IL-Miner: Instance-Level Discovery of Complex Event Patterns

Lars George, Bruno Cadonna, Matthias Weidlich
Humboldt-Universität zu Berlin
{lars.george,bruno.cadonna,matthias.weidlich}@informatik.hu-berlin.de

ABSTRACT

Complex event processing (CEP) matches patterns over a continuous stream of events to detect situations of interest. Yet, the definition of an event pattern that precisely characterises a particular situation is challenging: there are manifold dimensions to correlate events, including time windows and value predicates. In the presence of historic event data that is labelled with the situation to detect, event patterns can be learned automatically. To cope with the combinatorial explosion of pattern candidates, existing approaches work on a type-level and discover patterns based on predefined event abstractions, aka event types. Hence, discovery is limited to patterns of a fixed granularity and users face the burden to manually select appropriate event abstractions.

We present IL-MINER, a system that discovers event patterns by genuinely working on the instance-level, not assuming a priori knowledge on event abstractions. In a multi-phase process, IL-MINER first identifies relevant abstractions for the construction of event patterns. The set of events explored for pattern discovery is thereby reduced, while still providing formal guarantees on correctness, minimality, and completeness of the discovery result. Experiments using real-world datasets from diverse domains show that IL-MINER discovers a much broader range of event patterns compared to the state-of-the-art in the field.

Keywords

Event Processing, Pattern Mining, Machine Learning

1. INTRODUCTION

Complex event processing (CEP) detects situations of interest by matching *event patterns* over continuous streams of events. An event pattern defines how the events that shall be matched have to be *ordered* in the stream and how they are *correlated* along multiple dimensions, including predicates over their attribute values and time windows [7]. CEP is the foundation of applications in diverse domains [7], including financial trading [5], oil and gas [9], healthcare [13], business workflows [24], and urban transportation [3].

Proceedings of the VLDB Endowment, Vol. 10, No. 1

Event Pattern Specification. Traditional CEP applications assume that a domain expert knows how a situation of interest is manifested in the event stream. Hence, the expert is able to come up with the respective definition of an event pattern. The assumption of knowledgeable expert users is commonly met in reactive applications, where low-latency detection of a situation of interest enables immediate implementation of countermeasures.

As an example, consider the case of monitoring a compute cluster based on events that indicate state changes of executed jobs [19]. To detect jobs that are terminated, but not re-scheduled for execution, a pattern would match a termination event that is not followed by a scheduling event for the same job within a particular timeframe.

Recently, the application of CEP shifted towards pro-active processing, where an event pattern anticipates a situation of interest [6]. Matching an event pattern may then trigger actions to prepare for the situation or to circumvent it. In such applications, however, the knowledge of a domain expert is typically limited to the occurrence of the situation of interest. Knowledge about the event pattern that anticipates the situation is partial at best.

In cluster monitoring, for example, one can anticipate that a terminated job will not be re-scheduled. While a domain expert may have anecdotal evidence on what constitutes the respective pattern (e.g., the jobs belong to a particular class), they lack the insights needed for a comprehensive pattern definition (e.g., jobs have been re-scheduled at least twice with increasing priority and ran for more than two hours).

Automated Pattern Discovery. In the absence of an event pattern that detects a situation of interest, historic data that is labelled with the respective situation enables automatic discovery of event patterns [15]. Historic event data are split into sequences of events, aka *traces*, that serve as training examples for a learning process.

Given a set of traces, pattern discovery can be traced back to a large body of work on temporal knowledge discovery and, specifically, frequent sequence mining [1, 23, 26]. However, all these techniques have in common that sequential patterns are discovered on the *type-level*: they require that events of a stream are partitioned into types and then aim at identifying an order over these types. A consequence of this approach is also that any correlation condition defined based on the data carried by the events is necessarily *global*, i.e., it spans the complete pattern.

The limitations of existing sequence mining techniques are illustrated in Fig. 1. The top part shows a stream of six events of the aforementioned cluster monitoring scenario, each event having attributes such as the `class`, `status`, or `priority` of a compute job. Traditional pattern discovery would first require the definition of an event abstraction, as exemplified in the middle part of Fig. 1. Here, the *event type* is defined as the job `status` (`Submit`, `Terminate`, or `Fail`), so that a pattern that matches a sequence of two 'submit' and

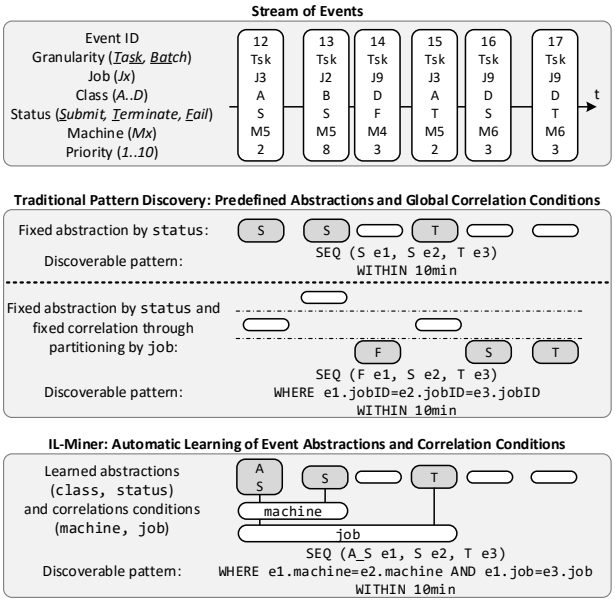

Figure 1: Unlike traditional pattern discovery, IL-MINER is not limited to predefined event abstractions and supports discovery of local correlation conditions.

a 'terminate' event may be discovered. Partitioning the stream based on attribute values further enables discovery of patterns with *global correlation conditions*. An example is shown in Fig. 1 in terms of a pattern that requires all events to relate to the same compute `job`.

The above approach to exploit sequence mining for the discovery of complex event patterns is neither efficient nor effective. Domain experts rarely know how to chose the appropriate event abstractions. Hence, an automatic approach needs to explore a large space of possible abstractions, which quickly becomes intractable in practice. Despite its inherent complexity, such an approach is also ineffective. Patterns that are composed of *different* event abstractions and *local* correlation conditions that hold true only for a subset of the matched events cannot be discovered. Recently, the iCEP miner [15] was proposed as the first attempt to incorporate local correlation conditions, yet it still works on the level of predefined event abstractions. As such, existing approaches limit the discovery to the *order* over correlated event abstractions, but do not aim at discovery of the relevant abstractions and correlation conditions.

Contributions. We describe the design and implementation of IL-MINER, **a system that discovers event patterns from labelled event data by automatically learning event abstractions and correlation conditions**. IL-MINER genuinely works on the instance-level and copes with the combinatorial explosion of pattern candidates by means of a multi-phase process. As such, it is able to discover patterns featuring diverse event abstractions and local correlation conditions. For the aforementioned example, the bottom part of Fig. 1 depicts such a pattern that could not be discovered by traditional approaches: It combines different event abstractions (`status` or a combination of `status` and `class`) and correlation conditions that span only a subset of the events to be matched.

IL-MINER's discovery process consists of several phases:

(1) Promising event abstractions are identified based on a given set of traces. Subsequently, frequent sequence mining is employed to obtain candidate sequences of event abstractions.

(2) The candidate sequences obtained in the first phase are linked to the events in the traces. This enables us to assess the relevance of particular event abstractions and correlation conditions on the instance-level.

(3) Discovery may yield a large set of event patterns. IL-MINER can thus filter patterns along several dimensions to identify a small, representative result set.

Our contribution is not limited to the definition of the discovery process: we also formally show correctness, minimality, and completeness of the discovery process for conjunctive event patterns. Finally, we outline optimisations of the implementation of pattern discovery to achieve scalability.

The remainder of the paper is organised as follows: §2 motivates event pattern discovery in different application domains and presents an event processing model; §3 defines the pattern discovery problem; §4 introduces our multi-phase process for pattern discovery; §5 discusses optimisations to achieve scalability; and §6 presents results from an evaluation with real-world datasets. The paper finishes with related work (§7) and conclusions (§8).

2. BACKGROUND

This sections introduces and formally defines the necessary background that is needed in order to discover event patterns.

2.1 Motivating Scenarios

Discovery of event patterns from labelled event data has applications in diverse domains, as exemplified below.

(i) Cluster management: To analyse resource usage and correct operation of compute clusters, events that report changes in resource utilisation and the job life-cycle may be exploited. Fig. 1 (top) illustrates such a stream of events as it is reported, for example, in the Google cluster traces [19, 20]. Yet, in many pro-active applications that aim at anticipating a situation of interest, there is no comprehensive understanding of the event pattern to be matched.

EXAMPLE 1. *Consider a scenario in which terminated compute jobs are not properly rescheduled for execution. A reactive application may identify this situation using a pattern that checks for termination events that are not followed by scheduling events of the same job within 1 hour. However, a pro-active application aims at anticipation of this situation immediately upon job termination. For instance, it may turn out that the respective jobs are of a particular class and terminate at most 10 minutes after their submission, when another job is scheduled on the same machine. A respective event pattern is shown in the bottom part of Fig. 1: an event with* `status=S` *and* `class=A` *is followed by an event with* `status=S` *and the same* `machine` *as the first one and, finally, a third event occurs, with* `status=T` *and the same* `job` *as the first one.*

To discover patterns that anticipate a situation of interest, traces can be constructed from historic event data, e.g., by splitting a stream whenever the situation occurred. Such traces are then used as input for automated learning.

