SEQUENCED, SPATIO-TEMPORAL AGGREGATION IN ROAD NETWORKS

Temporal, Valid-Time Relations

- A **temporal, valid-time** relation captures history of changes in the modeled reality
- Each tuple has a **valid timestamp (validity interval)** with a **start time point** and **finish time point**
- Temporal, valid-time relation `EMPLOYEE`:

Name	Salary	Dept	Start	Finish
Richard	46000	Accounting	18	31
Karen	45000	Shipping	8	20
Nathan	35000	Marketing	7	12
Nathan	38000	Accounting	18	21

Snapshot Aggregation

- **Snapshot aggregation** transforms an argument relation into a **summary result** relation
- Two-step process:
  1. Partition the argument relation into groups of tuples with identical values for one or more attributes (**aggregation groups**)
  2. Apply one or more aggregate functions (e.g., COUNT, SUM) to each of these groups in turn
- **Example**
  - Compute the average salary of all the employees:

```
SELECT AVG(SALARY)
FROM EMPLOYEE
```

  - Compute the average salary of employees grouped by department:

```
SELECT DEPT, AVG(SALARY)
FROM EMPLOYEE
GROUP BY DEPT
```

Temporal Aggregation

- Additionally use time dimension to group argument tuples
- **Temporal grouping**: Partition the timeline and group argument tuples over these time partitions
  - Then, compute the aggregate values over each of these groups
- **Instant grouping**: The timeline is partitioned into instants (*temporal granules/chronons*)
- **Sequenced, temporal aggregation**:
  - Do instant grouping
  - Report results over **constant intervals** (sequences of instants with the same argument tuples, i.e., the same aggregate values)

Sequenced, Temporal Aggregation: Example

- Query:
  - "For each month and department, what is the number of contracts?"

r1 = (Jan, 140, 1200, DB, [1;13))

r2 = (Ann, 141, 700, DB, [1;6))

r3 = (Ann, 150, 700, DB, [6;16))

r4 = (John, 143, 2000, AI, [4;10))

1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 **Time**

(DB, 2, [1;6))

(DB, 2, [6;13))

(DB, 1, [13;16))

(AI, 1, [4;10))

	Name	ContractID	Salary	Dept	T
r1	Jan	140	1200	DB	[1;13)
r2	Ann	141	700	DB	[1;6)
r3	Ann	150	700	DB	[6;16)
r4	John	143	2000	AI	[4;10)

Dept	Cnt	T
DB	2	[1;6)
DB	2	[6;13)
DB	1	[13;16)
AI	1	[4;10)

The Idea of Computing Sequenced, Temporal Aggregates

- **Time point sorting** for the COUNT aggregation:
  1. Load the entire tuples in main memory
  2. Extract time points from the tuples. Add to each time point a tag that indicates whether the time point is a star (S) or end (E) time point
  3. Sort the time points (together with the tags) in increasing order
  4. Scan the sorted time points, getting started with a `counter` initialized to 0.
     - At each time point, increment (decrement) the `counter` by the number of start (end) tags at that time

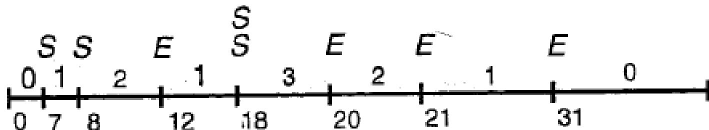
The Idea of Computing Sequenced, Temporal Aggregates

- **Example:**
  - “Compute the time-varying number of tuples for the following `Employee` relation”

Name	Salary	Dept	T
Richard	46000	Accounting	[18;31)
Karen	45000	Shipping	[8;20)
Nathan	35000	Marketing	[7;12)
Nathan	38000	Accounting	[18;21)

Count	T
1	[7;8)
2	[8;12)
1	[12;18)
3	[18;20)
2	[20;21)
1	[21;31)

- Sort the time points to compute COUNT aggregate function



Balanced Tree Algorithm

- The **Balanced Tree algorithm**:
  - sorts the time points *incrementally* in a *balanced*, binary search tree
  - computes *incrementally* the number of tags while constructing the tree

