SEQUENCED, SPATIO-TEMPORAL AGGREGATION IN ROAD NETWORKS

Igor Timko, Michael H. Böhlen, and Johann Gamper

Free University of Bozen-Bolzano, Italy

12th International Conference on Extending Database Technology (EDBT 2009)
Saint Petersburg, Russia
March 24–26, 2009
Talk Outline

- Motivation
- Definition of Sequenced, Spatio-Temporal (SST) Aggregation
- Data Structures and Algorithms for SST Aggregation
- Experiments
- Related Work, Conclusions, and Future Work
Time and Space Model

- **Our time model:**
  - The time line is a finite sequence of temporal granules (discrete time model)

- **Our space model** is similar to the time model:
  - The space is a collection of spacelines (1.5-dimensional model)
  - A spaceline is a finite sequence of spatial granules (discrete model):
    
    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

- **Spatio-temporal granule:**
  - A pair (temporal granule, spatial granule)
Motivation

- Tracking of vehicles:
  - Cars report their positions to a central database server
- Road traffic analysis:
  - “For each road, what is the number of cars at each point in time (between 5:00PM and 5:15PM) and space (position on the road network)?”

(a) 5:00PM – 5:10PM  
(b) 5:11PM – 5:15PM
Commercial tracking systems use simple **location update policies**:
- Each car updates its location (reports its position) *periodically*

**Reported car positions in GPS logs:**

```
Time=1  Time=4  Time=7
1 2 3 4 5 6 7 8 9 10 11 12
```

Road ID = 1101

**Spatio-temporal uncertainty:**
- GPS logs do not give any guarantees on where a car is between its two position reports

**Spatial interpolation:**
- During the time period between two reports, the car is *anywhere* between two reported positions

```
Time=[1;4)  Time=[4;7)
1 2 3 4 5 6 7 8 9 10 11 12
```

Road ID = 1101
Car Positions: Relational Perspective

(a) Original GPS logs, in a *conventional* relation

(b) GPS logs after *spatial interpolation*, in a *spatio-temporal* relation
Talk Outline

- Motivation
- **Definition of Sequenced, Spatio-Temporal (SST) Aggregation**
- Data Structures and Algorithms for SST Aggregation
- Experiments
- Related Work, Conclusions, and Future Work
Spatio-Temporal Relations

- Conceptually, a **spatio-temporal relation** is a set of **snapshot relations**, one per spatio-temporal granule.
- **Spatio-temporal validity rectangle**:  
  - A pair (**temporal validity interval**, **spatial validity interval**)
- A temporal validity interval has a **start time point** and a **finish time point**.
- A spatial validity interval has a **begin space point** and an **end space point**.
Spatio-Temporal Relations

- Conceptually, a spatio-temporal relation is a set of snapshot relations, one per spatio-temporal granule.
- Spatio-temporal validity rectangle:
  - A pair (temporal validity interval, spatial validity interval)
  - A temporal validity interval has a start time point and a finish time point.
  - A spatial validity interval has a begin space point and an end space point.

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1101</td>
<td>[1;4)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
<td>[4;7)</td>
<td>[6;11)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[3;6)</td>
<td>[3;8)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[6;9)</td>
<td>[7;11)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[3;6)</td>
<td>[6;9)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[6;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>
Conceptual definition:
- Apply the given aggregation function independently for each snapshot relation

Example SST aggregation query:
- “How many cars, for each point in time and space?”
Sequenced, Spatio-Temporal (SST) Aggregation

- The SST aggregation result:
  - Conceptually, a set of aggregate values (one value per spatio-temporal granule)
  - Practically, one set of **constant rectangles** per spaceline plus the aggregate value for each rectangle

- Graphical representation:
Sequenced, Spatio-Temporal (SST) Aggregation

► The SST aggregation result:
  ► Conceptually, a set of aggregate values (one value per spatio-temporal granule)
  ► Practically, one set of constant rectangles per spaceline plus the aggregate value for each rectangle

► Graphical representation:
Sequenced, Spatio-Temporal (SST) Aggregation

- The SST aggregation result:
  - Conceptually, a set of aggregate values (one value per spatio-temporal granule)
  - Practically, one set of **constant rectangles** per spaceline plus the aggregate value for each rectangle
- Graphical representation:
Sequenced, Spatio-Temporal (SST) Aggregation