Traditional frequent sequence mining requires manual definition of event abstractions, though, since automated exploration of possible patterns would have to cope with an exponential blow-up of the space of possible abstractions. Fig. 2 demonstrates this effect for the Google cluster traces [20]. Taking only a subset of the 13 attributes of events in this dataset (specifically, `status`, `job`, `priority`, `class`, and `machine`) yields already around 1,700,000 possible event abstractions, i.e., combinations of attribute values, over which patterns may be defined. In addition, even for a fixed event abstraction, there is an exponential number of possible local correlation conditions, so that naive frequent sequence mining becomes intractable in practice.

(ii) Finance: Events play an important role in computational finance applications, such as stock trading. Streams of trade events are monitored for particular patterns that hint at market changes and business

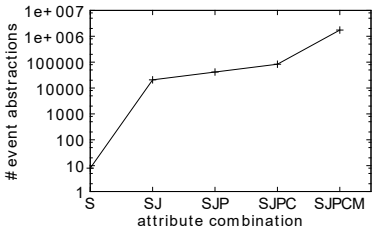

Figure 2: Exponential blow-up of the space of possible event abstractions for the Google cluster traces.

opportunities. An example would be the *head-and-shoulders trading pattern*, see [5], which is defined by two lower peak values that frame a higher peak value. Here, event pattern discovery based on historic data serves as a tool to find possible explanations for relevant situations (e.g., peaks in order volumes), thereby supporting decision making in the future.

The challenge, again, is the selection of appropriate event abstractions. Relevant patterns may combine events of different stocks, specific order volumes, or fixed price ranges. A large number of numeric event attributes with large domains makes this task particularly difficult in this domain: it is unclear how fine-granular order volumes or stock prices shall be abstracted. Further, local correlation conditions may be needed, e.g., correlating high-order events of the same stock with price peaks of another one.

(iii) Urban transportation: In traffic management [3], events capture the movement of buses and taxis in a city. Event patterns over such event streams may monitor the punctuality of buses and identify on which routes and at what times a delayed bus can catch up with the schedule again [27]. Event pattern discovery helps to identify, for instance, how bottlenecks in the transportation network build up or under which circumstances neighbourhoods become particularly profitable for taxi drivers.

Possible event abstractions in this setting are manifold: traffic patterns in bus routes may relate to specific bus routes, neighbourhoods, bus operators, or levels of accumulated delay. Also, patterns over taxi utilisation events may need to correlate a subset of matched events based on drivers, trip distances, or payment types.

2.2 Event Processing Model

Event Streams. We adopt a relational model of event streams, similar to the one defined by the continuous query language (CQL) [2]. An *event schema* is a finite sequence of attributes $A = \langle A_1, \ldots A_n \rangle$, each being of a primitive data type that is denoted by $dom(A_i)$ for attribute A_i. An *event* is an instance of an event schema, i.e., a finite sequence of attribute values $e = \langle a_1, \ldots a_n \rangle$ with a_i being the value of the respective attribute A_i. We write $e.A_i$ to denote the value a_i of attribute A_i of event e. We further assume a discrete, ordered time domain for timestamps, given as $\mathbb{N}_0$. An event e has a timestamp, denoted by $e.t$, which is the time of event occurrence. A *stream* is an infinite sequence $S = \langle e_1, e_2, \ldots \rangle$ of events, in which the order of events respects timestamps: for two event e_j and e_k, $j < k$ implies that $e_j.t \leq e_k.t$. Without loss of generality, we assume that all events in a stream have the same schema. Table 1 shows an overview of the notation used in this paper.

Event Pattern Matching. While many languages for the definition of event patterns, see [4, 25, 16], differ in syntax and semantics, they typically share a common set of concepts [28], such as *sequence* operators, *value predicates*, and *time windows*. That is, a pattern defines an order over some event abstractions (aka event types). Correlation conditions are expressed by predicates over the attributes of the respective event schema and in terms of a maximal timespan between the events that are matched.

Table 1: Overview of the used notation

Notation	Explanation
$A = \langle A_1, \ldots A_n \rangle$	An event schema, a sequence of attributes
$e = \langle a_1, \ldots a_n \rangle$	An event, a sequence of attribute values
$e.A_i$	The value a_i of attribute A_i of event e
$e.t$	The occurrence time of event E
$S = \langle e_1, e_2, \ldots \rangle$	A stream
$P = (V, \Theta, \tau)$	An event pattern
$V = \langle v_1, \ldots, v_r \rangle$	A sequence of event variables
$\Theta = \{\theta_1, \ldots, \theta_s\}$	A set of constraints
τ	A time window
$\Omega(P, S')$	The set of matches of pattern P in stream prefix S'

DEFINITION 1 (Event Pattern). *An event pattern, or pattern for short, is a triple $P = (V, \Theta, \tau)$, where*

- *$V = \langle v_1, \ldots, v_r \rangle$, $r \geq 1$, is a sequence of event variables;*
- *$\Theta = \{\theta_1, \ldots, \theta_s\}$, $s \geq 0$, is a set of constraints; and*
- *$\tau \in \mathbb{N}_0$ is a time window.*

An *event variable* v_j, $j \leq r$, is a placeholder for a set of events (of schema $A = \langle A_1, \ldots, A_n \rangle$) that are equivalent under a specific event abstraction. This abstraction is defined by means of *property constraints* of the form $v_j.A_i \; \phi \; C$, where $v_j.A_i$ refers to the value of attribute A_i of the event that shall be assigned to variable v_j; $C \in dom(A_i)$ is a constant of the same primitive type as attribute A_i, and $\phi \in \{\doteq, \dot{<}, \dot{\leq}, \dot{>}, \dot{\geq}\}$ is a comparison operator (distinguished from the respective mathematical relation by dotted notation). Constraints can also be of the form $v_j.A_i \; \phi \; v_k.A_l$, $k \leq r$, $j \neq k$, $l \leq n$. These constraints, referred to as *relation constraints*, model correlation conditions between events.

Semantics of a pattern are defined as follows. Let $P = (V, \Theta, \tau)$, $V = \langle v_1, \ldots, v_r \rangle$, be a pattern and $S' = \langle e_1, \ldots, e_z \rangle$ a finite sequence of events, e.g., a prefix of some stream. A *match* of P in S' is a sequence $\langle e_1, \ldots, e_r \rangle$, such that all constraints in Θ evaluate to true, if each event variable v_j is bound to event e_j, $j \leq r$. Note that an event can be part of multiples matches of P in S'. The set of matches of P in S' is denoted by $\Omega(P, S')$.

EXAMPLE 2. *Taking up Example 1, we illustrate how the respective pattern (Fig. 1, bottom part) is captured in our model. Let $A = \langle$ `id`, `granularity`, `job`, `class`, `status`, `machine`, `priority`$\rangle$ be the event schema. Then, the pattern is defined as $P = (\langle v_1, v_2, v_3 \rangle, \Theta, 10)$ with Θ containing:*

- *Property constraints defining the event abstractions:* v_1`.class` $\doteq$ `A`, v_1`.status` $\doteq$ `S`, v_2`.status` $\doteq$ `S`, *and* v_3`.status` $\doteq$ `T`;
- *Relation constraints defining correlation conditions:* v_1`.machine` $\doteq$ v_2`.machine` *and* v_1`.job` $\doteq$ v_3`.job`.

3. PROBLEM DEFINITION

We frame the problem of event pattern discovery based on the notion of a trace. A trace is a finite sequence of events, such that the situation of interest occurs towards the end of the timespan that is covered by the trace.

DEFINITION 2 (Trace). *A* trace *is a finite, non-empty sequence of events $h = \langle e_1, \ldots, e_m \rangle$, such that it holds $e_j.t \leq e_k.t$, for $1 \leq j < k \leq m$.*

We write $e \in h$, if $e = e_j$ for some $j \leq m$. Traces can be obtained in different ways. For instance, a maximal duration $d \in \mathbb{N}_0$ can be fixed for the patterns to discover: each occurrence of the situation of interest at time $t_i \in \mathbb{N}_0$ yields a trace comprising all events e of a stream with $t_i - d \leq e.t \leq t_i$. Another approach is to partition a stream (prefix) by the situations of interest, taking each part as a trace, as it is illustrated in Fig. 3.

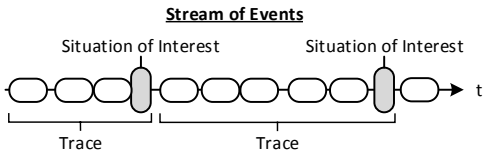

Figure 3: Partitioning an event stream (or a prefix thereof) in traces.

Traces serve as training examples for learning event patterns. However, there may be multiple patterns, say P and P', that create matches for the given traces. In that case, patterns P and P' may be comparable. In particular, we say that P is *stricter* than P', denoted by $P \prec P'$, if $\Omega(P, S') \subset \Omega(P', S')$ for any finite event sequence S'. In event pattern discovery, it is sufficient to discover the strictest patterns—the assumption being that they provide the most precise characterisations of how the situation of interest materialises in the event stream.

Given a set of traces and based on the notion of strictness of event patterns, we state the problem addressed in this work as follows:

PROBLEM 1. *Given a set of traces H, event pattern discovery is the construction of a set of event patterns $\mathcal{P}$, such that*

- *$\mathcal{P}$ is correct; each pattern creates at least a single match per trace: for all $P \in \mathcal{P}$ and $h \in H$ it holds that $|\Omega(P, h)| > 0$;*
- *$\mathcal{P}$ is minimal; only the strictest patterns are considered: for all $P, P' \in \mathcal{P}$ it holds that neither $P \prec P'$ nor $P' \prec P$.*
- *$\mathcal{P}$ is complete; if there exists a pattern P with $|\Omega(P, h)| > 0$ for all $h \in H$, but there does not exist a pattern P' with $P' \prec P$ and $|\Omega(P', h)| > 0$ for all $h \in H$, then it holds that $P \in \mathcal{P}$.*

For a pattern $P = (V, \Theta, \tau)$ to create a match for a trace, *all* event variables V must be bound and *all* constraints Θ must be satisfied. As such, event pattern discovery inherently requires generalisation: patterns are constructed based on event abstractions and correlation conditions that are relevant to create matches in all given traces.

