, probably guests who visited the apartment. In addition it should be noticed that the sensors of the guest bedroom have been activated in temporal relationship with corridor sensors. As during these two days different actions were performed, the dependencies discovered are not the same. Therefore, temporal dependencies rely on both the disposition of the sensors within the environment and on the usages that are made within this environment. Even if the testbed does not change, the temporal dependencies discovered are quite different. This experiment highlights the interest of our proposal. Indeed, background knowledge is not enough. Temporal dependencies have to be discovered to accurately describe what really happens.

Foxstream dataset

TEDDY's performances: Figure 18 reports the running time of TEDDY with respect to respectively all the batches (a), the batch size (b) and t_{max} (c). Figure 18 also displays the average number of dependencies found per couple of states and batch in **Foxstream** dataset varying Batch_size (b) and t_{max} (c). These results confirm the observations made in the previous section. TEDDY's execution time increases according to t_{max} and batch size but remains negligible compared to the real batch duration. Therefore, temporal dependency detection can be carried out without risk of generation batch overflow. Furthermore, it is so fast that there is still time for more sophisticated analysis in addition to temporal dependencies.



(a) (b)

Figure 18: Runtime of TEDDY on **Foxstream** with respect to the size of the batches (a) and t_{max} (b) (default values Batch_size=2 hours, $t_{min} = 0s$ and $t_{max} = 5s$).

TEDDY's ability to describe temporal phenomenon: To investigate whether the temporal dependencies are well suited to describing the weekly activity around the monitored building, we compare the temporal dependencies found in 4-hour batches. To this end, we perform a hierarchical clustering to automatically group batches having a similar dependency set. The distance between two batches is computed as $(1 - J(A, B))$ where $J(A, B)$ is the Jaccard similarity coefficient [22] between the temporal dependency sets A and B. For the computing distance between clusters, we used the shortest distance.

The resulting dendogram is given in Figure 19. For each day of the week, there are 6 batches numbered from 1 to 6 that correspond to the following time slots: (2am-6am), (6am-10am), (10am-2pm), (2pm-6pm), (6pm,10pm) and (10pm-2am). Notice that batches with no dependency found have been removed. In this dendrogram, five clusters are clearly identified. Two of them ($C1$ and $C4$) contain batches related to office hours on working days, while $C2$ and $C5$ are specific to weekend day batches. The last cluster $C3$ contains two batches of late hours (10pm-2am) of Thursday and Friday. As the batches are grouped within a cluster if they contain similar dependencies, we can conclude that some

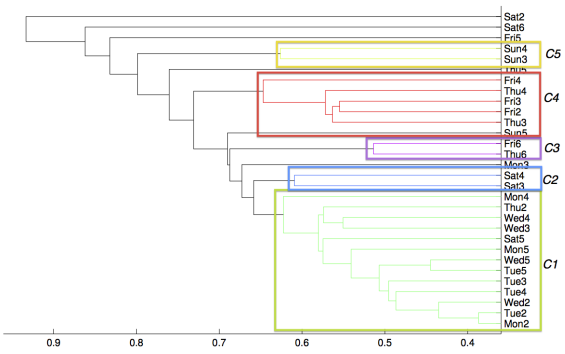


Figure 19: Hierarchical clustering of the batches.

of the extracted dependencies are specific to the office hour slots of working days, while others are specific to week-end days or nights. This hierarchical clustering demonstrates TEDDY's ability to identify temporal phenomena without prior knowledge.

The temporal dependencies extracted within a batch can be seen as the arcs of a directed graph whose nodes are the data states. Such a dependency graph describes the dynamic of events between data states during the batch period. Thus, analyzing the graphs associated with the batches enables us to understand the evolution of this dynamic through time. For example, we can consider the evolution of some macroscopic properties [37], such as the average degree as shown on Figure 20. Dependency graphs are dense during office hours

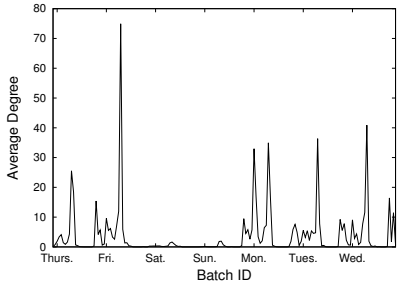


Figure 20: Evolution of the average degree of dependency graphs (Batch_size=1 hour, $t_{min} = 0s$ and $t_{max} = 5s$).

on working days, especially in the evening. On weekend days and at night, the dependency graphs are less dense. This is mainly due to the fact that the camera capture events in a working area. Thus, there is less activity during weekend days and at night.

