Stab-Forests: Dynamic Data Structures for Efficient Temporal Query Processing

Jelle Hellings Yuqing Wu

UC DAVIS UNIVERSITY OF CALIFORNIA

Pomona College

Temporal event-based data

Consider *events* of the form $\langle \text{start-time}, \text{end-time} \rangle$.

Example

- Process runtime in computer systems.
- Airline On-Time Performance Data (AOTPD).
- Civil Unrest Event Data (CUED).

Definition

Let R and S be sets of events. The temporal join $R \bowtie S$ is defined by

$$R \bowtie S = \{(\mathrm{E}_1, \mathrm{E}_2) \in R \times S \mid \mathrm{E}_1 \cap \mathrm{E}_2 \neq \emptyset\}.$$

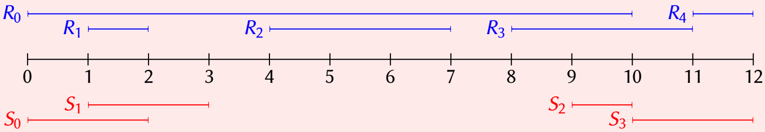
High-performance temporal joins $R \bowtie S$ via FwdScan

- All data in internal memory
  $\Longrightarrow$ R and S: *arrays* of events, ordered on time.
- Cache-friendly traversal of data
  $\Longrightarrow$ *sequential scan* of R and S.

Proposition

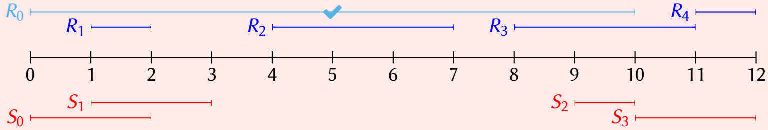
FwdScan(R, S) *computes* $R \bowtie S$ *in worst-case* $O(|R| + |S| + |output|)$.

Temporal joins via FwdScan (example)



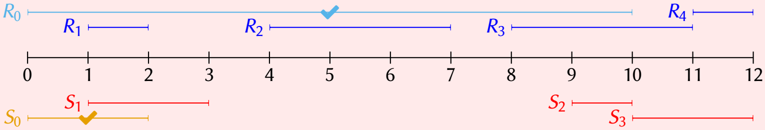
$$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$$

Temporal joins via FwdScan (example)



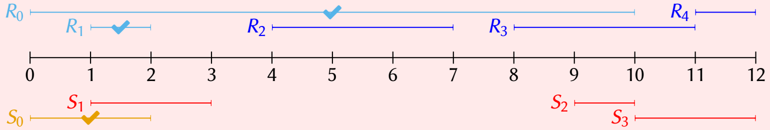
$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$

Temporal joins via FwdScan (example)



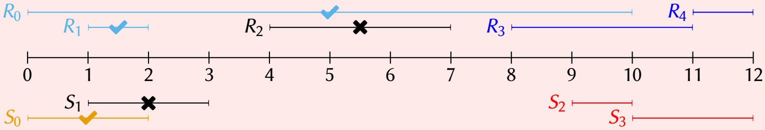
$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$

Temporal joins via FwdScan (example)



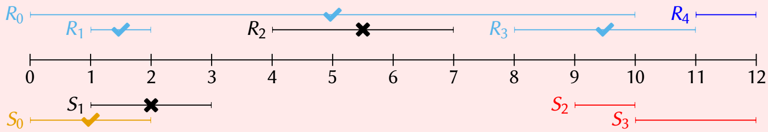
$$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$$

Temporal joins via FwdScan (example)



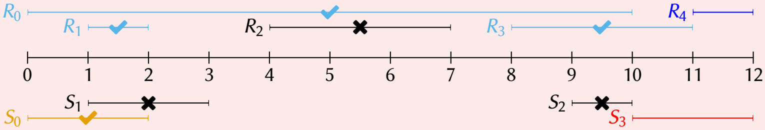
$$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$$

