A Bandwidth-Efficient Nearest Neighbor Search for Dynamic Time Warping Distance in Distributed Environment

Intel Team Topic 3

Pernghwa, Chin-Chi,
Yen-Hua, Kung-Ting
Problem Description

Time series are distributed among sites in a distributed environment.

Find query’s K nearest neighbors (KNN) under dynamic time warping (DTW).

Goal: reduce total communication cost (bandwidth) as much as possible.
Time Series

• A collection of observations in time sequentially
• Multi-dimensional feature vector where adjacent ordered features are highly dependent
Distributed Environment

- **Server**: A central site which has a time series as query to find its KNN.
K Nearest Neighbors

• Find **K candidate time series** that are the most **similar** to the query
• Similar: Measure by **distance**
  – Smaller distance $\leftrightarrow$ more similar
• **KNN**: K candidate time series that have the **smallest distance** to the query
• **KNN** is usually used to solve other problems such as classification
K Nearest Neighbors

• Example: KNN classification
Dynamic Time Warping

• DTW: A kind of distance measure
  – Beyond Dynamic Programming algorithm
• Considered better than Euclidean distance (ED) and many applications in different areas
• DTW(Q, C) = D(q_n, c_n) = d(q_n, c_n)
  + \min\{D(q_{n-1}, c_{n-1}), D(q_{n-1}, c_n), D(q_n, c_{n-1})\}
  – d(q_n, c_n) = (q_n - c_n)^2 (our setting is ED)
  – ED(Q, C) = D(q_n, c_n) = d(q_n, c_n) + D(q_{n-1}, c_{n-1})
Dynamic Time Warping

- Example: Compute distance between
  - \( A = (1, 3, 6, 4), B = (3, 6, 4, 2) \)
  - \( B \) looks like a left shift of \( A \)
    - \( B \) should be similar to \( A \) from human view
  - \( ED(A, B) = (1 - 3)^2 + \ldots + (4 - 2)^2 = 21 \)
    - \( ED((1, 3, 6, 4), (3, 6, 4, 2)) = (4 - 2)^2 + ED((1, 3, 6), (3, 6, 4)) \)
  - \( DTW(A, B) = DTW((1, 3, 6, 4), (3, 6, 4, 2)) = (4 - 2)^2 + \min \{ \\
    DTW((1, 3, 6), (3, 6, 4)), \\
    DTW((1, 3, 6), (3, 6, 4, 2)), \\
    DTW((1, 3, 6, 4), (3, 6, 4)) \\
  \} = 8 \)
Dynamic Time Warping

Euclidean Distance

Dynamic Time Warping

<table>
<thead>
<tr>
<th></th>
<th>ED</th>
<th>3</th>
<th>6</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>DTW</th>
<th>3</th>
<th>6</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>14</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>29</td>
<td>38</td>
<td>39</td>
<td></td>
</tr>
</tbody>
</table>
Naïve Approach

1. The whole query is transmitted to all sites
2. The sites compute exact DTWs between the query and the candidates
3. The server gathers all the exact DTWs and sort them to find KNN out
Improvement

• transmitting the complete query leads to very high bandwidth

• Two methods for saving bandwidth
  – Segmentation
  – Bounding technique

• The solutions can help the server prune candidate time series without computing exact DTW
Solutions

• Segmentation
  – Query is split into segments
  – Segments can be represented as **local maximum and minimum**, which are the sampled data points
  – Only the sampled data points are transmitted
Segmentation

At Level L

1. Time series is split into segments
2. Choose min / max in each segment as sampled data points
3. Copy min / max to construct an envelope of Q

Level 1
Level 2
Level 3
Level 4
Segmentation between Levels

• Observation: min / max at level L will be still min / max at level (L + 1)

• Transmitting **signals** instead of the same min / max between levels to save bandwidth

• At the final level (all segment size = 1), **all data points of the query are transmitted exactly once** =>
  Total bandwidth = (complete query length) * 1 + (some signal overhead)
  – In the worst case, not much lose to direct transmission
Segmentation between Levels

- Example: For 4 sub-segments, we use 2-bit signals

Naive: 5 1 0 1 1 -2 -1 3
New: 0 2 1 0 1 1 -1 3
where the first two number are signals
We can save bandwidth of
2 * (32 - 2) = 60
bits on this segmentation
Solutions

• Bounding Technique
  – Any site can construct an envelope of the query with sampled data points received
  – Using the approximate query, sites can compute an upper bound and a lower bound of DTW locally
  – prune candidate time series with these bounds
  – Lower bound $\leq$ Real DTW value always
  – Upper bound $\geq$ Real DTW value always
Bounding Technique

- Current Work: FTW2
  - Edited from FTW (proposed in 2005)
  - Calculation formula is like DTW
  - Providing a tight lower bound and upper bound of the real DTW value
Pruning Flow

At level L

Yes => To level (L+1), transmit more sampled data points of the query

Are more than K time series not pruned?

End because KNN are found

Pruning candidate time series with the information of bounds
Pruning Flow

Is it at the final level?

Are more than K time series not pruned?

Pruning candidate time series with the information of bounds

To next level

Computing the exact DTW of remaining time series

Sorting DTW and finding KNN

End
Pruning Framework

• How to prune candidates since we have the segmentation and bounding technique?
• There are two proposed frameworks
  – Framework 1: Global pruning
  – Framework 2: Local pruning
Framework 1 Overview

1. The partial query at level L is transmitted to all sites.

2. The sites compute upper / lower bounds between the query and the candidates.

3. The server gathers bounds.

4. The server sorts the upper bounds and sets the K-th smallest upper bound as the threshold.

5. The server prunes candidates where lower bound > threshold.

6. If there are more than K candidates, repeat 1 to 5 at level (L + 1).
1. The partial query at level L is transmitted to all sites.
Framework 1

2. The sites compute upper/lower bounds between the query and the candidates.
Framework 1

3. The server gathers bounds.
4. The server sorts the upper bounds and sets the K-th smallest upper bound as the threshold.
Framework 1

5. The server prunes candidates where lower bound > threshold.
Framework 1

6. If there are more than $K$ candidates, repeat 1 to 5 at level ($L + 1$).
1. The server transmits the complete query to some initial sites, gathers the exact DTWs, sorts them and sets the K-th smallest DTW as the threshold.

2. For other sites, the threshold and the partial query at level L is transmitted to one of them.

3. The site computes the lower bounds and prunes candidates where lower bound > threshold.

4. The site asks for the partial query at level (L + 1) until all candidates are pruned in the site.

5. If there are candidates not pruned at the final level, the site computes the exact DTWs and transmits them back to update the threshold.

6. The server repeats 2 to 5 for other unpruned sites.
1. The server transmits the complete query to some initial sites, gathers the exact DTWs, sorts them and sets the K-th smallest DTW as the threshold.
2. For other sites, the threshold and the partial query at level L is transmitted to one of them.
3. The site computes the lower bounds and prunes candidates where lower bound > threshold.
4. The site asks for the partial query at level \( L + 1 \) until all candidates are pruned in the site.
Framework 2

5. If there are candidates not pruned at the final level, the site computes the exact DTWs and transmits them back to update the threshold.
Framework 2

6. The server repeats 2 to 5 for other unpruned sites.
Challenges

• How to set proper parameters of segmentation? Better segmentation?
• What situation do we use Framework 1 or Framework 2 in?
• Balance between bandwidth and computation
  – Tighter bounds => Higher computation cost