Balanced Tree Algorithm: Top Level

Input: temporal relation $R = (A_1, A_2, \ldots, A_n, T_s, T_f)$
Output: temporal relation $Z = (Cnt, T_s, T_f)$

1 $\mathcal{T} \leftarrow$ LOADTREE(R);
2 $Z \leftarrow$ COMPUTECONSTINT($\mathcal{T}$);
3 **return** Z;

Balanced Tree Algorithm: LOADTREE

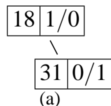
Input: temporal relation $R = (A_1, A_2, \ldots, A_n, T_s, T_f)$
Output: Balanced Tree $\mathcal{T}$

1 $\mathcal{T} \leftarrow$ empty Balanced Tree;
2 **foreach** $r \in R$ **do**
3 $\quad tn \leftarrow$ GETNODE$(\mathcal{T}, r.T_s)$;
4 $\quad tn.cnt_s + +$;
5 $\quad tn \leftarrow$ GETNODE$(\mathcal{T}, r.T_f)$;
6 $\quad tn.cnt_f + +$;
7 **return** $\mathcal{T}$;

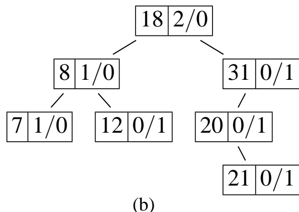
Balanced Tree Algorithm: LOADTREE *Example*

Name	Salary	Dept	Start	Finish
Richard	46000	Accounting	18	31
Karen	45000	Shipping	8	20
Nathan	35000	Marketing	7	12
Nathan	38000	Accounting	18	21

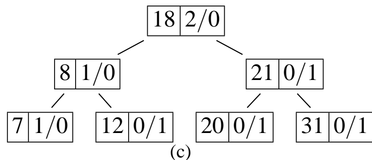
- The Balanced Tree at different stages:
  - *(a)* After having inserted the timestamps of the first tuple $(Richard, 46000, Accounting, 18, 31)$
  - *(b)* After having inserted all the tuples, but before balancing
  - *(c)* After balancing



(a)



(b)



(c)

Balanced Tree Algorithm: COMPUTECONSTRECT

Input: Balanced Tree $\mathcal{T}$

Output: temporal relation $Z = (Cnt, T_s, T_f)$

1 $Z \leftarrow \emptyset$;

2 $cnt \leftarrow 0$;

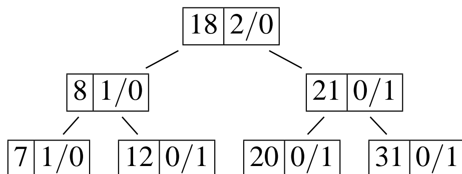
3 **foreach** *pair of consecutive nodes* $(tn, tn') \in \mathcal{T}$ *in in-order* **do**

4 $\quad cnt \leftarrow cnt + tn.cnt_s - tn.cnt_f$;

5 $\quad Z \leftarrow Z \cup \{(cnt, tn.T, tn'.T)\}$;

6 **return** Z;

Balanced Tree Algorithm: COMPUTECONSTRECT *Example*

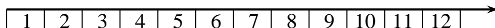


Count	**Start**	**Finish**
1	7	8
2	8	12
1	12	18
3	18	20
2	20	21
1	21	31

- In-order traversal of the tree to compute the aggregation result
  - E.g., $1 + 1 = 2$ for the time interval $[8; 12)$

Space Model

- The **space model** is the 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*):

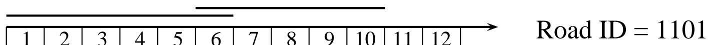


Car Positions

- Commercial positioning systems use simple **location update policies**:
  - Each car updates its location (reports its position) *periodically*
- Reported car positions in GPS logs:

1 2 3 4 5 6 7 8 9 10 11 12 Road ID = 1101

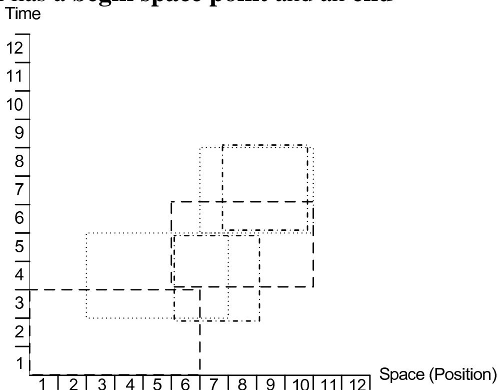
- **Spatio-temporal uncertainty**: GPS logs do not say how the speed is changing between two reported positions.
- **Spatial interpolation**: During the time period between two reports, the car is *anywhere* between two reported positions



Spatio-Temporal Relations

- Relation of the Spatio-Temporal Data Model (**STDM relation**) extends many notions of temporal, valid-time relations:
  - **Spatio-temporal granule:** A pair (temporal granule, spatial granule)
  - **Spatio-temporal validity rectangle**: A pair (temporal validity interval, **spatial validity interval**)
  - A spatial validity interval has a **begin space point** and an **end space point**

	CID	*RID*	*T*	*S*
*r*1	1	1101	[1;4)	[1;7)
*r*2	1	1101	[4;7)	[6;11)
*r*3	2	1101	[3;6)	[3;8)
*r*4	2	1101	[6;9)	[7;11)
*r*5	3	1101	[3;6)	[6;9)
*r*6	3	1101	[6;9)	[8;11)



Sequenced, Spatio-Temporal (SST) Aggregation

- Query: "How many cars, for each point in time and space?"
- Intersection of validity rectangles
- The aggregation result:
  - The set of **constant rectangles** plus the COUNT for each rectangle
- Graphical representation:

Time

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

1 2 3 4 5 6 7 8 9 10 11 12 Space (Position)

Time

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

1 2 3 4 5 6 7 8 9 10 11 12 Space (Position)

Sequenced, Spatio-Temporal (SST) Aggregation

- Query: "How many cars, for each point in time and space?"
- Intersection of validity rectangles
- The aggregation result:
  - The set of **constant rectangles** plus the COUNT for each rectangle
- Relational (STDM) representation:

	Cnt	*T*	*S*
1	1	[1;3)	[1;7)
2	1	[3;4)	[1;3)
3	2	[3;4)	[3;6)
4	3	[3;4)	[6;7)
5	2	[3;4)	[7;8)
6	1	[3;4)	[8;9)
7	1	[4;6)	[3;6)
8	3	[4;6)	[6;8)
9	2	[4;6)	[8;9)
10	1	[4;6)	[9;11)
11	1	[6;7)	[6;7)
12	2	[6;7)	[7;8)
13	3	[6;7)	[8;11)
14	1	[7;9)	[7;8)
15	2	[7;9)	[8;11)

	CID	*RID*	*T*	*S*
*r*1	1	1101	[1;4)	[1;7)
*r*2	1	1101	[4;7)	[6;11)
*r*3	2	1101	[3;6)	[3;8)
*r*4	2	1101	[6;9)	[7;11)
*r*5	3	1101	[3;6)	[6;9)
*r*6	3	1101	[6;9)	[8;11)

Data Structure for the SST Aggregation

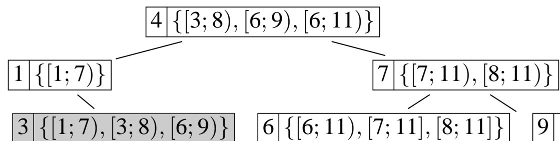
- Goal:
  - Extend the Balanced Tree to also handle the additional, **spatial** dimension
- Problem:
  - How to store space points?