Table 2: Two example traces

	ID	t	Gran.	Job	Class	Status	Machine	Prio.
h_1	e_1	10	Tsk	J3	A	S	M5	2
	e_2	12	Tsk	J2	B	S	M5	8
	e_3	15	Tsk	J3	A	T	M5	2
h_2	e_4	31	Tsk	J4	A	S	M2	3
	e_5	35	Tsk	J7	D	S	M2	1
	e_6	41	Tsk	J4	E	T	M6	3
	e_7	45	Tsk	J5	C	T	M6	0

EXAMPLE 3. *Table 2 lists two example traces for the aforementioned scenario. The pattern $P = (\langle v_1, v_2, v_3 \rangle, \Theta, 10)$, which is formalised in Example 2, creates matches for both of these traces. However, it would not be constructed by event pattern discovery. Rather, we would find a stricter pattern that defines additional property constraints. Specifically, discovered event patterns would include constraints of the form $v_j.\texttt{granularity} \doteq \texttt{Tsk}$, $j \in \{1, 2, 3\}$, encoding that all events provide information on the level of tasks of a compute job.*

4. MULTI-PHASE PATTERN DISCOVERY

This section presents IL-MINER's multi-phase process to discover event patterns from traces. We begin this section with an overview of the discovery process (§4.1) outlining the general approach. Subsequently, we provide details on the individual phases: learning of sequence templates (§4.2), pattern construction (§4.3), and pattern filtering (§4.4). Finally, we turn to the formal guarantees of the discovery process (§4.5).

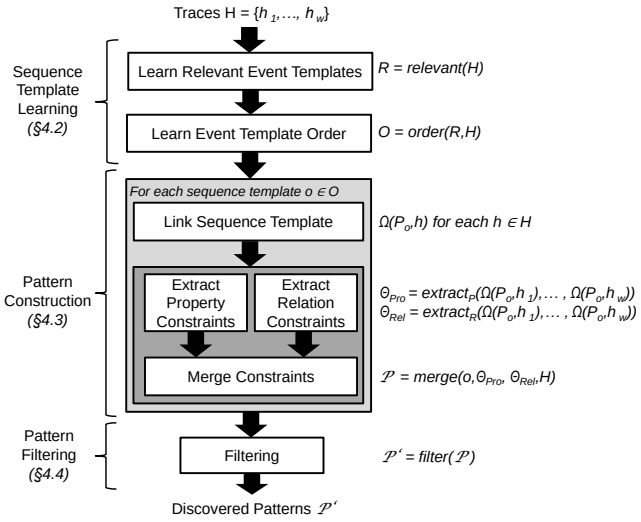

Figure 4: IL-MINER's multi-phase process to pattern discovery.

4.1 Overview

The discovery process of IL-MINER is organised in three phases. As illustrated in Fig. 4, these phases are executed sequentially and comprise several individual steps:

Sequence Template Learning. IL-MINER first identifies promising candidate event abstractions, referred to as *relevant event templates*. This is achieved by determining the set of equivalence predicates that are satisfied by the events of the given traces. Interpreting the traces in terms of the identified relevant event templates, frequent sequence mining yields a set of *sequence templates*, each serving as a skeleton for a set of event patterns.

Pattern Construction. To construct a pattern, each sequence template is first linked to the events of the given traces. Using the resulting sequences of events, property constraints as well as relation constraints are learned. Finally, plausible combinations of both types of constraints are identified and the time window of the resulting patterns is determined.

Pattern Filtering. To cope with a potentially large number of discovered event patterns, IL-MINER includes a filtering phase to identify a small, representative set of patterns. To this end, clustering based on syntactic and semantics measures for event patterns is leveraged.

4.2 Sequence Template Learning

Learning Relevant Event Templates.. An event template represents a possible event abstraction for the construction of a pattern. It is formalised as a set of equivalence predicates. An event template may later serve as the basis to define a property constraint in the construction of an event pattern, see §2.2. Let $A = \langle A_1, \ldots A_n \rangle$ be an event schema. Then, an event template is a set of predicates $A_i = C$, $i \leq n$, where $C \in dom(A_i)$ is a constant.

An event $e = \langle a_1, \ldots a_n \rangle$ induces a set of atomic equivalence predicates $A_i \doteq a_i$, each assigning to an attribute A_i the respective value a_i of e. The set of possible event templates induced by e is the powerset of such atomic predicates: $\zeta(e) = \wp\left(\bigcup_{i \leq n} \{A_i \doteq a_i\}\right)$.

Each event of a trace induces a set of possible event templates. As such, the set of possible event templates can be lifted to a trace $h \in H$ as $\zeta(h) = \bigcup_{e \in h} \zeta(e)$. IL-MINER considers event templates to be relevant, if they materialise in all given traces H. Consequently, the set of relevant event types R is the intersection of the possible templates per trace, $R = relevant(H) = \bigcap_{h \in H} \zeta(h)$.

It is worth to note that R contains *all* event templates that materialise in all traces, not only the most restrictive ones, i.e., not only those with the maximal number of equivalence predicates. At later stages, this will enable IL-MINER to discover patterns that are based on diverse event abstractions.

EXAMPLE 4. *We take the traces $H = \{h_1, h_2\}$ listed in Table 2 as an example. Then, a subset of the relevant event templates $R = relevant(H)$ is given below (we assign identifiers (1-5) to each of the templates for later reference):*

$$R = \{\ 1 : \{\texttt{gran.} = \texttt{Tsk}\},\ 2 : \{\texttt{gran.} \doteq \texttt{Tsk}, \texttt{status} \doteq \texttt{T}\},\ 3 : \{\texttt{gran.} \doteq \texttt{Tsk}, \texttt{status} \doteq \texttt{S}\},\ 4 : \{\texttt{gran.} \doteq \texttt{Tsk}, \texttt{class} \doteq \texttt{A}\},\ 5 : \{\texttt{gran.} \doteq \texttt{Tsk}, \texttt{status} \doteq \texttt{S}, \texttt{class} \doteq \texttt{A}\}, \dots\ \}.$$

Learning Sequence Templates. Using the identified relevant event templates, this step constructs *sequence templates*, sequences $o = \langle o_1, \dots, o_l \rangle$ with $o_i \in R$, $i \leq l$ of event templates that materialise in all given traces. These sequences are then used to define the order of event variables in a discovered pattern.

As a first step, each trace is interpreted in terms of the relevant event templates. From each trace $h = \langle e_1, \dots, e_m \rangle \in H$, we derive a sequence of sets of event templates, $h' = \langle t_1, \dots, t_m \rangle$, such that $t_i = R \cap \zeta(e_i)$, $i \leq m$. In other words, each event is assigned the set of relevant event templates for which it satisfies the respective equivalence predicates.

In a second step, IL-MINER identifies which of the possible orders of event templates are frequent. Here, the common problem of frequent sequence mining (FSM) [1] is solved, which, in our setting, discovers frequent sequences of sets of event templates based on the set of traces. Specifically, we consider FSM with a *minimum support value* of 1.0 and extract only *closed* sequences. The former means that only sequences that appear in all traces shall be considered [21]. That is to ensure that any discovered pattern indeed identifies all situations of interest (see Problem 1). The latter limits the result to maximal sequences [23].

Various algorithms have been present for FSM, e.g., Bide [23] and CloSpan [26]. Taking into account that we are only interested in closed sequences with a minimum support value of 1.0, we employ a simple FSM algorithm that adopts the idea of Pincer-Search [12] and implements a top-down search: it starts with the longest possible subsequence and step-wise shortens it until reaching the required support value.

The result of FSM in our setting are sequences of *sets* of event templates. At each position in such a sequence, all of the respective event templates are viable candidates for the definition of a sequence template—and, eventually, for the definition of property constraints in an event pattern. Hence, a single sequence identified by FSM actually induces a set of sequence templates, i.e., sequences of *individual* event templates. Each of these sequence templates is considered in the remaining steps of the discovery process. Since this may lead to an exponential number of sequence templates per sequence found by FSM, we later discuss an optimisation to avoid the exponential blow-up of sequences at the expense of potential incompleteness of the discovery process.

Algorithm 1 outlines the complete algorithm to learn a set of sequence templates O, given a set of relevant event templates R and a set of traces H. In a first step, a sequence of sets of relevant event templates is constructed per trace (lines 1 to 3). Subsequently, the shortest of these sequences is selected (line 4) as a starting point for the frequent sequence mining (lines 5 to 9). Here, sequences are iteratively simplified (by removing one template from one of the sets in the sequence) until a sequence of relevant event templates is found

Algorithm 1: order(R, H) – Learning Sequence Templates

Input: R – a set of relevant event templates
H – a set of traces
Output: O – a set of sequence templates

```
   /* Construct sequences of relevant event templates  */
1  seq ← ∅;
2  for ⟨e_1, ..., e_m⟩ ∈ H do
3      seq ← seq ∪ ⟨(R ∩ ζ(e_1)), ..., (R ∩ ζ(e_m))⟩;
   /* Conduct frequent sequence mining                 */
4  s ← arg min_{⟨t_1,...,t_m⟩ ∈ seq} m;
5  F ← {s};
6  for ⟨t_1, ..., t_m⟩ ∈ F do
7      if ∃ ⟨e_1, ..., e_m⟩ ∈ H, 1 ≤ i ≤ m : t_i ⋢ ζ(e_i) then
8          F ← F \ {⟨t_1, ..., t_m⟩};
9          F ← F ∪ {⟨t'_1, ..., t'_m⟩ | ∃ 1 ≤ i ≤ m : t'_i ⊂ t_i
               ∧ |t'_i| + 1 = |t_i| ∧ ∀ 1 ≤ j ≤ m, i ≠ j : t'_j = t_j};
   /* Derive sequences of individual event templates   */
10 O ← ⋃_{⟨t_1,...,t_m⟩ ∈ F} {⟨o_1, ..., o_m⟩ | ∀ 1 ≤ i ≤ m : o_i ∈ t_i};
11 return O;
```

that is in line with all traces (the templates of the respective events are subsets of the templates in the candidate sequence). Finally, sequences of individual templates are obtained from sequences of sets of templates (line 10).

