Ranked Subsequence Matching in Time-Series Databases

Wook-Shin Han (Kyungpook National University, Korea)
Jinsoo Lee (Kyungpook National University, Korea)
Yang-Sae Moon (Kangwon National University, Korea)
Haifeng Jiang (Google Inc., USA)

Contents

- Introduction
- Overview of DTW and Existing Lower Bounds
- Basic Ranked Subsequence Matching Algorithms
- Minimum Distance Matching Window Pair (MDMWP) and mdmwp-Distance Based Pruning
- Deferred Group Subsequence Retrieval
- Performance Evaluation
- Conclusions

Time-Series Databases [AFS93, FRM94, MWL01]

- Time-series data
  - Sequences of values sampled at a fixed time interval
  - Examples: music data, stock prices and network traffic data
- Time-series databases
  - Data sequence: time-series data stored in a database
  - Query sequence: time-series data given by a user for similarity search

Similarity Metric

- Measuring similarity as the distance between a data sequence and a given query sequence
- We use the dynamic time warping (DTW) distance [BC96, SC78]
  - One of most robust similarity measures
  - Widely used for various applications such as query by humming [ZS03], image searching [BCP05], and speech recognition [RJ93]

Motivation

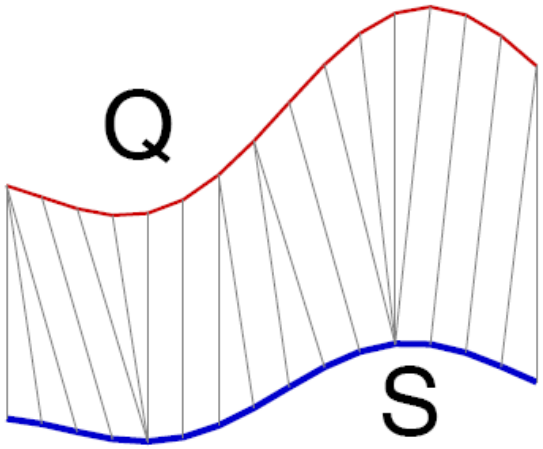
- Ranked subsequence matching under DTW
  - finds top-k similar subsequences to a query sequence from data sequences under DTW
- All the existing methods have been developed only for either ***whole*** matching or ***range subsequence*** matching

Category	Range query	k-NN query
Whole matching	[1, 6, 11, 12, 15, 20, 31]	[6, 11, 12]
Subsequence matching	[7, 16, 17, 18, 19, 28]	×

Contributions

- Propose the first and foremost approach for ranked subsequence matching
- Propose the concept of *minimum-distance matching-window pair* and pruning with MDMWP distance
- Propose deferred group subsequence retrieval along with another lower bound, *window-group distance*
- Show efficiency of the proposed methods using many real and synthetic datasets

Review of DTW

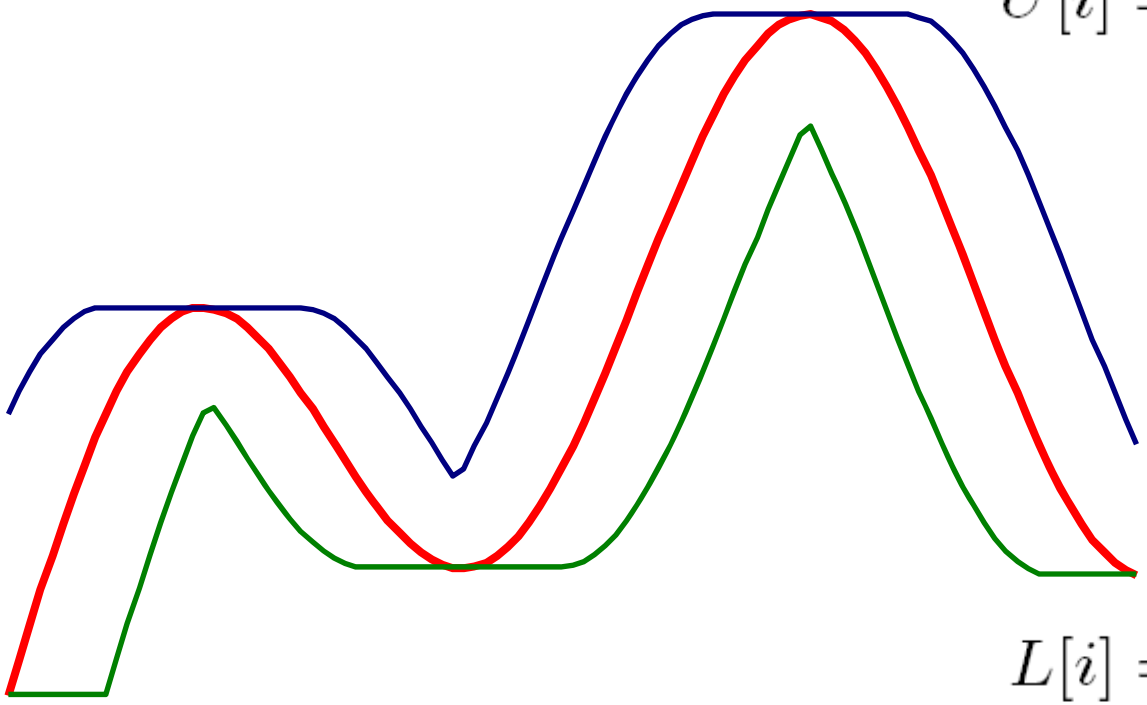


(a) DTW comparison.

$$DTW(\langle\rangle, \langle\rangle) = 0$$
$$DTW(S, \langle\rangle) = DTW(\langle\rangle, Q) = \infty$$
$$DTW(S,Q) = \sqrt[p]{|S[1] - Q[1]|^p + \min \begin{cases} DTW(Rest(S), Rest(Q)) \\ DTW(Rest(S), Q) \\ DTW(S, Rest(Q)) \end{cases}}$$

Query Envelope [Keo02, ZS03]

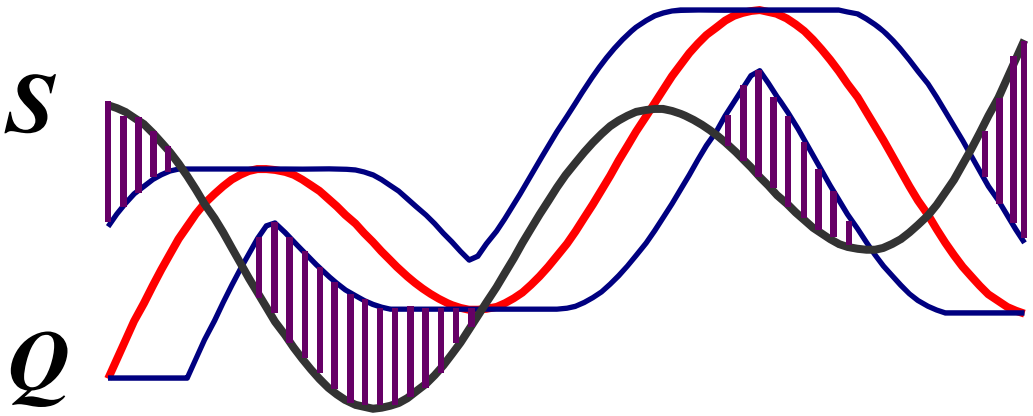
$$U[i] = \max_{-\rho \le r \le \rho} (Q[i+r])$$



$$L[i] = \min_{-\rho \le r \le \rho} (Q[i+r])$$

LB_Keogh [Keo02]

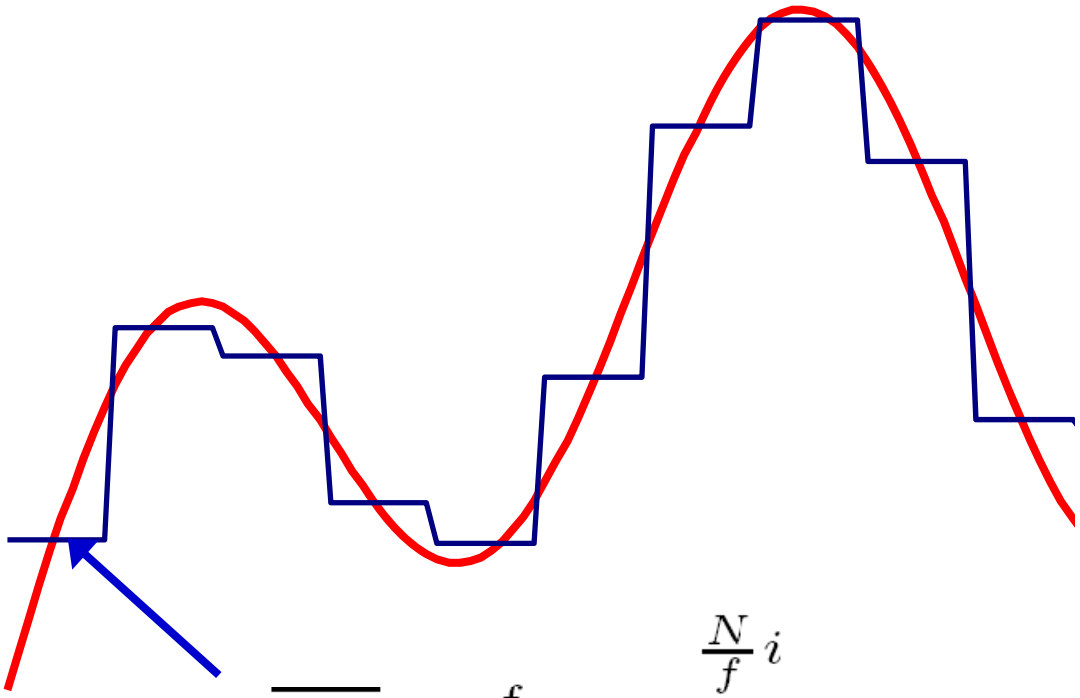
- Distance between a query envelope E(Q) and a data sequence S
- Lower bounding distance under DTW at the sequence level



$$LB\_Keogh(\mathbb{E}(Q), S) = \sqrt[p]{\sum_{i=1}^{N} \begin{cases} |S[i] - U[i]|^p & \text{if } S[i] > U[i] \\ |S[i] - L[i]|^p & \text{if } S[i] < L[i] \\ 0 & \text{otherwise} \end{cases}}$$

Piecewise Aggregate Approximation (PAA) [YF00, Keo02]