Naively Extended Balanced Tree

- Idea:
  - At each node of the Balanced 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

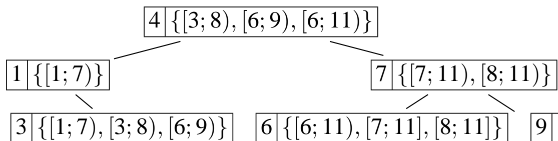
	CID	*RID*	*T*	*S*
*r*1	1	1101	[1;4)	[1;7)
*r*2	1	1101	[4;7)	[6;11)
*r*3	2	1101	[3;6)	[3;8)
*r*4	2	1101	[6;9)	[7;11)
*r*5	3	1101	[3;6)	[6;9)
*r*6	3	1101	[6;9)	[8;11)

	Cnt	*T*	*S*
1	1	[1;3)	[1;7)
2	1	[3;4)	[1;3)
3	2	[3;4)	[3;6)
4	3	[3;4)	[6;7)
5	2	[3;4)	[7;8)
6	1	[3;4)	[8;9)
7	1	[4;6)	[3;6)
8	3	[4;6)	[6;8)
9	2	[4;6)	[8;9)
10	1	[4;6)	[9;11)
11	1	[6;7)	[6;7)
12	2	[6;7)	[7;8)
13	3	[6;7)	[8;11)
14	1	[7;9)	[7;8)
15	2	[7;9)	[8;11)

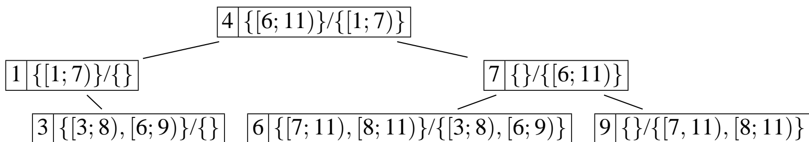


Naively Extended Balanced Tree: Optimizations

- **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:

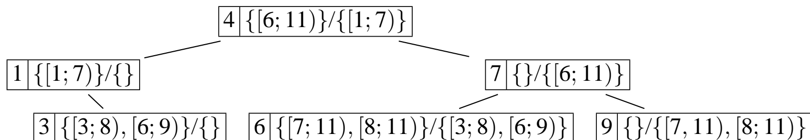


- After Optimization 1:



Naively Extended Balanced Tree: Optimizations

- **Optimization 2**:
  - Instead of space intervals, store counters of space points
- Tuple groups:
  - tuples that start at t and begin at s:
    $G_{s,b}(R,t,s) = \{r \mid r \in R \wedge r.T_s = t \wedge r.S_b = s\}$
  - tuples that start at t and end at s:
    $G_{s,e}(R,t,s) = \{r \mid r \in R \wedge r.T_s = t \wedge r.S_e = s\}$
  - tuples that finish at t and begin at s:
    $G_{f,b}(R,t,s) = \{r \mid r \in R \wedge r.T_f = t \wedge r.S_b = s\}$
  - tuples that finish at t and end at s:
    $G_{f,e}(R,t,s) = \{r \mid r \in R \wedge r.T_f = t \wedge r.S_e = s\}$

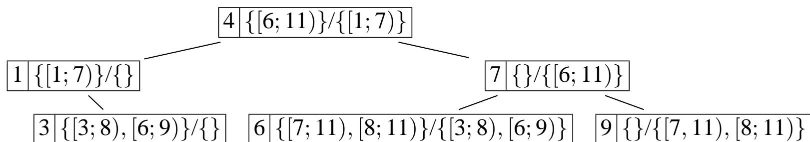


Naively Extended Balanced Tree: Optimizations

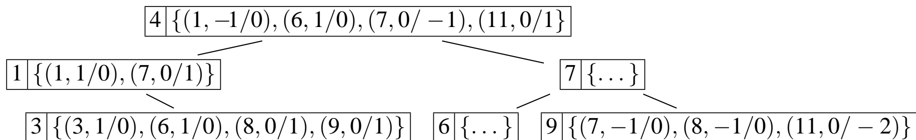
- A node in the tree: (t, S_t), where
  - t is a start/finish time point and
  - S_t is a set of *(start/finish space point, begin counter, end counter)*:

$$S_t = \{(s, cnt_b, cnt_e) | s \in P_S \wedge cnt_b = |G_{s,b}(R,t,s)| - |G_{f,b}(R,t,s)| \wedge cnt_e = |G_{s,e}(R,t,s)| - |G_{f,e}(R,t,s)|\}$$