EXAMPLE 5. *Using the identifiers assigned to the event templates in Example 4, the traces of Table 2 are encoded for frequent sequence mining as:*

h_1: $\langle\{1,3,4,5\},\{1,3\},\{1,2,4\}\rangle$
h_2: $\langle\{1,3,4,5\},\{1,3\},\{1,2\},\{1,2\}\rangle$

Applying Algorithm 1, we first construct the following frequent sequence of sets of event templates: $\langle\{1,3,4,5\},\{1,3\},\{1,2\}\rangle$. From this sequence, various sequence templates are constructed, e.g., $\langle 1,1,2\rangle$ or $\langle 4,3,1\rangle$. The latter, for instance, corresponds to a sequence of three events with $\texttt{granularity} \doteq \texttt{Tsk}$, while it also holds $\texttt{class} \doteq \texttt{A}$ for the first event and $\texttt{status} \doteq \texttt{S}$ for the second event.

4.3 Pattern Construction

The pattern construction phase of IL-MINER is executed for each sequence template in O and involves four steps: (i) linking the sequence template to events of traces; extracting (ii) property constraints and (iii) relation constraints from the respective events; (iv) merge the constraints to derive an event pattern. As illustrated in Fig. 4, steps (ii) and (iii) are independent of each other and are thus executed in parallel.

Linking Sequence Templates. A sequence template represents a frequent sequence of promising event abstractions, thereby providing a first hint on how to discover an event pattern. However, a sequence template is still a rather coarse-grained abstraction: the identified sequences may not be relevant for *all* traces without further refinement. To explore the relevance of event abstractions and correlation conditions, IL-MINER considers the instance-level, i.e., the actual events of the given traces. Therefore, this step determines all sequences of events of any trace that are instantiations of the sequence template.

Technically, IL-MINER relies on event pattern matching: for a sequence template $o = \langle o_1, \dots, o_m \rangle$, an event pattern $P_o = (V_o, \Theta_o, \tau_o)$ is derived: $V_o = \langle v_1, \dots, v_m \rangle$ are event variables, one per event template in o; Θ_o contains property constraints that assign for each variable v_i the equivalence predicates of the event template o_i; and τ_o is set as the maximum timespan between the first and the last event of all traces in H.

The result of this step is then obtained by matching pattern P_o over all traces H, i.e., by computing $\Omega(P_o, h)$ for all $h \in H$.

EXAMPLE 6. *Let P_o be the event pattern of sequence template $\langle 4, 3, 1\rangle$ (see Example 5). Then, matching P_o to trace h_1 of Table 2 yields $\Omega(P_o, h_1) = \{\langle e_1, e_2, e_3\rangle\}$. Matching it to trace h_2 yields two matches, $\Omega(P_o, h_2) = \{\langle e_4, e_5, e_6\rangle, \langle e_4, e_5, e_7\rangle\}$.*

Extracting Property Constraints. Based on the matches obtained for pattern P_o for a sequence template $o \in O$, this step extracts the property constraints for the resulting event pattern. As such, this step is similar to learning of relevant event templates as presented above (§4.2). However, it differs in two aspects: first, we consider only constraints that hold true for at least one event in all matches; second, not only constraints with equivalence predicates, but all types of comparison predicates are taken into account.

Inequality predicates are explored solely for numeric attributes. While this is typically intractable for the whole set of events considered in the first phase (sequence template learning), it becomes feasible once the set of considered events has been narrowed based on sequence templates.

Let Γ_{Pro} be the set of candidate property constraints $v_j.A_i \; \phi \; C$, such that v_j, $j \leq m$, is an event variable of pattern P_o with $V_o = \langle v_1, \ldots, v_m\rangle$; A_i is an attribute; $C \in dom(A_i)$ a constant; and $\phi \in \{\doteq, \dot{<}, \dot{\leq}, \dot{>}, \dot{\geq}\}$ is a comparison operator. Then, the set of extracted property constraints, which is denoted by $\Theta_{Pro} = extract_P(\Omega(P_o, h_1), \ldots, \Omega(P_o, h_w))$, contains all the constraints $(v_j.A_i \; \phi \; C) \in \Gamma_{Pro}$, such that for all $h \in H = \{h_1, \ldots, h_w\}$ there exists a match $\langle e_1, \ldots, e_m\rangle \in \Omega(P_o, h)$ and it holds that $e_j.A_i \; \phi \; C$ for some $j \leq m$.

Extracting Relation Constraints. Relation constraints model correlation conditions between pairs of events to be matched. Again, IL-MINER extracts these constraints on the instance-level, using the matches obtained for pattern P_o. Let Γ_{Rel} be the set of all candidate relation constraints $v_j.A_i \; \phi \; v_k.A_l$, such that v_j and v_k, $j \leq m$, $k \leq m$, $j \neq k$, are event variables of pattern P_o with $V_o = \langle v_1, \ldots, v_m\rangle$; A_i and A_l are attributes; and $\phi \in \{\doteq, \dot{<}, \dot{\leq}, \dot{>}, \dot{\geq}\}$ is a comparison operator. Extraction of relation constraints identifies all constraints satisfied by at least one match per trace: $\Theta_{Rel} = extract_R(\Omega(P_o, h_1), \ldots, \Omega(P_o, h_w))$ contains all the constraints $(v_j.A_i \; \phi \; v_k.A_l) \in \Gamma_{Rel}$, such that for all $h \in H = \{h_1, \ldots, h_k\}$ there exists a match $\langle e_1, \ldots, e_m\rangle \in \Omega(P_o, h)$ and it holds that $e_j.A_i \; \phi \; e_k.A_l$ for some $j \leq m$, $k \leq m$, $j \neq k$.

EXAMPLE 7. *We consider $\Omega(P_o, h_1) = \{\langle e_1, e_2, e_3\rangle\}$ as well as $\Omega(P_o, h_2) = \{\langle e_4, e_5, e_6\rangle, \langle e_4, e_5, e_7\rangle\}$ of Example 6. For example, there exists a match for h_1 as well as for h_2, in which the first and the third event have equivalent values for attribute* `priority`. IL-MINER *would therefore extract the relation constraint $v_1.\texttt{priority} \doteq v_3.\texttt{priority}$. In the same vein,* IL-MINER *would extract the constraint $v_2.\texttt{priority} \dot{>} v_3.\texttt{priority}$, since it holds that $e_2.\texttt{priority} \dot{>} e_3.\texttt{priority}$ (match of h_1) and $e_5.\texttt{priority} \dot{>} e_7.\texttt{priority}$ (match of h_2). We note that both constraints have been extracted from different matches of h_2.*

Merging Constraints. As the last step in pattern construction, IL-MINER explores possible conjunctions of the candidate property constraints (Θ_{Pro}) and candidate relation constraints (Θ_{Rel}) based on the respective sequence template $o \in O$ and the traces H. Exploration of possible constraint conjunctions has to take into account that constraints may originate from *different* matches per trace, as illustrated in Example 7. Hence, constraints may be incompatible as their conjunction may no longer be satisfied by any of the matches of a trace. In the above example, this is indeed the case: The conjunction of constraints $v_1.\texttt{priority} \doteq v_3.\texttt{priority}$ and $v_2.\texttt{priority} \dot{>} v_3.\texttt{priority}$ is not satisfied by any of the matches ($\langle e_4, e_5, e_6\rangle, \langle e_4, e_5, e_7\rangle$) of trace h_2.

Algorithm 2: merge$(o, \Theta_{Pro}, \Theta_{Rel}, H)$ – Merging Constraints

Input: o – a sequence template
Θ_{Pro} – a set of property constraints
Θ_{Rel} – a set of relation constraints
H – a set of traces
Output: $\mathcal{P}$ – a set of event patterns

```
/* Construct all maximal subsets of constraints per
   trace, TC is a set of sets of constraints       */
1  TC ← ∅;
2  for h ∈ H do
3      TC ← TC ∪ combine(Θ_Pro, Θ_Rel, h)

   /* Construct all maximal subsets of intersections of
      sets of constraints that have been obtained per
      trace                                          */
4  init_TC ← random_element(TC);
5  PC ← init_TC;
6  for tc ∈ (TC \ {init_TC}) do
7      for c ∈ tc do
8          TC' ← ∅;
9          for pc ∈ PC do
10             if (pc ∩ c) ≠ ∅ then TC' ← TC' ∪ (pc ∩ c) ;
11         PC ← TC';

   /* Construct event patterns based on the constraint
      sets                                           */
12 P ← ∅;
13 for pc ∈ PC do
14     V ← ⟨v_1, ..., v_m⟩ for o = ⟨o_1, ..., o_m⟩;
15     τ ← max_{h∈H}( minimal_timestamp_diff(pc, h));
16     P ← (V, pc, τ);
17 return P;
```

Against this background, IL-MINER creates a set of event patterns based on the maximal sets of property and relation constraints that are compatible, i.e., at least a single match per trace satisfies their conjunction. Moreover, the time window of a pattern is determined: For each trace, we consider the time differences of the first and last event per match and determine the minimum of these differences, referred to as the trace window. Then, the window of the respective event pattern is set to the maximal trace window over all traces.