- The SST aggregation result:
  - Conceptually, a set of aggregate values (one value per spatio-temporal granule)
  - Practically, one set of constant rectangles per spaceline plus the aggregate value for each rectangle

- Relational representation:
Idea of SST aggregation evaluation

- Ignore the spatial part and compute temporal components of constant rectangles
- “Refine” the temporal components with the spatial part
Idea of SST aggregation evaluation

- Ignore the spatial part and compute temporal components of constant rectangles
- “Refine” the temporal components with the spatial part
Idea of SST aggregation evaluation

- Ignore the spatial part and compute temporal components of constant rectangles
- “Refine” the temporal components with the spatial part
Idea of SST aggregation evaluation

- Ignore the spatial part and compute temporal components of constant rectangles
- “Refine” the temporal components with the spatial part
Talk Outline

- Motivation
- Definition of Sequenced, Spatio-Temporal (SST) Aggregation
- Data Structures and Algorithms for SST Aggregation
- Experiments
- Related Work, Conclusions, and Future Work
Computing Temporal Components: Idea

- Sort the start and finish time points from the input relation
- Traverse the sorted list of time points
Computing Temporal Components: Idea

- Sort the start and finish time points from the input relation
- Traverse the sorted list of time points

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
<th>$T$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1101</td>
<td>[1;4)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
<td>[4;7)</td>
<td>[6;11)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[3;6)</td>
<td>[3;8)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[6;9)</td>
<td>[7;11)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[3;6)</td>
<td>[6;9)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[6;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>

1 3 4 6 7 9
Computing Temporal Components: Idea

- Sort the start and finish time points from the input relation
- Traverse the sorted list of time points

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1101</td>
<td>[1;4)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
<td>[4;7)</td>
<td>[6;11)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[3;6)</td>
<td>[3;8)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[6;9)</td>
<td>[7;11)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[3;6)</td>
<td>[6;9)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[6;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cnt</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1;3)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[1;3)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[3;6)</td>
</tr>
<tr>
<td>3</td>
<td>[3;4)</td>
<td>[6;7)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[7;8)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[8;9)</td>
</tr>
<tr>
<td>1</td>
<td>[4;6)</td>
<td>[3;6)</td>
</tr>
<tr>
<td>3</td>
<td>[4;6)</td>
<td>[6;8)</td>
</tr>
<tr>
<td>2</td>
<td>[4;6)</td>
<td>[8;9)</td>
</tr>
<tr>
<td>1</td>
<td>[4;6)</td>
<td>[9;11)</td>
</tr>
<tr>
<td>1</td>
<td>[6;7)</td>
<td>[6;7)</td>
</tr>
<tr>
<td>2</td>
<td>[6;7)</td>
<td>[7;8)</td>
</tr>
<tr>
<td>3</td>
<td>[6;7)</td>
<td>[8;11)</td>
</tr>
<tr>
<td>1</td>
<td>[7;9)</td>
<td>[7;8)</td>
</tr>
<tr>
<td>2</td>
<td>[7;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>
Computing Temporal Components: Implementation

- Load the start and end time points into a \textit{balanced, binary search tree}
- Traverse the loaded tree in-order
Computing Temporal Components: Implementation

▶ Load the start and end time points into a balanced, binary search tree

▶ Traverse the loaded tree in-order

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1101</td>
<td>[1;4)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
<td>[4;7)</td>
<td>[6;11)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[3;6)</td>
<td>[3;8)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[6;9)</td>
<td>[7;11)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[3;6)</td>
<td>[6;9)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[6;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>
Computing Temporal Components: Implementation

- Load the start and end time points into a *balanced, binary search tree*
- Traverse the loaded tree in-order

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1101</td>
<td>[1;4)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
<td>[4;7)</td>
<td>[6;11)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[3;6)</td>
<td>[3;8)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[6;9)</td>
<td>[7;11)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[3;6)</td>
<td>[6;9)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[6;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>
Computing Spatial Components