- Dimension reduction: N dimension → f dimension

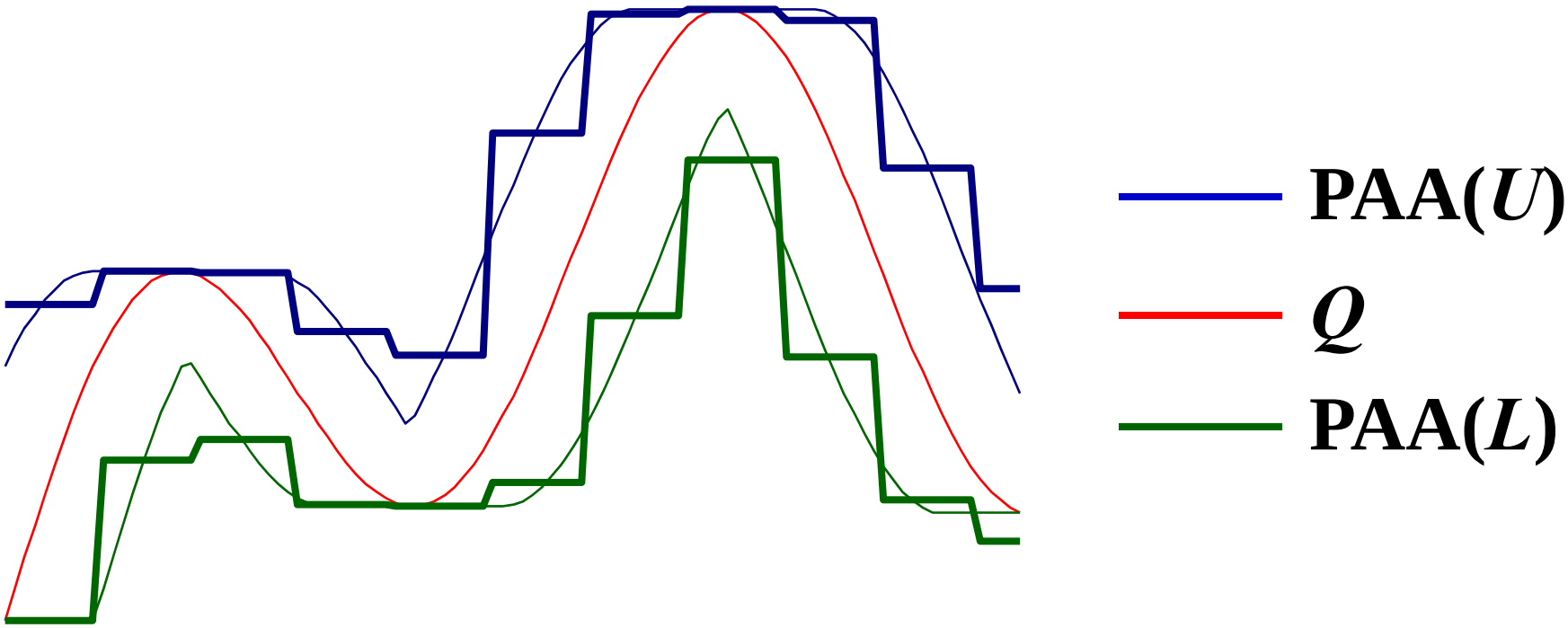


$\overline{S}$ = (PAA(S))

S

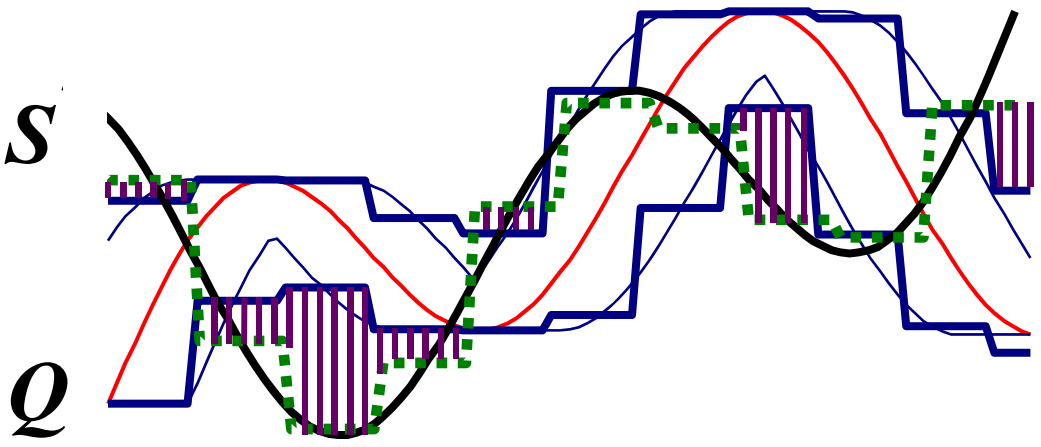
$$\overline{S[i]} = \frac{f}{N} \sum_{j=\frac{N}{f}(i-1)+1}^{\frac{N}{f}i} S[j]$$

PAA(ENV(Q))



LB_PAA [ZS03]

- Distance between the PAA of the query envelope $P(\mathrm{E}(Q))$ and the PAA of the data sequence $P(S)$
- Lower bounding distance under DTW at the index level



$$LB\_PAA(\mathcal{P}(\mathbb{E}(Q)),\mathcal{P}(S)) = \sqrt[p]{\sum_{i=1}^{f} \frac{N}{f} \begin{cases} |\overline{S[i]} - \overline{U[i]}|^p & \text{if } \overline{S[i]} > \overline{U[i]} \\ |\overline{S[i]} - \overline{L[i]}|^p & \text{if } \overline{S[i]} < \overline{L[i]} \\ 0 & \text{otherwise} \end{cases}}$$

Lower Boundness of the Two Distances for Whole Matching [Keo02, ZS03]

Lemma 1. *Given two subsequence Q and S of the same length and a warping width ρ, the following equation holds*:

$$DTW_\rho(Q,S) \geq LB\_Keogh(\mathbb{E}(Q),S)$$
$$\geq LB\_PAA(\mathcal{P}(\mathbb{E}(Q)),\mathcal{P}(S))$$

We can exploit these lower bounds whenever pruning is possible at the index level or at the sequence level.

Related Work

- Range Whole Matching [AFC93]
- Ranked Whole Matching
  - Under Euclidean Distance [Keo01, Cha03]
  - Under DTW [Keo02]
- Range Subsequence Matching
  - Dividing a data sequence into sliding windows, a query sequence into disjoint windows [FRM94]
  - Dual Match: dual approach of FRM [MWL01]
  - General Match [MWH02]

Two Basic Algorithms for Ranked Subsequence Matching

- *DualMatchTopK*
  - applies the window construction mechanism of DualMatch [MWL01] to the *ranked whole* matching algorithm [Cha03, Keo02]
- *RangeTopK*
  - Obtains top-k entries at the index level using *DualMatchTopK* and an upper bound ε by retrieving the corresponding data subsequences for the entries
  - and then finds top-*k* subsequences using the *range* subsequence matching algorithm with ε

Algorithm 1 *DualMatchTopK*

Input: Q, k, ρ
Output: k-nearest data subsequences for Q

1: **Variable** queue : Minimum priority queue
2: **Variable** results : List
3: **Variable** $\delta_{cur} \leftarrow \infty$; /*$\delta_{cur}$ is the DTW$_\rho$ distance between the Q and the top k-th subsequence obtained so far*/
4: **for each** i-th sliding window $\mathbb{E}(q_i)$ in $\mathbb{E}(Q)$ **do**
5: queue.Push($\langle$*RootNode*,MINDIST($\mathcal{P}(\mathbb{E}(q_i))$,*RootNode*),$i$,$-1$,$-1\rangle$);
6: **while not** queue.IsEmpty() **do**
7: $\langle obj, d, j, sid, off\rangle \leftarrow$ queue.Pop();
8: **if** *obj* is a subsequence **then**
9: add *obj* to *results*;
10: **if** $|results| = k$ **then**
11: **return** *results*;
12: **else if** *obj* is a leaf entry **then**
13: $soff \leftarrow off - j + 1$; /*start offset*/
14: $eoff \leftarrow soff + Len(Q) - 1$; /*end offset*/
15: *SequenceRetrieval*(queue, *soff*, *eoff*, *sid*, δ_{cur}, ρ);
16: **else if** *obj* is a leaf node *LN* **then**
17: **for each** leaf entry E $\langle$Point P, SeqID *sid2*, offset *off2*$\rangle$ in *LN* **do**
18: **if** LB_PAA($\mathcal{P}(\mathbb{E}(q_j))$,$P$) $< \delta_{cur}$ **then**
19: queue.Push($\langle E$, LB_PAA($\mathcal{P}(\mathbb{E}(q_j))$,$P$), j, *sid2*, *off2*$\rangle$);
20: **else**
21: **for each** child node E $\langle$MBR M, Child $ptr\rangle$ in *obj* **do**
22: **if** MINDIST($\mathcal{P}(\mathbb{E}(q_j))$,$M$) $< \delta_{cur}$ **then**
23: queue.Push($\langle E$, MINDIST($\mathcal{P}(\mathbb{E}(q_j))$,$M$), j, -1, $-1\rangle$);

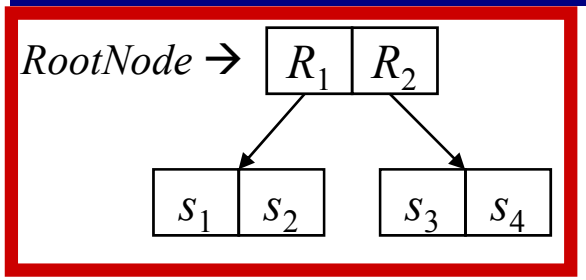
Pruning at the index level

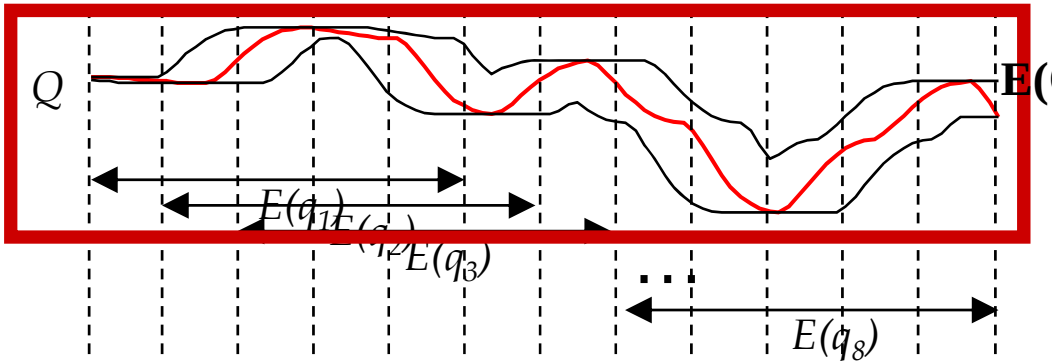
Function 2 *SequenceRetrieval*

Input: queue, *soff*, *eoff*, *sid*, δ_{cur}, ρ

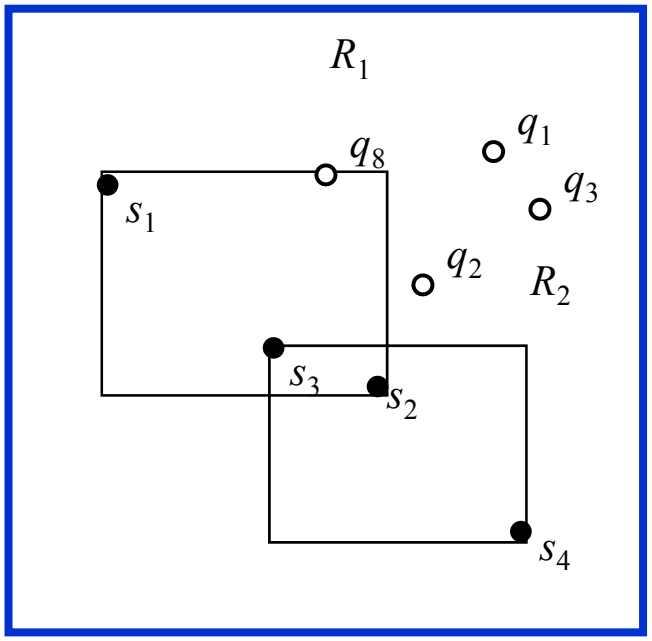
1: **if** $Sub(= S_{sid}[soff : eoff])$ is not yet retrieved **then**
2: retrieve Sub from $S_{sid}[soff : eoff]$;
3: **if** LB_Keogh($\mathbb{E}(Q)$,Sub) $< \delta_{cur}$ **then**
4: **if** DTW$_\rho$(Q,Sub) $< \delta_{cur}$ **then**
5: queue.Push(Sub, DTW$_\rho$(Q,Sub),-1, *sid*,*soff*);
6: update δ_{cur};