We formalise this procedure in Algorithm 2, which takes as input a sequence template, sets of property and relations constraints, and the traces. In a first step, all maximal subsets of constraints are computed per trace, such that each event variable can be assigned a *single* event in the trace and all constraints are satisfied (lines 1 to 3). This gives us a set of sets of constraints for each trace. Next, all possible non-empty intersections of the sets of constraints per trace are extracted (lines 4 to 11). Starting with a random set of constraints (selected from the constraint sets of a random trace), intersections are computed with all constraint sets of all other traces. Finally, the actual patterns are constructed for each obtained set of constraints (lines 12 to 16). That is, a set of event variables is defined (induced by the sequence template). In addition, we compute the trace windows (i.e., the minimal difference between the timestamps of events in matches that satisfy the current set of constraints) and set the maximum of the trace windows as the window of the pattern.

EXAMPLE 8. *To illustrate the merging of constraints and construction of event patterns, we consider the following relation constraints, which are a subset of the constraints actually extracted for our running example (again, we assign identifiers to the constraints for later reference):*

$$\begin{aligned}\Theta_{Rel} = \{ \quad I &: v_1.\texttt{status} \doteq v_2.\texttt{status} \\ II &: v_1.\texttt{machine} \doteq v_2.\texttt{machine} \\ III &: v_1.\texttt{job} \doteq v_3.\texttt{job} \\ IV &: v_1.\texttt{priority} \doteq v_3.\texttt{priority} \\ V &: v_1.\texttt{priority} \dot{>} v_3.\texttt{priority} \; \}.\end{aligned}$$

We observe that a conjunction of all these constraints is satisfied by the only match (see Example 6) obtained for trace h_1. For trace h_2, constraints IV and V cannot be used together, so that the maximal sets of constraints for the construction of an event pattern are given as $\{I, II, III, IV\}$ and $\{I, II, III, V\}$.

4.4 Pattern Filtering

The first two phases of IL-MINER's discovery process may yield a large set of event patterns. However, many of these patterns are likely to be rather similar, so that returning all of them is of limited usefulness in practice. Hence, the last phase of IL-MINER's discovery process aims at identifying a representative sample of the discovered patterns. To this end, IL-MINER employs clustering to distinguish of patterns that have different characteristics. Selecting a few patterns from each cluster then yields a sample that exemplifies the basic properties of the discovered patterns.

We consider two dimensions to cluster event patterns, outlined in Table 3. On the one hand, patterns may be assessed based on their syntax or their semantics. On the other hand, each pattern can be considered in isolation (absolute measure) or in relation to other patterns (relative measure). To assess patterns along these dimensions, Table 3 lists several measures. With Δ as the symmetric difference of two sets and $\Omega(P, H) = \bigcup_{h \in H} \Omega(P, h)$ as the set of all matches of pattern P in traces H, these measures are defined as:

- *Structural Simplicity* measures the complexity of an individual pattern (V, Θ, τ) and is defined as $^1/_{1+|\Theta|+|V|}$.
- *Structural Similarity* quantifies the similarity of two patterns in terms of their syntax. For patterns (V, Θ, τ) and (V', Θ', τ'), we define this measure based on the symmetric difference of their constraint sets as $^1/_{1+|\Theta \,\Delta\, \Theta'|}$.
- The *Number of Matches* $|\Omega(P, H)|$ of pattern P in traces H is a measure to quantify its strictness (aka selectivity).
- *Matches Similarity* quantifies the similarity of two patterns P and P' in terms of their semantics. It is defined based on the symmetric difference of their sets of matches obtained for the traces H as $^1/_{1+|\Omega(P,H) \,\Delta\, \Omega(P',H)|}$.

Using the above measures, the filtering procedure of works as follows: For the absolute measures evaluating each pattern in isolation, IL-MINER employs a density-based clustering algorithm and selects one pattern from either end of the measure spectrum. For the relative measures, IL-MINER relies on hierarchical clustering and selects a fixed amount of patterns from the hierarchy. Starting with the maximal measure value, we move down the hierarchy until the number of patterns to select is larger or equal than the number of clusters. Then, one pattern is chosen randomly from each cluster.

4.5 Formal Guarantees

The definition of the problem of event pattern discovery (Problem 1) outlines three dimensions on which any solution is to be evaluated: correctness, minimality, and completeness. To assess to which extent the multi-phase discovery process of IL-MINER meets these requirements, we consider solely the first two phases, *sequence template learning* and *pattern construction*. The third phase, *pattern filtering*, aims at increasing the practical usefulness of the approach and is inherently heuristic.

First and foremost, we observed that IL-MINER's discovery process is correct and minimal. In other words, each of the discovered patterns creates at least a single match for each trace and the discovered set does not include a stricter pattern. Completeness, in turn, can be achieved solely for conjunctive patterns: a pattern $P = (V, \Theta, \tau)$ is *conjunctive*, if all (property and relation) constraints in Θ are built of equivalence predicates (operator $\doteq$). The restriction to conjunctive patterns stems from the fact that the discovery of the constant C of property constraints built of inequality predicates (operators $\dot{<}$, $\dot{\leq}$, $\dot{>}$, $\dot{\geq}$) cannot be guaranteed as it depends on the attribute values of the events in the traces.

Table 3: Overview of measures to filter event patterns.

	Individual Pattern (absolute)	**Pairs of Patterns (relative)**
Syntax	Structural Simplicity	Structural Similarity
Semantics	Number of Matches	Matches Similarity

Following the structure of Problem 1, we formalise these guarantees as follows:

THEOREM 1. *Let H be a set of traces and $\mathcal{P}$ be the set of patterns discovered by* IL-MINER *(without filtering).*

- *$\mathcal{P}$ is correct: for all patterns $P \in \mathcal{P}$ and traces $h \in H$ it holds that $|\Omega(P, h)| > 0$;*
- *$\mathcal{P}$ is minimal: for all patterns $P, P' \in \mathcal{P}$ it holds that neither $P \prec P'$ nor $P' \prec P$.*
- *$\mathcal{P}$ is complete for conjunctive patterns; if there exists a conjunctive pattern P with $|\Omega(P, h)| > 0$ for all $h \in H$, but there does not exist a conjunctive pattern P' with $P' \prec P$ and $|\Omega(P', h)| > 0$ for all $h \in H$, then it holds that $P \in \mathcal{P}$.*

The proof can be found in Appendix A.

5. SCALABILITY CONSIDERATIONS

Event pattern discovery typically has to cope with a large number of traces, each having a large number of events. To address the resulting challenges in terms of scalability of the discovery process, we outline optimisations that have been incorporated in IL-MINER.

Efficient Set Intersection. IL-MINER's discovery process often requires computation of intersections of sets (e.g., of event templates or constraints). To achieve an efficient implementation of the intersection operation for sets, bitwise operations over a binary encoding of sets can be exploited: each set is encoded as a binary feature vector, so that a bitwise AND yields a vector representing the intersection.

Exploiting Event Template Dependencies. When preparing the sequences of sets of event templates for frequent sequence mining, some templates may be removed to speed-up the mining process. First, if an event template t can be expressed as the conjunction of two other templates, say t' and t'', template t can be removed before the mining process. In the obtained sequences of sets of event templates, template t is added again to each set containing t' and t''. Consequently, no information is lost in the discovery process.

Assessing Event Template Relevance. A particular event template may be considered to be irrelevant, if it occurs in each set of a sequence of all sequences obtained from all traces. In this case, it may be removed before the frequent sequence mining and added again to the obtained frequent sequences of event templates. However, we note that this optimisation may be lossy, if the removal yields an empty set for some position in a sequence of sets of templates. In that case, correctness, minimality, and completeness of discovery (see §4.5) are no longer guaranteed.

If trading the formal guarantees of discovery for increased runtime performance is a viable option, a more drastic approach may be followed to reduce the number of event templates that are subject to frequent sequence mining. The general idea is to remove templates that appear to be less relevant for identifying sequence templates. With R as the set of relevant event templates, we estimate the probability $P(t)$ for template $t \in R$ being part of an

extracted set of templates (i.e., those that appear in one of the sequence templates O) by the ratio of the number of sets that contain t and the number of all sets. For $P(t)$, we then calculate the Shannon information content as $c(t) = log_2(^1/_{P(t)})$, see [14]. The entropy of R is calculated as the average Shannon information content $\mathcal{H} = \sum_{t \in R} P(t) log_2(^1/_{P(t)})$. Then, each event template for which it holds that $c(t) < \mathcal{H}$ is not considered further in the discovery process.

6. EVALUATION

We evaluate IL-MINER using four real-world datasets from different domains. We first show that event pattern discovery with IL-MINER is indeed highly effective (§6.2). Second, we demonstrate that IL-MINER discovers event patterns in large-scale datasets within a matter of seconds (§6.3). Third, we compare the effectiveness of IL-MINER with iCEP, a state-of-the-art system for event pattern discovery (§6.4).

6.1 Experimental Setup and Datasets

Experimental Workflow. To assess the effectiveness of IL-MINER, we rely on the experimental workflow outlined in Fig. 5. The input of IL-MINER are traces in which the situation of interest occurred. Our real-world event streams do not provide labels for particular situations to detect. Therefore, for each dataset, we consider a workload of several patterns and split the event data into a training dataset D_{train} (∼20% of the data) and an evaluation dataset D_{eval} (∼80% of the data). Then, for each pattern P, we obtain the matches M_{train} and M_{eval} for datasets D_{train} and D_{eval}, respectively. These matches serve as the situation of interest that shall be detected by the discovered pattern.

