Temporal data is important in numerous domains:
- financial market
- scientific applications
- biomedical field
Temporal data is important in numerous domains:
- financial market
- scientific applications
- biomedical field

Extensive efforts have been made towards efficiently storing, processing, and querying temporal data.
Temporal data is important in numerous domains:
- financial market
- scientific applications
- biomedical field

Extensive efforts have been made towards efficiently storing, processing, and querying temporal data.

Ranking temporal data has only recently been studied. [LYL10]
The *instant* top-$k$ query returns objects $o_i$s with the $k$ highest scores at query time $t$. \[LYL10\]
The *instant* top-\(k\) query returns objects \(o_i\)'s with the \(k\) highest scores at query time \(t\). [LYL10]

Example: Return top-10 weather stations with highest average temperature from 1 Aug to 27 Aug.

[LYL10] Li et al., Top-\(k\) queries on temporal data. In *VLDBJ*, 2010.
The *instant* top-*k* query returns objects $o_i$s with the *k* highest scores at query time $t$. [LYL10]

What is a good value for $t$? 

![Graph showing score over time with rankings]

[LYL10] Li et al., Top-*k* queries on temporal data. In VLDBJ, 2010.
The *instant* top-\(k\) query returns objects \(o_i\)s with the \(k\) highest scores at query time \(t\). [LYL10]

What is a good value for \(t\)?

[LYL10] Li et al., Top-\(k\) queries on temporal data. In *VLDBJ*, 2010.
The *instant* top-\(k\) query returns objects \(o_i\)'s with the \(k\) highest scores at query time \(t\). [LYL10]

What is a good value for \(t\)?

[LYL10] Li et al., Top-\(k\) queries on temporal data. In *VLDBJ*, 2010.
The *instant* top-\(k\) query returns objects \(o_i\)'s with the \(k\) highest scores at query time \(t\). \([LYL10]\)

Use aggregation within a temporal interval instead!!!

Example: Return top-10 weather stations with highest average temperature from 1 Aug to 27 Aug.

[LYL10] Li et al., Top-\(k\) queries on temporal data. In *VLDBJ*, 2010.
The *instant* top-$k$ query returns objects $o_i$s with the $k$ highest scores at query time $t$. [LYL10]

**Example:** Return top-10 weather stations with highest average temperature from 1 Aug to 27 Aug.
Temporal database consists of $m$ objects $o_1, o_2, \ldots, o_m$. 

- **Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang**

**Ranking Large Temporal Data**
Problem Formulation

Temporal database consists of $m$ objects $o_1, o_2, \ldots, o_m$

- $o_i$ is represented by piecewise linear function $g_i$ with $n_i$ segments.

$g_i : \mathbb{R} \rightarrow \mathbb{R}$ (time $\rightarrow$ score)

$g_3(t) = 90$

$n_1 = 3$

$n_2 = 5$

$n_3 = 6$
Temporal database consists of $m$ objects $o_1, o_2, \ldots, o_m$

- $o_i$ is represented by piecewise linear function $g_i$ with $n_i$ segments.
- $\text{top-} k(t_1, t_2, \sigma)$ is an aggregate top-$k$ query for aggregate function $\sigma$
  - $g_i(t_1, t_2)$ represent all possible values of $g_i$ in $[t_1, t_2]$
  - $\sigma(g_i(t_1, t_2))$ ($= \sigma_i(t_1, t_2)$) is the aggregate score of $o_i$ in $[t_1, t_2]$
Temporal database consists of \( m \) objects \( o_1, o_2, \ldots, o_m \)

- \( o_i \) is represented by piecewise linear function \( g_i \) with \( n_i \) segments.
- top-\( k(t_1, t_2, \sigma) \) is an aggregate top-\( k \) query for aggregate function \( \sigma \)
  - \( g_i(t_1, t_2) \) represent all possible values of \( g_i \) in \([t_1, t_2] \)
  - \( \sigma(g_i(t_1, t_2)) (\sigma_i(t_1, t_2)) \) is the aggregate score of \( o_i \) in \([t_1, t_2] \)
  - For \( \sigma = \text{sum} \), \( \sigma(g_i(t_1, t_2)) = \int_{t_1}^{t_2} g_i(t) dt \)
Problem Formulation

$A(k, t_1, t_2) :$ ordered top-$k$ objects for top-$k(t_1, t_2, \sigma)$

Let $\sigma = \text{sum} = \int_{t_1}^{t_2} g(t) dt$
Problem Formulation

\[ A(2, t_1, t_2) = \{o_3, o_1\} \]

- \( A(k, t_1, t_2) \): ordered top-\( k \) objects for top-\( k(t_1, t_2, \sigma) \)
- Let \( \sigma = \text{sum} = \int_{t_1}^{t_2} g(t)dt \)
\[ A(1, t_1, t_2) = \{ o_1 \} \]

- \( A(k, t_1, t_2) \): ordered top-\( k \) objects for top-\( k(\tau_{1}, \tau_{2}, \sigma) \)
- Let \( \sigma = \text{sum} = \int_{t_1}^{t_2} g(t)dt \)
Outline

1. Introduction and Problem Formulation

2. Exact Solutions
   - Baseline Solution
   - Improved Solution using Prefix Sums and B-tree Forest
   - Improved Solution using Prefix Sums and Interval Tree

3. Approximate Solutions
   - Overview
   - Breakpoints
   - Approaches for Approximation Queries
   - Combining Breakpoints with Queries

4. Experiments

5. Conclusions
Baseline Solution

- Compute $\sigma_i(t_1, t_2)$ for all objects by scanning each segment.
Baseline Solution

- Compute $\sigma_i(t_1, t_2)$ for all objects by scanning each segment.
- Simple improvement: use B-tree to avoid segments outside query interval.
- Query cost: $O(\log_B N + \sum_{i=1}^{m} \frac{q_i}{B} + (m/B)\log_B k)$
  - $q_i = \text{number of segments overlapping } [t_1, t_2]$
- We denote this query $\text{Exact1}$. 
Improved Solution using Prefix Sums and B-tree Forest

- We can avoid scanning all overlapping segments with \([t_1, t_2]\) by using prefix sums:
  - Index segment and prefix sums for an object in a B-tree.
  - Compute \(\sigma_i(t_1, t_2)\) by retrieving two segments from B-tree.
We can avoid scanning all overlapping segments with \([t_1, t_2]\) by using prefix sums:

- Index segment and prefix sums for an object in a B-tree.
- Compute \(\sigma_i(t_1, t_2)\) by retrieving two segments from B-tree.