TEDDY's ability to detect a spatial phenomenon: Dependency graphs may contain up to ten thousand temporal dependencies. This high number is mainly due to the important number of data sources mined. Actually, a moving object generates events in many areas, leading to as many reliable temporal dependencies. Figure 21 shows the temporal dependencies between the data

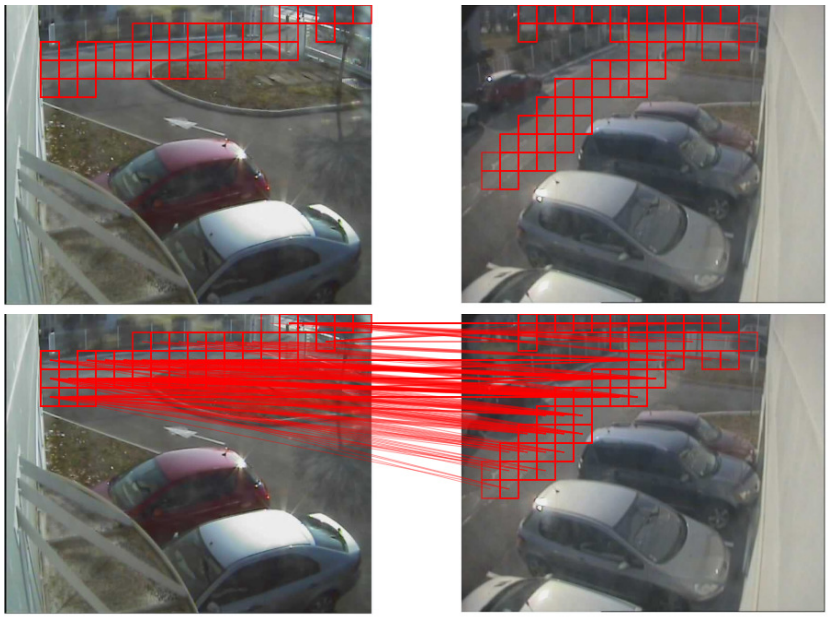


Figure 21: Temporal dependencies between data sources corresponding to subareas of two cameras. Images on top represent the subareas involved in dependencies; Bottom images display the temporal dependencies between them ($t_{max} = 10s$, Batch_size=3600s).

sources corresponding to subareas of two overlapping cameras. These dependencies have been extracted from the batch of events that occur on Tuesday between 10am and 2pm. The corresponding dependency graph is represented in Figure 22. This graph contains two connected components. If we study the evolution of this graph through time, we observe that it changes very little during the office hours of working days, whereas at night, the dependencies discovered are those from the upper connected component of the graph in Figure 22. Therefore, our approach makes it possible to automatically discover two distinct behaviours: one component captures the activity around the building (cars and persons) and the other depicts the activity (car traffic, pedestrians) within the street located in the neighborhood of the building, but not in the private parking lot.

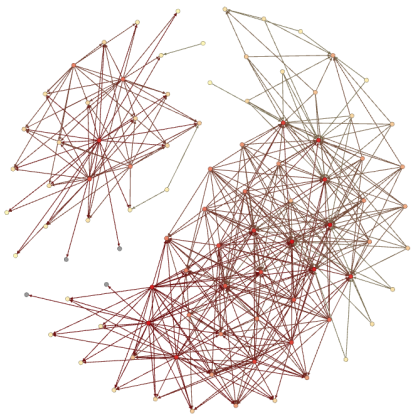


Figure 22: Dependency graph corresponding to data sources of Figure 21. The two connected components corresponds to two different behaviors: motion around the building (bottom/right component) and the background street activity (up/left component).