Pruning at the sequence level





RootNode

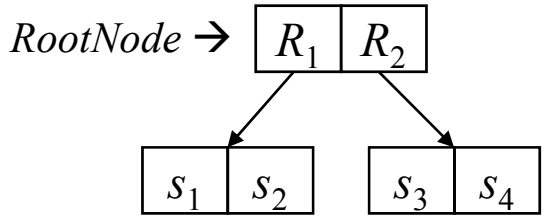


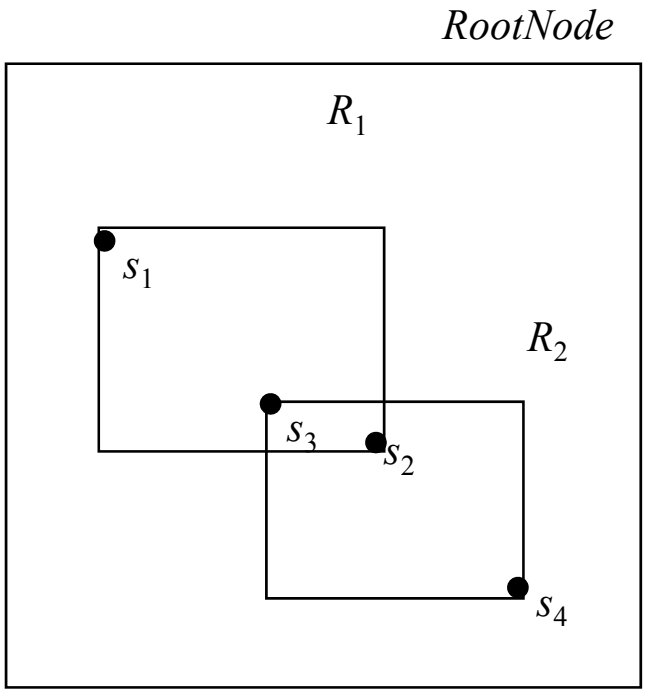
Distance

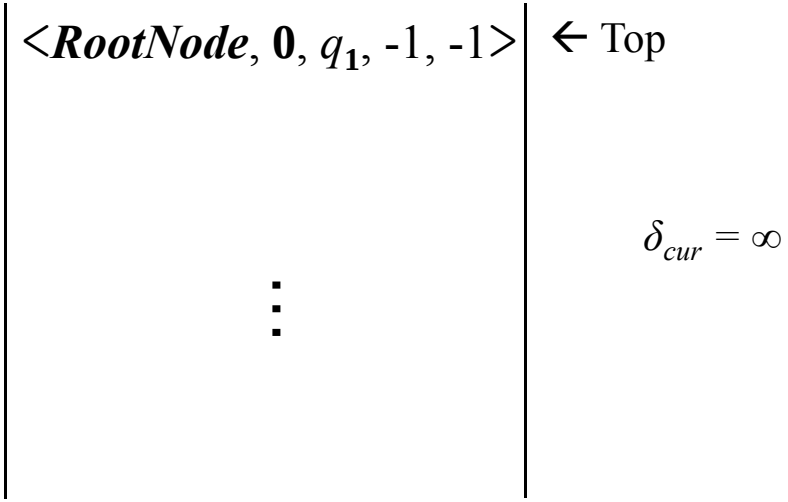
<***RootNode***, **0**, q_1, -1, -1> ← Top
<***RootNode***, **0**, q_2, -1, -1>
<***RootNode***, **0**, q_3, -1, -1>
⋮
<***RootNode***, **0**, q_8, -1, -1>

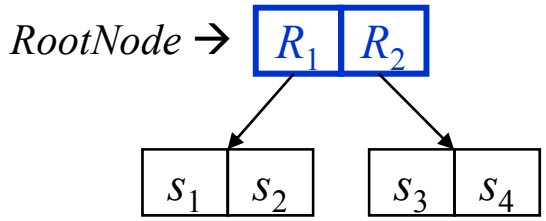
$\delta_{cur} = \infty$

Priority Queue



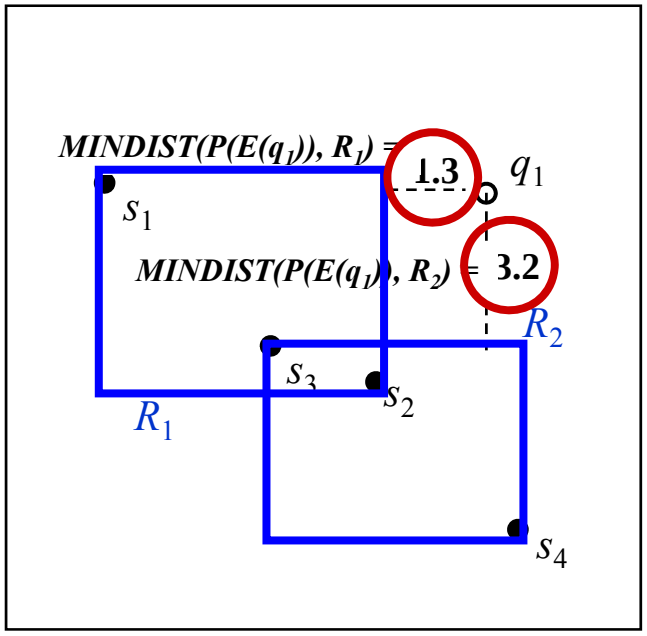


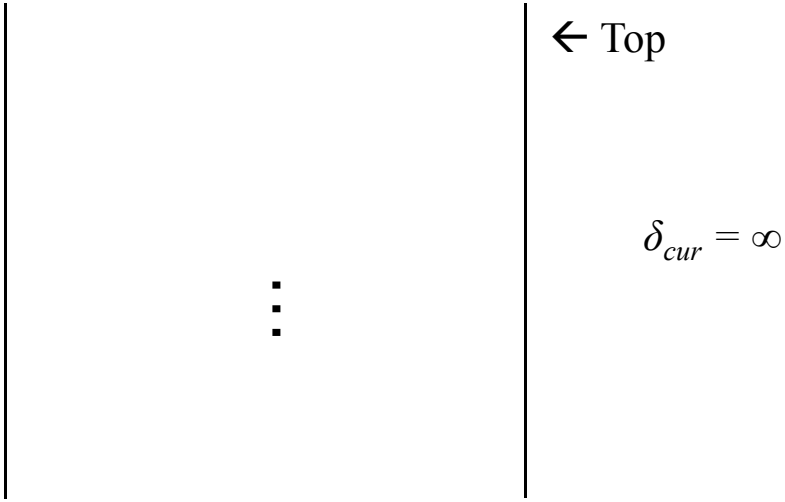


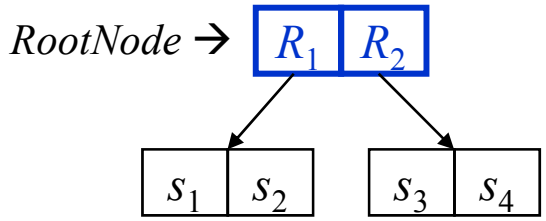


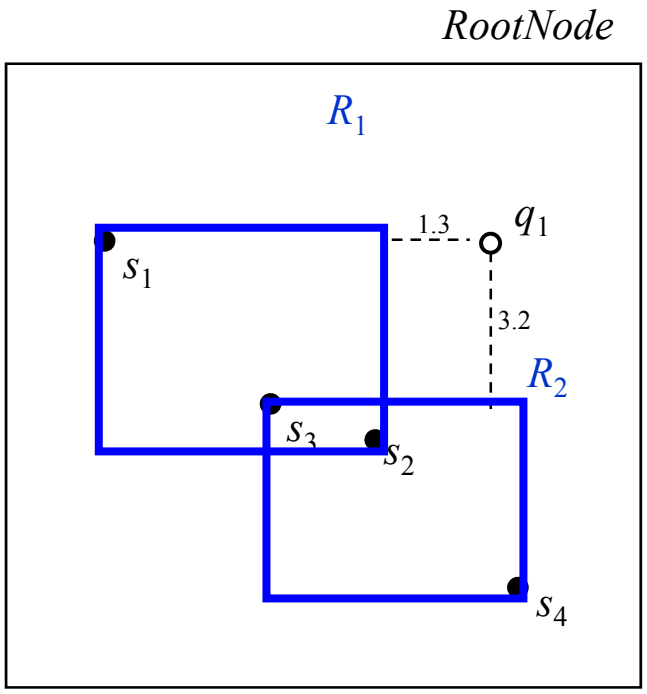
<q_1, RootNode, 0, q_1, -1, -1>

RootNode









$\vdots$

$\langle \boldsymbol{R_1}, \mathbf{1.3}, q_1, -1, -1 \rangle$

$\vdots$

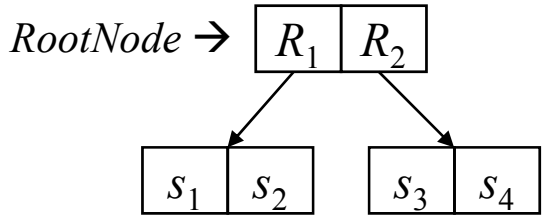
$\langle \boldsymbol{R_2}, \mathbf{3.2}, q_1, -1, -1 \rangle$

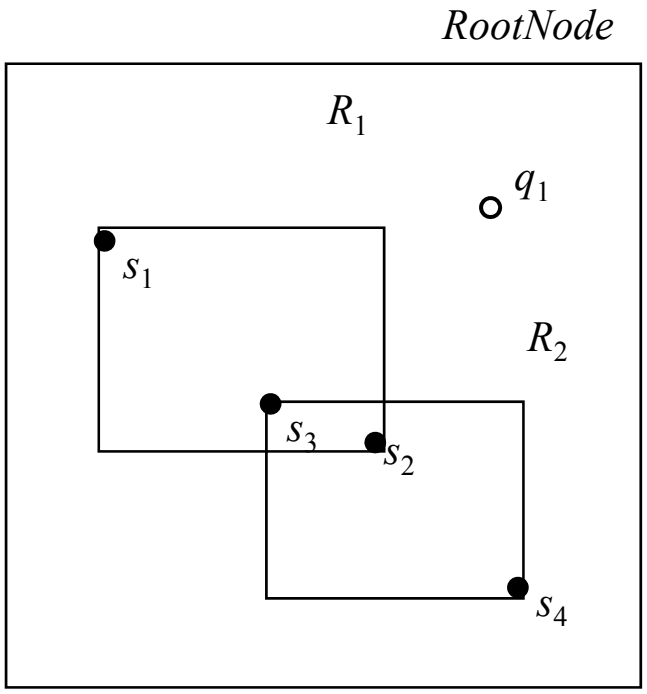
$\vdots$

Priority Queue

← Top

$\delta_{cur} = \infty$





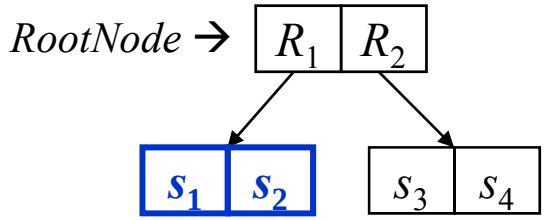
<$\boldsymbol{R_1}$, **1.3**, q_1, -1, -1>