Query cost is \(O(\sum_{i=1}^{m} \log_B n_i + (m/B)\log_B k)\)

This solution is denoted \(\text{EXACT2}\).
Consider an object $o_i$ with intervals $l_{i,1}, \ldots, l_{i,n_i}$

- $g_{i,j} = j$th segment of $o_i$ is $((t_{i,j-1}, v_{i,j-1}), (t_{i,j}, v_{i,j}))$
- $l_{i,\ell} = [t_{i,0}, t_{i,\ell}]$ for $\ell = 1, \ldots, n_i$
We define $l_{i,1}, \ldots, l_{i,n_i}$ s.t. $l_{i,\ell} = [l_{i,\ell-1}, l_{i,\ell}]$

The data entries for $i = 1, \ldots, m$ and $\ell = 1, \ldots, n_i$ are

- key: $(l_{i,\ell})$ and value: $(g_{i,\ell}, \sigma_i(l_{i,\ell}))$
We define $I_{i,1}, \ldots, I_{i,n_i}$ s.t. $I_{i,\ell} = [I_{i,\ell-1}, I_{i,\ell}]$

The data entries for $i = 1, \ldots, m$ and $\ell = 1, \ldots, n_i$ are

- key: $(I_{i,\ell})$ and value: $(g_{i,\ell}, \sigma_i(I_{i,\ell}))$
We define $I_{i,1}, \ldots, I_{i,n_i}$ s.t. $I_{i,\ell} = [l_{i,\ell-1}, l_{i,\ell}]$

The data entries for $i = 1, \ldots, m$ and $\ell = 1, \ldots, n_i$ are

- key: $(l_{i,\ell})$ and value: $(g_{i,\ell}, \sigma_i(l_{i,\ell}))$
We define $l_{i,1}, \ldots, l_{i,n_i}$ s.t. $l_{i,\ell} = [l_{i,\ell-1}, l_{i,\ell}]$

The data entries for $i = 1, \ldots, m$ and $\ell = 1, \ldots, n_i$ are
- key: $l_{i,\ell}$ and value: $(g_{i,\ell}, \sigma_i(l_{i,\ell}))$
We define $I_{i,1}^-, \ldots, I_{i,n_i}^-$ s.t. $I_{i,\ell}^- = [l_{i,\ell-1}, l_{i,\ell}]$

The data entries for $i = 1, \ldots, m$ and $\ell = 1, \ldots, n_i$ are
- key: $(l_{i,\ell}^-)$ and value: $(g_{i,\ell}, \sigma_i(l_{i,\ell}))$
We define $l_{i,1}, \ldots, l_{i,n_i}$ s.t. $l_{i,\ell} = [l_{i,\ell-1}, l_{i,\ell}]$

The data entries for $i = 1, \ldots, m$ and $\ell = 1, \ldots, n_i$ are

- key: $(l_{i,\ell})$ and value: $(g_{i,\ell}, \sigma_i(l_{i,\ell}))$
Improved Solution using Prefix Sums and Interval Tree

We have \( g_{3,2}; g_{3,4}; I_{3,2}; I_{3,4} \)

Compute as \( I_{3,4} - I_{3,2} \)

We define \( I_{i,1,1}, \ldots, I_{i,n_i} \) s.t. \( I_{i,1,\ell} = [l_{i,\ell-1}, l_{i,\ell}] \)

The data entries for \( i = 1, \ldots, m \) and \( \ell = 1, \ldots, n_i \) are

- key: \((l_{i,\ell})\) and value: \((g_{i,\ell}, \sigma_i(l_{i,\ell}))\)
We define $l_{i,1}, \ldots, l_{i,n_i}$ s.t. $l_{i,\ell} = [l_{i,\ell-1}, l_{i,\ell}]$

The data entries for $i = 1, \ldots, m$ and $\ell = 1, \ldots, n_i$ are

- key: $(l_{i,\ell})$ and value: $(g_{i,\ell}, \sigma_i(l_{i,\ell}))$
We define $l_{i,1}, \ldots, l_{i,n_i}$ s.t. $l_{i,\ell} = [l_{i,\ell-1}, l_{i,\ell}]$

The data entries for $i = 1, \ldots, m$ and $\ell = 1, \ldots, n_i$ are
- key: $(l_{i,\ell})$ and value: $(g_{i,\ell}, \sigma_i(l_{i,\ell}))$
Total stabbing query cost is $O(\log_B N + m/B)$. 

Using priority queue to get top-k is $O(\log_B N + (m/B)\log_B k)$. 

We denote this query $\text{EXACT}3$. 

---

Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang 

Ranking Large Temporal Data
Outline

1. Introduction and Problem Formulation
2. Exact Solutions
   - Baseline Solution
   - Improved Solution using Prefix Sums and B-tree Forest
   - Improved Solution using Prefix Sums and Interval Tree
3. Approximate Solutions
   - Overview
   - Breakpoints
   - Approaches for Approximation Queries
     - Nested B-tree Approximate Query
     - Dyadic Interval Approximate Query
   - Combining Breakpoints with Queries
4. Experiments
5. Conclusions
Our most query-efficient technique costs $O(\log_B N + m/B)$.

- Must compute all $m$ aggregates $\sigma_i(t_1, t_2)$.
- Still too expensive for large datasets with large $m$. 
Approximate Solution Overview

- Our most query-efficient technique costs $O(\log_B N + m/B)$.
  - Must compute all $m$ aggregates $\sigma_i(t_1, t_2)$.
  - Still too expensive for large datasets with large $m$.

- Our approximate methods construct breakpoints $B = \{b_1, \ldots, b_r\}$, $b_i \in [0, T]$. 
Approximate Solution Overview

- Our most query-efficient technique costs $O(\log_B N + m/B)$.
  - Must compute all $m$ aggregates $\sigma_i(t_1, t_2)$.
  - Still too expensive for large datasets with large $m$.

