Results & Analysis

FlexDTW: Algorithm

1. Initialize

- Cumulative cost matrix $D \in \mathbb{R}^{N \times M}$
- Backtrace matrix $B \in \mathbb{Z}^{N \times M}$
- Starting point matrix $S \in \mathbb{Z}^{N \times M}$

2. Dynamic Programming

- Paths compared using normalized cost measure

3. Backtracking

- Select best endpoint as backtrace

$$E_{best} = \arg\min_{(i,j) \in E_{start}} D[i,j]$$

Experimental Setup

- We modified the Chopin Mazurka dataset [1] to simulate different boundary conditions. Our modifications resulted in a suite of 16 separate benchmarks, where each benchmark tests performance under a specific boundary condition.

- Boundary Conditions

  - Full Match: align full recordings of both (original dataset)
  - Subsequence: align random segment of A against full recording of B
  - Partial Start: both recordings start together but one ends early
  - Partial End: both recordings end together, but one starts late
  - Partial Overlap: recording A starts late and recording B ends early
  - Pre: silence is prepended to A and aligned against full recording of B
  - Post: silence is appended to A and aligned against full recording of B
  - Pre-Post: silence is prepended to A and appended to B

The key insight of FlexDTW is that Manhattan distance can be computed by simply knowing the starting point of the alignment path (not the actual path itself). This information can be computed recursively during dynamic programming, eliminating the need for backtracking.

FlexDTW: Overview

- FlexDTW allows the alignment path to start anywhere on the left or bottom edge, and to end anywhere on the right or top edge. A short buffer region is imposed to avoid short, degenerate alignment paths near the top left and bottom right corners.
- To fairly compare alignment paths of very different length, we must use a path cost measure that normalizes by the path length. We could backtrack from every position $(i,j)$ to determine the length of the alignment path ending at $(i,j)$, but this would require a prohibitive amount of additional computation.
- The key insight of FlexDTW is that Manhattan distance can be computed by simply knowing the starting point of the alignment path (not the actual path itself). This information can be computed recursively during dynamic programming, eliminating the need for backtracking.

Runtime

- Average runtime (10 trials) in seconds to process a cost matrix of size $N \times N$:

<table>
<thead>
<tr>
<th>System</th>
<th>1k</th>
<th>2k</th>
<th>5k</th>
<th>10k</th>
<th>20k</th>
<th>50k</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTW</td>
<td>0.033</td>
<td>0.14</td>
<td>0.87</td>
<td>3.5</td>
<td>13.8</td>
<td>87.3</td>
</tr>
<tr>
<td>SubseqDTW</td>
<td>0.044</td>
<td>0.15</td>
<td>0.96</td>
<td>3.82</td>
<td>15.3</td>
<td>96.8</td>
</tr>
<tr>
<td>NWTW</td>
<td>0.037</td>
<td>0.16</td>
<td>0.99</td>
<td>3.93</td>
<td>15.8</td>
<td>101.1</td>
</tr>
<tr>
<td>FlexDTW</td>
<td>0.038</td>
<td>0.16</td>
<td>1.05</td>
<td>4.21</td>
<td>16.9</td>
<td>111.1</td>
</tr>
</tbody>
</table>

- FlexDTW incurs a 20-25% runtime overhead compared to DTW and a 10-15% runtime overhead compared to subsequence DTW.

Memory

- FlexDTW has memory overhead for storing the additional starting point matrix $S \in \mathbb{Z}^{N \times M}$.
- For sequence lengths $<2^{15}$, the overhead is 2NM bytes (12% increase in total memory).
- For sequence lengths $>2^{15}$, the overhead is 4NM bytes (24% increase).

References & Acknowledgements


This material is based upon work supported by the National Science Foundation under Grant No. 2144050.