Pattern Matching on Event Streams & Time Series & ... 

Bernhard Seeger 
University of Marburg
Overview of This Talk

- Introduction to Event Processing
- Introduction to (Event) Pattern Matching
- Highlight Features of current state-of-the-art
- Interval-based Pattern Matching Developed @ UMR
- Group Patterns
- Brief Outlook: Pattern Matching on Property Graphs
Introduction to Event Processing
1. Motivation

- reactive monitoring of time-critical business processes
- predictions about the near future and recommendations for actions
Situations of Interest

**Root Cause Event**

- Benefit Opportunity

**Impact**

- Reaction Costs

**Options**

Time

- E-2
- E-1
- E
- E+1
- E+2
- E+3
- E+4
- E+7
- E+8
- E+9
- E+10
Many application domains

• Algorithmic trading

• Logistics

• Traffic management

• Internet of Things & Industry 4.0

• System Monitoring & Security
Example: Traffic Management

- Data from Sensors
  \[ \text{HighwayStream( lane, speed, length, timestamp )} \]
- Continuous Flow \( \rightarrow \) Data Stream
  - variable data rates
  - time and space as default dimensions
- Queries
  - continuously processed

"At which positions of the highway was the average speed below 30 km/h within the last 15 minutes?"
What is an Event?

- **Basic events** (aka temporal data item): \((o, p, t)\)
  - Object \(o\)
  - Property \(p\)
  - Time point or a time interval \(t\)

- Derived events (aka *complex events*)

- Event stream
  - Temporally ordered *sequence of events* with the same schema received from an active event source
Basic Idea of Event Processing

- **Event Sources**
  - Continuous production of events

- **Event Processing Agents (EPAs)**
  - Processing building blocks: Consume input events & process them & deliver output events

- **Event Sinks**
  - Consumers of events
Comparison

Time Series Database Systems

• Data items are persistent
  • Time Series
• Queries are ephemeral
  • Throughput optimization

Event Processing Systems

• Data items are ephemeral
  • Event Streams
• Queries are persistent
  • Latency optimization

• Applications need both systems working hand in hand
  • COMMONITIES
    • Data Model
    • Query Language
Problems and Issues

• Performance
  • High Throughput vs. Low Latency

• Event Store (= Time Series Database)
  • Persistent management of very fast arriving events

• Functionality
  • Powerful pattern matching
  • Spatial processing
Problems and Issues

• Performance
  • High Throughput vs. Low Latency

• Event Store
  • Persistent Management of Events

• Functionality
  • Powerful pattern matching
  • Spatial and graph processing
The Performance Issue of CEP

- Esper
- Spark Streaming
- Spark

- Low latency
- High latency
- Low throughput
- High throughput
The Persistence Issue of Event Processing

• Event processing systems are in-memory only.
  • Volatile Data and Persistent Queries

• Applications require persistence of events.
  • Analysis of historical data → Anomaly Patterns
  • Revision and reproducibility

• Special Requirements
  • Extremely high input rates (millions of events/s)
  • Time-based queries on massive databases

• Are standard DBMS or NoSQL systems suitable?
The Functionality Issue

• Pattern Matching is the Core Operator for Event Processing
  • Example of an alarm pattern
    Detect a *sharp increase* in temperature together with *sufficiently large amount* of smoke within a *short period* of time.

• Example of an aggressive driver pattern (see AAA)
  A sharp acceleration followed by hard braking, both accompanied by a period of speeding.
Selection of Literature


Pattern Matching: Introduction
Pattern Matching

• Process a stream of data and find a specific pattern within the stream
  • Basically a sequence of conditions mapped to a subsequence of the input stream

• Example
  • Find a sequence of temperature sensor events for which the following holds:
    • The first event provides a temperature below 50°
    • The subsequent events deliver a monotonically increasing temperature
    • Finally, the temperature raises above 90°

• Difference to traditional relational data processing
  • (temporal) order is important
Pattern Matching - Example

A: value < 50°

B: value > previous value

C: value >= 90°
Pattern Matching - Example

A: value < 50°
B: value > previous value
C: value >= 90°

Pattern: \[ AB[... ]BC \]
Pattern Matching - Example

A: value < 50°
B: value > previous value
C: value >= 90°

Pattern: AB[...]BC

Filter
Pattern Matching - Example

A: value < 50°
B: value > previous value
C: value >= 90°

Pattern: A[... ]B
Pattern Matching - Example

A: value < 50°
B: value > previous value
C: value >= 90°
Pattern Matching - Example