- Our approximate methods construct breakpoints $\mathcal{B} = \{b_1, \ldots, b_r\}$, $b_i \in [0, T]$.

- Queries are snapped to align to breakpoints.
  - A query snapped to $(b_i, b_j)$ uses $\sigma_i(b_i, b_j)$ as an object’s score.
G is an \((\varepsilon, \alpha)\)-approximation algorithm if:

- \(G\) returns \(\tilde{\sigma}_i(t_1, t_2)\) such that
  \[
  \frac{\sigma_i(t_1, t_2)}{\alpha} - \varepsilon M \leq \tilde{\sigma}_i(t_1, t_2) \leq \sigma_i(t_1, t_2) + \varepsilon M
  \]
- \(\alpha \geq 1, \varepsilon > 0\)
- \(M = \sum_{i=1}^{m} \sigma_i(0, T)\)

Must hold for all objects and temporal intervals.
Approximate Solution Notations

- $A(j)$ ($\tilde{A}(j)$) = the $j$th ranked object in $A(k, t_1, t_2)$ ($\tilde{A}(k, t_1, t_2)$)
- $R$ is an $(\varepsilon, \alpha)$-approximation algorithm of top-$k(t_1, t_2, \sigma)$ if:
  - $R$ returns $\tilde{A}(k, t_1, t_2)$ and $\tilde{\sigma}_{\tilde{A}(j)}(t_1, t_2)$ for $j \in [1, k]$, s.t.
\[
\sigma_2(t_1, t_2) / \alpha - \varepsilon M \leq \tilde{\sigma}_2(t_1, t_2) \leq \sigma_2(t_1, t_2) + \varepsilon M
\]

- \(\mathcal{A}(j) (\tilde{\mathcal{A}}(j))\) = the \(j\)th ranked object in \(\mathcal{A}(k, t_1, t_2) (\tilde{\mathcal{A}}(k, t_1, t_2))\)
- \(R\) is an \((\varepsilon, \alpha)\)-approximation algorithm of top-\(k(t_1, t_2, \sigma)\) if:
  - \(R\) returns \(\tilde{\mathcal{A}}(k, t_1, t_2)\) and \(\tilde{\sigma}_{\tilde{\mathcal{A}}(j)}(t_1, t_2)\) for \(j \in [1, k]\), s.t.
    - \(\tilde{\sigma}_{\tilde{\mathcal{A}}(j)}(t_1, t_2)\) is an \((\varepsilon, \alpha)\)-approximation of \(\sigma_{\mathcal{A}(j)}(t_1, t_2)\)
Approximate Solution Notations

\[ \frac{\sigma_3(t_1, t_2)}{\alpha} - \varepsilon M \leq \tilde{\sigma}_2(t_1, t_2) \leq \sigma_3(t_1, t_2) + \varepsilon M \]

- \( A(j) (\tilde{A}(j)) = \) the \( j \)-th ranked object in \( A(k, t_1, t_2) (\tilde{A}(k, t_1, t_2)) \)
- \( R \) is an \((\varepsilon, \alpha)\)-approximation algorithm of top-\( k(t_1, t_2, \sigma) \) if:
  - \( R \) returns \( \tilde{A}(k, t_1, t_2) \) and \( \tilde{\sigma}_{\tilde{A}(j)}(t_1, t_2) \) for \( j \in [1, k] \), s.t.
    1. \( \tilde{\sigma}_{\tilde{A}(j)}(t_1, t_2) \) is an \((\varepsilon, \alpha)\)-approximation of \( \sigma_{\tilde{A}(j)}(t_1, t_2) \)
    2. \( \tilde{\sigma}_{\tilde{A}(j)}(t_1, t_2) \) is an \((\varepsilon, \alpha)\)-approximation of \( \sigma_{A(j)}(t_1, t_2) \)
- Must hold for all \( k \) and all temporal intervals.
Starting from $b_0$ and moving forward we have:

\[
\begin{align*}
    b_{j+1} \quad \text{so} \quad \sum_{i=1}^{m} \sigma_i(b_j, b_{j+1}) &= \varepsilon M, \\
    \max_{i=1}^{m} \sigma_i(b_j, b_{j+1}) &= \varepsilon M,
\end{align*}
\]

in BreakPoints1($B_1$) and BreakPoints2($B_2$)
Starting from $b_0$ and moving forward we have:

$$b_{j+1} \text{ so that } \begin{cases} 
\sum_{i=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BreakPoints1}(B_1) \\
\max_{i=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BreakPoints2}(B_2)
\end{cases}$$

Score

Time

$\times$ breakpoint

$$\sigma_1(b_j, b_{j+1}) + \sigma_2(b_j, b_{j+1}) + \sigma_3(b_j, b_{j+1}) = \varepsilon M$$
Starting from $b_0$ and moving forward we have:

\[
\begin{aligned}
    b_{j+1} & \quad \text{so} \\
    \sum_{i=1}^{m} \sigma_i(b_j, b_{j+1}) &= \varepsilon M, \\
    \max_{i=1}^{m} \sigma_i(b_j, b_{j+1}) &= \varepsilon M,
\end{aligned}
\]

in \text{BreakPoints1}(B_1) and \text{BreakPoints2}(B_2).

\[
\sigma_1(b_j, b_{j+1}) + \sigma_2(b_j, b_{j+1}) + \sigma_3(b_j, b_{j+1}) = \varepsilon M
\]
Starting from \( b_0 \) and moving forward we have:

\[
\begin{align*}
\sigma_i(b_j, b_{j+1}) &= \varepsilon M, \\
\max_{i=1}^m \sigma_i(b_j, b_{j+1}) &= \varepsilon M,
\end{align*}
\]

in BreakPoints1(\( B_1 \))

in BreakPoints2(\( B_2 \))

\[
\sigma_1(b_j, b_{j+1}) + \sigma_2(b_j, b_{j+1}) + \sigma_3(b_j, b_{j+1}) = \varepsilon M
\]
Starting from $b_0$ and moving forward we have:

\[
\begin{align*}
\exists j & \sum_{i=1}^m \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in } \text{BreakPoints}_1(B_1) \\
\max_{i=1}^m \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in } \text{BreakPoints}_2(B_2)
\end{align*}
\]

\[\sigma_1(b_j, b_{j+1}) + \sigma_2(b_j, b_{j+1}) + \sigma_3(b_j, b_{j+1}) = \varepsilon M\]
Starting from $b_0$ and moving forward we have:

$$ b_{j+1} \text{ so that } \begin{cases} \sum_{i=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BreakPoints}_1(B_1) \\ \max_{i=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BreakPoints}_2(B_2) \end{cases} $$