- **Idea:**
  - Use the balanced, binary search tree to also handle the additional, **spatial** dimension
  - A node key is a start or finish time point

- **Problem:**
  - How to store spatial information?

```
        4
       /|
      1 7
     /  |
    3 6 9
```
Naively Extended Balanced Tree

- Idea:
  - At each node of the balanced binary tree, store space intervals of the tuples that are valid at the node’s time point
  - While traversing the tree, intersect the space intervals of each node
Naively Extended Balanced Tree

- Idea:
  - At each node of the balanced binary tree, store space intervals of the tuples that are valid at the node’s time point
  - While traversing the tree, intersect the space intervals of each node

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1101</td>
<td>[1;4)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
<td>[4;7)</td>
<td>[6;11)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[3;6)</td>
<td>[3;8)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[6;9)</td>
<td>[7;11)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[3;6)</td>
<td>[6;9)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[6;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>

Idea representation:

```
4  {{[3; 8), [6; 9), [6; 11)}}
```

```
1  {{[1; 7)}}
```

```
3  {{[1; 7), [3; 8), [6; 9)}}
```

```
6  {{[6; 11), [7; 11), [8; 11)}}
```

```
7  {{[7; 11), [8; 11)}}
```

```
9
```
Naively Extended Balanced Tree

- **Idea:**
  - At each node of the balanced binary tree, store space intervals of the tuples that are valid at the node’s time point
  - While traversing the tree, intersect the space intervals of each node

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
<th>$T$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1101</td>
<td>[1;4)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
<td>[4;7)</td>
<td>[6;11)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[3;6)</td>
<td>[3;8)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[6;9)</td>
<td>[7;11)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[3;6)</td>
<td>[6;9)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[6;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cnt</th>
<th>$T$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1;3)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[1;3)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[3;6)</td>
</tr>
<tr>
<td>3</td>
<td>[3;4)</td>
<td>[6;7)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[7;8)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[8;9)</td>
</tr>
<tr>
<td>1</td>
<td>[4;6)</td>
<td>[3;6)</td>
</tr>
<tr>
<td>3</td>
<td>[4;6)</td>
<td>[6;8)</td>
</tr>
<tr>
<td>2</td>
<td>[4;6)</td>
<td>[8;9)</td>
</tr>
<tr>
<td>1</td>
<td>[4;6)</td>
<td>[9;11)</td>
</tr>
<tr>
<td>1</td>
<td>[6;7)</td>
<td>[6;7)</td>
</tr>
<tr>
<td>2</td>
<td>[6;7)</td>
<td>[7;8)</td>
</tr>
<tr>
<td>3</td>
<td>[6;7)</td>
<td>[8;11)</td>
</tr>
<tr>
<td>1</td>
<td>[7;9)</td>
<td>[7;8)</td>
</tr>
<tr>
<td>2</td>
<td>[7;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>
Naively Extended Balanced Tree: Optimizations

- **Optimization 1:**
  - The set of space intervals of a node is a modification of the set of space intervals of the previous node
  - At each node, store only space intervals that enter and exit from the set (incremental approach)

- **Before Optimization 1:**

- **After Optimization 1:**
Naively Extended Balanced Tree: Optimizations

- **Optimization 1:**
  - The set of space intervals of a node is a modification of the set of space intervals of the previous node
  - At each node, store only space intervals that enter and exit from the set (*incremental approach*)

- Before Optimization 1:
  
  - 4: \{[3; 8), [6; 9), [6; 11]\}
  - 1: \{[1; 7]\}
  - 7: \{[7; 11), [8; 11]\}
  - 3: \{[1; 7), [3; 8), [6; 9]\}
  - 6: \{[6; 11), [7; 11], [8; 11]\}
  - 9

- After Optimization 1:
Naively Extended Balanced Tree: Optimizations

- **Optimization 1:**
  - The set of space intervals of a node is a modification of the set of space intervals of the previous node
  - At each node, store only space intervals that enter and exit from the set (*incremental approach*)

- **Before Optimization 1:**
  - 4 \{[3; 8), [6; 9), [6; 11)]
  - 1 \{[1; 7)]
  - 3 \{[1; 7), [3; 8), [6; 9)}
  - 7 \{[7; 11), [8; 11)}
  - 6 \{[6; 11), [7; 11], [8; 11]}  