A: value < 50°
B: value > previous value
C: value >= 90°
Pattern Matching - Example

A: value < 50°
B: value > previous value
C: value >= 90°

Pattern: AB[...]BC
Pattern Matching - Example

A: value < 50°
B: value > previous value
C: value >= 90°

Pattern: AB[...]BC
Pattern Matching - Example

A: value < 50°
B: value > previous value
C: value >= 90°

Pattern: AB[...]BC
Pattern Matching - Example

A: value < 50°
B: value > previous value
C: value >= 90°
Pattern Matching - Example

- A: value < 50°
- B: value > previous value
- C: value >= 90°

Pattern: AB[...]BC

MATCH
MATCH RECOGNIZE

STATE-OF-THE-ART PATTERN MATCHING
Introduction

• Row based pattern matching language supporting
  • Aggregations
  • Dependencies between pattern variables and arbitrary rows
  • Fine grained control of how, how many, and when results are delivered

• SQL:2016 Standard

• (Partially) implemented in a few database systems and event systems
  • Oracle (since 12c)
  • Esper
  • Apache Flink
Example

Find a **steadily increasing** temperature, eventually reaching a **value above a threshold**

```
SELECT *
FROM Sensors MATCH_RECOGNIZE (PARTITION BY SENSOR_ID
  MEASURES FIRST(A.TIME) AS START,
  B.TIME AS END
  PATTERN A+ B
  DEFINE
    A AS A.TEMP > PREV(A.TEMP)
    B AS B.TEMP > 90
)
```
Basic Match Recognize Syntax

SELECT ...
FROM ... MATCH_RECOGNIZE (
  [PARTITION BY ...]
  [ORDER BY ...]
  MEASURES ...
  [ONE | ALL] ROW PER MATCH
  AFTER MATCH SKIP TO ...
  PATTERN ...
  DEFINE ...
  )
Basic Match Recognize Syntax

```
SELECT ...
FROM ...
MATCH_RECOGNIZE ( 
    [PARTITION BY ...]
    [ORDER BY ...]
    MEASURES ...
    [ONE|ALL] ROW PER MATCH
    AFTER MATCH SKIP TO ...
    PATTERN ...
    DEFINE ...
)
```

Source relation/ event stream
Basic Match Recognize Syntax

SELECT ...
FROM ... MATCH_RECOGNIZE ( 
  [PARTITION BY ...]
  [ORDER BY ...]
  MEASURES ...
  [ONE\|ALL] ROW PER MATCH
  AFTER MATCH SKIP TO ...
  PATTERN ...
  DEFINE ...
)

Variable definitions
Basic Match Recognize Syntax

```
SELECT ...
FROM ... MATCH_RECOGNIZE (  
    [PARTITION BY ...]  
    [ORDER BY ...]  
    MEASURES ...
    [ONE|ALL] ROW PER MATCH  
    AFTER MATCH SKIP TO ...  
    PATTERN ...
    DEFINE ...
)
```
Basic Match Recognize Syntax

SELECT ...
FROM ...
MATCH_RECOGNIZE (   
[PARTITION BY ...]   
[ORDER BY ...]   
MEASURES ...
[ONE|ALL] ROW PER MATCH
AFTER MATCH SKIP TO ...
PATTERN ...
DEFINE ...
)

Output definition
Basic Match Recognize Syntax

```
SELECT ...
FROM ...
MATCH_RECOGNIZE ( 
  [PARTITION BY ...]
  [ORDER BY ...]
  MEASURES ...
  [ONE|ALL] ROW PER MATCH
  AFTER MATCH SKIP TO ...
  PATTERN ...
  DEFINE ...
)
```

Optional partitioning/ordering
- Matchig is performed separately on each partition
- Data is reordered accordingly
Basic Match Recognize Syntax

SELECT ...
FROM ... MATCH_RECOGNIZE (  
    [PARTITION BY ...]  
    [ORDER BY ...]  
    MEASURES ...
    [ONE|ALL] ROW PER MATCH  
    AFTER MATCH SKIP TO ...
    PATTERN ...
    DEFINE ...
)  

Output
• only one row per successful match or
• all rows that participated
Basic Match Recognize Syntax