Score

Time

- $o_1$
- $o_2$
- $o_3$

$x$ breakpoint

$$ \max\{\sigma_1(b_j, b_{j+1}), \sigma_2(b_j, b_{j+1}), \sigma_3(b_j, b_{j+1})\} = \varepsilon M $$
Starting from $b_0$ and moving forward we have:

$$b_{j+1} \text{ so that } \begin{cases} \sum_{i=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BREAKPOINTS}_1(B_1) \\ \max_{j=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BREAKPOINTS}_2(B_2) \end{cases}$$

$\max\{\sigma_1(b_j, b_{j+1}), \sigma_2(b_j, b_{j+1}), \sigma_3(b_j, b_{j+1})\} = \varepsilon M$
Properties of Breakpoints

Starting from $b_0$ and moving forward we have:

$$b_{j+1} \quad \text{so} \quad \begin{cases} \sum_{i=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BreakPoints1}(B_1) \\ \max_{i=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BreakPoints2}(B_2) \end{cases}$$

- We show how to efficiently construct both types of breakpoints
- A cost of $O((N/B)\log_B N)$ IOs for both types.
Properties of Breakpoints

Starting from \( b_0 \) and moving forward we have:

\[
 b_{j+1} \quad \text{so} \quad \begin{cases} 
 \sum_{i=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BreakPoints1}(B_1) \\
 \max_{i=1}^{m} \sigma_i(b_j, b_{j+1}) = \varepsilon M, & \text{in BreakPoints2}(B_2)
\end{cases}
\]

- We show how to efficiently construct both types of breakpoints
  - A cost of \( O((N/B) \log_B N) \) IOs for both types.
  - The theoretical number of breakpoints is \( O(1/\varepsilon) \) for both types.
  - \text{BreakPoints2} has much fewer breakpoints than \text{BreakPoints1} in practice.
We show how to answer queries using $B_1$ or $B_2$ approximately.

$\forall (t_1, t_2)$, let $(B(t_1), B(t_2))$ be the approximate interval

- $B(t_1) = \min_{b_i \in B} \text{ s.t. } B(t_1) \geq t_1$
- $B(t_2) = \min_{b_i \in B} \text{ s.t. } B(t_2) \geq t_2$
We show how to answer queries using $B_1$ or $B_2$ approximately.

- $\forall(t_1, t_2)$, let $(B(t_1), B(t_2))$ be the approximate interval
  - $B(t_1) = \min_{b_i \in B} \text{ s.t. } B(t_1) \geq t_1$
  - $B(t_2) = \min_{b_i \in B} \text{ s.t. } B(t_2) \geq t_2$
Answering Queries with Breakpoints

We show how to answer queries using $B_1$ or $B_2$ approximately.

\[ \forall (t_1, t_2), \text{ let } (B(t_1), B(t_2)) \text{ be the approximate interval} \]

- $B(t_1) = \min_{b_i \in B} \text{ s.t. } B(t_1) \geq t_1$
- $B(t_2) = \min_{b_i \in B} \text{ s.t. } B(t_2) \geq t_2$
We show how to answer queries using $B_1$ or $B_2$ approximately.

$\forall (t_1, t_2)$, let $(B(t_1), B(t_2))$ be the approximate interval

$B(t_1) = \min_{b_i \in B} \text{ s.t. } B(t_1) \geq t_1$

$B(t_2) = \min_{b_i \in B} \text{ s.t. } B(t_2) \geq t_2$

**Lemma**

$\forall (t_1, t_2)$ and its approximate interval $(B(t_1), B(t_2))$: $\forall o_i,$

$|\sigma_i(t_1, t_2) - \sigma_i(B(t_1), B(t_2))| \leq \varepsilon M.$
We show how to answer queries using $B_1$ or $B_2$ approximately.

$\forall (t_1, t_2)$, let $(B(t_1), B(t_2))$ be the approximate interval

- $B(t_1) = \min_{b_i \in B} \text{ s.t. } B(t_1) \geq t_1$
- $B(t_2) = \min_{b_i \in B} \text{ s.t. } B(t_2) \geq t_2$

Lemma

$\forall (t_1, t_2)$ and its approximate interval $(B(t_1), B(t_2))$: $\forall o_i$, $|\sigma_i(t_1, t_2) - \sigma_i(B(t_1), B(t_2))| \leq \varepsilon M$. 
Outline

1 Introduction and Problem Formulation

2 Exact Solutions
   - Baseline Solution
   - Improved Solution using Prefix Sums and B-tree Forest
   - Improved Solution using Prefix Sums and Interval Tree

3 Approximate Solutions
   - Overview
   - Breakpoints
     - Approaches for Approximation Queries
       - Nested B-tree Approximate Query
       - Dyadic Interval Approximate Query
     - Combining Breakpoints with Queries

4 Experiments

5 Conclusions
Querying Breakpoints with Nested B-trees

- breakpoint

\[ \begin{array}{c}
  \text{Left end-point index.} \\
  \text{Time}
\end{array} \]

- **QUERY1** indexes all \( \binom{n}{2} \) intervals of breakpoints \( B \).
- For each interval \([b_j, b'_j]\), \( A(k_{max}, b_j, b'_j) \) is computed.

Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang

Ranking Large Temporal Data
Querying Breakpoints with Nested B-trees

× breakpoint

QUERY1 indexes all \( \binom{n}{2} \) intervals of breakpoints \( B \).
- For each interval \([b_j, b_j']\), \( A(k_{max}, b_j, b_j') \) is computed.
Querying Breakpoints with Nested B-trees

Query1 indexes all \( \binom{n}{2} \) intervals of breakpoints \( \mathcal{B} \).

- For each interval \([b_j, b'_j]\), \( A(k_{\text{max}}, b_j, b'_j) \) is computed.
QUERY1 indexes all $\binom{n}{2}$ intervals of breakpoints $B$.