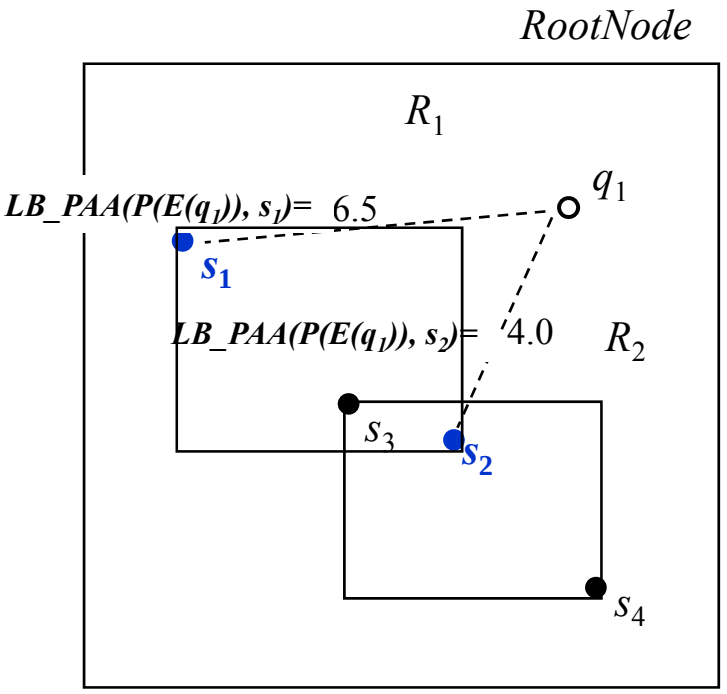
⋮

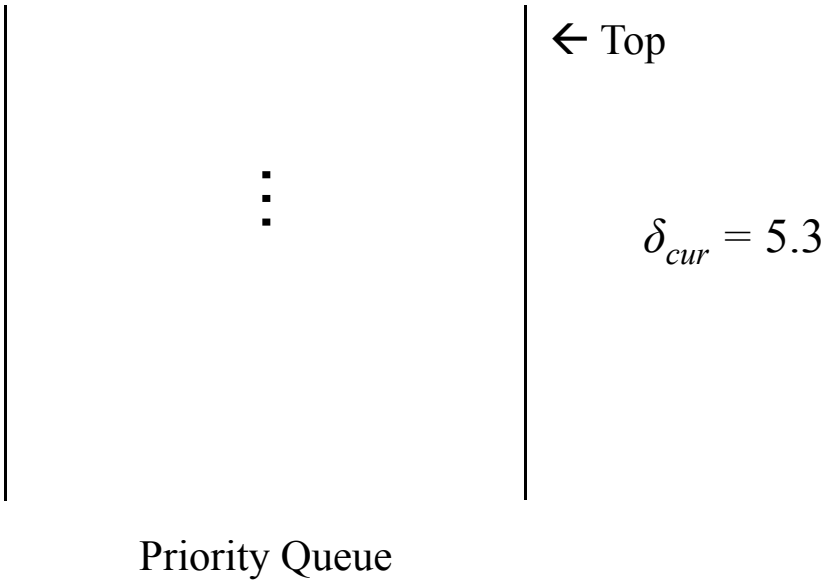
Priority Queue

← Top

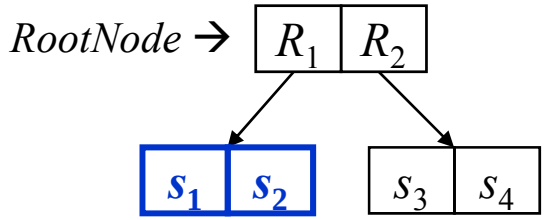
$\delta_{cur} = 5.3$



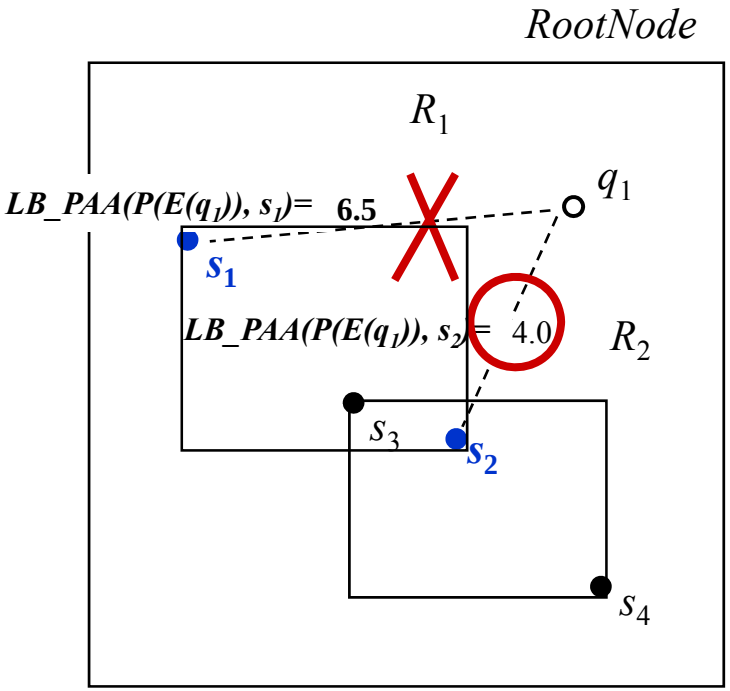


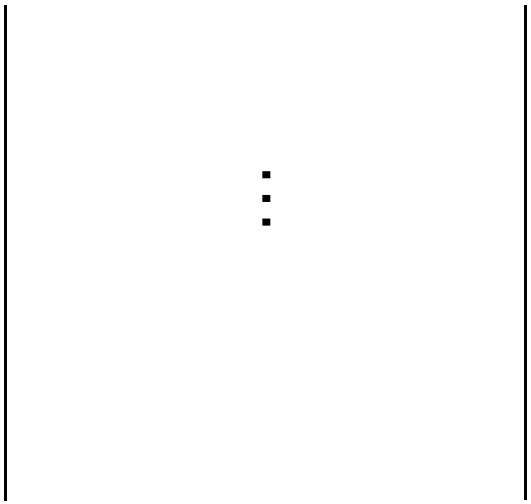


$<\boldsymbol{R_1}, \mathbf{1.3}, q_1, -1, -1>$



<R_1, **1.3**, q_1, -1, -1>



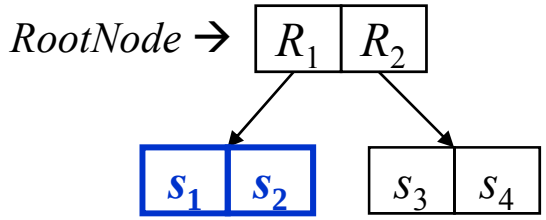


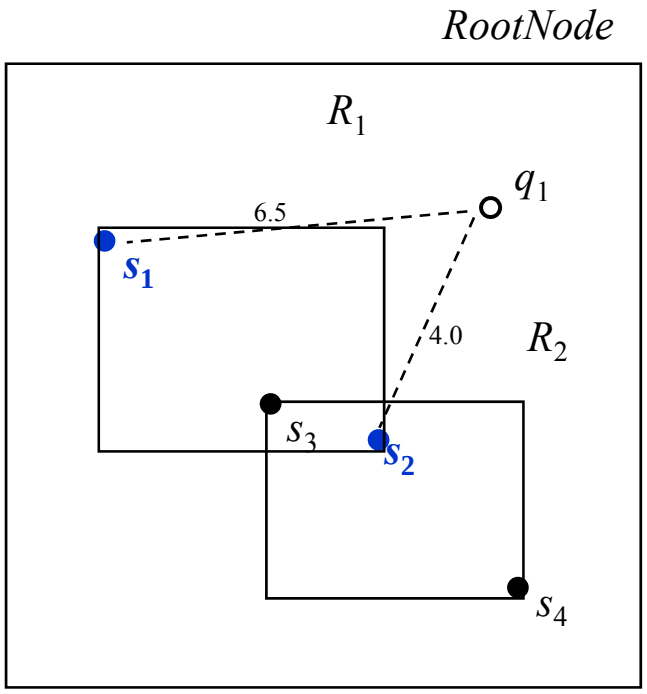
← Top

$\delta_{cur} = 5.3$

since 6.5 > δ_{cur}, s_1 is pruned

Priority Queue





$\vdots$

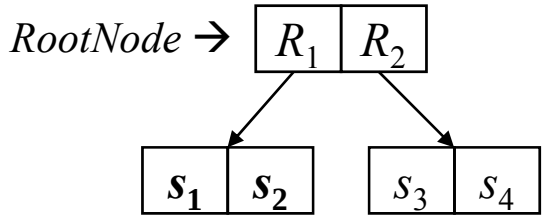
$\langle s_2, \mathbf{4.0}, q_1, \mathbf{3}, \mathbf{8} \rangle$

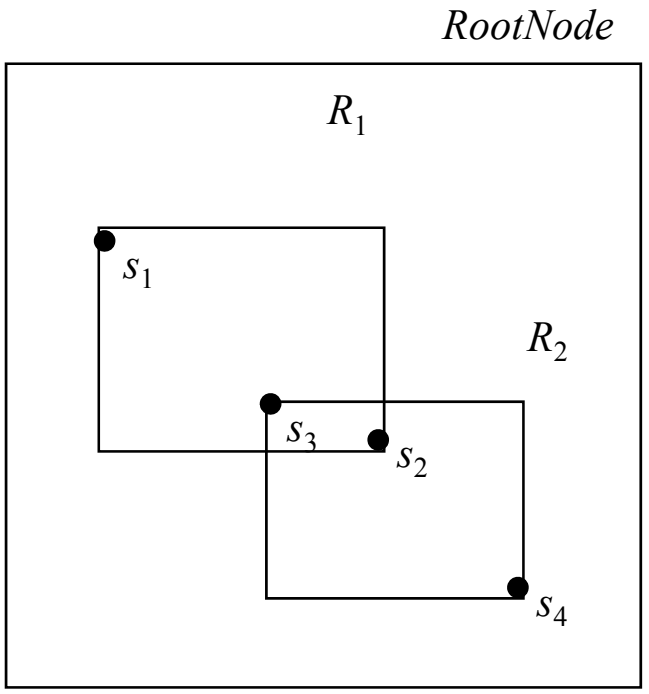
$\vdots$

← Top

$\delta_{cur} = 5.3$

Priority Queue





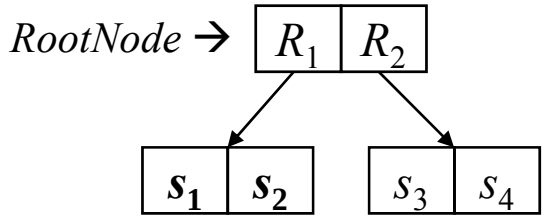
$<s_2, \mathbf{4.0}, q_1, \mathbf{3}, \mathbf{8}>$