- **After Optimization 1:**
  - 4 \{[6; 11)}/{[1; 7)}
  - 1 \{[1; 7)}/{
  - 3 \{[3; 8), [6; 9)}/{
  - 6 \{[7; 11), [8; 11]}/ {[3; 8), [6; 9)}
  - 7 \{}/[6; 11)}
  - 9 \{}/ {[7, 11), [8; 11)}}
Naively Extended Balanced Tree: Optimizations

- **Optimization 2:**
  - Instead of space intervals, store two counters for each space point:
    - **begin counter:** how many intervals *begin* at this space point?
    - **end counter:** how many intervals *end* at this space point?

- **Before Optimization 2:**

- **After Optimization 2:**
Optimization 2:

- Instead of space intervals, store two counters for each space point:
  - **begin counter**: how many intervals *begin* at this space point?
  - **end counter**: how many intervals *end* at this space point?

Before Optimization 2:

\[
\begin{array}{|c|c|c|}
\hline
1 & [1; 7), [7; 10]] & {[2; 7), [1; 7]} \\
\hline
\end{array}
\]

After Optimization 2:
Naively Extended Balanced Tree: Optimizations

- **Optimization 2:**
  - Instead of space intervals, store two counters for each space point:
    - **begin counter:** how many intervals *begin* at this space point?
    - **end counter:** how many intervals *end* at this space point?

- **Before Optimization 2:**
  
  \[
  1 \{[[1; 7), [7; 10)]/[[2; 7), [1; 7)]\}
  \]

- **After Optimization 2:**
  
  \[
  1 \{(1, 1 – 1/0 – 0), (7, 1 – 0/1 – 2), (2, 0/ – 1), (10, 0/1)}\]
Sequenced, Spatio-Temporal Tree (SST-tree)

The SST-tree is the Naively Extended Balanced Tree after Optimizations 1 and 2

```
4 {(1, −1/0), (6, 1/0), (7, 0/−1), (11, 0/1)}

1 {(1, 1/0), (7, 0/1)}

3 {(3, 1/0), (6, 1/0), (8, 0/1), (9, 0/1)}

6 {...}

7 {...}

9 {(7, −1/0), (8, −1/0), (11, 0/−2)}
```

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1101</td>
<td>[1;4)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
<td>[4;7)</td>
<td>[6;11)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[3;6)</td>
<td>[3;8)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[6;9)</td>
<td>[7;11)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[3;6)</td>
<td>[6;9)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[6;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>
SST-tree Implementation

- How to efficiently implement the set of space points/counters?
- Two variants:

<table>
<thead>
<tr>
<th>Variant</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hashmap</strong></td>
<td>smaller size, faster (constant) insert/lookup</td>
<td>resize overhead</td>
</tr>
<tr>
<td><strong>Balanced Tree</strong></td>
<td>no resize overhead</td>
<td>larger size, slower (log) insert/lookup</td>
</tr>
</tbody>
</table>

![Diagram](image_url)
SST-tree Algorithm: Top Level

**Input**: spatio-temporal relation \( R = (A_1, A_2, \ldots, A_n, T, S) \)

**Output**: spatio-temporal relation \( Z = (Cnt, T, S) \)

1. \( Z \leftarrow \emptyset; \)
2. foreach road-ID rid \( \in \pi[RID]R \) do
   3. \( \mathcal{T} \leftarrow \text{LOADTREE}(\sigma[RID = rid]R); \)
   4. \( Z_{rid} \leftarrow \{rid\} \times \text{COMPUTECONSTRECT}(\mathcal{T}); \)
   5. \( Z \leftarrow Z \cup Z_{rid}; \)
3. return \( Z; \)
SST-tree Algorithm: LOADTREE Example

- Input relation:

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1101</td>
<td>[1;4)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
<td>[4;7)</td>
<td>[6;11)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[3;6)</td>
<td>[3;8)</td>
</tr>
<tr>
<td>2</td>
<td>1101</td>
<td>[6;9)</td>
<td>[7;11)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[3;6)</td>
<td>[6;9)</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
<td>[6;9)</td>
<td>[8;11)</td>
</tr>
</tbody>
</table>