For each interval $[b_j, b'_j]$, $A(k_{max}, b_j, b'_j)$ is computed.
Querying Breakpoints with Nested B-trees

- **breakpoint**

\[ \text{Left end-point index.} \]

\[ \text{Right end-point index.} \]

\[ \text{Time} \]

- **QUERY1** indexes all \( \binom{n}{2} \) intervals of breakpoints \( B \).

- For each interval \([b_j, b'_j]\), \( A(k_{max}, b_j, b'_j) \) is computed.
QUERY1 indexes all \( \binom{n}{2} \) intervals of breakpoints \( B \).

- For each interval \([b_j, b_j']\), \( A(k_{\text{max}}, b_j, b_j') \) is computed.
- At query time we probe first-level B-tree with \( t_1 \) to get \( B(t_1) \).
QUERY1 indexes all $\binom{n}{2}$ intervals of breakpoints $\mathcal{B}$.
- For each interval $[b_j, b'_j]$, $A(k_{\text{max}}, b_j, b'_j)$ is computed.
- At query time we probe first-level B-tree with $t_1$ to get $\mathcal{B}(t_1)$. 
Querying Breakpoints with Nested B-trees

- **QUERY1** indexes all $\binom{n}{2}$ intervals of breakpoints $B$.
  - For each interval $[b_j, b'_j]$, $A(k_{\text{max}}, b_j, b'_j)$ is computed.
- At query time we probe first-level B-tree with $t_1$ to get $B(t_1)$.
- We probe $B(t_1)$'s associated nested B-tree to get $B(t_2)$. 

\[ \text{Left end-point index.} \]
\[ \text{Right end-point index.} \]

Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang

Ranking Large Temporal Data
Querying Breakpoints with Nested B-trees

\[ B(t_1) \]  
\[ B(t_2) \]

- **QUERY1** indexes all \( \binom{n}{2} \) intervals of breakpoints \( B \).
  - For each interval \([b_j, b'_j]\), \( A(k_{max}, b_j, b'_j) \) is computed.

At query time we probe first-level B-tree with \( t_1 \) to get \( B(t_1) \).
- We probe \( B(t_1) \)’s associated nested B-tree to get \( B(t_2) \).
Querying Breakpoints with Nested B-trees

- **QUERY1** indexes all \( \binom{n}{2} \) intervals of breakpoints \( B \).
  - For each interval \([b_j, b'_j]\), \( A(k_{\text{max}}, b_j, b'_j) \) is computed.
- At query time we probe first-level B-tree with \( t_1 \) to get \( B(t_1) \).
- We probe \( B(t_1) \)'s associated nested B-tree to get \( B(t_2) \).
Querying Breakpoints with Nested B-trees

Objects ordered in descending order of $\sigma_i(.)$

- **QUERY1** indexes all $\binom{n}{2}$ intervals of breakpoints $B$.
  - For each interval $[b_j, b'_j]$, $A(k_{max}, b_j, b'_j)$ is computed.
- At query time we probe first-level B-tree with $t_1$ to get $B(t_1)$.
- We probe $B(t_1)$’s associated nested B-tree to get $B(t_2)$.

$\begin{array}{|c|c|}
\hline
o_i & \sigma_i(B(t_1), B(t_2)) \\
\hline
o_{i_1} & \sigma_{i_1}(B(t_1), B(t_2)) \\
\vdots & \vdots \\
o_{i_{k_{max}}} & \sigma_{i_{k_{max}}}(B(t_1), B(t_2)) \\
\hline
\end{array}$
Querying Breakpoints with Nested B-trees

Objects ordered in descending order of $\sigma_i(.)$

<table>
<thead>
<tr>
<th>$o_i$</th>
<th>$\sigma_i(B(t_1), B(t_2))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_{\ell_1}$</td>
<td>$\sigma_{\ell_1}(B(t_1), B(t_2))$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$o_{\ell_{k_{\max}}}$</td>
<td>$\sigma_{\ell_{k_{\max}}}(B(t_1), B(t_2))$</td>
</tr>
</tbody>
</table>

Take top-$k$

$\tilde{A}(k, t_1, t_2)$

- **QUERY1** indexes all $\binom{n}{2}$ intervals of breakpoints $B$.
  - For each interval $[b_j, b'_j]$, $A(k_{\max}, b_j, b'_j)$ is computed.
  - At query time we probe first-level B-tree with $t_1$ to get $B(t_1)$.
  - We probe $B(t_1)$’s associated nested B-tree to get $B(t_2)$.
  - The approximate answer $\tilde{A}(k, t_1, t_2)$ is returned.
We prove \textsc{Query1} has the following properties:

- Index size $O((1/\varepsilon)^2 k_{\text{max}}/B)$.
- Query cost $O(k/B + \log_B (1/\varepsilon))$.
- $(\varepsilon, 1)$-approximation.
We prove **QUERY1** has the following properties:
- Index size $O((1/\varepsilon)^2 k_{\text{max}}/B)$.
- Query cost $O(k/B + \log_B (1/\varepsilon))$.
- $(\varepsilon, 1)$-approximation.

**QUERY2** reduces space to $O((1/\varepsilon) k_{\text{max}}/B)$.
- $(\varepsilon, 2 \log(1/\varepsilon))$-approximation.
- Query cost $O(k \log(1/\varepsilon) \log_B k)$. 

Objects ordered in descending order of $\sigma_i(.)$
**QUERY2** indexes all dyadic intervals over the breakpoints $B$.

- The intervals represent the span of nodes in a balanced binary tree.
**QUERY2** indexes all dyadic intervals over the breakpoints $B$.

- The intervals represent the span of nodes in a balanced binary tree.
**Querying Breakpoints with Dyadic Intervals**

**QUERY2** indexes all dyadic intervals over the breakpoints $B$.

- The intervals represent the span of nodes in a balanced binary tree.
Querying Breakpoints with Dyadic Intervals

\[ \text{Query2 indexes all dyadic intervals over the breakpoints } B \]

- The intervals represent the span of nodes in a balanced binary tree.
Querying Breakpoints with Dyadic Intervals

Query2 indexes all dyadic intervals over the breakpoints $B$

- The intervals represent the span of nodes in a balanced binary tree.