Temporal joins via FwdScan (example)



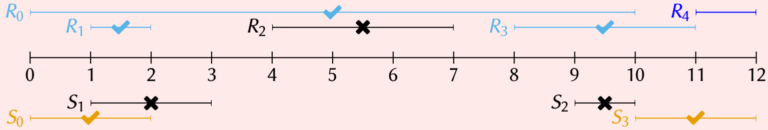
$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$

Temporal joins via FwdScan (example)



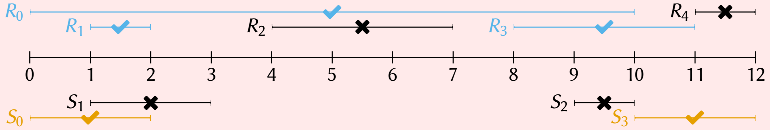
$$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$$

Temporal joins via FwdScan (example)



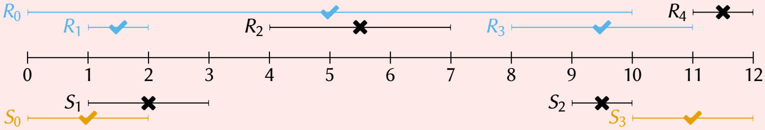
$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$

Temporal joins via FwdScan (example)



$$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$$

Temporal joins via FwdScan (example)



$[(R_0, S_0), (R_0, S_1), (R_0, S_2), (R_1, S_0), (R_1, S_1), (R_3, S_2), (R_3, S_3), (R_4, S_3)]$

Question

Can we prevent inspection of R_2?

The SkipJoin algorithm

Algorithm SkipJoin(R, S):

```
i, j := 0, 0
while i < |R| and j < |S| do
    if R[i].start ≤ S[j].start then
        if S[j].start ≤ R[i].end then          ▷ Normal Join (e.g., Forward Scan).
            Join R[i] with S[j ...]
            i := i + 1
        else                                   ▷ Skip Events.
            (i, L) := Stab(R[i ...], S[j].start)
            For each event e ∈ L, join e with S[j ...]
    else analogous (swap roles of R and S)
```

$$\textsc{Stab}(R, t) = \{\textsc{e} \in R \mid \textsc{e}.\texttt{start} \le t \le \textsc{e}.\texttt{end}\}$$

High-performance sequences of stab queries

Problem

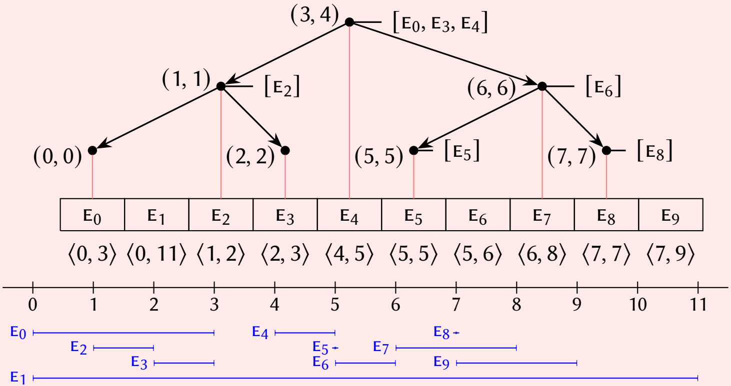
Cannot efficiently query an array A for all events active at t.

Solution

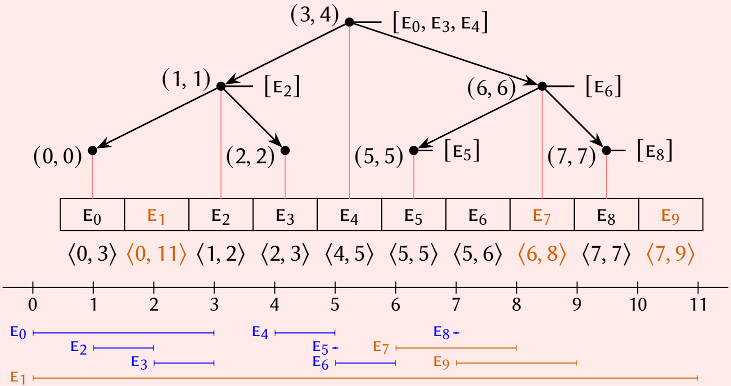
Stab-forest: index-structure supporting sequences of stab-queries.

