Processing windowed Top-k Queries on Uncertain Streams

Cheqing Jin 金澈清 cqjin@ecust.edu.cn
East China Normal University, Shanghai, China

華東師范大學 上海 中國
Outline

- Introduction
  - Top-k queries
  - contribution
- Our solution
- Experiments
- Conclusion
Possible World Model

A small record set of reading logs

<table>
<thead>
<tr>
<th>ID</th>
<th>Speed(*10)</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>0.4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0.4</td>
</tr>
</tbody>
</table>

If we ignore the probability field, the semantic of top-k query is very clear. 16 possible world instances

\[
0.096 = 0.4 \times 0.8 \times (1-0.5) \times (1-0.4)
\]

\[
0.036 = (1-0.8) \times (1-0.5) \times (1-0.4) \times (1-0.4)
\]
Windowed Model on Uncertain Streams

Assume $W=3$

<table>
<thead>
<tr>
<th>time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

At time 3, possible worlds are:

- $\{8, 7, 2\}$
- $\{7, 2\}$
- $\{8, 7\}$
- $\{7\}$
- $\{8, 2\}$
- $\{2\}$
- $\{8\}$

At time 4, possible worlds are:

- $\{9, 7, 2\}$
- $\{7, 2\}$
- $\{9, 7\}$
- $\{7\}$
- $\{9, 2\}$
- $\{2\}$
- $\{9\}$

Obviously, when a new tuple arrives, half of possible worlds will change.
Uncertain Top-$k$ queries

- **U-Top$k$ [4]**
  - returns the top-$k$ tuples in all possible worlds with maximum probability.

- **U-$k$Ranks [4]**
  - returns the winner for the $i$-th rank for all $1 \leq i \leq k$.

- **PT-$k$ [2]**
  - returns all the tuples with maximum aggregate probability greater than a user-given threshold $p$
    - aggregate probability: the prob. of being the top-$k$ among all.

- **P$k$-Top$k$ [this paper]**
  - returns the $k$ most probable tuples of being the top-$k$ among all.
  - A slight modification of PT-$k$, while without threshold $p$. 
Example \((k=2)\)

16 possible world instances

<table>
<thead>
<tr>
<th>Tuples</th>
<th>Pr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8, 6, 5, 2</td>
<td>.064</td>
</tr>
<tr>
<td>8, 5</td>
<td>.096</td>
</tr>
<tr>
<td>8, 2</td>
<td>.016</td>
</tr>
<tr>
<td>6, 5</td>
<td>.144</td>
</tr>
<tr>
<td>5, 2</td>
<td>.096</td>
</tr>
<tr>
<td>Empty</td>
<td>.036</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tuples</th>
<th>Pr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8, 6, 5</td>
<td>.096</td>
</tr>
<tr>
<td>8, 6, 2</td>
<td>.016</td>
</tr>
<tr>
<td>8</td>
<td>.024</td>
</tr>
<tr>
<td>6, 2</td>
<td>.024</td>
</tr>
<tr>
<td>6</td>
<td>.036</td>
</tr>
<tr>
<td>2</td>
<td>.024</td>
</tr>
</tbody>
</table>

Query results

<table>
<thead>
<tr>
<th>Query</th>
<th>U-Topk</th>
<th>U-kRanks</th>
<th>PT-k</th>
<th>Pk-Topk ((p=0.3))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>{6, 5}</td>
<td>{8, 5}</td>
<td>{5, 6, 8}</td>
<td>{5, 6}</td>
</tr>
</tbody>
</table>

**Claim:** our goal is not to propose yet another new top-\(k\) query definition, but a new framework that all top-\(k\) queries can be processed in the sliding window model.
Contribution:
Ours is the first work
In this area, especially for Top-k queries!
Working model

- In the ultimate situation, all tuples in the window must be saved in memory!
  - Example
    - $t_i$: the $i$-th tuple in stream, (value: $1/i$, probability: $1/i$).
    - The window size is $W$, at time $n$, for any $k$, tuple $t_{n-W+1}$ is at the query result of $Pk$-Top$k$ query.
  - So, worst-case bounds are trivial and meaningless.