Consider a query over $[t_1, t_2]$. 
Querying Breakpoints with Dyadic Intervals

- Query 2 indexes all dyadic intervals over the breakpoints $B$.
- The intervals represent the span of nodes in a balanced binary tree.
- Consider a query over $[t_1, t_2]$. 

---

Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang

Ranking Large Temporal Data
**Querying Breakpoints with Dyadic Intervals**

- **QUERY2** indexes all dyadic intervals over the breakpoints \( B \).
- The intervals represent the span of nodes in a balanced binary tree.
- Consider a query over \([t_1, t_2]\).
The intervals represent the span of nodes in a balanced binary tree. Consider a query over \([t_1, t_2]\).
Querying Breakpoints with Dyadic Intervals

- **QUERY2** indexes all dyadic intervals over the breakpoints $\mathcal{B}$.
  - The intervals represent the span of nodes in a balanced binary tree.
- Consider a query over $[t_1, t_2]$. 

The diagram illustrates the dyadic intervals and breakpoints, with the query range highlighted between $t_1$ and $t_2$. The intervals are shown as horizontal bars, and the breakpoints are marked with x's.
Querying Breakpoints with Dyadic Intervals

Query2 indexes all dyadic intervals over the breakpoints $B$

- The intervals represent the span of nodes in a balanced binary tree.
- Consider a query over $[t_1, t_2]$. 
**Querying Breakpoints with Dyadic Intervals**

- **QUERY2** indexes all dyadic intervals over the breakpoints $B$.
- The intervals represent the span of nodes in a balanced binary tree.
- Consider a query over $[t_1, t_2]$. 

\[ t_1 \quad \text{time} \quad t_2 \]
Querying Breakpoints with Dyadic Intervals

\[ A(k_{\text{max}}, b_2, b_3) : \sigma_{A(j)}(b_2, b_3) \forall j \in [1, \ldots, k_{\text{max}}] \]

- **QUERY2** indexes all dyadic intervals over the breakpoints \( B \)
  - The intervals represent the span of nodes in a balanced binary tree.
- Consider a query over \([t_1, t_2]\).
- At each dyadic interval \([b_i, b_j]\) we store \( A(k_{\text{max}}, b_i, b_j) \).
  - There are at most \( 2\log(1/\varepsilon) \) intervals and \( 2k\log(1/\varepsilon) \) candidates.
Querying Breakpoints with Dyadic Intervals

\[ A(k_{\text{max}}, b_2, b_3) : \sigma_{A(j)}(b_2, b_3), \forall j \in [1, \ldots, k_{\text{max}}] \]

- **QUERY2** indexes all dyadic intervals over the breakpoints \( B \)
  - The intervals represent the span of nodes in a balanced binary tree.
  - Consider a query over \([t_1, t_2]\).
  - At each dyadic interval \([b_i, b_j]\) we store \( A(k_{\text{max}}, b_i, b_j) \).
  - There are at most \( 2\log(1/\varepsilon) \) intervals and \( 2k\log(1/\varepsilon) \) candidates.

\[
\begin{array}{|c|c|}
\hline
o_i & \text{running sum } \sigma'_i \\
\hline
o_6 & \sigma'_6 \\
\hline
\vdots & \vdots \\
\hline
o_{19} & \sigma'_{19} \\
\hline
\end{array}
\]

at most \( 2k\log(1/\varepsilon) \) candidates!!!
We prove Query2 has the following properties:

- Index size $O\left(\frac{1}{\varepsilon} k_{\text{max}} / B\right)$.
- Query cost $O\left(k \log \frac{1}{\varepsilon} \log B k\right)$.
- $(\varepsilon, 2 \log(1/\varepsilon))$-approximation.
We consider the following algorithms:

- **Appx1-B**: \((\text{Query1, BreakPoints1})\)
- **Appx2-B**: \((\text{Query2, BreakPoints1})\)
- **Appx1**: \((\text{Query1, BreakPoints2})\)
- **Appx2**: \((\text{Query2, BreakPoints2})\)
- **Appx2+**: \((\text{Query2, BreakPoints2})\) and Discovers candidates’ exact aggregate score using B-tree from \text{Exact2} (B-tree forest).
We consider the following algorithms:

- **Appx1-B**: \((\text{Query1}, \text{BreakPoints1})\)
- **Appx2-B**: \((\text{Query2}, \text{BreakPoints1})\)
- **Appx1**: \((\text{Query1}, \text{BreakPoints2})\)
- **Appx2**: \((\text{Query2}, \text{BreakPoints2})\)
- **Appx2+**: \((\text{Query2}, \text{BreakPoints2})\) and Discovers candidates’ exact aggregate score using B-tree from \text{Exact2} (B-tree forest).
Our algorithms are designed to efficiently handle I/Os.

- All algorithms are implemented in C++ using TPIE.
Our algorithms are designed to efficiently handle I/Os.
  - All algorithms are implemented in C++ using TPIE.
All experiments performed on Linux machine with:
  - Intel Core i7-2600 3.4GHz CPU
  - 8GB of memory
  - 1TB hard drive

We use two real large datasets:
- Temp is a temperature dataset from the MesoWest Project. It contains measurements from Jan 1997 to Oct 2011. There are \( m = 145,628 \) objects with average \( n_{avg} = 17,833 \).
- Meme is obtained from the Memetracker Project. It tracks the frequency of popular quotes over time. There are \( m = 1,500 \) million objects with \( n_{avg} = 67 \).
Experiments: Setup

- Our algorithms are designed to efficiently handle I/Os.
  - All algorithms are implemented in C++ using TPIE.
- All experiments performed on Linux machine with:
  - Intel Core i7-2600 3.4GHz CPU
  - 8GB of memory
  - 1TB hard drive
- We use two real large datasets:
  - *Temp* is a temperature dataset from the MesoWest Project.
    - there are $m = 145,628$ objects with average $n_{avg} = 17,833$.
  - *Meme* is obtained from the Memetracker Project.
    - tracks the frequency of popular quotes over time.
    - there are $m = 1.5$ million objects with $n_{avg} = 67$. 
## Experiments: Default Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td></td>
<td>$Temp$</td>
</tr>
<tr>
<td>number of objects</td>
<td>$m$</td>
<td>50,000</td>
</tr>
<tr>
<td>average object line segments</td>
<td>$n_{avg}$</td>
<td>1,000</td>
</tr>
<tr>
<td>max top-$k$ value</td>
<td>$k_{max}$</td>
<td>200</td>
</tr>
<tr>
<td>top-$k$ value</td>
<td>$k$</td>
<td>50</td>
</tr>
<tr>
<td>number of breakpoints</td>
<td>$r = (1/\varepsilon)$</td>
<td>500</td>
</tr>
<tr>
<td>query interval size</td>
<td>$(t_2 - t_1)$</td>
<td>20% $T$</td>
</tr>
<tr>
<td>TPIE disk block size</td>
<td></td>
<td>4KB</td>
</tr>
</tbody>
</table>

Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang

Ranking Large Temporal Data
## Experiment: Index size.