⋮

← Top

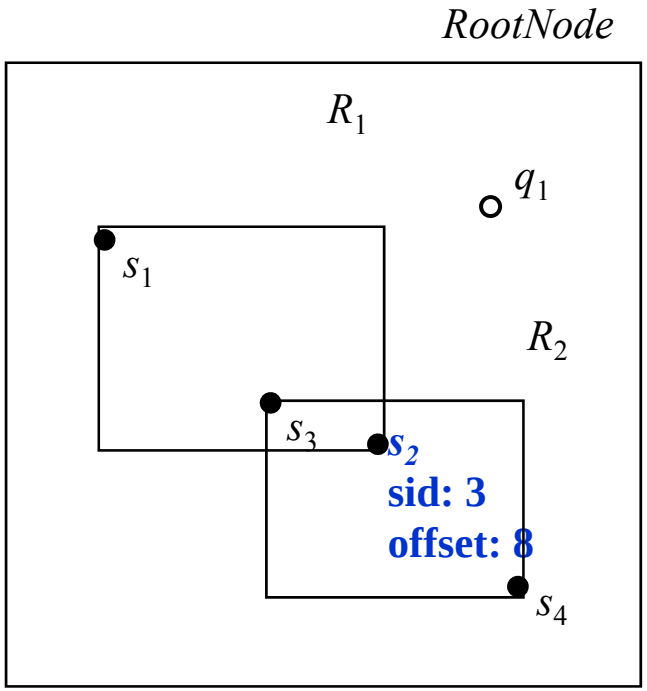
$\delta_{cur} = 5.3$

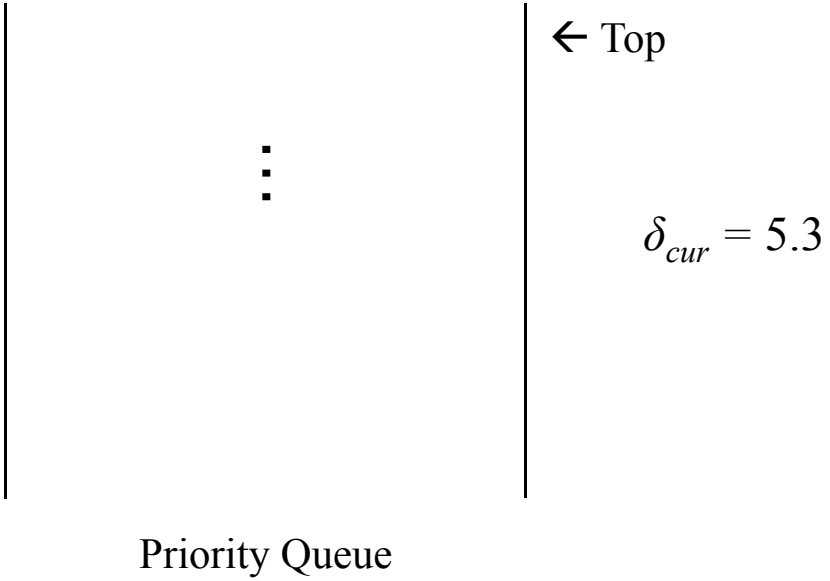
Priority Queue

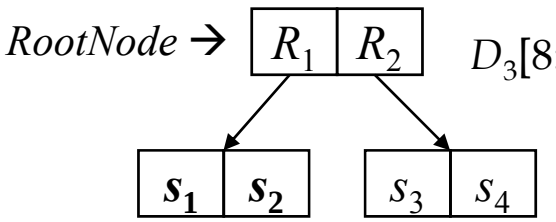


sid,offset

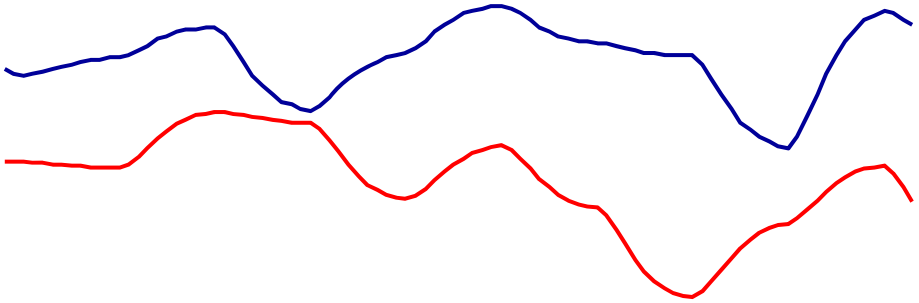
$\langle s_2, \mathbf{4.0}, q_1, \mathbf{3}, \mathbf{8} \rangle$



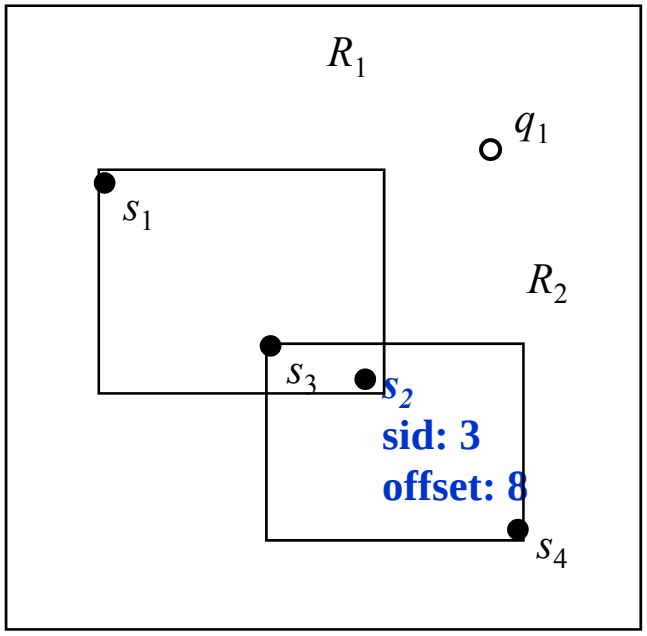




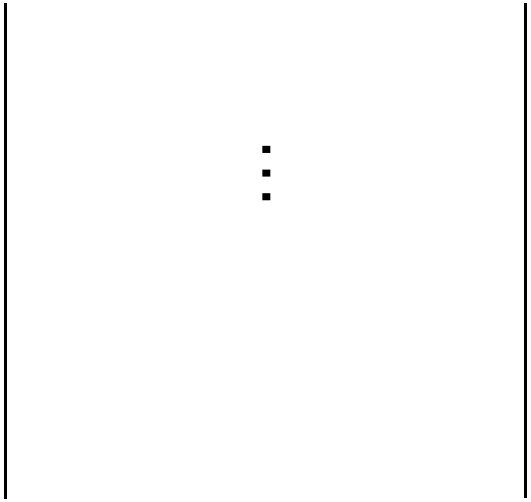
$D_3[8:8+\text{Len}(Q)-1]$



Q



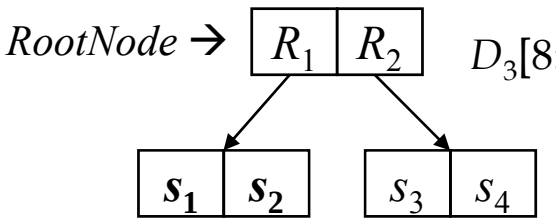
$LB\_Keogh(E(Q), D_3[8:8+Len(Q)-1]) = 5.0 < \delta_{cur}$



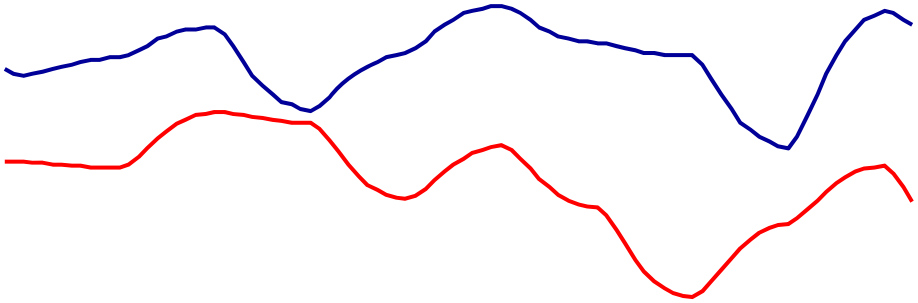
← Top

$\delta_{cur} = 5.3$

Priority Queue

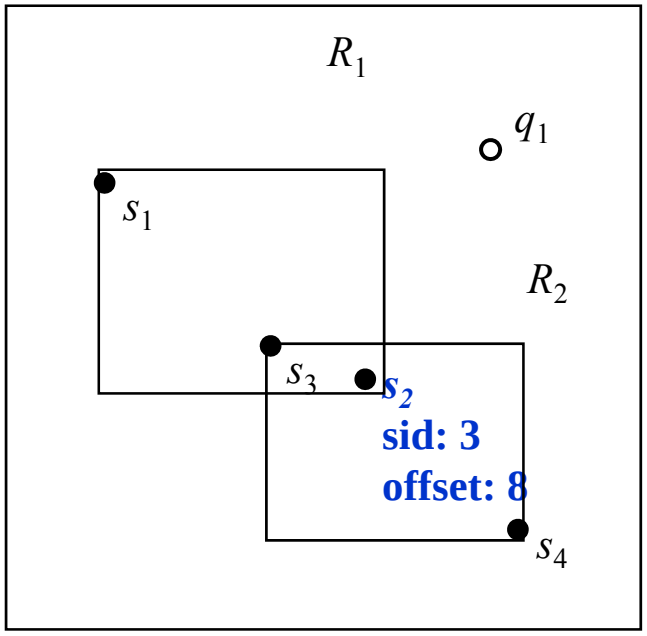


D_3[8:8+Len(Q)-1]