- The SST-tree after inserting the first tuple:
The Dynamic Tree:
- is used to compute aggregate values of spatio-temporal rectangles
- is a Balanced Tree (+ deletion algorithm)
The SST-tree:

The Dynamic Tree and the result relation after processing 1 node:
SST-tree Algorithm: COMPUTECONSTRECT Example

- The SST-tree:

```
4  {(1, −1/0), (6, 1/0), (7, 0/−1), (11, 0/1)}
  1  {(1, 1/0), (7, 0/1)}
   3  {(3, 1/0), (6, 1/0), (8, 0/1), (9, 0/1)}
      6  {...}
        9  {...}
```

- The Dynamic Tree and the result relation after processing 2 nodes

```
6  0+1/0+0
  3  0+1/0+0
    1  0+1/0+0
      7  0+0/0+1
        8  0+0/0+1
          9  0+0/0+1
```

<table>
<thead>
<tr>
<th>Cnt</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1;3)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[1;3)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[3;6)</td>
</tr>
<tr>
<td>3</td>
<td>[3;4)</td>
<td>[6;7)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[7;8)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[8;9)</td>
</tr>
</tbody>
</table>
SST-tree Algorithm: \texttt{COMPUTECNSTRICT} Example

- The SST-tree:

\[
\begin{array}{c}
4 \{ (1, -1/0), (6, 1/0), (7, 0/−1), (11, 0/1) \} \\
1 \{ (1, 1/0), (7, 0/1) \} \\
3 \{ (3, 1/0), (6, 1/0), (8, 0/1), (9, 0/1) \} \\
6 \{ \ldots \} \\
9 \{ \ldots \} \\
\end{array}
\]

- The Dynamic Tree and the result relation after processing 3 nodes

<table>
<thead>
<tr>
<th>Cnt</th>
<th>( T )</th>
<th>( S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1;3)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[1;3)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[3;6)</td>
</tr>
<tr>
<td>3</td>
<td>[3;4)</td>
<td>[6;7)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[7;8)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[8;9)</td>
</tr>
<tr>
<td>1</td>
<td>[4;6)</td>
<td>[3;6)</td>
</tr>
<tr>
<td>3</td>
<td>[4;6)</td>
<td>[6;8)</td>
</tr>
<tr>
<td>2</td>
<td>[4;6)</td>
<td>[8;9)</td>
</tr>
<tr>
<td>1</td>
<td>[4;6)</td>
<td>[9;11)</td>
</tr>
</tbody>
</table>
The SST-tree:

\[
\begin{align*}
4 & \{ (1, -1/0), (6, 1/0), (7, 0/1), (11, 0/1) \} \\
1 & \{ (1, 1/0), (7, 0/1) \} \\
3 & \{ (3, 1/0), (6, 1/0), (8, 0/1), (9, 0/1) \} \\
7 & \{ \ldots \} \\
6 & \{ \ldots \} \\
9 & \{ \ldots \}
\end{align*}
\]

The Dynamic Tree and the result relation after processing 3 nodes:

<table>
<thead>
<tr>
<th>Cnt</th>
<th>T</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1;3)</td>
<td>[1;7)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[1;3)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[3;6)</td>
</tr>
<tr>
<td>3</td>
<td>[3;4)</td>
<td>[6;7)</td>
</tr>
<tr>
<td>2</td>
<td>[3;4)</td>
<td>[7;8)</td>
</tr>
<tr>
<td>1</td>
<td>[3;4)</td>
<td>[8;9)</td>
</tr>
<tr>
<td>1</td>
<td>[4;6)</td>
<td>[3;6)</td>
</tr>
<tr>
<td>3</td>
<td>[4;6)</td>
<td>[6;8)</td>
</tr>
<tr>
<td>2</td>
<td>[4;6)</td>
<td>[8;9)</td>
</tr>
<tr>
<td>1</td>
<td>[4;6)</td>
<td>[9;11)</td>
</tr>
</tbody>
</table>
Talk Outline

- Motivation
- Definition of Sequenced, Spatio-Temporal (SST) Aggregation
- Data Structures and Algorithms for SST Aggregation
- Experiments
- Related Work, Conclusions, and Future Work
Experiments: Settings (1)