Anomaly detection: The video surveillance cameras used in this dataset are equipped with infrared LEDs, used to illuminate the scene in the infrared spectrum. This technology produces heat which attracts all sorts of insects, especially spiders. Spider's webs partially obstruct the view of the camera and blur the captured images. This phenomenon is even more important when it rains, water droplets hanging from the web causing serious obstruction of the camera view. Our goal is to automatically detect cameras that are subject to interference by insects. To this end, we apply a spider's web mask on the camera images captured during Wednesday and study the impact of the spider's web on the dependency graphs as follows. For each one-hour batch of Wednesday, we build the dependency graph from TEDDY's output and select the dependencies that involve a subarea of the camera. This so-called camera egocentric dependency graph is compared to the similar graph built from the batch of events produced 24 hours before, that is on Tuesday. The Jaccard index is used to evaluate the similarity of the set of arcs of these two graphs. Comparing two batches associated with the same time slots of two working days makes sense as the batches of these two days are grouped within the same cluster C_1 of the hierarchical clustering of Figure 19, indicating that the activity recorded during these two days is analogous. Furthermore, it enables us to overcome the fluctuations related to night/day phenomena. We assume that if the similarity measure is smaller than a given threshold, the corresponding camera is suspected of undergoing an unusual phenomenon, that may be due to the presence of insects on the camera lens.

Figure 24 (right) reports the Jaccard index values between the camera ego-

Figure 23: Temporal dependencies between two cameras for a batch corresponding to a period of no activity around the building (night-time). The corresponding dependency graph is the up/left component of the graph from Figure 22.

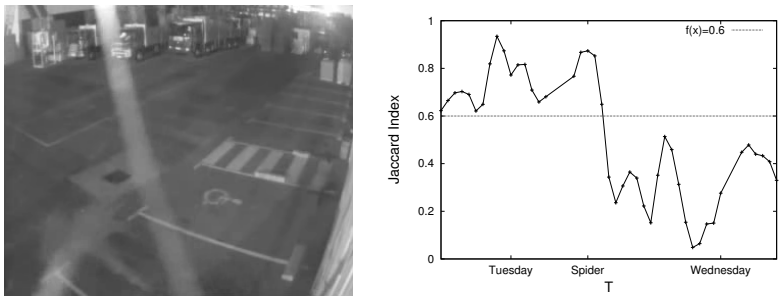


Figure 24: An image with a spider's web (left). Jaccard index evolution over Tuesday and Wednesday (right) (Batch_size=1 hour, $t_{min} = 0s$ and $t_{max} = 5s$).

centric dependency graphs obtained at time T and $T - 24$. The spider mask is introduced at $T = 21$. We can observe a drop in the Jaccard index that coincides with the appearance of the spider's web. Afterwards, the value of the Jaccard index remains smaller than 0.6. Therefore, a threshold of 0.6 on the Jaccard index can be used to detect abnormal behaviors in the monitoring system. This experiment demonstrates that it is possible to capture unusual phenomena / anomalies with TEDDY's output. It is a proof of concept and some more sophisticated treatments can be defined.

5 Related work

This paper makes a significant contribution to data stream management. The novelty of our approach is represented by the knowledge nuggets discovered over multiple heterogeneous data streams. Due to the type of processed data and the confidence measure used, our proposal might be related to the contributions from time series area. In this topic of research, algorithms are devised for measuring the similarity between two time series [43]. Most of them extend

the Dynamic Time Warping (DTW) algorithm [41, 7, 27] that makes it possible to find an optimal time alignment between two time series. However, the time-series are warped non-linearly in the time dimension so as to determine a measure of their similarity independent of certain non-linear time variations. In our work, we aim to consider linear transformation of the state stream in order to find out temporal state dependencies and their most specific time-delays.

Our work is related to various research areas: (i) temporal pattern mining on event intervals, (ii) trajectory and workflow mining, and (iii) mining smart environments.

