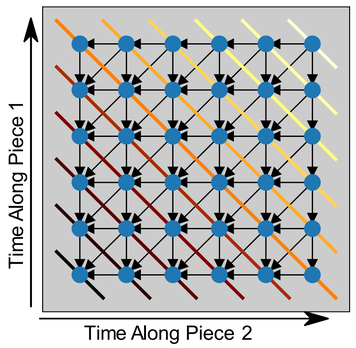
Exact, Parallelizable Dynamic Time Warping with Linear Memory

Christopher J. Tralie, Assistant Professor, Math/Computer Science, Ursinus College, ctralie@alumni.princeton.edu
Elizabeth Dempsey, Senior Computer Science Major, Ursinus College, eldempsey@ursinus.edu

Abstract

Audio alignment is a fundamental preprocessing step in many MIR pipelines. For two audio clips with M and N frames, respectively, the most popular approach, dynamic time warping (DTW), has O(MN) requirements in both memory and computation, which is prohibitive for frame-level alignments at reasonable rates. To address this, a variety of memory efficient algorithms exist to approximate the optimal alignment under the DTW cost. To our knowledge, however, no exact algorithms exist that are guaranteed to break the quadratic memory barrier. In this work, we present a divide and conquer algorithm that computes the exact globally optimal DTW alignment using O(M+N) memory. Its runtime is still O(MN), trading off memory for a 2x increase in computation. However, the algorithm can be parallelized up to a factor of min(M, N) with the same memory constraints, so it can still run more efficiently than the textbook version with an adequate GPU. We use our algorithm to compute exact alignments on a collection of orchestral music, which we use as ground truth to benchmark the alignment accuracy of several popular approximate alignment schemes at scales that were not previously possible.

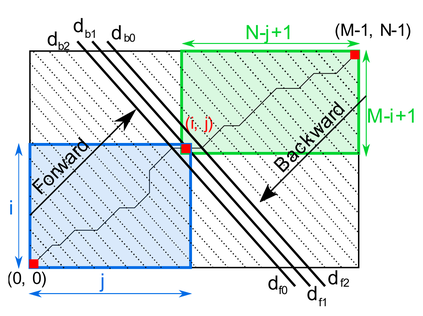
Subroutine: Linear Systolic Array [1, 2]



- Process elements along diagonals instead of left to right

Linear Memory	✓
Parallel Processing	✓
Extract Alignment	✗

Main Algorithm: Divide And Conquer Linear Systolic Arrays



Linear Memory	✓
Parallel Processing	✓
Extract Alignment	✓

1. Forward on half
2. Backward on other half
3. Meet in the middle
4. Recurse on both halves

- **CPU MEMORY:** O(min(M, N))
- **CPU TIME:** O(MN)
- **GPU MEMORY:** O(min(M, N))
- **GPU TIME:** O(min(M, N))