- Generated 10 spatio-temporal relations:
  - Used Brinkhoff’s generator of moving objects to generate GPS logs
  - Generated 10 GPS logs (from 3,000 moving objects to 30,000, step 3,000, in approx. 7K roads)
  - Converted each GPS log to a spatio-temporal relation by applying spatio-temporal interpolation (from 62K to 732K tuples)
Experiments: Settings (2)

- Compared 3 methods for SST aggregation:
  - Brute Force:
    - one Balanced Tree per *each* temporal granule
  - Balanced Tree*-based SST-tree
  - Hashmap-based SST-tree

(a) SST-tree

(b) Brute Force
Experiments: Conclusions

- The Brute Force is very inefficient ⇒ should not be used
- Other methods offer a trade-off between the speed and RAM usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balanced Tree*</td>
<td>faster</td>
<td>less RAM efficient</td>
</tr>
<tr>
<td>Hashmap</td>
<td>more RAM efficient</td>
<td>slower</td>
</tr>
</tbody>
</table>

*Note: Hashmap typically outperforms all other methods in terms of speed and RAM usage, making it the preferred choice.
Talk Outline

- Motivation
- Definition of Sequenced, Spatio-Temporal (SST) Aggregation
- Data Structures and Algorithms for SST Aggregation
- Experiments
- Related Work, Conclusions, and Future Work
Related Work

- **Spatial aggregation:**
  - Range (box) aggregation: aR-tree [1], aP-tree [2]
  - Sequenced, spatial (1D) aggregation: none

- **Temporal aggregation:**
  - Range (box) aggregation: MVSB-tree [3]
  - Sequenced, temporal (1D) aggregation: Aggregation Tree [4], Balanced Tree [5], TMDA operator [6], SB-tree [7]

- **Spatio-temporal aggregation:**
  - Range (box) aggregation: aRB-tree [8], sketch index [9], AMH [10]
  - Sequenced, spatial aggregation: none

Bibliography:


Conclusions

- We propose an efficient method for evaluation of COUNT/SUM sequenced, spatial-temporal aggregation queries
- Specifically, we propose:
  - the SST-tree that efficiently stores info about the input relation
  - algorithms for efficiently computing the aggregation result (including constant rectangles) from the SST-tree
  - two variants of the SST-tree implementation
- The SST-tree is more RAM-efficient and time-efficient than a brute force approach
Future Work

- More efficient data structures/algorithms (e.g., plane-sweep based)
- More experiments with the current data structures/algorithms (e.g., larger data sets)
- Other aggregation functions (e.g., MIN/MAX)
- Approximate aggregation (coalescing)
- Cost model for the order of dimensions
Questions?
Experiments: Load Time

![Load Time Graph]

- **Dense SST-H-tree**
- **Sparse SST-H-tree**
- **SST-T-tree**
- **Brute force**

Load time (secs) vs. No of tuples (thousands)

I. Timko et al.

SEQUENCED, SPATIO-TEMPORAL AGGREGATION IN ROAD NETWORKS
Experiments: Traversal Time

![Traversal Time Graph]

- **Dense SST-H-tree**
- **Sparse SST-H-tree**
- **SST-T-tree**
- **Brute Force**

**No of tuples (thousands):**
- 62
- 131
- 205
- 280
- 352
- 430
- 505
- 580
- 656
- 732

**Traverse time (secs):**
- 0
- 5
- 10
- 15
- 20
- 25

I. Timko et al.
SEQUENCED, SPATIO-TEMPORAL AGGREGATION IN ROAD NETWORKS
Experiments: Total (Load + Traversal) Time

![Graph showing total load + traversal time for different methods and tuple counts. The y-axis represents total time (seconds) and the x-axis represents the number of tuples (thousands). The methods compared are Dense SST-H-tree, Sparse SST-H-tree, SST-T-tree, and Brute force. The graph indicates that the Brute force method has the highest total time, followed by the Dense SST-H-tree, while the Sparse SST-H-tree and SST-T-tree have significantly lower times.]
Experiments: RAM

I. Timko et al.