- Before Optimization 2:

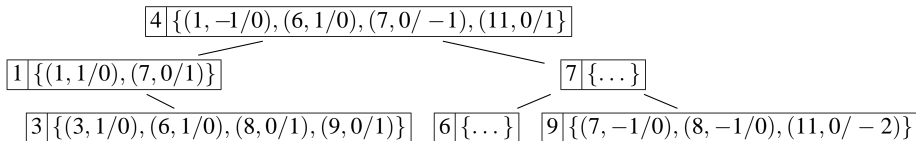


- After Optimization 2:



Sequenced, Spatio-Temporal Tree (SST-tree)

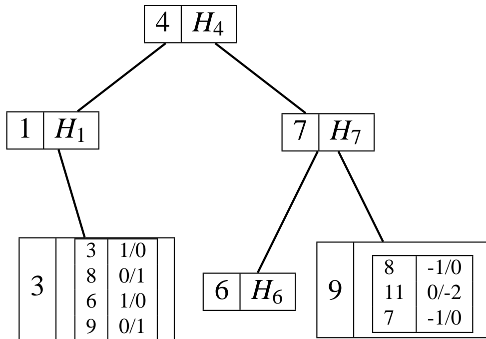
- The SST-tree is the Naively Extended Balanced Tree after Optimizations 1 and 2



	CID	*RID*	*T*	*S*
$r1$	1	1101	[1;4)	[1;7)
$r2$	1	1101	[4;7)	[6;11)
$r3$	2	1101	[3;6)	[3;8)
$r4$	2	1101	[6;9)	[7;11)
$r5$	3	1101	[3;6)	[6;9)
$r6$	3	1101	[6;9)	[8;11)

SST-tree Implementation

- For each node, the set of counters, S_t, is implemented as a hashmap:
  - A key is a space point
  - Constant space point insert and lookup time



SST-tree Algorithm: Top Level

Input: STDM relation $R = (A_1, A_2, \ldots, A_n, T, S)$
Output: STDM relation $Z = (Cnt, T, S)$

1 $Z \leftarrow \emptyset$;
2 **foreach** *road-ID rid* $\in \pi[RID]R$ **do**
3 $\quad \mathcal{T} \leftarrow$ LOADTREE$(\sigma[RID = rid]R)$;
4 $\quad Z_{rid} \leftarrow \{rid\} \times$ COMPUTECONSTRECT$(\mathcal{T})$;
5 $\quad Z \leftarrow Z \cup Z_{rid}$;

6 **return** Z;

SST-tree Algorithm: LOADTREE

Input: STDM relation $R = (A_1, A_2, \ldots, A_n, T, S)$
Output: SST-tree $\mathcal{T}$

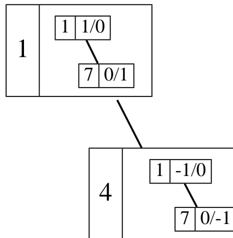
```
1  T ← empty SST-tree;
2  foreach r ∈ R do
3      tn ← GETNODE(T, r.T_s) ;
4      sn ← GETENTRY(tn.H, r.S_b) ;
5      sn.cnt_b ++ ;
6      sn ← GETENTRY(tn.H, r.S_e) ;
7      sn.cnt_e ++ ;
8      tn ← GETNODE(T, r.T_f) ;
9      sn ← GETENTRY(tn.H, r.S_b) ;
10     sn.cnt_b -- ;
11     sn ← GETENTRY(tn.H, r.S_e) ;
12     sn.cnt_e -- ;
13 return T;
```

SST-tree Algorithm: LOADTREE *Example*

- Input relation:

	CID	*RID*	*T*	*S*
*r*1	1	1101	[1;4)	[1;7)
*r*2	1	1101	[4;7)	[6;11)
*r*3	2	1101	[3;6)	[3;8)
*r*4	2	1101	[6;9)	[7;11)
*r*5	3	1101	[3;6)	[6;9)
*r*6	3	1101	[6;9)	[8;11)

- The SST-tree after inserting the first tuple:



SST-tree Algorithm: ComputeConstRect

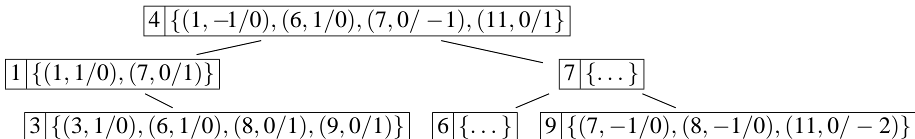
Input: SST-tree $\mathcal{T}$

Output: STDM relation $Z = (Cnt, T, S)$

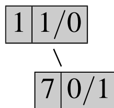
```
1  Z ← ∅ ;
2  T_dyn ← empty tree;
3  foreach pair of consecutive nodes (tn, tn′) ∈ T in in-order do
       /* Update the dynamic tree                                  */
4      foreach sn ∈ tn.T do
5          dn ← GetNode(T_dyn, sn.S);
6          dn.cnt_b = dn.cnt_b + sn.cnt_b ;
7          dn.cnt_e = dn.cnt_e + sn.cnt_e ;
8          if dn.cnt_b = 0 ∧ dn.cnt_e = 0 then
9              Delete dn from T_dyn;
       /* Traverse the dynamic tree                                */
10     cnt ← 0 ;
11     foreach pair of consecutive nodes (dn, dn′) ∈ T_dyn in in-order do
12         cnt ← cnt + dn.cnt_b − dn.cnt_e ;
13         Z ← Z ∪ {(cnt, [tn.T, tn′.T), [dn.S, dn′.S))} ;
14 return Z;
```

SST-tree Algorithm: COMPUTECONSTRECT *Example*

- The SST-tree:



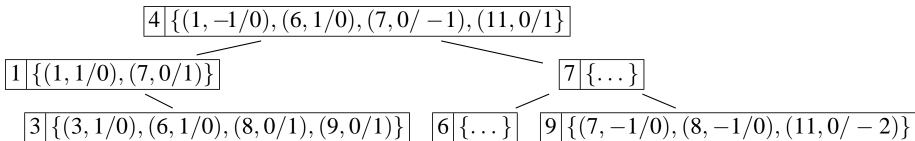
- The dynamic tree and the result relation after processing 1 node:



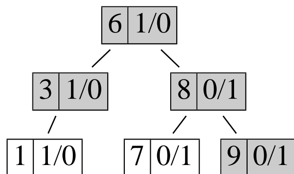
Cnt	*T*	*S*
1	[1;3)	[1;7)

SST-tree Algorithm: COMPUTECONSTRECT *Example*

- The SST-tree:



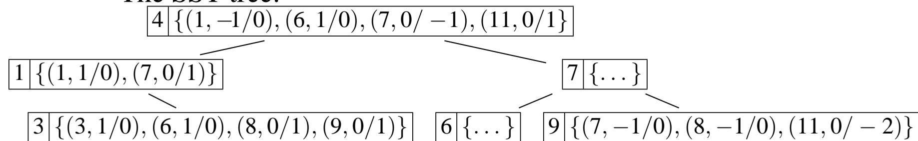
- The dynamic tree and the result relation after processing 2 nodes



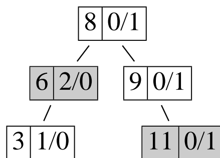
Cnt	*T*	*S*
1	[1;3)	[1;7)
1	[3;4)	[1;3)
2	[3;4)	[3;6)
3	[3;4)	[6;7)
2	[3;4)	[7;8)
1	[3;4)	[8;9)

SST-tree Algorithm: ComputeConstRect *Example*

- The SST-tree:



- The dynamic tree and the result relation after processing 3 nodes



Cnt	*T*	*S*
1	[1;3)	[1;7)
1	[3;4)	[1;3)
2	[3;4)	[3;6)
3	[3;4)	[6;7)
2	[3;4)	[7;8)
1	[3;4)	[8;9)
1	[4;6)	[3;6)
3	[4;6)	[6;8)
2	[4;6)	[8;9)
1	[4;6)	[9;11)