SELECT ...
FROM ... MATCH_RECOGNIZE (
    [PARTITION BY ...]
    [ORDER BY ...]
    MEASURES ...
    [ONE|ALL] ROW PER MATCH
    AFTER MATCH SKIP TO ...
    PATTERN ...
    DEFINE ...
)

Where to continue processing after a match was found?
Match Recognize Details – Pattern Definition

- Patterns are **regular expressions** over pattern variables
  - Zero or one occurrence: $A?$
  - Kleene: $A^*/A+$
  - Quantifies: $A\{n,m\}, A\{n\}, A\{m\}$
  - Alternatives: $A|B$
  - Anchors: Before first event (^), after last event ($) of a partition

- Examples:
  - $A \; B^* \; C$  // A followed by zero or more Bs, followed by C
  - $A \; (B|C)^+ \; D$  // A followed by at least 1 B or C, followed by D
  - $A \; B\{3,\} \; C$  // A followed by at least 3 Bs, followed by C
  - $^A+$  // The entire partition must satisfy A
Match Recognize Details – Variable Definitions

• Variables
  • Building blocks for the pattern definition
  • Representation of user defined conditions

• Basic Syntax: `DEFINE <NAME> AS <CONDITION>`
  • Condition consists of Boolean expressions and the following PM specific constructs:
    • `PREV, NEXT`
      • refer to rows (events) that occur before/after the currently processed row
      • operate on physical rows, not restricted to pattern variables
    • `FIRST, LAST`
      • navigate rows that were mapped to a pattern variable
      • operate on logical rows, restricted to pattern variables
  • Aggregates
    • Access to `running` aggregates over a specific pattern variable
Variable Definitions with \texttt{PREV} and \texttt{NEXT}

- Access rows/events by physical relative offset
  - \texttt{A AS A.temp > PREV(A.temp)} // refer to previous row
  - \texttt{A AS A.temp > PREV(A.temp, 50)} // move back 50 events
  - \texttt{A AS PREV(A.temp \times A.accuracy) > 90} // refer to multiple columns
  - \texttt{A AS A.temp > 2 \times ( PREV (A.temp, 2) + \texttt{PREV} (A.temp, 1) + \texttt{NEXT} (A.temp, 1) + \texttt{NEXT} (A.temp, 2)) / 4} // find outlier (> than twice the average of its four neighbors)
  - \texttt{A AS A.temp > PREV(B.temp)} // refer to another variable:
    - // previous row of the last row mapped to B
Variable Definitions with FIRST and LAST

• Access rows with logical offset based on pattern variable

• DEFINE A AS A.TEMP > 90
  
  B AS B.TEMP < LAST(A.TEMP)  // Refer to the last occurrence of A

• DEFINE A AS A.TEMP > 90
  
  B AS B.TEMP < LAST(A.TEMP,20)  // with offset 20
Example

- **Event Stream**

<table>
<thead>
<tr>
<th>Record sequence</th>
<th>Value</th>
<th>Variable Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>R2</td>
<td>20</td>
<td>B</td>
</tr>
<tr>
<td>R3</td>
<td>30</td>
<td>A</td>
</tr>
<tr>
<td>R4</td>
<td>40</td>
<td>C</td>
</tr>
<tr>
<td>R5</td>
<td>50</td>
<td>A</td>
</tr>
</tbody>
</table>

- FIRST (A.Price) = FIRST (A.Price, 0) = LAST (A.Price, 2) = 10
- FIRST (A.Price, 1) = LAST (A.Price, 1) = 30
- FIRST (A.Price, 2) = LAST (A.Price, 0) = LAST (A.Price) = 50
- FIRST (A.Price, 3) is null, LAST (A.Price, 3) is null
Variable Definitions with Aggregates

• Define Variable based on running aggregates

• DEFINE A AS A.TEMP > AVG(A.TEMP) // self reference

• DEFINE A AS true

• B AS B.TEMP >= 2*AVG(A.TEMP) // refer to another // variable A
Match Recognize Details – Output Definition

• Output defined via MEASURES clause
  • access to defined pattern variables
  • Basic syntax: \(<\text{EXPRESSION}>\) AS \(<\text{NAME}>\)

• Expressions refer to specific values via
  • FIRST/LAST expressions
  • Aggregates over attributes of a pattern variable

• Expressions can be \textit{running} or \textit{final}
  • If the user specifies to output all rows contributing to a match
    • running means: Use the values seen so far
    • final means: Always use the values obtained after processing the last row
Match Recognize Details – Output Definition