- A fully-balanced binary search tree to minimize size.
- Event-based augmentations via compact static arrays.
- Efficiently supports append-only operations.

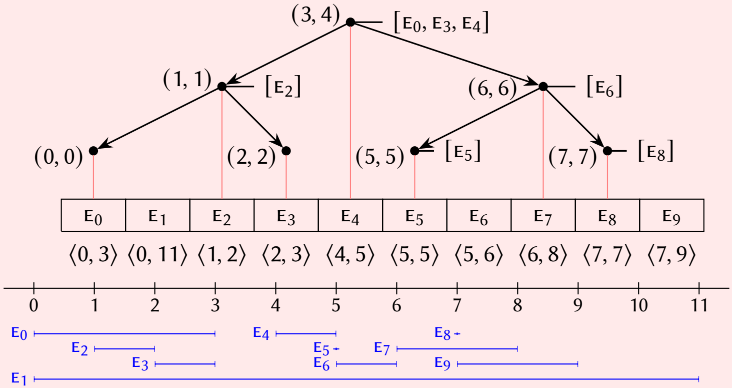
Stab-forests: Collections of stab-trees



Stab-forests: Collections of stab-trees

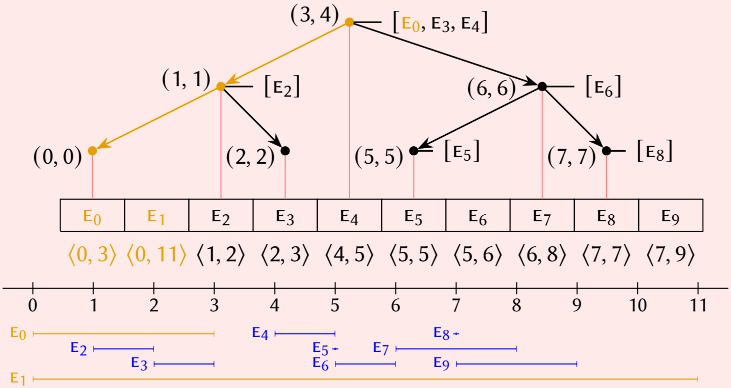


Stab-forests: Querying a stab-tree



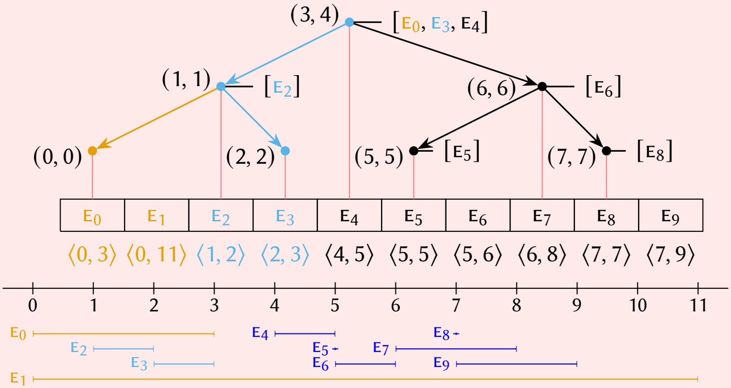
Query: "MultiStab($\mathcal{S}$, [0, 2, 7])".