As the starting point for event pattern discovery, traces are obtained from the training dataset D_{train} by splitting the respective event stream into traces based on the matches M_{train}. The obtained traces are used as input for event pattern discovery, which yields a pattern P^* (or a set thereof). The discovered pattern P^* is compared with pattern P in terms of the matches M_{eval} and M^*_{eval} obtained by both patterns for the evaluation dataset. It is important to note that even if P is conjunctive, P^* may differ from P as it can be stricter, $P^* \prec P$. If so, pattern P^* may yield less matches in dataset D_{eval} compared to P, i.e., $|M^*_{eval}| < |M_{eval}|$.

Evaluation Measures. A comparison of matches M^*_{eval} and M_{eval} yields sets of true positive (TP), false positive (FP), and false negative (FN) matches. A match in M^*_{eval} is a TP, if the timestamp of the last event in the match has the same timestamp as the last event of a match in M^*_{eval}, otherwise it is a FP. A match in M_{eval} is a FN, if there exists no match in M^*_{eval} for which the timestamp of the last event coincides with the one of the last event of this match.

Based on these measures, precision and recall are calculated. Precision denotes how well a pattern only detects the situation of interest and is defined as $pre = {}^{TP}/_{TP+FP}$. Recall denotes how well a pattern finds all situations of interest. It is defined as $rec = {}^{TP}/_{TP+FN}$.

Datasets. We use four real-world datasets from different domains, namely urban traffic, finance, and cluster management. We consider a workload of three event patterns per dataset to execute the above experimental workflow. Half of the patterns are conjunctive. Below, we describe the intuition of all patterns. Their comprehensive specification can be found in Appendix B.

Dublin: This dataset comprises events used for urban traffic management in the city of Dublin[1]. The events describe the status and position of buses for one month. We consider three patterns that are relevant for managing the punctuality of buses (time windows between 1min and 10min): Pattern $Dublin_{p1}$ describes a monotonic increase of the delay of buses of a particular line, measured at bus stops, over a sequence of three events. Pattern $Dublin_{p2}$ matches two events of the same vehicle that indicate decreasing delay. Pattern $Dublin_{p3}$ describes a busy bus stop, i.e., three buses need to arrive at the same stop within a short period of time.

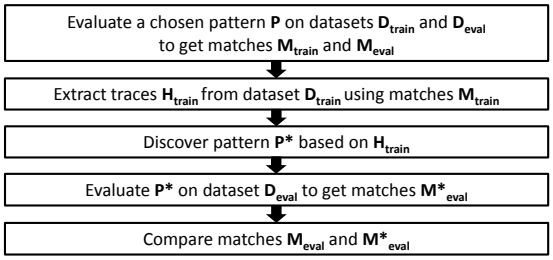

Figure 5: Experimental workflow

Stock: The second dataset describes intraday stock trade events from the NASDAQ in November 2010. Again, we consider three patterns (time windows between 50sec and 10min): Pattern $Stock_{p1}$ captures an increase of the lowest stock trade value of two consecutive trades of Google stocks. Pattern $Stock_{p2}$ has been adopted from [11] and detects trades of three specific stocks (MSFT, GOOG, and AAPL) with increasing volumes. $Stock_{p3}$ detects sequences of four trade events (any stock) with equal volumes.

DEBS: The third dataset was published as part of the ACM DEBS 2015 Grand Challenge [10] and also represents an urban transportation scenario. The events are a stream of trip reports of taxis in New York City in 2013. The first pattern, $DEBS_{p1}$, is related to a task posted in the Grand Challenge: it identifies taxis driving in a specific direction based on their GPS coordinates (time window 6min). Pattern $DEBS_{p2}$ detects sequences of three trip events (within 80sec) with equal payment types, also comparing the distances of these trips. Pattern $DEBS_{p3}$ detects sequences of three trip events (within 80sec) with different payment types and further compares four properties of these trips (distance, tip received by driver, fare, total amount paid).

Cluster: The Google cluster traces [19, 20] used for illustration in this paper are used as a fourth dataset. They contain the events related to job scheduling in a cluster of about 12.5k machines over a whole month. We consider three patterns that monitor the execution of jobs (time windows between 100min and 100sec): Pattern $Cluster_{p1}$ detects jobs that are submitted, killed and resubmitted on the same machine. Pattern $Cluster_{p2}$ matches four events that indicate finished jobs on the same machine. The third pattern, $Cluster_{p3}$, introduced already in Example 1, detects termination of a job due to another job being scheduled on the same machine.

Implementation. IL-MINER is implemented using the Java programming language and is publicly available[2]. All experiments where conducted on a Desktop with an Intel Core i7-3770 CPU @3.40Ghz and 16GB RAM.

6.2 How effective is event pattern discovery with IL-Miner?

We first explore the effectiveness of IL-MINER by executing the above mentioned experimental workflow (Fig. 5) for each workload pattern P. Assessing the quality of the best discovered pattern P^* based on the comparison of M_{eval} and M^*_{eval}, the precision and

[1] http://dublinked.com/datastore/datasets/dataset-304.php

[2] https://github.com/Xerxekyran/IL-Miner

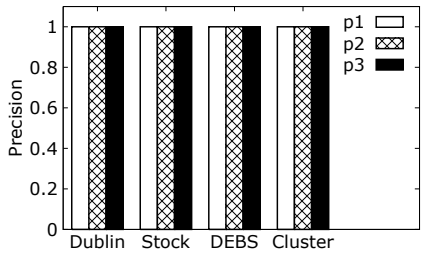

Figure 6: Precision for all patterns

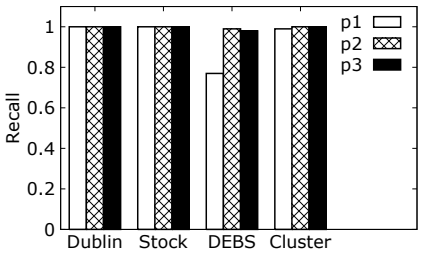

Figure 7: Recall for all patterns

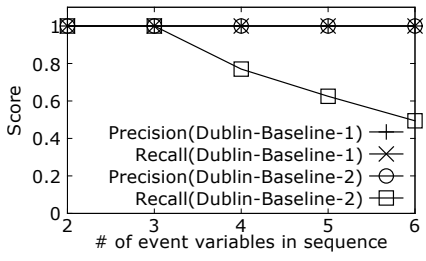

Figure 8: Increasing number of event variables

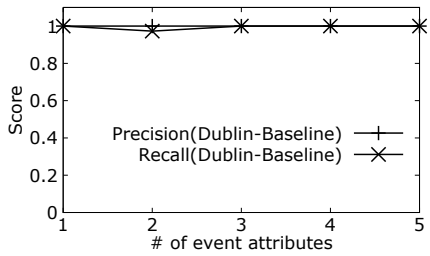

Figure 9: Increasing number of property constraints per event variable

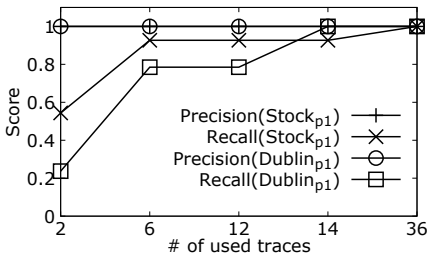

Figure 10: Recall and precision when increasing number of used traces

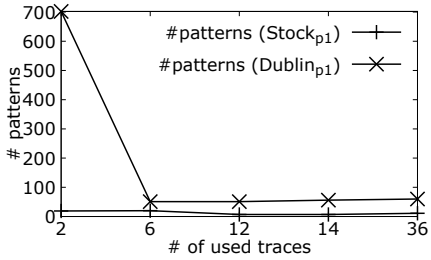

Figure 11: Number of discovered patterns when increasing number of used traces

recall is shown in Fig. 6 and Fig. 7, respectively. For all patterns, precision is 1.0. While this is expected for the six conjunctive patterns, it highlights IL-MINER's effectiveness in terms of precision also if there are no formal guarantees for the discovery process. Recall is generally high and only drops to 0.8 for one pattern. We stress that a recall value of 1.0 is not guaranteed even for conjunctive patterns (IL-MINER may learn a stricter pattern from the training dataset), so that Fig. 7 witnesses generally high effectiveness of IL-MINER in terms of recall.

Next, we explore the impact of various parameters of the pattern P, which characterises the situation of interest, on effectiveness of discovery. Based on pattern $Dublin_{p1}$, we create two baseline patterns, each comprising only property constraints over a single attribute (`lineID` and `atStop`, respectively). Increasing the length of the sequence of event variables in both patterns, Fig. 8 shows that the obtained results. Recall for the pattern based on attribute `lineID` is not affected, whereas recall worsens with increased sequence length for the pattern based on `atStop`—which suggests that the effect largely depends on the stream characteristics.

Similarly, Fig. 9 shows the results obtained for a baseline pattern (using the $Dublin$ data) of fixed sequence length when varying the number of property constraints per event variable. Despite a glitch in recall when a stricter pattern was discovered, we do not observe that this parameter influences the results.

Turning to the influence of the number of traces on the discovery effectiveness, Fig. 10 shows the results obtained for patterns $Dublin_{p1}$ and $Stock_{p1}$. Increasing the number of traces increases the recall, since more traces allow for a better generalisation and thus enable discovery of more accurate patterns. Still, perfect recall is reached in both cases with 36 traces, which is remarkably small subset of the up to 8000 traces obtained from the training dataset. However, a minimal number of traces is needed to obtain a set of discovered patterns of manageable size, see Fig. 11.

