FLEXDTW: DYNAMIC TIME WARPING WITH FLEXIBLE BOUNDARY CONDITIONS

Irmak Bükey
Pomona College
ibab2018@mymail.pomona.edu

Jason Zhang
University of Michigan
zhangjt@umich.edu

TJ Tsai
Harvey Mudd College
ttsai@hmc.edu

ABSTRACT

Alignment algorithms like DTW and subsequence DTW assume specific boundary conditions on where an alignment path can begin and end in the cost matrix. In practice, the boundary conditions may not be known a priori or may not satisfy such strict assumptions. This paper introduces an alignment algorithm called FlexDTW that is designed to handle a wide range of boundary conditions. FlexDTW allows alignment paths to start anywhere on the bottom or left edge of the cost matrix (adjacent to the origin) and to end anywhere on the top or right edge. In order to properly compare paths of very different lengths, we use a normalized path cost measure that normalizes the cumulative path cost by the path length. The key insight of FlexDTW is that the Manhattan length of a path can be computed by simply knowing the starting point of the path, which can be computed recursively during dynamic programming. We artificially generate a suite of 16 benchmarks based on the Chopin Mazurka dataset in order to characterize audio alignment performance under a variety of boundary conditions. We show that FlexDTW has consistently strong performance that is comparable or better than commonly used alignment algorithms, and it is the only system with strong performance in some boundary conditions.

1. INTRODUCTION

Dynamic Time Warping (DTW) is a dynamic programming algorithm for computing the optimal alignment between two sequences under certain assumptions. In the MIR literature, it is the most widely used method for aligning two audio recordings of the same piece of music. One of its assumptions is the boundary condition of the alignment path: it assumes that the alignment path begins at the origin of the pairwise cost matrix and ends in the opposite corner of the cost matrix. When working with real data like (say) Youtube recordings of a piece of classical music, the boundary conditions are usually unknown a priori and may not satisfy the restrictive assumptions of standard DTW. This may be due to silence or applause at the beginning or end of videos, or perhaps due to some videos containing only one movement of a piece. This paper seeks to develop a more flexible variant of DTW that can handle a wider range of boundary conditions.

Previous work. There is a very large body of work on variations or extensions of DTW. These works generally fall into one of two categories. The first category focuses on mitigating the quadratic computation and memory costs of DTW. Some works approach this by speeding up an exact DTW computation through the use of lower bounds [1, 2], early abandoning [3, 4], using multiple cores [5, 6], or specialized hardware [7, 8]. Tralie and Dempsey [9] introduce a method for computing exact DTW with linear memory by processing diagonals rather than rows/columns. Other works propose approximations to DTW that require less computation, runtime, or memory. Some approaches include approximate lower bounds [10, 11], approximations of DTW distance [12, 13], imposing bands in the cost matrix to limit extreme time warping [14, 15], computing alignments at multiple resolutions [16, 17], parallelizable approximations of DTW [18], or working with a fixed amount of memory [19]. The second category focuses on extending the behavior of DTW in some way. Some examples in the MIR literature include handling structural differences like repeats and jumps in music [20–22], performing alignment in an online setting [23–25], handling partial alignments [26, 27], using multiple performances to improve alignment accuracy [28], accounting for pitch drift in a capella music [29], and aligning sets of source recordings and mixtures [30].

Shortcomings. Our work aims to make DTW more flexible by focusing on an often overlooked aspect: boundary conditions. The vast majority of previous works on DTW or its variants focus on handling one specific type of boundary condition. For example, DTW (and any of its approximations or efficient implementations) assumes that an alignment path begins at the origin of the cost matrix and ends in the opposite corner. Similarly, subsequence DTW assumes that an alignment path begins somewhere on the longer edge of the cost matrix and ends on the opposite edge. As mentioned above, in many situations the boundary conditions are unknown a priori or may be incompatible with the assumptions of standard alignment methods.