- So, we consider a more general scenario: random order stream model.
  - The value and probability of a tuple are both randomly and independently drawn from some (arbitrary) distribution.
Naïve solution

- Basic Synopsis (BS)
  - Reserve all recent $W$ tuples in memory
  - Use traditional method to answer top-$k$ queries

- Analysis
  - Time-efficient, but space-inefficient.
  - The space complexity is $O(W)$.
**GOAL:** designing a general framework for all kinds of top-k queries, not only for a special kind of query.

1. A small subset of the original dataset
2. Self-maintenance $C(D \cup \{t\}) \subseteq C(D) \cup \{t\}$
3. Capable of answering a top-k query

1. $W$ different windows. i.e., $[t-j, t]$, for $j=0..W-1$
2. One compact set for each window

1. CSQ, CCSQ, SCSQ, SCSQ-buffer
2. Space-efficient
3. Time-efficient
Example: Compact Set for P\textsubscript{k}-Top\textsubscript{k}

- **Symbols**
  1. \( D_i \): the subset of \( D \) containing the first \( i \) tuples in \( D \).
  2. \( r_{ij} \): the probability that a randomly generated world from \( D_i \) has exactly \( j \) tuples.
     1. \( r_{ij} \) can be maintained through dynamic program.
  3. \( p(t_i) \): the probability of tuple \( t_i \).
  4. \( p(t_i)r_{i-1,j-1} \): the probability that \( t_i \) ranks the \( j \)-th in a randomly generated world from \( D \).
  5. \( p(t_i)\sum_{i=1..k} r_{i-1,j-1} \): the probability that \( t_i \) ranks top-\( k \) in all possible worlds generated from \( D \).
  6. \( \sum_{i=1..k} r_{i-1,d-1} \): The up-bound probability of any other tuple outside of \( D \) that ranks top-\( k \) in all possible worlds generated from \( D \).

- **Compact set for P\textsubscript{k}-Top\textsubscript{k}**
  - *Smallest* \( D \) with \( k \) tuples \( (t_a) \) satisfying: \( p(t_a)\sum_{i=1..k} r_{i-1,a-1} > \sum_{i=1..k} r_{i-1,d-1} \)
Possible sub-windows

Assume window size $W=8$, $k=3$

<table>
<thead>
<tr>
<th>time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Possible sub-windows

- [6,8]
- [5,8]
- [4,8]
- [3,8]
- [2,8]
- [1,8]

Create a compact set for each sub-window!
Create compact sets

Goal: Compress the remaining compact sets.
1. A new tuple arrives

2. Generate a new compact set

3. Update compact sets

4. Compact sets are unchanged

5. Remove Compact set if expired

6. Duplicate pruning
Compact Set Queue: Analysis

- Advantages
  - Easy to understand
  - The space consumption and the per-tuple processing cost are small.

- Disadvantages
  - Redundancy exists between neighbor compact sets.

- Solution
  - compress neighbor compact sets!
Synopsis (2):
Compressed Compact Set Queue

1. A new tuple arrives, reserve it in CCSQ


3. Process tuple 5

4. Process tuple 3

5. State after processing tuple 7

6. Remove expired tuple 8
Compressed Compact Set Queue: Analysis

- **Advantages**
  - The space consumption is reduced.

- **Disadvantages**
  - Lots of compact sets must be generated and checked for each incoming tuple, which results in high per-tuple processing cost.

- **Solution**
  - Group neighbor compact sets!
  - In fact, it’s the combination of CSQ and CCSQ.
Synopsis (3):
Segmental Compact Set Queue

1. A new tuple arrives
2. Update compact sets
3. Duplicate pruning
4. Remove expired tuple
5. Generate a new SCS
6. Merge

RULE: Sum of the affiliated tuples in two neighboring compact sets is smaller than k
Segmental Compact Set Queue: Analysis

- Advantages
  - Low space consumption.
  - Low per-tuple processing cost.

- Disadvantages
  - Some medial compact sets are unnecessary to be maintained per tuple.