Since discovery may yield tens or even hundreds of patterns, we also evaluate the filtering phase of IL-MINER. For all experiment runs resulting in more than one pattern, we found that a random selection of five patterns includes the best pattern in only 21% of the cases. Selecting five patterns using IL-MINER's filtering strategy results in the best pattern being selected in 71% of the experiments.

Table 4: Excerpt of dataset sizes.

Dataset	# total events	Pattern	# traces	# events in traces
$Dublin$	9,546,612	$Dublin_{p1}$	827	6,204
$Stock$	7,176,930	$Stock_{p1}$	3,413	3,199,033
$DEBS$	980,319	$DEBS_{p1}$	14	1,880
$Cluster$	9,489,484	$Cluster_{p1}$	4,001	17,744

6.3 How efficient is event pattern discovery with IL-Miner?

To assess the efficiency of IL-MINER, we measure the absolute execution times of the discovery process for the first pattern of each dataset. The measurements cover the third step (discover pattern $P*$ based on H_{train}) of the experimental workflow outlined in Fig. 5. We note that the construction of traces H_{train} yields diverse results for the four datasets, in terms of the number of traces as well as the number of events in these traces, see Table 4.

Fig. 12 shows the measured absolute execution times. Run-times between milliseconds and tens of seconds witness a high variability, but demonstrate general feasibility. Interestingly, we observe high variability of the amount of time spent for particular phases of the discovery process. Fig. 13 depicts the relative share of the learning of relevant event templates (*Relevant*), the learning of sequence templates (*Order*), and the pattern construction phase (*Constraint*). A follow-up analysis reveals that these differences are largely due to the structure of the patterns and the characteristics of the event stream: a large number of relation constraint candidates as well as very similar and long traces influence the relative amount of time used for the phases of the discovery process.

6.4 How does IL-Miner compare to the state-of-the-art?

We compare the performance of IL-MINER with iCEP [15], a recently proposed system for event pattern discovery. To this end, we follow the above experimental workflow. Since iCEP has limited support for learning event abstractions, for the comparison, we had to simplify the discovery task and rely on patterns that define a unique property constraint per event variable. We considered six such patterns, which are also included in Appendix B.

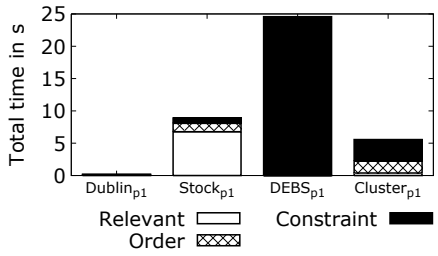

Figure 12: Execution time absolute

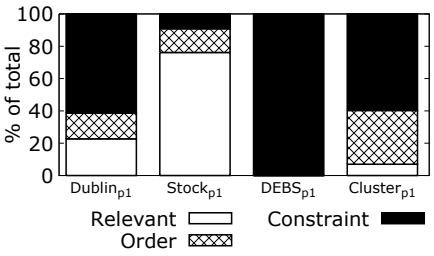

Figure 13: Execution time relative

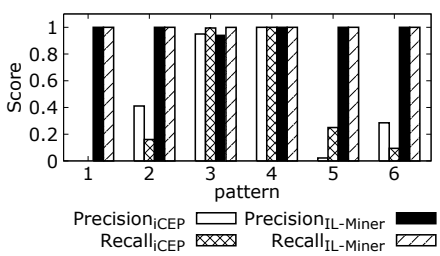

Figure 14: Comparison with iCEP

The obtained precision and recall values, shown in Fig. 14, indicate large differences between iCEP and IL-MINER. iCEP fails to discover accurate event patterns for many scenarios. In particular, event patterns that are built of event variables for which overlapping sets of property constraints have been defined turn out to be problematic. IL-MINER discovers a much broader range of event patterns, thereby consistently achieving high precision and high recall.

7. RELATED WORK

Closest to our work is the aforementioned iCEP miner introduced by Margara et al. [15]. It aims at discovering event patterns from a given set of traces, thereby assuming the same setting as IL-MINER. iCEP supports limited discovery of event abstractions. Given an initial definition, it discovers additional property constraints that may refine an event abstraction. However, the actual detections of the order of an event pattern is done purely on the type-level, which renders it impossible to discover patterns that require multiple occurrences of the same event abstraction. As shown in §6.4, this is often not sufficient to discover comprehensive patterns that detect a situation of interest.

Moreover, event pattern discovery relates to work on mining of sequential patterns, see [1, 23, 26]. In particular, frequent sequences may be identified based on an interval-based event model [18], which accommodates for the need to consider time windows. However, as virtually all traditional approaches for mining of sequential patterns, this approach requires the a priori definition of event abstractions. The definition of constraints for sequential patterns has been explored in [8]. Yet, in [8], constraints are used to inject external knowledge into the discovery process to improve mining performance. In our work, constraints are an integral part of the pattern model, defining event abstractions and correlation conditions.

Pattern discovery has also been investigated for data streams, streams of atomic data entities. The CrossMatch algorithm [22] identifies local patterns in data streams using dynamic time warping, while SPIRIT [17] detects correlations and hidden variables corresponding to trends in co-evolving data streams. All these approaches have been devised for a coarse-grained data model and do not target the discovery of fine-granular event abstractions and correlation conditions.

8. CONCLUSION

We have presented IL-MINER, a system that, given a set of traces, discovers event patterns. In particular, IL-MINER does not require an a priori definition of event abstractions. In a multi-phase process, IL-MINER identifies relevant abstractions for the construction of event patterns, exploits frequent sequence mining over these abstractions, and finally learns constraints that encode correlation conditions. We have shown that this process is not only correct and minimal, but also complete for conjunctive event patterns. Finally, our experiments with four real-world datasets highlight that IL-MINER is able to discover a broad range of event patterns that could not be found by a state-of-the-art discovery algorithm. In future work, we aim at exploring event pattern discovery for noisy traces. Moving to a probabilistic model, this raises the question of how to derive error bounds for the correctness and completeness of the discovery process.

9. REFERENCES

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In ICDE, pages 3–14. IEEE, 1995.

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

[3] A. Artikis et al. Heterogeneous stream processing and crowdsourcing for urban traffic management. In EDBT, pages 712–723. OpenProceedings.org, 2014.

[4] L. Brenna, A. J. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, and W. M. White. Cayuga: a high-performance event processing engine. In SIGMOD, pages 1100–1102. ACM, 2007.

[5] B. Chandramouli, M. H. Ali, J. Goldstein, B. Sezgin, and B. S. Raman. Data stream management systems for computational finance. *IEEE Computer*, 43(12):45–52, 2010.

[6] Y. Engel, O. Etzion, and Z. Feldman. A basic model for proactive event-driven computing. In DEBS, pages 107–118. ACM, 2012.

[7] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Company, 2010.

[8] M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In *VLDB*, volume 99, pages 7–10, 1999.

[9] M. L. Hill, M. Campbell, Y. Chang, and V. S. Iyengar. Event detection in sensor networks for modern oil fields. In DEBS, pages 95–102. ACM, 2008.

[10] Z. Jerzak and H. Ziekow. The DEBS 2015 grand challenge. In DEBS, pages 266–268. ACM, 2015.

[11] I. Kolchinsky, I. Sharfman, and A. Schuster. Lazy evaluation methods for detecting complex events. In DEBS, pages 34–45. ACM, 2015.

[12] D.-I. Lin and Z. M. Kedem. Pincer-search: A new algorithm for discovering the maximum frequent set. In EDBT, pages 103–119. Springer, 1998.

[13] M. Liu, M. Ray, D. Zhang, E. A. Rundensteiner, D. J. Dougherty, C. Gupta, S. Wang, and I. Ari. Realtime healthcare services via nested complex event processing technology. In EDBT, pages 622–625. ACM, 2012.

[14] D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[15] A. Margara, G. Cugola, and G. Tamburrelli. Learning from the past: automated rule generation for complex event processing. In DEBS, pages 47–58. ACM, 2014.

[16] Y. Mei and S. Madden. Zstream: a cost-based query processor for adaptively detecting composite events. In SIGMOD, pages 193–206. ACM, 2009.
[17] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In VLDB, pages 697–708. VLDB Endowment, 2005.
[18] D. Patel, W. Hsu, and M. L. Lee. Mining relationships among interval-based events for classification. In SIGMOD, pages 393–404. ACM, 2008.
[19] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In SOCC, page 7. ACM, 2012.
[20] C. Reiss, J. Wilkes, and J. Hellerstein. Google cluster-usage traces: format and schema. 2013.
[21] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In EDBT, LNCS 1057, pages 1–17. Springer, 1996.
[22] M. Toyoda, Y. Sakurai, and Y. Ishikawa. Pattern discovery in data streams under the time warping distance. *VLDB J.*, 22(3):295–318, 2013.
[23] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In ICDE, pages 79–90. IEEE, 2004.
[24] M. Weidlich, H. Ziekow, A. Gal, J. Mendling, and M. Weske. Optimizing event pattern matching using business process models. *IEEE Trans. Knowl. Data Eng.*, 26(11):2759–2773, 2014.
[25] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In SIGMOD, pages 407–418. ACM, 2006.
[26] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large datasets. In SDM, pages 166–177, 2003.
[27] B. Yu, W. H. Lam, and M. L. Tam. Bus arrival time prediction at bus stop with multiple routes. *Transportation Research Part C: Emerging Technologies*, 19(6):1157 – 1170, 2011.
[28] H. Zhang, Y. Diao, and N. Immerman. On complexity and optimization of expensive queries in complex event processing. In SIGMOD, pages 217–228. ACM, 2014.