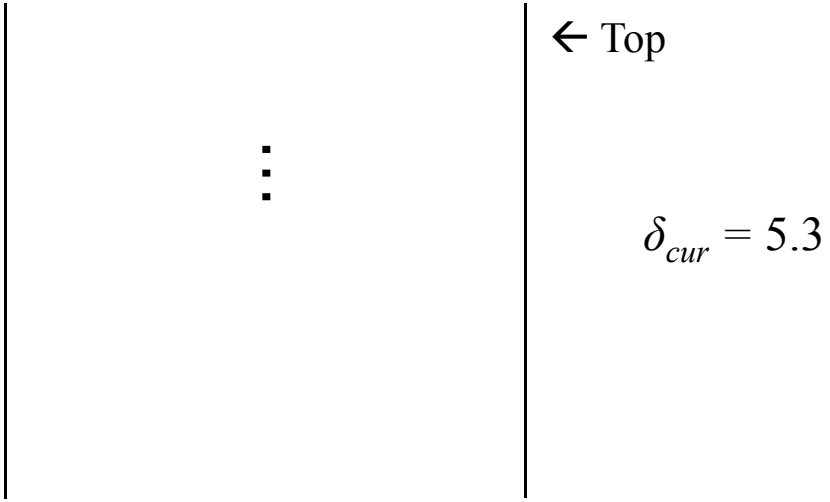


Q

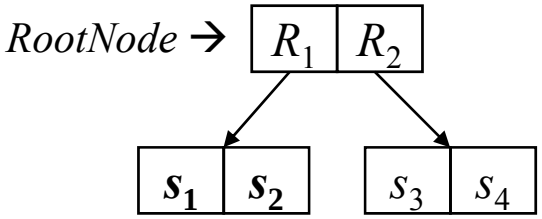
RootNode

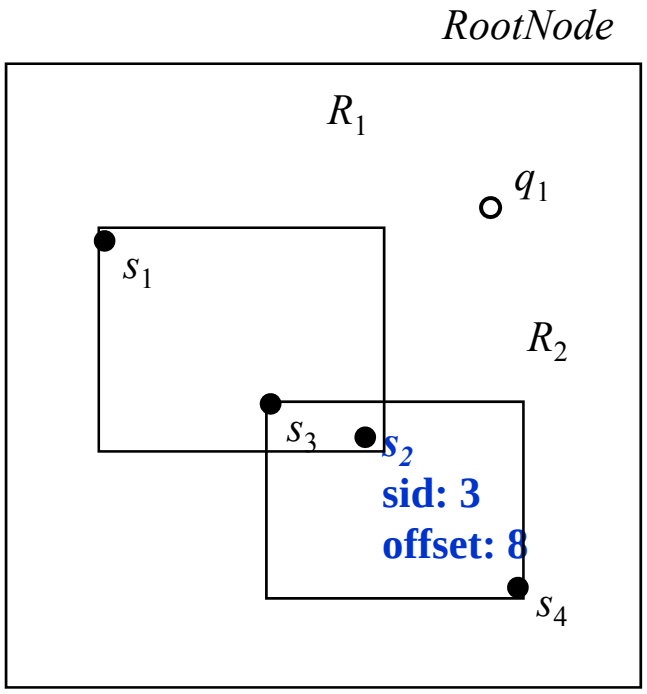


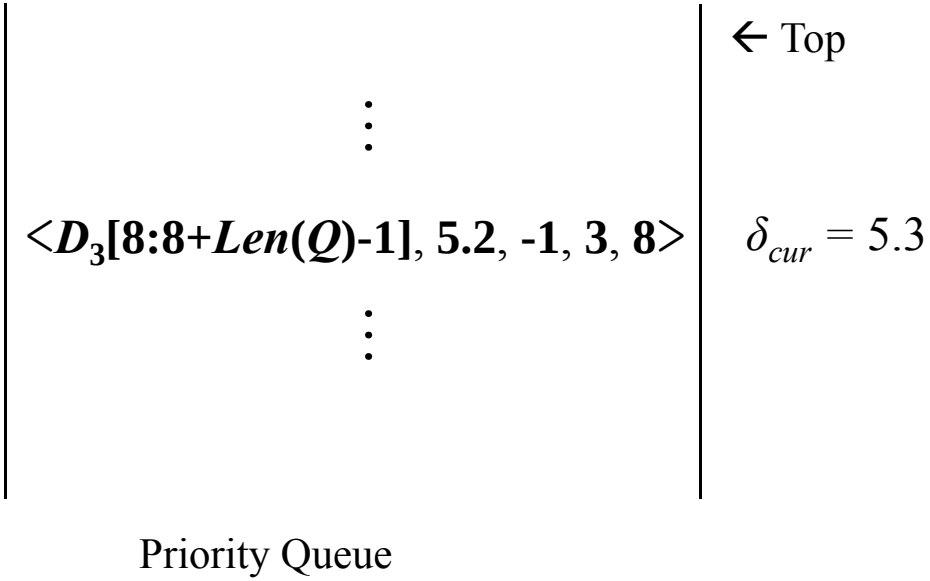
$DTW_\rho(Q, D_3[8:8+Len(Q)-1]) = 5.2 < \delta_{cur}$



Priority Queue



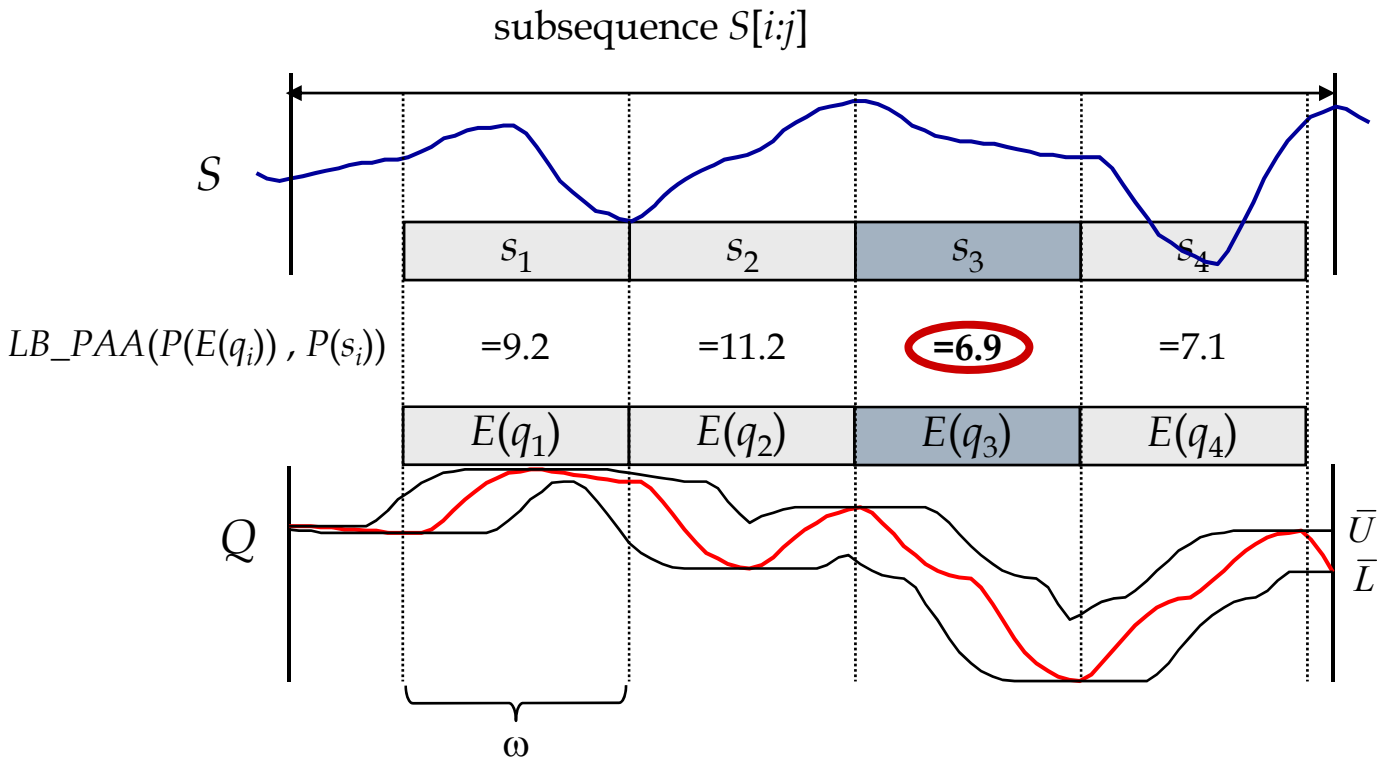




Comments on *DualMatchTopK*

- Many unnecessary subsequences are likely to be retrieved due to the loose lower bound
- To solve this problem, we propose an approach that prunes the index search space leveraging the novel notion of *minimum-distance matching-window pair*

Minimum-Distance Matching-Window Pair



MDMWP Distance

- Suppose that MDMWP of $\mathcal{P}(\mathbb{E}(Q))$ and $\mathcal{P}(S[i:j])$ is $(\mathcal{P}(\mathbb{E}(q_m)), \mathcal{P}(s_m))$
- mdmwp-distance = $\sqrt[p]{r} \times LB\_PAA(\mathcal{P}(\mathbb{E}(q_m)), \mathcal{P}(s_m))$

Lower Boundness of MDMWP-distance

Lemma 3. *Given a query envelope* $\mathbb{E}(Q)$ *and a data subsequence* $S[i:j]$*, the following Eq. (5) holds:*

$$DTW_\rho(Q, S[i:j]) \geq mdmwp\text{–}distance(\mathcal{P}(\mathbb{E}(Q)), \mathcal{P}(S[i:j])) \quad (5)$$

We call the algorithm that incorporates *mdmwp-distance* based pruning in DualMatchTopK, AdvTopK

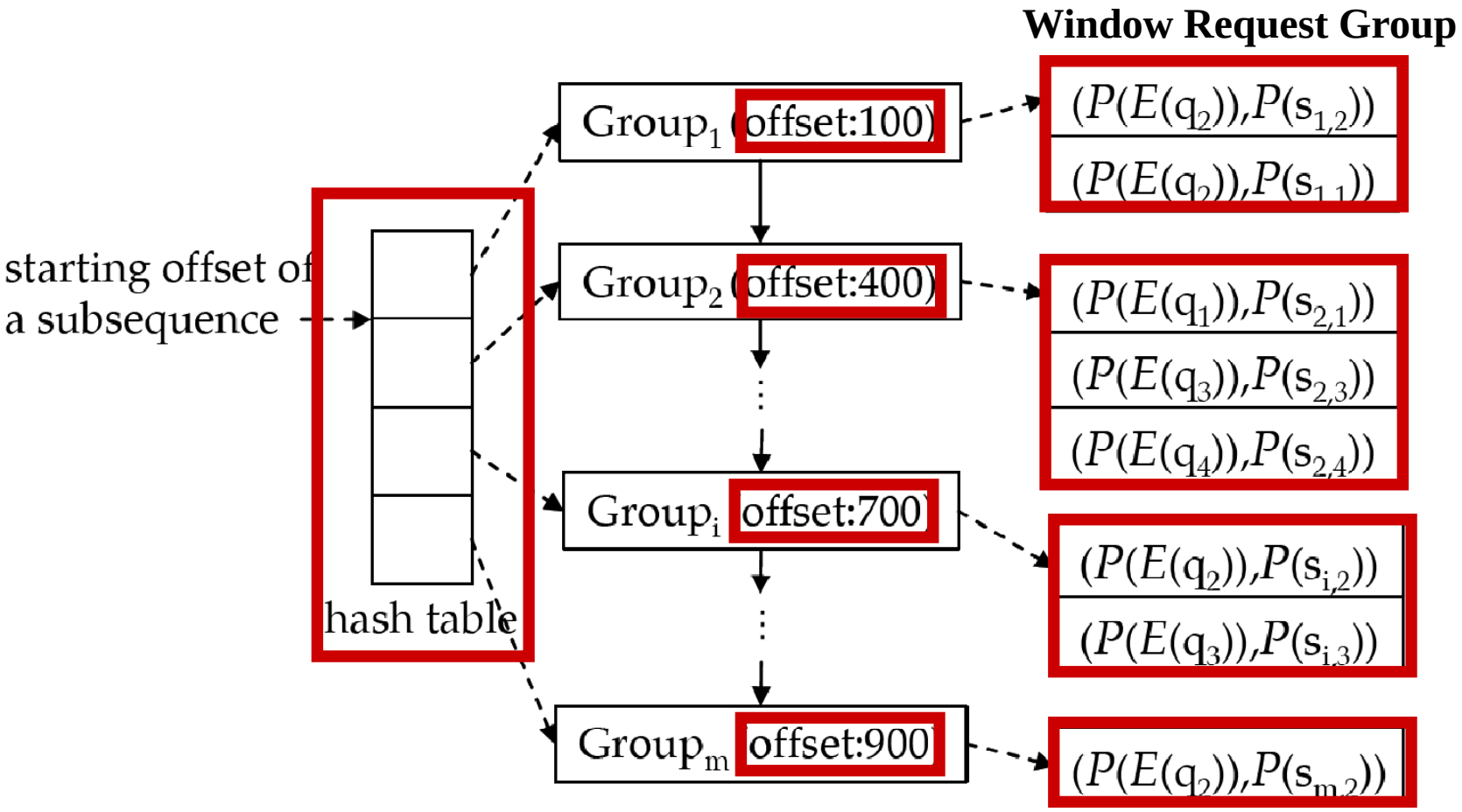
Correctness of AdvTopK

Theorem 1. *Suppose that the current popped entry is given by <obj, d, j, sid, off> in DualMatchTopK, where obj is a leaf entry and the corresponding subsequence for obj is not yet retrieved. Let* δ_{cur} *be the* DTW_ρ *distance between the query sequence and the top k-th data subsequence obtained so far. If* $\sqrt[p]{r} \times LB\_LAA(\mathcal{P}(\mathbb{E}(q_j)), \mathcal{P}(S_{sid}[off : off + \omega - 1]))$ *is greater than* δ_{cur}*, then the corresponding subsequence is not included in the top-k subsequences. Here,* q_j *is the j-th sliding window of Q, and* $r = \lfloor (Len(Q) + 1)/\omega \rfloor - 1$.