Our approach. FlexDTW is designed to be flexible in handling a wide range of boundary conditions. Assuming that the origin of the cost matrix is in the lower left corner,

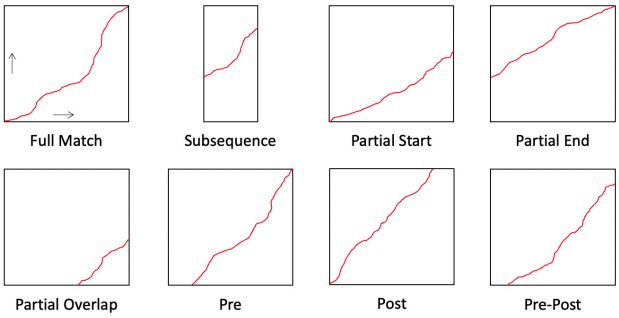

Figure 1. Different boundary conditions for the alignment path between two sequences. The full match and subsequence conditions are well handled by standard algorithms, but the other conditions are not.

FlexDTW allows an alignment path to begin anywhere on the left or bottom edge, and it allows the alignment path to end anywhere on the top or right edge. Figure 1 shows several examples of boundary conditions that FlexDTW can handle. To properly compare alignment paths of very different length, it is necessary to use a normalized path cost measure that normalizes the cumulative path cost by the path length. While it is possible to determine the optimal alignment path ending at any position by following the backpointers in the backtrace matrix, this would result in an impractically high computation overhead. The key insight with FlexDTW is that the Manhattan length of an alignment path can be computed by simply knowing the starting and ending location of the alignment path (without knowing the actual path itself). The starting location information can be computed in a recursive manner and stored during the dynamic programming stage, making it possible to compute normalized path costs in an efficient manner.

Contributions. This paper has three main contributions. First, we introduce an alignment algorithm called FlexDTW that handles a wide range of boundary conditions. FlexDTW allows an alignment path to start anywhere on the two edges of the cost matrix adjacent to the origin (e.g. bottom and left edge), and it allows alignment paths to end anywhere on the other two edges (top and right edge). Second, we design a suite of 16 benchmarks based on the Chopin Mazurka dataset [31] in order to characterize audio alignment performance under a variety of specific boundary conditions. Third, we present experimental results showing that FlexDTW has consistently strong performance across all 16 benchmarks that is comparable to or better than the best-performing system from a set of widely used audio alignment algorithms. We provide source code for our implementation of FlexDTW, along with code for running all experiments in this paper. [1]

2. SYSTEM DESIGN

In this section we describe the FlexDTW algorithm in detail. To make it clear how FlexDTW relates to previous work, we begin with a brief overview of DTW and subsequence DTW.

2.1 DTW and Subsequence DTW

Standard DTW estimates the alignment between two feature sequences $x_0, x_1, \ldots, x_{N-1}$ and $y_0, y_1, \ldots, y_{M-1}$. It accomplishes this by using dynamic programming to find the optimal path through a pairwise cost matrix $C \in \mathbb{R}^{N \times M}$ under a set of allowable transitions. DTW assumes that the alignment path begins at (0,0) and ends at ($N-1$, $M-1$) in the cost matrix. Subsequence DTW is a variant of DTW that finds the optimal alignment between a query sequence $x_0, x_1, \ldots, x_{N-1}$ and any subsequence within a (typically longer) reference sequence $y_0, y_1, \ldots, y_{M-1}$. Subsequence DTW assumes that the alignment path includes the entire query sequence but can begin and end anywhere in the reference sequence.

2.2 FlexDTW: Algorithm

FlexDTW is a variant of DTW that seeks to handle a much wider range of boundary conditions. It is designed to handle the boundary conditions of standard DTW, subsequence DTW, as well as many other conditions that are not handled by DTW or subsequence DTW. We first give an overview of the boundary conditions that FlexDTW is designed to handle, describe the main challenge in allowing flexible boundary conditions, introduce a key insight, and then explain the algorithm in detail.

Boundary conditions. Figure 1 shows an overview of the boundary conditions that FlexDTW is designed to handle. In the given figure, the alignment path may begin anywhere along the left edge or bottom edge of the cost matrix, and the alignment path can end anywhere along the top edge or right edge. [2] Note that the resulting set of allowable alignment paths is a superset that contains all allowable DTW paths and all allowable subsequence DTW paths, in addition to many other types of alignment paths.