APPENDIX

A. PROOF OF THEOREM 1

THEOREM 1. *Let H be a set of traces and $\mathcal{P}$ be the set of patterns discovered by* IL-MINER *(without filtering).*

- *$\mathcal{P}$ is correct: for all patterns $P \in \mathcal{P}$ and traces $h \in H$ it holds that $|\Omega(P,h)| > 0$;*
- *$\mathcal{P}$ is minimal: for all patterns $P, P' \in \mathcal{P}$ it holds that neither $P \prec P'$ nor $P' \prec P$.*
- *$\mathcal{P}$ is complete for conjunctive patterns; if there exists a conjunctive pattern P with $|\Omega(P,h)| > 0$ for all $h \in H$, but there does not exist a conjunctive pattern P' with $P' \prec P$ and $|\Omega(P',h)| > 0$ for all $h \in H$, then it holds that $P \in \mathcal{P}$.*

Proof. Let H be a set of traces and $\mathcal{P}$ be the set of patterns discovered by IL-MINER (without filtering).

Correctness. Suppose that there exist a pattern $P \in \mathcal{P}$ and a trace $h \in H$ such that $\Omega(P,h) = \emptyset$. Backtracking the discovery process yields a contradiction of this assumption: Since P was returned by Algorithm 2, there exists a sequence template o and a constraint set pc from which P was created. The time window of P is selected as the maximal difference between events satisfying pc over all traces, including h. Hence, $\Omega(P,h) = \emptyset$ can only be caused by pc being not satisfied by any subsequence of events in h. Constraint set pc is a maximal subset of intersections of constraint sets obtained per trace, including h. Hence, all constraints in pc are part of $tc \in TC$ (see Algorithm 2) created for h. Set tc, in turn, is constructed from a *single* match m, obtained by evaluating the pattern P_o for sequence template o over h. This contradicts the assumption of pc being not satisfied by any subsequence of events in h.

Minimality. Suppose there exist patterns $P, P' \in \mathcal{P}$ and $P \prec P'$. Hence, $\Omega(P,S') \subset \Omega(P',S')$ for any event sequence S'. This means that either $\tau' < \tau$ or $\Theta' \subset \Theta$ for the respective time windows and sets of constraints, respectively. Since time windows are determined based on the maximum over all traces of the minimal timespan between events satisfying the respective constraint sets, $\tau' < \tau$ actually implies $\Theta' \subset \Theta$, so that it suffices to show that this yields a contradiction. $\Theta' \subset \Theta$ implies that both sets are part of PC in Algorithm 2, which contradicts the fact that PC contains only maximal sets (lines 8 to 11 in Algorithm 2).

Completeness. We show completeness by construction. Let $P = (V, \Theta, \tau)$ be a pattern with $|\Omega(P,h)| > 0$ for all $h \in H$. Suppose that there is no stricter pattern P' with $|\Omega(P',h)| > 0$ for all $h \in H$ (this case is covered by the minimality property already). Consider Algorithm 1: Set seq contains a sequence of sets of event templates that comprises all predicates of property constraints in Θ of P. This sequence must be part of F, since P creates matches based on the property constraints built from these templates for all traces H. Further, each sequence of individual event templates of property constraints in Θ is part of O, the result of Algorithm 1.

Consider one of these sequences $o \in O$: For the respective pattern P_o, it holds $\Omega(P,h) \subseteq \Omega(P_o,h)$ for all $h \in H$. Now, consider Algorithm 2: since P is conjunctive, for each event and attribute in a match in $\Omega(P_o,h)$, there is solely a single property constraint that can be part of TC and, since it is the intersection of constraints per trace, also in PC. Hence, by computing the maximal subsets of constraint intersections, we have $\Theta \in PC$. Finally, consider the time window of the pattern created for this constraint set, i.e., the maximum over all traces of the minimal timespan between events satisfying Θ per trace. Any time window smaller than τ, the window of P, would mean that there exists a trace h for the which minimal timespan between events satisfying Θ is larger than τ, so that $\Omega(P,h) =$. On the other hand, any time window larger than τ contradicts the assumption of not having a stricter conjunctive pattern P'. Hence, we conclude that $P \in \mathcal{P}$. □

B. PATTERN WORKLOADS

Patterns are expressed using the SASE language, as defined in [28].

Urban transportation: Dublin bus events

```
-- Schema of Bus: vehicleID (int),
--    lineID (int), stopID (int),
--    atStop ({0,1}), delay (float),
--    operator (string), journey (string)

-- Pattern Dublin_p1
Pattern SEQ(Bus a, Bus b, Bus c)
Where a.lineID=b.lineID=c.lineID
And a.delay<b.delay<c.delay
And a.atStop=b.atStop=c.atStop=1
Within 10 minutes

-- Pattern Dublin_p2
Pattern SEQ(Bus a, Bus b)
Where a.atStop=b.atStop=0
And a.vehicleID=b.vehicleID
And a.delay>b.delay
Within 10 minutes
```

```
-- Pattern Dublin_p3
Pattern SEQ(Bus a, Bus b, Bus c)
Where a.atStop=b.atStop=c.atStop=1
And a.stopID=b.stopID=c.stopID
Within 1 minutes
```

Patterns (unique property constraint per event variable) for the comparison with iCEP, based on the Dublin bus events

```
-- Pattern_1
Pattern SEQ(Bus a, Bus b)
Where a.lineID=46 And b.lineID=38
And a.delay=b.delay
Within 200 seconds

-- Pattern_2
Pattern SEQ(Bus a, Bus b, Bus c)
Where a.lineID=c.lineID=46 And b.lineID=38
Within 15 minutes

-- Pattern_3
Pattern SEQ(Bus a, Bus b)
Where a.lineID=46 And b.lineID=38
And a.delay>-1000 And a.delay<1000
And b.delay>-1000 And b.delay<1000
Within 15 minutes

-- Pattern_4
Pattern SEQ(Bus a, Bus b, Bus c, Bus d)
Where a.lineID=46 And b.lineID=38
And c.lineID=27 And d.lineID=83
Within 15 minutes

-- Pattern_5
Pattern SEQ(Bus a, Bus b, Bus c, Bus d)
Where a.lineID=46 And b.lineID=38
And c.lineID=27 And d.lineID=83
And a.delay<b.delay And c.delay=d.delay
Within 15 minutes

-- Pattern_6
Pattern SEQ(Bus a, Bus b)
Where a.lineID=31 And b.lineID=38
And a.journey='00311001' And b.atStop=1
Within 15 minutes
```

Finance: NASDAQ stock ticker events

```
-- Schema of Stock: ticker (string),
--   per (string), date (long),
--   open (float), high (float),
--   low (float), close (float), vol (float)

-- Pattern Stock_p1
Pattern SEQ(Stock a, Stock b)
Where a.ticker=b.ticker='GOOG'
And a.low>b.low
Within 800 seconds

-- Pattern Stock_p2
Pattern SEQ(Stock a, Stock b, Stock c)
Where a.ticker='MSFT' And b.ticker='GOOG'
And c.ticker='AAPL' And a.vol<b.vol
And b.vol<c.vol
Within 10 minutes

-- Pattern Stock_p3
Pattern
SEQ(Stock a, Stock b, Stock c, Stock d)
Where a.vol=b.vol And a.vol=c.vol
And a.vol=d.vol
Within 50 seconds
```

Urban transportation: NYC taxi events

```
-- Schema of Taxi: medallion (string),
--   license (string), pickup (datetime),
--   dropoff (datetime), trip_time (int),
--   distance (float), pickup_lon (float),
--   pickup_lat (float),
--   dropoff_lon (float),
--   dropoff_lat (float), payment (string),
--   fare (float), tip (float),
--   total (float)

-- Pattern DEBS_p1
Pattern SEQ(Taxi a, Taxi b)
Where a.medaillon=b.medaillon
And b.pickup_lon<a.dropoff_lon
And b.pickup<a.dropoff_lat
Within 6 minutes

-- Pattern DEBS_p2
Pattern SEQ(Taxi a, Taxi b, Taxi c)
Where a.payment='CSH' And b.payment='CSH'
And c.payment='CSH'
And a.distance<b.distance
And b.distance=c.distance
Within 80 seconds

-- Pattern DEBS_p3
Pattern SEQ(Taxi a, Taxi b, Taxi c)
Where a.payment='CRD' And c.payment='CRD'
And b.payment='CSH' And b.tip<a.tip
And b.fare<a.fare And c.total=b.total
And c.distance=b.distance
Within 80 seconds
```

Cluster management: Google cluster traces

```
-- Schema of Task: missing (string),
--   job (string), task (int),
--   machine (string), class (string),
--   taskType ({0,...,8}), user (string),
--   priority (float), reqCPU (float),
--   reqRAM (float), reqHDD (float),
--   con ({0,1})

-- Pattern Cluster_p1
Pattern SEQ(Task a, Task b, Task c)
Where a.status=c.status=0 And b.status=5
And a.job=b.job And a.job=c.job
And a.machine=b.machine
And a.machine=c.machine
Within 60 minutes

-- Pattern Cluster_p2
Pattern SEQ(Task a, Task b, Task c, Task d)
Where a.machine=b.machine
And a.machine=c.machine
And a.machine=d.machine
And a.status=4 And b.status=4
And c.status=4 And d.status=4
Within 100 minutes

-- Pattern Cluster_p3
Pattern SEQ(Task a, Task b, Task c)
Where a.machine=b.machine And a.job=c.job
And a.status=1 And b.status=1 And c.status=2
Within 100 seconds
```