Deferred Group Subsequence Retrieval

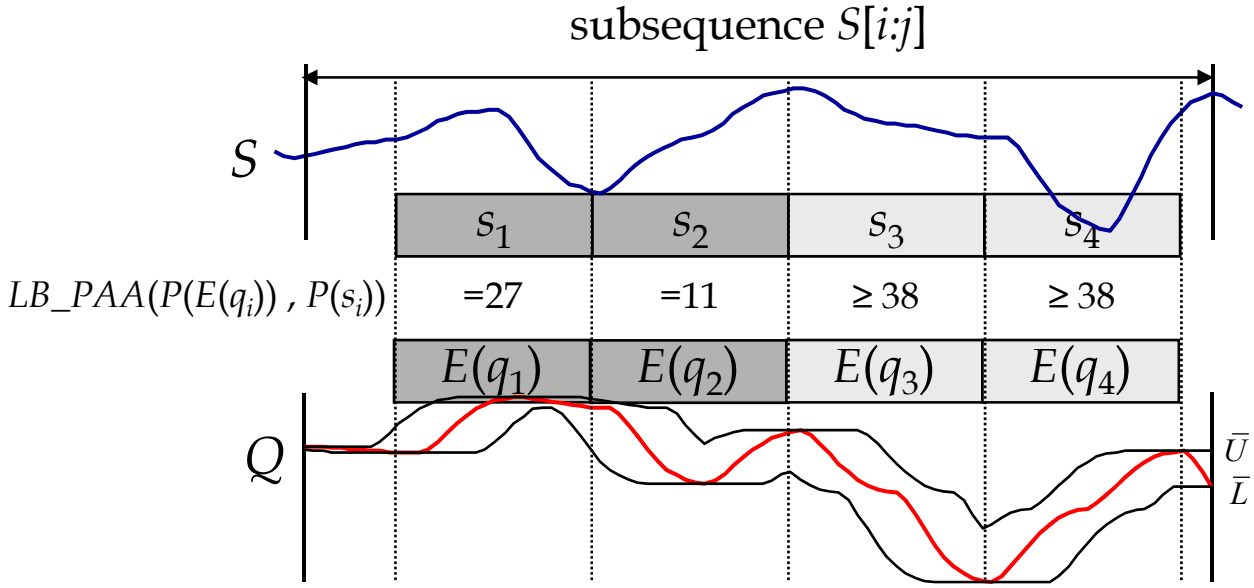
- I/O optimization over AdvTopK
  - avoid excessive random disk I/Os
  - maximize buffer utilization
- Delay a fixed size set of subsequence retrieval requests and enables batch retrieval in a sequential access manner
- Introduce the *group subsequence access list* for storing all requests delayed for the next bulk access

Example of Group Subsequence Access List



Window-Group Distance

- Derived by exploiting both delayed matching windows in each group and the largest distance in the group subsequence access list



$$WG\text{-}dist(P(E(Q), P(S[i:j])) : \sqrt[p]{11^p + 27^p + 38^p \times (4-2)}$$

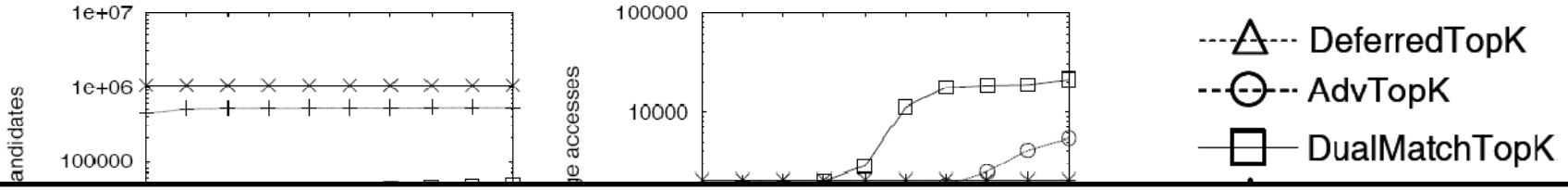
Experimental Setup

- Algorithms compared
  - DualMatchTopK, RangeTopK, AdvTopK, DeferredTopK
  - SeqTopK: sequential scan based algorithm exploiting LB_Keogh
- Datasets used
  - UCR-DATA (33 data sets of different characteristics in the UCR time-series archive, 1,055,525 entries)
  - WALK-DATA (random walk data consisting of one million entries)
  - STOCK-DATA (real data set consisting of 329,112 entries)
  - MUSIC-DATA (pitch data set consisting of 2,373,120 entries extracted from 500 MIDI files)
- Linux Kernel 2.6 PC with 512 Mbytes RAM and Pentium IV 2.8 GHz CPU

Experimental parameters

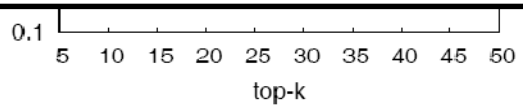
Parameter	Default	Range
k	25	$5 \sim 50$
Buffer size	5%	$1\% \sim 10\%$
$Len(Q)$	384	256, 384, 512
ω (window size)	64	32, 64, 128

Effect of *k* Using UCR-DATA



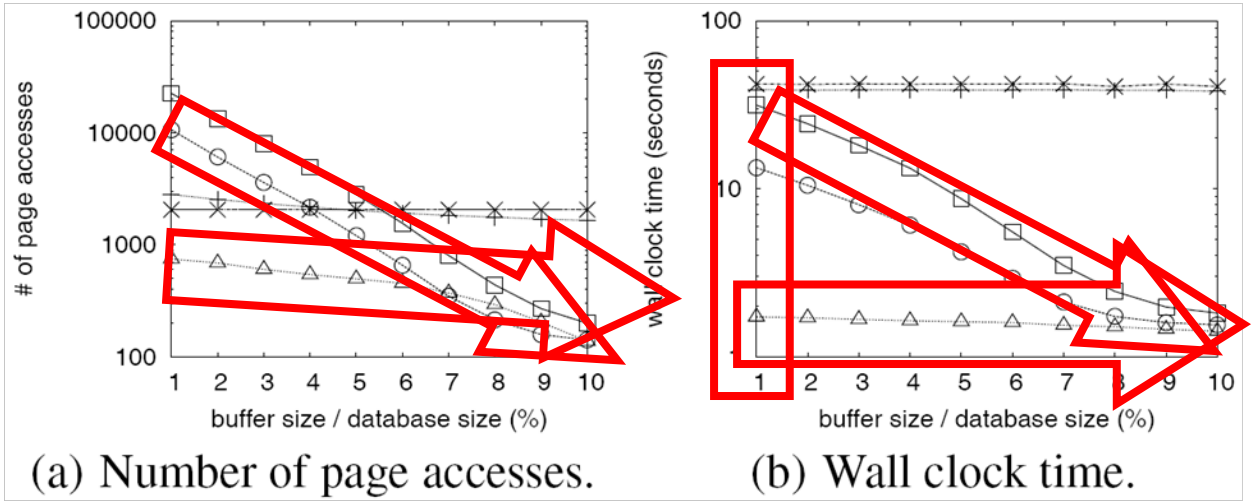
We see similar trends in terms of wall clock time.

In terms of # of page accesses, for small k, all index-based algorithms perform much better than SeqTopK and RangeTopK. As k increases, # of page access of all the index-based algorithms increase.



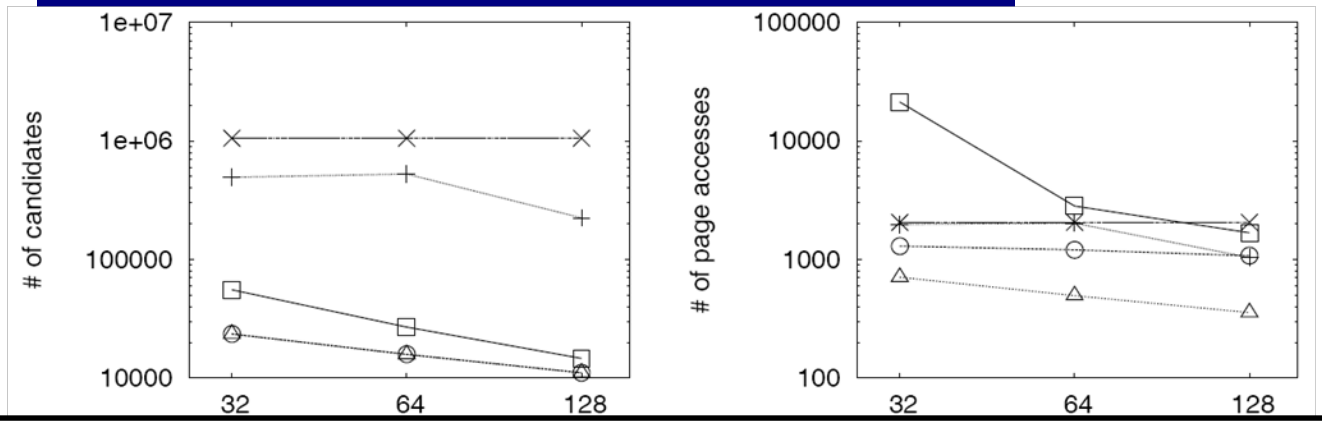
(c) Wall clock time.