Stab-forests: Querying a stab-tree



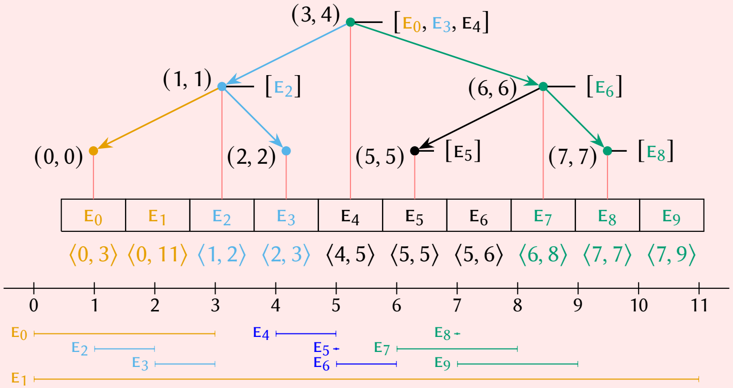
Query: "MultiStab($\mathcal{S}$, [0, 2, 7])".

Stab-forests: Querying a stab-tree



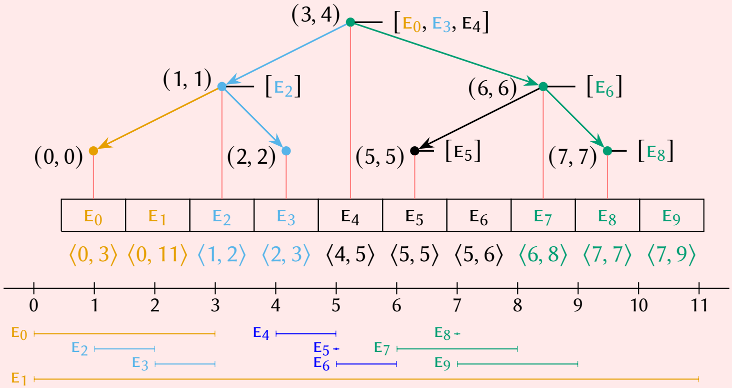
Query: "MULTISTAB($\mathcal{S}$, [0, 2, 7])".

Stab-forests: Querying a stab-tree



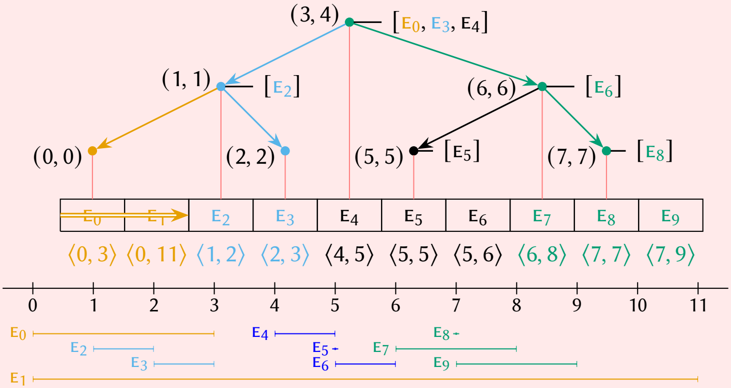
Query: “MultiStab($\mathscr{S}$, [0, 2, 7])”.

Stab-forests: Cache-friendly querying a stab-tree



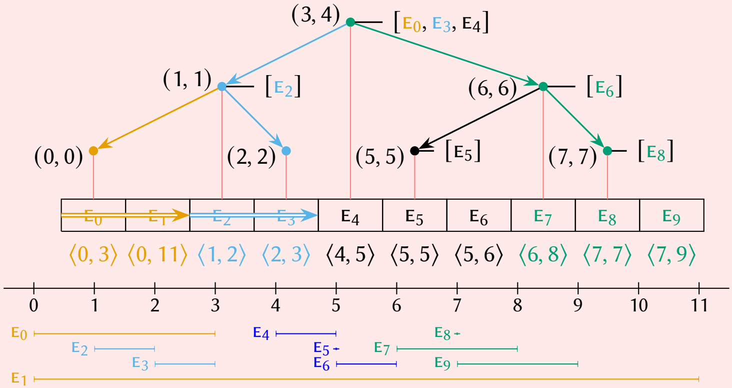
Query: "MultiStab($\mathcal{S}$, [0, 2, 7])".