MEASURES FIRST(A.TIME) AS START,
   B.TIME AS END,
   AVG(A.TEMP) as AVG_TMP
PATTERN A+ B
DEFINE A AS A.TEMP > PREV(A.TEMP)
   B AS B.TEMP > 90
Match Recognize Details – Output Definition

MEASURES FIRST(A.TIME) AS START,
    B.TIME AS END,
    AVG(A.TEMP) as AVG_TMP
PATTERN A+ (B|C)
DEFINE A AS A.TEMP > PREV(A.TEMP)
    B AS B.TEMP > 90
    C AS C.TEMP < FIRST(A.TEMP)

B.TIME is undefined, if the pattern ended with C
Match Recognize Details – Output Definition

MEASURES FIRST(A.TIME) AS START,
   COALESCE(B.TIME,C.TIME) AS END,
   AVG(A.TEMP) as AVG_TMP

PATTERN  A+ (B|C)

DEFINE   A AS A.TEMP > PREV(A.TEMP)
          B AS B.TEMP > 90
          C AS C.TEMP < FIRST(A.TEMP)

Use first non-null value instead
Match Recognize Details – Continue Search

• Position to start looking for the next match after a successful match:
  • AFTER MATCH SKIP TO NEXT ROW
    • Resume pattern matching at the row after the first row of the current match.

  • AFTER MATCH SKIP PAST LAST ROW (Default)
    • Resume pattern matching at the next row after the last row of the current match.

  • AFTER MATCH SKIP TO FIRST <pattern_variable>
    • Resume pattern matching at the first row that is mapped to the pattern variable.

  • AFTER MATCH SKIP TO LAST <pattern_variable>
    • Resume pattern matching at the last row that is mapped to the pattern variable.
Another Example

- Find flickering alarms
- SELECT * FROM ALARMS MATCH_RECOGNIZE(
  PARTITION BY ALARM_ID
  ORDER BY TIME
  AFTER MATCH SKIP TO FIRST B
  MEASURES FIRST(A.ALARM_ID) AS ID,
    FIRST(A.TIME) AS BEGIN,
    LAST(B.TIME) AS END,
  PATTERN (A+B+){50,}
  DEFINE A AS A.alarm = true,
    B AS B.alarm = false AND B.time - FIRST(A.TIME) <= 5 min
)
Summary – Match Recognize

• Pattern definitions via regular expression over pattern variables

• Complex variable- and result-definitions via:
  • Aggregates
  • Physical offsets (PREV,NEXT)
  • Logical offsets (FIRST,LAST)

• Control output via:
  • Resume options (AFTER MATCH …)
  • ALL ROWS/ONE ROW

• Not discussed so far: Window-Clause
  • Define a time window in which the pattern must occur completely
Literature


Complex Temporal Pattern Matching
Motivation

• Sequential Patterns are limited to before/after/at the same time relationships

• Many real-world applications require the definition of complex temporal relationships between situations
  • Running Example: Traffic monitoring

• Cumbersome/impossible to express with sequential patterns
  • Queries are hard to understand
  • Evaluation performance is poor
Example: Detect Aggressively Driving Cars

• Input: sensor data from connected cars
  • ID, position, speed, acceleration, etc.

• Indicators for aggressive driving according to AAA
  • “[…] suddenly changing speeds” and “Driving […] in excess of posted speed limit”

• Query: „A sharp acceleration followed by hard braking, both accompanied by a period of speeding”
Option 1: Raw Events + Pattern Matching

- Process raw event stream
  - Infinite sequence of timestamped relational tuples
- Detection of sequential patterns
  - Pattern expressed via regular expressions or equivalent
  - Poor evaluation performance
  - Awkward query definitions

```sql
FROM CARS PARTITION BY CAR_ID
PATTERN A+ B+ C+ D+ E+
WHERE A.SPEED <= 75 mph
AND A.ACCEL > 8 m/s²
AND B.SPEED > 75 mph
AND B.ACCEL > 8 m/s²
AND C.SPEED > 75 mph
AND C.ACCEL >= -9 m/s²
AND C.ACCEL <= 8 m/s²
AND D.SPEED > 75 mph
AND D.ACCEL < -9 m/s²
```
Option 2: Interval Based Systems

• Process streams of interval events
  • Infinite sequence of relational tuples associated with a time interval