Effect of Buffer Size Using UCR-DATA

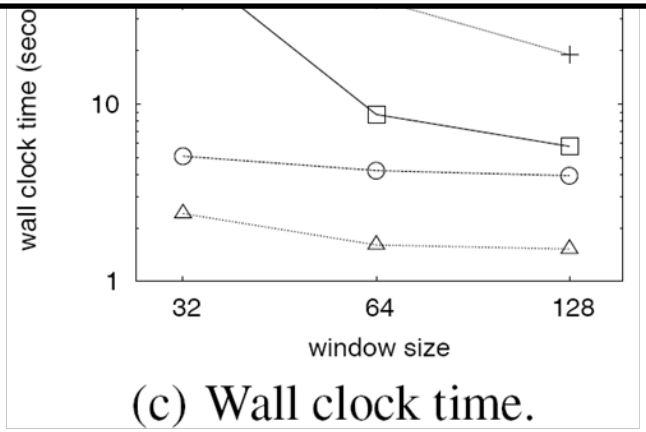


DeferredTopK shows almost constant performance and much better performance with a very small buffer size

Effect of Window Size Using UCR-DATA

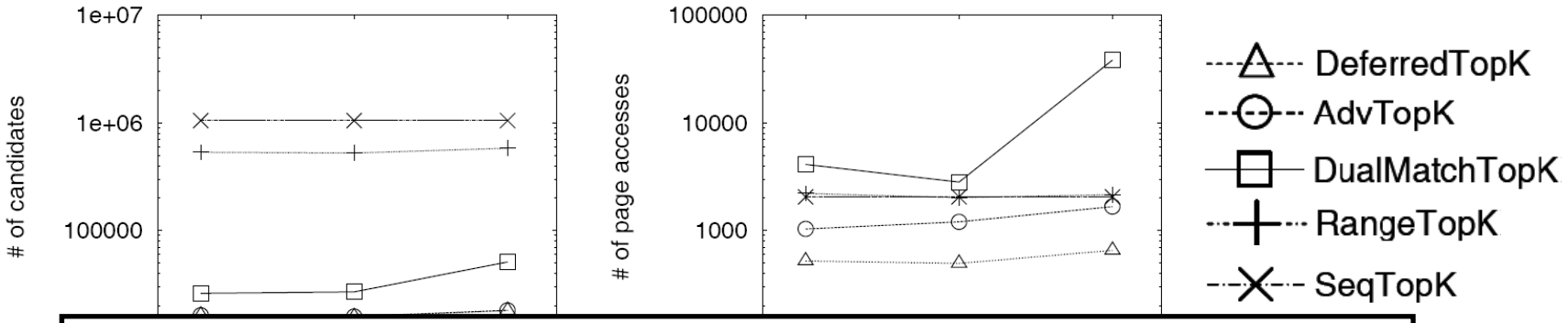


As the window size increases, all three measures of these index-based algorithms decrease due to window size effect.

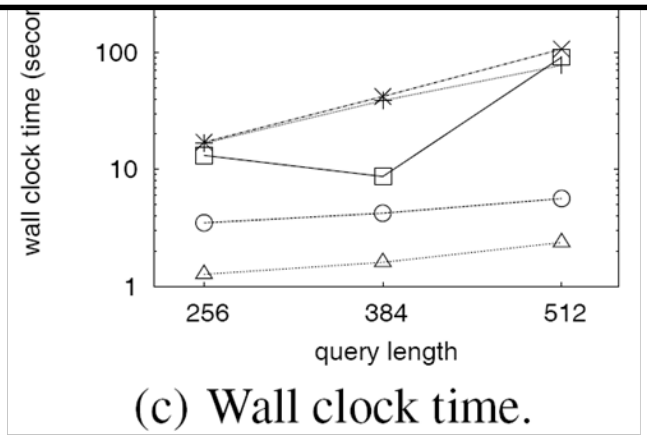


(c) Wall clock time.

Effect of Query Length Using UCR-DATA

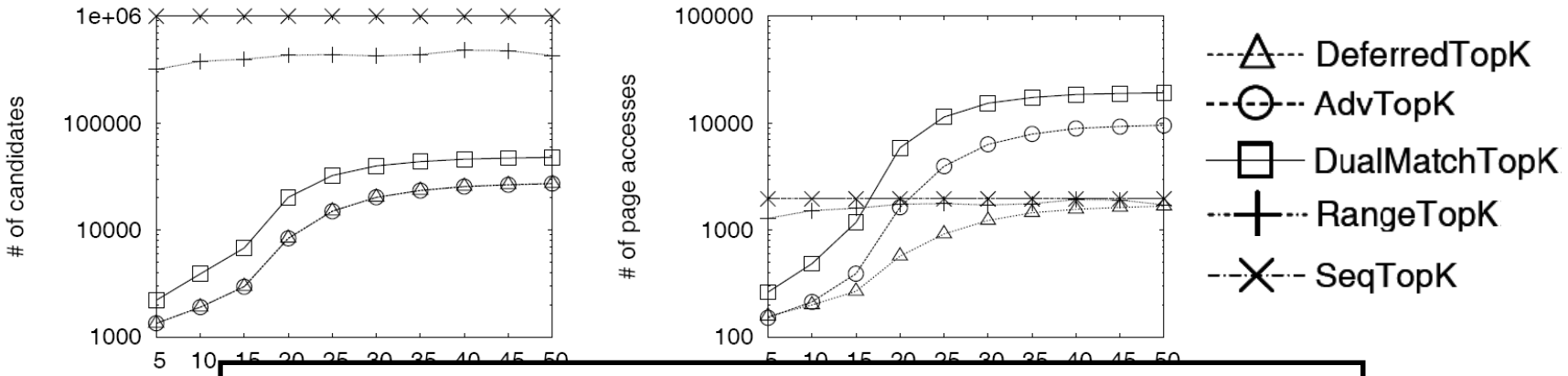


As the query length increases, the relative size of the corresponding window decreases, and thus, more candidates occur due to the window size effect.



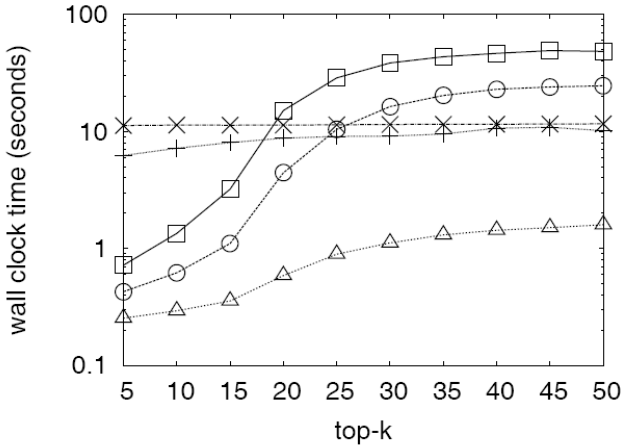
(c) Wall clock time.

Experimental Results for WALK-DATA by Varying *k*



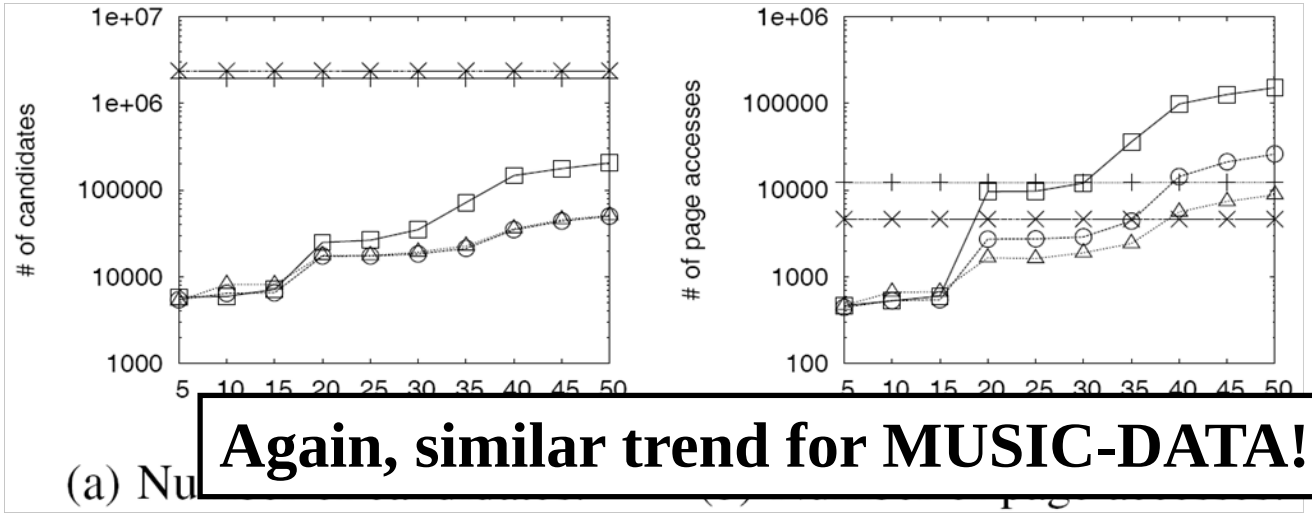
(a) Number of candidates. (b) Number of page accesses.

The trend is similar to that for UCR-DATA.



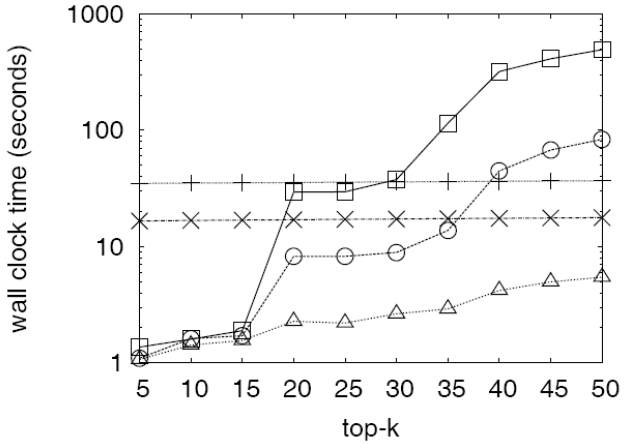
(c) Wall clock time.

Experimental Result for MUSIC-DATA by Varying *k*



(a) Nu

Again, similar trend for MUSIC-DATA!



(c) Wall clock time.

Conclusions

- proposed a novel notion of the *minimum-distance matching-window pair* and derived a lower bound, *mdmwp-distance*
- proposed the *deferred group subsequence retrieval* to avoid excessive random disk I/Os and bad buffer utilization
- derived another lower bound *window-group distance* that can be used together with deferred group subsequence retrieval
- proposed four ranked subsequence matching methods, *DualMatchTopK*, *RangeTopK*, *AdvTopK*, and *DeferredTopK*
- Extensive experiments showed that our advanced methods outperform competing methods by up to orders of magnitude

Thank You Very Much!

Any Questions?

Appendix

RangeTopK

Algorithm 3 *RangeTopK*

Input: Q, k, ρ
Output: results (k-nearest data subsequences for Q)

```
 1: Variable queue : Minimum priority queue;
 2: Variable max_queue : Maximum priority queue;
 3: Variable candidates : List;
 4: Variable ε;
 5: for each i-th sliding window E(q_i) in E(Q) do
 6:    queue.Push(⟨RootNode,MINDIST(P(E(q_i)),RootNode),i,−1,−1⟩)
 7: while not queue.IsEmpty() do
 8:    ⟨obj, d, j, sid, off⟩ ← queue.Pop();
 9:    if obj is a leaf entry then
10:       soff ← off − j + 1; /*start offset*/
11:       eoff ← soff + Len(Q) − 1; /*end offset*/
12:       max_queue.Push(Sub, DTW_ρ(Q,Sub),−1, sid,soff);
13:       if max_queue.Size() = k then
14:          ⟨obj2, d2, j2, sid2, off2⟩ ← max_queue.Top();
15:          ε ← d2;
16:          break;
17:    else if obj is a leaf node LN then
18:       for each leaf entry E ⟨Point P, SeqID sid2, offset off2⟩ in LN do
19:          queue.Push(⟨E, LB_PAA(P(E(q_j)),P), j, sid2, off2⟩);
20:    else if obj is a non-leaf node then
21:       for each child node E ⟨MBR M, Child ptr⟩ in obj do
22:          queue.Push(⟨E, MINDIST(P(E(q_j)),M), j, −1, −1⟩);

23: candidates ← RangeScan(E(Q), ε);
24: results ← Refinement(candidates, k);
```