Stab-forests: Cache-friendly querying a stab-tree



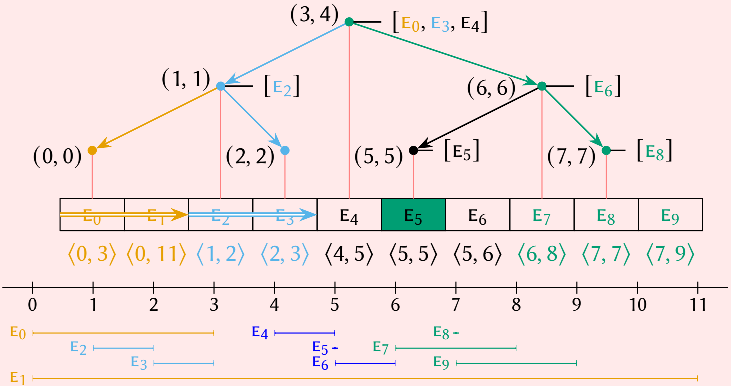
Query: "MultiStab($\mathcal{S}$, [0, 2, 7])".

Stab-forests: Cache-friendly querying a stab-tree



Query: "MultiStab($\mathcal{S}$, [0, 2, 7])".

Stab-forests: Cache-friendly querying a stab-tree



Query: "MultiStab($\mathcal{S}$, [0, 2, 7])".

SkipJoin with stab-forests: Results

Theorem

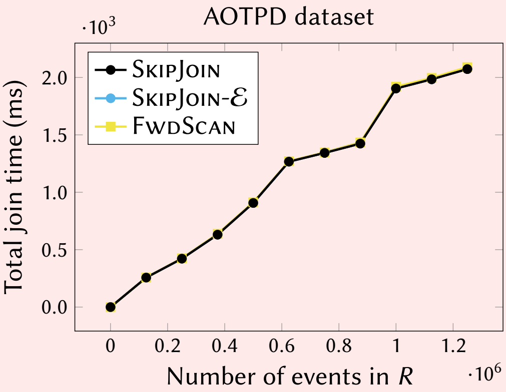
A stab-forest indexing L

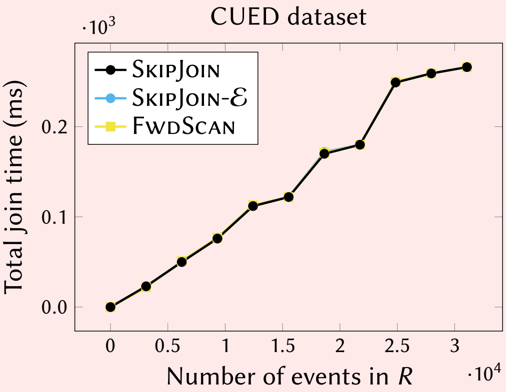
- *can be stored in worst-case $\mathcal{O}(|L|)$ space.*
- *can be constructed in $\mathcal{O}(|L|\log|L|)$.*
- *can be appended-to in amortized $\mathcal{O}(\log|L|)$.*
- *can be stabbed for timestamps ϕ in $\mathcal{O}(\min(|\phi|\log|L|, |\phi| + |L|) + |output|)$.*

Theorem

SkipJoin(R, S) *computes $R \bowtie S$ in worst-case $\mathcal{O}(M(R, S) + M(S, R) + |output|)$, with $M(A, B)$ the cost of stabbing A with $|B|$ timestamps.*