<table>
<thead>
<tr>
<th>Objects $m \times 10^3$</th>
<th>Index size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exact1</td>
</tr>
<tr>
<td></td>
<td>Exact2</td>
</tr>
<tr>
<td></td>
<td>Exact3</td>
</tr>
<tr>
<td></td>
<td>Appx1</td>
</tr>
<tr>
<td></td>
<td>Appx2</td>
</tr>
<tr>
<td></td>
<td>Appx2+</td>
</tr>
</tbody>
</table>

**Diagram:**
- **Exact1**
- **Exact2**
- **Exact3**
- **Appx1**
- **Appx2**
- **Appx2+**

**Legend:**
- Red triangle
- Pink circle
- Black square
- Green asterisk
- Blue star
- Cyan star

**Objects $m$ ($\times 10^3$):**
- 10
- 30
- 50
- 100
- 145

**Index size (bytes):**
- $10^4$
- $10^6$
- $10^8$
- $10^{10}$
- $10^{12}$
Experiment: Build time.

Objects $m \times 10^3$

Build time (seconds)

EXACT1  EXACT2  EXACT3
APPX1   APPX2   APPX2+

Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang
Ranking Large Temporal Data
<table>
<thead>
<tr>
<th>Objects $m$ ($\times 10^3$)</th>
<th>I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>145</td>
<td>145</td>
</tr>
</tbody>
</table>

**Experiment: Query I/Os.**

The graph illustrates the relationship between the number of objects $m$ and the number of I/Os for different query methods. The methods include:

- **Exact1**
- **Exact2**
- **Exact3**
- **Appx1**
- **Appx2**
- **Appx2+**

The y-axis represents the number of I/Os on a logarithmic scale, ranging from $10^0$ to $10^8$. The x-axis represents the number of objects $m$ ($\times 10^3$), ranging from 10 to 145.
Experiment: Query time.

Objects $m \times 10^3$ vs. Time (seconds)
Experiment: Precision/Recall.

\[(t_2 - t_1) \text{ as } \% \text{ of } T\]
Experiment: Ratio.

\[ (t_2 - t_1) \text{ as } \% \text{ of } T \]

**Approximation ratio**

- Appx1
- Appx2
- Appx2+

Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang

Ranking Large Temporal Data
We studied ranking large temporal data using aggregate scores over a query interval.
Conclusions

- We studied ranking large temporal data using aggregate scores over a query interval.
- Our most efficient exact technique $\text{Exact}3$ is more efficient than baseline solutions.
  - Approximations offer even more improvements.

Future work includes ranking with holistic aggregations and extending to distributed settings.
Conclusions

- We studied ranking large temporal data using aggregate scores over a query interval.
- Our most efficient exact technique $\texttt{EXACT3}$ is more efficient than baseline solutions.
  - Approximations offer even more improvements.
- Future work includes ranking with holistic aggregations and extending to distributed settings.
Thank You

Q and A
Baseline Solution

Computing $\sigma(g_3(t_1, t_2))$

1. Initialize sum $s_3 = 0$ for object $o_3$
Baseline Solution

Computing $\sigma(g_3(t_1, t_2))$

1. Initialize sum $s_3 = 0$ for object $o_3$
2. For each segment $\ell$ of $g_3$ defined by $(t_{3,j}, v_{3,j}), (t_{3,j+1}, v_{3,j+1})$
Baseline Solution

Computing $\sigma(g_3(t_1, t_2))$

1. Initialize sum $s_3 = 0$ for object $o_3$
2. For each segment $\ell$ of $g_3$ defined by $(t_{3,j}, v_{3,j}), (t_{3,j+1}, v_{3,j+1})$
   - Define $I = [t_1, t_2] \cap [t_{3,j}, t_{3,j+1}]$
Baseline Solution

Computing $\sigma(g_3(t_1, t_2))$

1. Initialize sum $s_3 = 0$ for object $o_3$
2. For each segment $\ell$ of $g_3$ defined by $(t_{3,j}, v_{3,j}), (t_{3,j+1}, v_{3,j+1})$
   - Define $I = [t_1, t_2] \cap [t_{3,j}, t_{3,j+1}]$
   - Update $s_3 = s_3 + \sigma_3(I)$
Baseline Solution

Computing $\sigma(g_3(t_1, t_2))$

1. Initialize sum $s_3 = 0$ for object $o_3$
2. For each segment $\ell$ of $g_3$ defined by $(t_{3,j}, v_{3,j}), (t_{3,j+1}, v_{3,j+1})$
   - Define $\mathcal{I} = [t_1, t_2] \cap [t_{3,j}, t_{3,j+1}]$
   - Update $s_3 = s_3 + \sigma_3(\mathcal{I})$
Baseline Solution

Computing $\sigma(g_3(t_1, t_2))$

1. Initialize sum $s_3 = 0$ for object $o_3$
2. For each segment $\ell$ of $g_3$ defined by $(t_{3,j}, v_{3,j}), (t_{3,j+1}, v_{3,j+1})$
   - Define $\mathcal{I} = [t_1, t_2] \cap [t_{3,j}, t_{3,j+1}]$
   - Update $s_3 = s_3 + \sigma_3(\mathcal{I})$
Baseline Solution

Computing $A(k, t_1, t_2)$

- Compute $s_i$ for all objects $i \in [1, m]$.
  - Insert $s_i$'s into priority queue of size $k$ to get $A(k, t_1, t_2)$.
Computing $A(k, t_1, t_2)$

- Compute $s_i$ for all objects $i \in [1, m]$.
  - Insert $s_i$’s into priority queue of size $k$ to get $A(k, t_1, t_2)$.
- Naive cost: $O(N + m\log k)$
For each line segment $\ell = \{(t_{i,j}, v_{i,j}), (t_{i,j+1}, v_{i,j+1})\}$
- Index left end-point $t_{i,j}$ in B-tree.
- The value associated with $t_{i,j}$ is $\ell$. 