Taking temporal information into account, most of the existing approaches aim at discovering frequent patterns among a set of sequences (i.e., temporal information is used to order the events within the sequences). Such approaches include mining of sequential patterns [3] or episodes [31] on sequences of "point events" (i.e., events with no duration), with applications in data stream processing [10, 32, 9]. In addition, some approaches show a particular interest in the time transition between events, either pushing aside some specific constraints like the well-known - mingap, maxgap and window-size - time constraints or trying to characterize the lag intervals between two event types [19, 42] or between items within a sequence [17]. Furthermore, based on the fact that sequential pattern mining on point event sequences is inadequate in discovering more sophisticated relations than the "before" / "after" relation, some works consider "interval events", i.e., events that have a duration. This introduces more complex relations between events extending Allen's algebra [26, 21, 8, 35, 45, 44]. Some approaches define events based on the interval model, but only the "before" / "after" temporal relation is supported [13, 4, 14]. We emphasize the fact that these approaches, except [30, 28, 29], are not dedicated to data stream mining tasks. Moreover, they aim at discovering regularities in a collection of sequences, whereas we wish to highlight some temporized dependencies between data sources (that are sequence producers) based on their states. Furthermore, incorporating statistical metric like χ^2 test within the pattern mining process is a well-studied issue [33, 23]. But these measures are often considered in addition to others such as confidence and support measures. In this paper, this statistical assessment is also used to automatically set some thresholds. Indeed, threshold setting is a difficult issue for end-user who are often not data miners and not familiar with these techniques. Thus, χ^2 test used in our proposal enables us to obtain more valuable results while limiting the number of parameters the end-user has to set. Finally, we are convinced that the two tasks are different and complementary. Indeed, two data sources may support several frequent sequences without having a dependency relation between them and vice versa.

Temporally annotated sequences [17] have been successfully applied to workflow mining [6] and trajectory pattern mining [18]. Once again, our approach differs in the nuggets that are discovered. For process or trajectory mining, a collection of logs is examined to highlight the regularities. In our approach, we search for dependencies between data sources.

From an application point of view, our work is close to the research conducted in the smart environment community, where one of the main challenges

is activity discovery. In [39], Rashidi and Cook proposed mining sequential patterns over time from streaming non-transactional data using tilted-time windows [20]. They extended their previous work [40], thereby introducing the first stream mining method for discovering human activity patterns in sensor data. Based on these proposals, association rule mining was applied to discover temporal relations of daily activities in [34]. Nevertheless, the motivations are different and these approaches are supervised while ours is unsupervised.

6 Conclusion

Designing new methods to discover relations over multiple heterogeneous data streams is a timely challenge. To the best of our knowledge, recently proposed methods focus on the discovery of regularities among events within streams. No proposal that we are aware of takes on the challenge of discovering particular relations between data sources that produce multiple heterogeneous data streams. Our work has investigated a new direction in data stream mining. We aimed at identifying temporal dependencies between data streams. We represented event streams by state streams that are induced by the streams' events themselves. First, we defined the novel problem of mining temporal state dependencies over multiple heterogeneous data streams. Then, we designed and implemented a complete, though scalable, algorithm that efficiently computes such temporal dependencies. Our approach is robust to the temporal variability of events and characterizes the time intervals during which the events are dependent. It links two types of events if the occurrence of one is often followed by the appearance of the other in a certain time interval. The proposed interval-based approach determines the most appropriate time intervals of a temporal dependency whose validity is assessed by a χ^2 test. As several intervals may redundantly describe the same dependency, the approach retrieves only the few most specific intervals with respect to a dominance relationship over temporal dependencies, and thus avoids the classical problem of pattern flooding in data mining. The TEDDY algorithm takes advantage of various properties to prune the search space while certifying the discovery of all valid and significant temporal dependencies. We conducted an extensive experimental study of both synthetic and real-world data streams from smart environments equipped with various kinds of sensors (cameras, motion sensors, etc.). From these experiments, we conclude that the pruning techniques are very efficient and speed up TEDDY running time by a factor that varies between 2 and 60. A qualitative analysis of the output shows that TEDDY produces a small set of non-redundant dependencies that accurately describe the phenomenon captured by the data.