Self-joining real-world datasets

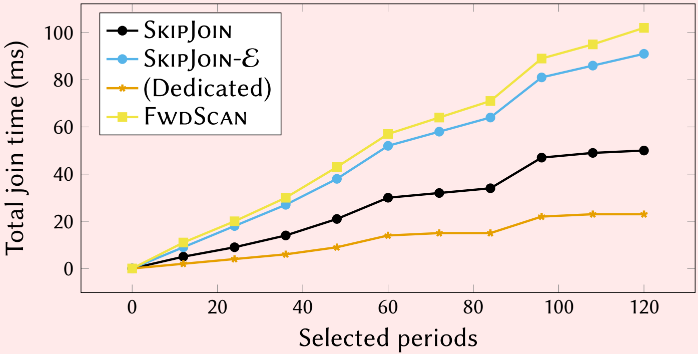




- 61,100,539 events in 10 years.
- Shortest: 0 min. Longest: 1350 min.
- Used: 2,500,000 events.

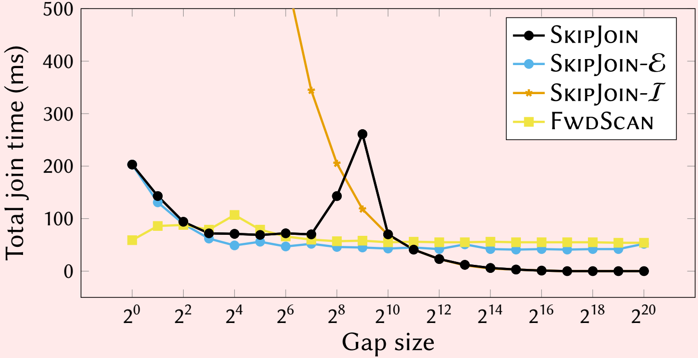
- 62,141 events in 60 years.
- Shortest: 0 d. Longest: 18 407 d.

Sparse joins and dedicated multi-window-queries

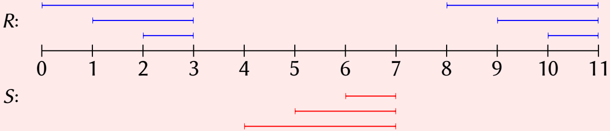


Query: *"select all events on the 7-th day from the first n months in AOTPD"*.

The behavior of SkipJoin

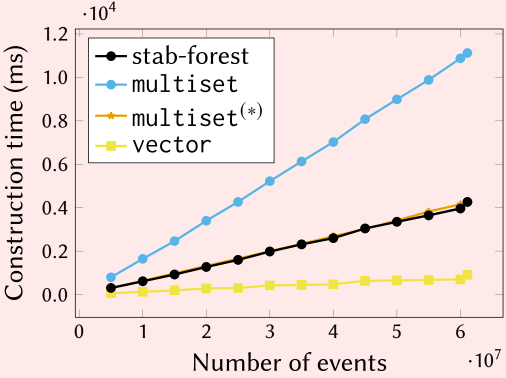


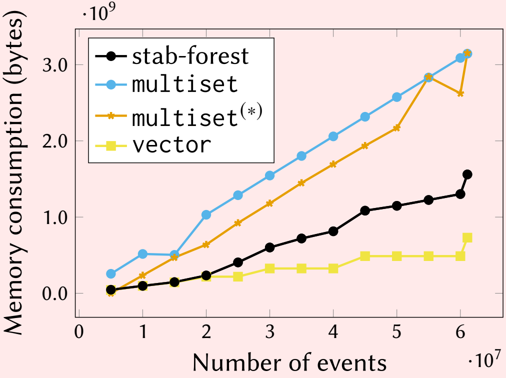
Query: “*$R \bowtie S$ with*



(gap size 3)”.

The costs of stab-forests





Conclusion

SkipJoin: (windowed) temporal joins on skewed datasets.
Stab-forests: append-only index for sequences of stab-queries.

Future Work

- Using stab-forests for other temporal operations.
- Block-based external-memory designs based on the append-only stab-forests.
- Parallel and distributed high-performance temporal joins.

https://jhellings.nl/