• Pattern stated as interval endpoint order
  • $A.\text{start} < S.\text{start} < A.\text{end} < D.\text{start} < \ldots$

• Intervals created externally
  • Visible to the system after interval ended
    ➔ No early results
    ➔ No optimization opportunities

• Poorly supported by existing systems
Requirements for Temporal Pattern Matching

- Easily readable queries
- Compatible with event processing systems
  - Process raw event streams
- Early result detection (low latency)
- Efficient query processing

FROM CARS PARTITION BY CAR_ID
DEFINE A AS ACCEL > 8 m/s²,
    B AS SPEED > 75 mph,
    C AS ACCEL < -9 m/s²
PATTERN A OVERLAPS B AND
    B OVERLAPS C
TPStream*: Basic Ideas

• Input and Output are event streams

• Temporal patterns as binary constraints using Allen’s interval algebra
  • E.g. Acceleration \textit{overlaps} Speeding

• Tight coupling of derivation and matching
  • Early results

*M. Körber, N. Glombiewski, B. Seeger: TPStream: Low-Latency and High-Throughput Temporal Pattern Matching on Event Streams, will appear in Distributed and Parallel Databases
TPStream: Basic Ideas

- Input and Output are event streams
- Temporal patterns as binary constraints using Allen’s interval algebra
  - E.g. Acceleration *overlaps* Speeding
- Tight coupling of derivation and matching
  - Early results
Deriving Situations

- **Situation**: A period of time for which a set of conditions holds true
  - Summarization of a continuous subsequence of events

- Defined via
  - $\varphi$: Predicate, e.g. $speed > 75 \text{ mph}$
    - Identify continuous subsequences
  - $\gamma$: Set of aggregates, e.g. $avg(speed)$, $max(acceleration)$
    - Summarize events of subsequence
  - $\Delta$: Optional duration constraints, e.g. $5s < duration < 2\text{min}$
    - Restrict situations of interest

- Derived on the fly from incoming events
Deriving Situations - Example

Speeding:
\[ \varphi: speed > 75 \text{mph} \]
\[ \gamma: \text{avg}(speed) \]

Acceleration:
\[ \varphi: \text{accel} > 8 \text{m/s}^2 \]
\[ \gamma: \text{max}(\text{accel}) \]

Deceleration:
\[ \varphi: \text{accel} < -9 \text{m/s}^2 \]
\[ \gamma: \text{min}(\text{accel}) \]

Events:

Speed: 70 mph
Accel: 7 m/s²
...  
Timestamp: 1
Deriving Situations - Example

Speeding:
\( \varphi: speed > 75 \text{mph} \)
\( \gamma: \text{avg}(speed) \)

Acceleration:
\( \varphi: \text{accel} > 8 \text{m/s}^2 \)
\( \gamma: \text{max}(\text{accel}) \)

Deceleration:
\( \varphi: \text{accel} < -9 \text{m/s}^2 \)
\( \gamma: \text{min}(\text{accel}) \)

Events:

Speed: 72 mph
Accel: 9 m/s^2
Timestamp: 2

max(accel): 9 m/s^2
Validity: [2, *)
Deriving Situations - Example

Speeding:
\[ \varphi: speed > 75\text{mph} \]
\[ \gamma: \text{avg}(speed) \]

Acceleration:
\[ \varphi: \text{accel} > 8\text{m/s}^2 \]
\[ \gamma: \text{max}(\text{accel}) \]

Deceleration:
\[ \varphi: \text{accel} < -9\text{m/s}^2 \]
\[ \gamma: \text{min}(\text{accel}) \]

Events:

---

avg(speed): 76 mph
Validity: [3,*)

max(accel): 10 m/s²
Validity: [2,*)

Speed: 76 mph
Accel: 10 m/s²
Timestamp: 3
Deriving Situations - Example

Speeding:
\(\varphi: speed > 75 \text{mph}\)
\(\gamma: \text{avg}(speed)\)

Acceleration:
\(\varphi: \text{accel} > 8 \text{m/s}^2\)
\(\gamma: \text{max}(\text{accel})\)

Deceleration:
\(\varphi: \text{accel} < -9 \text{m/s}^2\)
\(\gamma: \text{min}(\text{accel})\)

Events:

avg(speed): 77 mph
Validity: [3,*)

max(accel): 10 m/s²
Validity: [2,4)