There are several ways of extending the main results of this paper. First of all, we plan to investigate the dynamics of dependency graphs through time. The set of temporal dependencies can be viewed as an attributed graph whose nodes describe streams' states. Mining such dynamic attributed graphs would enable us to discover periodic phenomenon and other evolving behaviors that cannot be easily discovered without such a graph abstraction. We also plan

to make temporal dependencies more actionable from a database perspective. We are convinced that such dependencies can be integrated into continuous query engines. Indeed, some temporal dependencies could be the basis of a semantic indexation of data sources to better support human monitoring or object tracking in a set of cameras.

References

[1] Aggarwal, C.C. (ed.): Data Streams - Models and Algorithms, *Advances in Database Systems*, vol. 31. Springer (2007)

[2] Aggarwal, C.C., Li, Y., Yu, P.S., Jin, R.: On dense pattern mining in graph streams. PVLDB **3**(1), 975–984 (2010)

[3] Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE, pp. 3–14 (1995)

[4] Akdere, M., Çetintemel, U., Tatbul, N.: Plan-based complex event detection across distributed sources. PVLDB **1**(1), 66–77 (2008)

[5] Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983)

[6] Berlingerio, M., Pinelli, F., Nanni, M., Giannotti, F.: Temporal mining for interactive workflow data analysis. In: KDD, pp. 109–118 (2009)

[7] Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: KDD Workshop, pp. 359–370 (1994)

[8] Böttcher, M., Höppner, F., Spiliopoulou, M.: On exploiting the power of time in data mining. SIGKDD Explorations **10**(2), 3–11 (2008)

[9] Chang, L., Wang, T., Yang, D., Luan, H.: Seqstream: Mining closed sequential patterns over stream sliding windows. In: IEEE ICDM, pp. 83–92 (2008)

[10] Chen, G., Wu, X., Zhu, X.: Sequential pattern mining in multiple streams. In: IEEE ICDM, pp. 585–588 (2005)

[11] Cook, D., Schmitter-Edgecombe, M.: Assessing the quality of activities in a smart environment. Methods of Information in Medicine **48**(5) (2009)

[12] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms (3. ed.). MIT Press (2009)

[13] Demers, A.J., Gehrke, J., Panda, B., Riedewald, M., Sharma, V., White, W.M.: Cayuga: A general purpose event monitoring system. In: CIDR, pp. 412–422 (2007)

[14] Ding, L., Chen, S., Rundensteiner, E.A., Tatemura, J., Hsiung, W.P., Candan, K.S.: Runtime semantic query optimization for event stream processing. In: ICDE, pp. 676–685 (2008)

[15] Faloutsos, C., Kolda, T.G., Sun, J.: Mining large graphs and streams using matrix and tensor tools. In: SIGMOD Conference, p. 1174 (2007)

[16] Giannella, C., Han, J., Pei, J., Yan, X., Yu, P.S.: Mining frequent patterns in data streams at multiple time granularities. In: Data Mining: Next Generation Challenges and Future Directions. AAAI/MIT Press (2004)

[17] Giannotti, F., Nanni, M., Pedreschi, D.: Efficient mining of temporally annotated sequences. In: SDM (2006)

[18] Giannotti, F., Nanni, M., Pedreschi, D., Pinelli, F., Renso, C., Rinzivillo, S., Trasarti, R.: Unveiling the complexity of human mobility by querying and mining massive trajectory data. VLDB J. **20**(5), 695–719 (2011)

[19] Golab, L., Karloff, H.J., Korn, F., Saha, A., Srivastava, D.: Sequential dependencies. PVLDB **2**(1), 574–585 (2009)

[20] Han, J., Chen, Y., Dong, G., Pei, J., Wah, B.W., Wang, J., Cai, Y.D.: Stream cube: An architecture for multi-dimensional analysis of data streams. Distributed and Parallel Databases **18**(2), 173–197 (2005)

[21] Höppner, F., Klawonn, F.: Finding informative rules in interval sequences. Intell. Data Anal. **6**(3), 237–255 (2002)