Score

$\downarrow$

$t_1$

$\downarrow$

$t_2$

$\downarrow$

$O_3$

Time
For each line segment $\mathbf{\ell} = \{(t_{i,j}, v_{i,j}), (t_{i,j+1}, v_{i,j+1})\}$

- Index left end-point $t_{i,j}$ in B-tree.
- The value associated with $t_{i,j}$ is $\mathbf{\ell}$.

Query cost: $O(\log_B N + \frac{\sum_{i=1}^{m} q_i}{B} + (m/B)\log_B k)$

- $q_i =$ number of $\mathbf{\ell}$ overlapping $[t_1, t_2]$

We denote this query $\text{EXACT1}$. 
Improved Solution using Prefix Sums and B-tree Forest

Let $I_i, \ell = [t_{i,0}, 0, t_{i,\ell}]$ for $\ell = 1, \ldots, n_i$ and compute $\sigma_{i}(I_i, \ell)$

- $g_i = \bigcup g_{i,j}$
- $g_{i,j}$ is defined by $((t_{i,j-1}, v_{i,j-1}), (t_{i,j}, v_{i,j}))$ for $j \in \{1, \ldots, n_i\}$
Improved Solution using Prefix Sums and B-tree Forest

\[ g_i = \bigcup g_{i,j} \]
\[ g_{i,j} \text{ is defined by } ((t_{i,j-1}, v_{i,j-1}), (t_{i,j}, v_{i,j})) \text{ for } j \in \{1, \ldots, n_i\} \]
\[ \text{Let } I_{i,\ell} = [t_{i,0}, t_{i,\ell}] \text{ for } \ell = 1, \ldots, n_i \text{ and compute } \sigma_i(I_{i,\ell}) \]
Improved Solution using Prefix Sums and B-tree Forest

\[ g_i = \bigcup g_{i,j} \]

- \( g_{i,j} \) is defined by \((t_{i,j-1}, v_{i,j-1}), (t_{i,j}, v_{i,j})\) for \( j \in \{1, \ldots, n_i\} \)

- Let \( l_{i,\ell} = [t_{i,0}, t_{i,\ell}] \) for \( \ell = 1, \ldots, n_i \) and compute \( \sigma_i(l_{i,\ell}) \)
Improved Solution using Prefix Sums and B-tree Forest

$g_i = \bigcup g_{i,j}$

$g_{i,j}$ is defined by $((t_{i,j-1}, v_{i,j-1}), (t_{i,j}, v_{i,j}))$ for $j \in \{1, \ldots, n_i\}$

Let $l_{i,\ell} = [t_{i,0}, t_{i,\ell}]$ for $\ell = 1, \ldots, n_i$ and compute $\sigma_i(l_{i,\ell})$
Let $t_{i,L} = \text{successor}(t_{i,1})$ and $t_{i,R} = \text{successor}(t_{i,2})$
Let \( t_{i,L} = \text{successor}(t_{i,1}) \) and \( t_{i,R} = \text{successor}(t_{i,2}) \)

\[
\sigma_i(t_1, t_2) = \sigma_i(l_{i,R}) - \sigma_i(l_{i,L}) - \sigma_i(t_2, t_{i,R}) + \sigma_i(t_1, t_{i,L})
\]
Let $t_{i,L} = \text{successor}(t_{i,1})$ and $t_{i,R} = \text{successor}(t_{i,2})$

$\sigma_i(t_1, t_2) = \sigma_i(l_{i,R}) - \sigma_i(l_{i,L}) - \sigma_i(t_2, t_{i,R}) + \sigma_i(t_1, t_{i,L})$
Improved Solution using Prefix Sums and B-tree Forest

Let $t_{i,L} = \text{successor}(t_{i,1})$ and $t_{i,R} = \text{successor}(t_{i,2})$

$$\sigma_i(t_1, t_2) = \sigma_i(l_{i,R}) - \sigma_i(l_{i,L}) - \sigma_i(t_2, t_{i,R}) + \sigma_i(t_1, t_{i,L})$$
Let \( t_{i,L} = \text{successor}(t_{i,1}) \) and \( t_{i,R} = \text{successor}(t_{i,2}) \).

\[
\sigma_i(t_1, t_2) = \sigma_i(l_{i,R}) - \sigma_i(l_{i,L}) - \sigma_i(t_2, t_{i,R}) + \sigma_i(t_1, t_{i,L})
\]
Let $t_{i,L} = \text{successor}(t_{i,1})$ and $t_{i,R} = \text{successor}(t_{i,2})$

Let $\sigma_i(t_1, t_2) = \sigma_i(l_{i,R}) - \sigma_i(l_{i,L}) - \sigma_i(t_2, t_{i,R}) + \sigma_i(t_1, t_{i,L})$

Use a B-tree forest to index $(t_{3,\ell}, (g_{i,\ell}, \sigma_i(l_{i,\ell}))$

- Each $o_i$ indexed in a separate B-tree
- Query cost is $O(\sum_{i=1}^{m} \log_B n_i + (m/B)\log_B k)$

We denote this query $\text{EXACT2}$. 

Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang

Ranking Large Temporal Data
Improved Solution using Prefix Sums and B-tree Forest

- Our B-tree forest solution requires $m$ B-trees.
- Query time improves from baseline.
- Opening/Closing $m$ B-trees expensive for large $m$. 

Jeffrey Jestes, Jeff M. Phillips, Feifei Li, Mingwang Tang

Ranking Large Temporal Data
Improving Solution using Prefix Sums and B-tree Forest

- Our B-tree forest solution requires $m$ B-trees.
  - Query time improves from baseline.
  - Opening/Closing $m$ B-trees expensive for large $m$.
- We show how to solve a query using a single interval tree.