Speed: 78 mph
Accel: 5 m/s²
... Timestamp: 4
Deriving Situations - Example

Speeding:
\[ \varphi: \text{speed} > 75 \text{mph} \]
\[ \gamma: \text{avg(speed)} \]

Acceleration:
\[ \varphi: \text{accel} > 8 \text{m/s}^2 \]
\[ \gamma: \text{max(accel)} \]

Deceleration:
\[ \varphi: \text{accel} < -9 \text{m/s}^2 \]
\[ \gamma: \text{min(accel)} \]

Events:

Speed: 76 mph
Accel: -10 m/s²
Timestamp: 5

avg(speed): 76,6 mph
Validity: [3,*)

max(accel): 10 m/s²
Validity: [2,4)

min(accel): -10 m/s²
Validity: [5,*)

Timestamp: 5
Deriving Situations - Example

Speeding:
\[\phi : speed > 75 \text{mph} \]
\[\gamma : \text{avg}(speed) \]

Acceleration:
\[\phi : \text{accel} > 8 \text{m/s}^2 \]
\[\gamma : \text{max}(accel) \]

Deceleration:
\[\phi : \text{accel} < -9 \text{m/s}^2 \]
\[\gamma : \text{min}(accel) \]

Events:

\[
\begin{align*}
\text{avg(speed): 76.6 mph} \\
\text{Validity: [3,6)} \\
\text{max(accel): 10 m/s}^2 \\
\text{Validity: [2,4)} \\
\text{min(accel): -11 m/s}^2 \\
\text{Validity: [5,*)} \\
\text{Speed: 72 mph} \\
\text{Accel: -11 m/s}^2 \\
\text{Timestamp: 6}
\end{align*}
\]
Deriving Situations - Example

Speeding:
\[ \varphi: speed > 75 \text{mph} \]
\[ \gamma: \text{avg}(speed) \]

Acceleration:
\[ \varphi: \text{accel} > 8 \text{m/s}^2 \]
\[ \gamma: \text{max}(\text{accel}) \]

Deceleration:
\[ \varphi: \text{accel} < -9 \text{m/s}^2 \]
\[ \gamma: \text{min}(\text{accel}) \]

Events:

avg(speed): 76.6 mph
Validity: [3,6)

max(accel): 10 m/s²
Validity: [2,4)

min(accel): -11 m/s²
Validity: [5,7)

Speed: 68 mph
Accel: -5 m/s²
... Timestamp: 7
TPStream: Basic Ideas

- Input and Output are event streams
- Temporal patterns as binary constraints using Allen’s interval algebra
  - E.g. Acceleration overlaps Speeding
- Tight coupling of derivation and matching
  - Early results
Temporal Pattern

• Temporal Relation
  • Basic building block of a pattern
  • Based on Allen’s interval algebra
  • Example: Acceleration *overlaps* Speeding

• Temporal Constraint
  • Disjunction of temporal relations
  • Definition of alternatives
  • Example: Acceleration *overlaps OR meets* Speeding

• Temporal Pattern
  • Conjunction of temporal constraints
  • Additional time window
  • Example: (Acceleration *overlaps OR meets* Speeding) AND (Speeding *overlaps OR contains* Deceleration) AND (Acceleration *before* Deceleration)

<table>
<thead>
<tr>
<th>Relation</th>
<th>Equivalent</th>
<th>Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>A before B</td>
<td>B after A</td>
<td></td>
</tr>
<tr>
<td>A starts B</td>
<td>B started-by A</td>
<td></td>
</tr>
<tr>
<td>A meets B</td>
<td>B met-by A</td>
<td></td>
</tr>
<tr>
<td>A overlaps B</td>
<td>B overlapped-by A</td>
<td></td>
</tr>
<tr>
<td>A during B</td>
<td>B contains A</td>
<td></td>
</tr>
<tr>
<td>A finishes B</td>
<td>B finished-by A</td>
<td></td>
</tr>
<tr>
<td>A equals B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Low Latency Matching (1)

• Revisit definitions of Allen’s interval relations:

<table>
<thead>
<tr>
<th>Relation (R)</th>
<th>Definition</th>
<th>$t_d(R)$</th>
<th>Prefix-Group (G)</th>
<th>$t_d(G)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A overlaps B</td>
<td>$A.ts &lt; B.ts &lt; A.te &lt; B.te$</td>
<td>$A.te$</td>
<td>$A.ts &lt; B.ts$</td>
<td>$B.ts$</td>
</tr>
<tr>
<td>A contains B</td>
<td>$A.ts &lt; B.ts &lt; B.te &lt; A.te$</td>
<td>$B.te$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Revisit definitions of Allen’s interval relations:

<table>
<thead>
<tr>
<th>Relation (R)</th>
<th>Definition</th>
<th>$t_d(R)$</th>
<th>Prefix-Group (G)</th>
<th>$t_d(G)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A overlaps B</td>
<td>$A.ts &lt; B.ts &lt; A.te &lt; B.te$</td>
<td>A.te</td>
<td>$A.ts &lt; B.ts$</td>
<td>B.ts</td>
</tr>
<tr>
<td>A finishes B</td>
<td>$A.ts &lt; B.ts &lt; A.te = B.te$</td>
<td>A.te = B.te</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A contains B</td>
<td>$A.ts &lt; B.ts &lt; B.te &lt; A.te$</td>
<td>B.te</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each temporal relation can be detected/validated at $t_3 \left( t_d(R) \right)$

- Example: $t_d(A \text{ overlaps } B) = A.\text{te}$
- We know $A.ts < B.ts$ and B will definitely end at a later point in time
Low Latency Matching (1)

• Revisit definitions of Allen’s interval relations:

<table>
<thead>
<tr>
<th>Relation (R)</th>
<th>Definition</th>
<th>( t_d(R) )</th>
<th>Prefix-Group (G)</th>
<th>( t_d(G) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A overlaps B</td>
<td>( A.ts &lt; B.ts &lt; A.te &lt; B.te )</td>
<td>A.te</td>
<td>A.ts &lt; B.ts</td>
<td>B.ts</td>
</tr>
<tr>
<td>A finishes B</td>
<td>( A.ts &lt; B.ts &lt; A.te = B.te )</td>
<td>A.te = B.te</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A contains B</td>
<td>( A.ts &lt; B.ts &lt; B.te &lt; A.te )</td>
<td>B.te</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Each temporal relation can be detected/validated at \( t_3 \left( t_d(R) \right) \)
  • Example: \( t_d\left( A \text{ overlaps } B \right) = A.te \)
  • We know \( A.ts < B.ts \) and \( B \) will definitely end at a later point in time

• Prefix groups allow detection/validation at \( t_2 \left( t_d(G) \right) \)
  • Example: \( t_d\left( A \text{ overlaps OR finishes OR contains } B \right) = B.ts \)
  • Ordering of \( A.te \) and \( B.te \) does not matter
Experimental Results

- Query with 3 situations
  - A before B AND B overlaps C
- Synthetic data (300M events)
- Varying window size

- Detection latency per temporal relation
- 2 situation streams
- 55s average duration
- Application time (event timestamps)
Group Patterns
Group Patterns I

• Detect groups of objects with similar behavior over time
  • Similarity often refers to the spatial position of (moving) objects

• Two kinds of basic patterns
  • **Concurrency**: Find all birds currently flying in south direction
  • **Mutual relationship**: Find a group of gazelles heading towards a specific location

• Complex patterns are composed of temporal relationships instances of simple patterns

• Assumptions
  • Every event has at least the following attributes
    • unique object ID
    • attribute to be analyzed (e.g., position)
    • timestamp
Basic Group Patterns

• Generate snapshots of the objects at given points in time

• Concurrency
  • Evaluate the predicate for every object on each snapshot
  • All objects satisfying the predicate form a group
  • Time intervals describe the lifetime of a group

• Mutual relationship
  • Evaluate predicates for all pairs of objects on each snapshot (e.g. distance between)
    • Generation of a graph with an edge if two objects fulfill the predicate
  • Connected components inside the graph form groups
  • Time intervals describe the lifetime of a group
Complex Group Patterns with TPStream

- Complex patterns
  - Combination of simple patterns using TPStream

- Challenges
  - Keep track of group development (leaves, joins, splits, merges)
  - Efficient computation

- Example: Find a group of gazelles heading towards a specific location
  - Predicate: First derivative of the distance < 0
    - Distance shrinks
  - Find a connected component inside the graph
    - Distance between all members of the subgraph shrinks
Example

- Find planes that are moving close to each other over a certain distance
  - What is the meaning of proximity?
  - What is the minimum distance?
Literature


Discussion