- Solution
  - Use a buffer!
Synopsis (4): SCSQ-buffer

- **Basic structure contains:**
  - A buffer with size $kH$ to reserve new tuples
  - A SCSQ for all tuples except the buffer
  - A compact set $C(S_W)$ for query result

- **When a tuple $t$ arrives**
  - Insert $t$ into $B$
  - remove out-of-date tuples in SCSQ if possible
  - If $B$ is full, update SCSQ with $B$
  - Else, update $C(S_W)$
The state at time $t$

During $[t+1, t+12]$,  
1. Fill buffer till full,  
2. Remove expired tuples in SCSQ

Then,  
Update SCSQ,  
Clear buffer
# Performance summary

<table>
<thead>
<tr>
<th></th>
<th>Space consumption</th>
<th>Processing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Synopsis</td>
<td>$O(W + kH)$</td>
<td>$O(kH^2/W + \log W)$</td>
</tr>
<tr>
<td>Compact Set Queue</td>
<td>$O(H^2\log W)$</td>
<td>$O(kH^2)$</td>
</tr>
<tr>
<td>Compressed Compact Set Queue</td>
<td>$O(H(k + \log W))$</td>
<td>$O(kH^2)$</td>
</tr>
<tr>
<td>Segmental Compact Set Queue</td>
<td>$O(H(k + \log W))$</td>
<td>$O(kH \log W)$</td>
</tr>
<tr>
<td><strong>SCSQ-buffer</strong></td>
<td>$O(H(k + \log W))$</td>
<td>$O(kH^2/W + \log W)$</td>
</tr>
</tbody>
</table>
Experiments

- **Dataset:** International Ice Patrol (IIP) Iceberg Sightings Database
  - information on iceberg activity in North Atlantic to monitor iceberg danger near the Grand Banks
  - Sighting signals
    - R/V (radar and visual) – 0.8
    - VIS (visual only) – 0.7
    - RAD (radar only) – 0.6
    - SAT-LOW (low earth orbit satellite) – 0.5
    - SAT-MED (medium earth orbit satellite) – 0.4
    - SAT-HIGH (high earth orbit satellite) – 0.3
    - EST (estimated, used before 2005) – 0.4
  - we created a 1,000,000-record data stream by repeatedly selecting records randomly.
Space consumption

(a) Varying $k$ ($W = 100,000$)  
(b) Varying $W$ ($k=20$)
Per-tuple processing cost

(a) Varying $k$ ($W = 100,000$)    (b) varying $W$ ($k=20$)
Sliding-window Top-$k$ Queries on Uncertain Streams, VLDB 2008

Figure 8: PT-$k$ query on real dataset

(a) Space used ($W = 10^5$)  
(b) Space used ($k = 20$)  
(c) Per-tuple cost ($W = 10^5$)  
(d) Per-tuple cost ($k = 20$)

Figure 9: U-$k$Ranks query on real dataset

(a) Space used ($W = 10^5$)  
(b) Space used ($k = 20$)  
(c) Per-tuple cost ($W = 10^5$)  
(d) Per-tuple cost ($k = 20$)

Figure 10: U-top$k$ query on real dataset

(a) Space used ($W = 10^5$)  
(b) Space used ($k = 20$)  
(c) Per-tuple cost ($W = 10^5$)  
(d) Per-tuple cost ($k = 20$)
My idea to produce a good paper:
———Doing as much as possible

Find a valuable problem

Application arguable?

Hard to sell

Get a basic idea

Write it down

At most a small paper

Application arguable?

Keep working

with simple experiments

A potential good paper

Submit it now

Keep working

Doing more extended work and experiments

A qualified paper

Keep working

Doing more extended work and experiments

Keep working

Improving writing skill, Reorganize contents, Add examples

Submit it now
Conclusion

- We propose a general framework to process Sliding-window Top-k queries on uncertain streams.
- Support U-Topk, U-kRanks, PT-k, Pk-Topk.
- Our work is the first work in processing sliding-window queries on uncertain streams.
Future work

- Handle other kinds of queries with this framework
- Handle more complex uncertain models, such as with constraints?
- Attribute level uncertainty?
- Using probability density function?
- Distributed uncertain stream?
- From top-k to NN query, range query, or skyline query?
Reference (uncertain top-k queries)

Reference (uncertain stream processing)


Questions?

Thanks.