[22] Jain, A., Dubes, R.: Algorithms for clustering data. Prentice Hall, Englewood cliffs, New Jersey (1988)

[23] Jermaine, C.: Finding the most interesting correlations in a database: how hard can it be? Inf. Syst. **30**(1), 21–46 (2005)

[24] Jin, R., Agrawal, G.: An algorithm for in-core frequent itemset mining on streaming data. In: IEEE ICDM, pp. 210–217 (2005)

[25] Jin, R., Agrawal, G.: Frequent pattern mining in data streams. In: C.C. Aggarwal (ed.) Data Streams - Models and Algorithms, *Advances in Database Systems*, vol. 31, pp. 61–84. Springer (2007)

[26] shan Kam, P., Fu, A.W.C.: Discovering temporal patterns for interval-based events. In: DaWaK, pp. 317–326 (2000)

[27] Keogh, E.J., Ratanamahatana, C.A.: Exact indexing of dynamic time warping. Knowl. Inf. Syst. **7**(3), 358–386 (2005)

[28] Li, M., Mani, M., Rundensteiner, E.A., Lin, T.: Constraint-aware complex event pattern detection over streams. In: DASFAA, pp. 199–215 (2010)

[29] Li, M., Mani, M., Rundensteiner, E.A., Lin, T.: Complex event pattern detection over streams with interval-based temporal semantics. In: DEBS, pp. 291–302 (2011)

[30] Liu, M., Li, M., Golovnya, D., Rundensteiner, E.A., Claypool, K.T.: Sequence pattern query processing over out-of-order event streams. In: ICDE, pp. 784–795 (2009)

[31] Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Min. Knowl. Discov. **1**(3), 259–289 (1997)

[32] Mendes, L.F., Ding, B., Han, J.: Stream sequential pattern mining with precise error bounds. In: IEEE ICDM, pp. 941–946 (2008)

[33] Morishita, S., Sese, J.: Traversing itemset lattice with statistical metric pruning. In: PODS, pp. 226–236 (2000)

[34] Nazerfard, E., Rashidi, P., Cook, D.J.: Using association rule mining to discover temporal relations of daily activities. In: ICOST, pp. 49–56 (2011)

[35] Patel, D., Hsu, W., Lee, M.L.: Mining relationships among interval-based events for classification. In: SIGMOD Conference, pp. 393–404 (2008)

[36] Pearson, K.: On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. Psychology Magazine **1**, 157–175 (1900)

[37] Prado, A.B., Plantevit, M., Robardet, C., Boulicaut, J.F.: Mining graph topological patterns: Finding co-variations among vertex descriptors. IEEE Transactions on Knowledge and Data Engineering **1** (2013)

[38] Raïssi, C., Plantevit, M.: Mining multidimensional sequential patterns over data streams. In: DaWaK, pp. 263–272 (2008)

[39] Rashidi, P., Cook, D.J.: Mining sensor streams for discovering human activity patterns over time. In: IEEE ICDM, pp. 431–440 (2010)

[40] Rashidi, P., Cook, D.J., Holder, L.B., Schmitter-Edgecombe, M.: Discovering activities to recognize and track in a smart environment. IEEE Trans. Knowl. Data Eng. **23**(4), 527–539 (2011)

[41] Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. Acoustics, Speech and Signal Processing, IEEE Transactions on **26**(1), 43 – 49 (1978). DOI 10.1109/TASSP.1978.1163055

[42] Tang, L., Li, T., Shwartz, L.: Discovering lag intervals for temporal dependencies. In: KDD, pp. 633–641 (2012)

[43] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E.J.: Experimental comparison of representation methods and distance measures for time series data. Data Min. Knowl. Discov. **26**(2), 275–309 (2013)

[44] Winarko, E., Roddick, J.F.: Armada - an algorithm for discovering richer relative temporal association rules from interval-based data. Data Knowl. Eng. **63**(1), 76–90 (2007)

[45] Wu, S.Y., Chen, Y.L.: Mining nonambiguous temporal patterns for interval-based events. IEEE Trans. Knowl. Data Eng. **19**(6), 742–758 (2007)