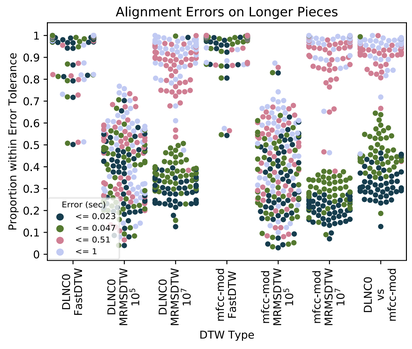
Experiments: Memory Requirements

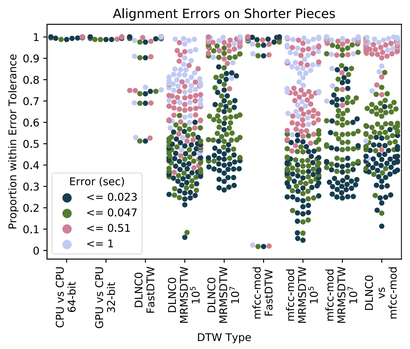
Piece	Version 1	Version 2	DTW	FastDTW	Ours
Vivaldi Spring	Abbado (188 sec)	Gunzenhauser (209 sec)	277 MB	3.86 MB	194 KB
Candide Overture	Bernstein (268 sec)	Dudamel (279 sec)	527 MB	5.5 MB	270 KB
Beethoven. Symph. No.5	Thielemann (445 sec)	Bernstein (514 sec)	1.58 GB	9.12 MB	448 KB
Schumann - Symph. No. 3	Bernstein (2124 sec)	Muti (2199 sec)	23.2 GB	36.9 MB	1.77 MB
Stravinsky The Rite of Spring	Rattle (2053 sec)	Bernstein (2082 sec)	29.4 GB	42.1 MB	2.02 MB
Tchaikovsky Symph. No. 4	Bernstein (2645 sec)	Rozhdestve.. (2530 sec)	46.1 GB	51.9 MB	2.48 MB
Shostakovich: Symph. No. 11	Søndergård (3647 sec)	Nelsons (3765 sec)	94.6 GB	74.8 MB	3.6 MB
Verdi Requiem	Bychkov (4983 sec)	Solti (5042 sec)	173 GB	102 MB	4.9 MB
Wagner - Das Rheingold	Kuhn (8799 sec)	Solti (8759 sec)	542 GB	180 MB	8.6 MB

- DTW refers to the naive algorithm
- FastDTW refers to the algorithm in [21] using a band width of δ = 30.
- The memory requirements for MrMsDTW[4] for 10^5 and 10^7 constant cells is 391KB and 38Mb, respectively

Experiments: Approximate Algorithm Frame Level Alignment Errors

- 50 “short” pairs of performances, under 20 minutes each
- 50 “long” pairs of performances, between 20 minutes and ~2 hours
- Compare To FastDTW [3] and Memory-Restricted Multiscale DTW (MrMsDTW) [4]





- Most frames in approximate alignments agree to within a second of those in our exact alignment
- Fewer frames agree to within frame length of 23 milliseconds.
- Overall, approximate algorithms have reasonably good performance even on larger pieces. We able to run these experiments show this since the memory scales

Python Software (pip install linmdtw)

The first step is to load in the audio

```
[1]: import linmdtw
     import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib notebook
     import warnings
     warnings.filterwarnings("ignore")
     import IPython.display as ipd

     sr = 44100
     x0_0, sr = linmdtw.load_audio("../experiments/OrchestralPieces/Short/0_0.mp3", sr)
     x0_1, sr = linmdtw.load_audio("../experiments/OrchestralPieces/Short/0_1.mp3", sr)
```

Next, we’ll compute the “MFCC mod” features for each audio clip [5]

```
[2]: hop_length = 512
     X0_0 = linmdtw.get_mfcc_mod(x0_0, sr, hop_length)
     X0_1 = linmdtw.get_mfcc_mod(x0_1, sr, hop_length)
```

Now, we can extract a warping path between the two audio streams using the main DTW library

```
[3]: import time
     metadata = {'totalCells':0, 'M':X0_0.shape[0], 'N':X0_1.shape[0],
                 'timeStart':time.time(), 'perc':10}
     path0 = linmdtw.linmdtw(X0_0, X0_1, do_gpu=True, metadata=metadata)
```

- NOTE: Can also sonify alignments using the rubberband library [6]

Select References

[1] Evert Dijkstra and Christian Piguet. On minimizing memory in systolic arrays for the dynamic time warping algorithm. Integration, 4(2):155–173, 1986

[2] Chi Wai Yu, KH Kwong, Kin-Hong Lee, and Philip Heng Wai Leong. A smith-waterman systolic cell. In New Algorithms, Architectures and Applications for Reconfigurable Computing, pages 291–300. Springer,2005.

[3] Stan Salvador and Phillip Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. Proc. of ACM Knowledge Data And Discovery (KDD), 3rd Wkshp. on Mining Temporal and Sequential Data, 2004.

[4] Thomas Prätzlich, Jonathan Driedger, and Meinard Müller. Memory-restricted multiscale dynamic time warping. In Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), pages 569–573. IEEE, 2016.

[5] Thassilo Gadermaier and Gerhard Widmer. A study of annotation and alignment accuracy for performance comparison in complex orchestral music. In Proc. of the Int. Soc. for Music Information Retrieval Conf. (ISMIR), in print, 2019.

[6] C Cannam. Rubber band library. Software released under GNU General Public License (version 1.8. 1), 2012.