Challenge. The main challenge in allowing such flexible boundary conditions is normalization. Because the set of allowable paths has such an enormous variation in path length, one must use a normalized path cost to fairly compare one alignment path with another. (Otherwise, the path with minimum cumulative path cost will simply be the alignment path with fewest elements.) This means that our metric for comparing different alignment paths must normalize the cumulative path cost by some measure of alignment path length. To determine the length of an alignment path ending at position (i,j), we could simply follow the backpointers in the backtrace matrix, but this introduces an impractically high computational overhead to the algorithm.

Key insight. The key insight with FlexDTW is that the Manhattan length of an alignment path does not require knowing what the actual alignment path is. Assuming that the alignment path is monotonically non-decreasing (as is the case with DTW), computing the Manhattan length of

[1] Code can be found at `https://github.com/anonymized/`.

[2] We exclude a buffer region near the top left and bottom right corners to avoid short, degenerate paths, as will be explained later.

an alignment path only requires knowing the starting point and ending point of the path. The starting location of any optimal alignment path can be computed recursively with minimal computational overhead and simply stored as an additional piece of information (similar to the backtrace information). Having the starting location of all optimal alignment paths allows us to efficiently calculate normalized path costs without having to perform any backtracking. We can then compare the goodness of alignment paths by comparing their path cost per Manhattan block.

Algorithm. We now describe the FlexDTW algorithm for aligning two feature sequences x_0, x_1,..., x_{N-1} and y_0, y_1, ..., y_{M-1}. Similar to DTW, one must specify a set of allowable transitions and corresponding transition weights. In addition, there is one hyperparameter `buffer` that specifies a minimum allowable path length, which helps to avoid short, degenerate alignment paths. The algorithm consists of five steps, which are described below.

The first step is to compute a pairwise cost matrix $C \in \mathbb{R}^{N \times M}$, where each element $C[i, j]$ indicates the distance between x_i and y_j under some distance metric.

The second step is to initialize three matrices: a cumulative cost matrix $D \in \mathbb{R}^{N \times M}$, a backtrace matrix $B \in \mathbb{Z}^{N \times M}$, and a starting point matrix $S \in \mathbb{Z}^{N \times M}$. In order to allow alignment paths to begin anywhere in either sequence without penalty, we initialize $D[0, j] = C[0, j]$, $j = 0, 1, \ldots, M-1$ and $D[i, 0] = C[i, 0]$, $i = 0, 1, \ldots, N-1$. We also initialize S for all valid starting points for alignment paths. Since the starting locations are all of the form $(0, j)$ or $(i, 0)$, we can efficiently encode the starting locations as a single integer, where positive integers indicate a starting location $(0, j)$ and negative integers indicate a starting location $(i, 0)$. This reduces the memory overhead of matrix S. Accordingly, we initialize $S[0, j] = j$, $j = 0, 1, \ldots, M-1$ and $S[i, 0] = -i$, $i = 0, 1, \ldots, N-1$.

The third step is to compute the elements in D, B, and S using dynamic programming. For a given set of allowable transitions $\{t_1, t_2, t_3\}$ (assumed to be $\{(1, 1), (1, 2), (2, 1)\}$ in the equation below) and corresponding multiplicative weights w_1, w_2, w_3, the optimal transition $B[i, j]$ can be computed with the following recursive formula:

$$B[i,j] = \underset{k=1,2,3}{\arg\min} \begin{cases} \frac{D[i-1,j-1]+w_1 \cdot C[i,j]}{i+j-|S[i-1,j-1]|} & \text{if } k=1 \\ \frac{D[i-1,j-2]+w_2 \cdot C[i,j]}{i+j-|S[i-1,j-2]|} & \text{if } k=2 \\ \frac{D[i-2,j-1]+w_3 \cdot C[i,j]}{i+j-|S[i-2,j-1]|} & \text{if } k=3 \end{cases} \quad (1)$$

The numerator elements in the equation above are cumulative path costs, and the denominator elements are the Manhattan lengths of each candidate path. Once the best transition has been determined, the value of $D[i, j]$ can be updated as:

$$D[i,j] = \begin{cases} D[i-1,j-1] + w_1 \cdot C[i,j] & \text{if } B[i,j]=1 \\ D[i-1,j-2] + w_2 \cdot C[i,j] & \text{if } B[i,j]=2 \\ D[i-2,j-1] + w_3 \cdot C[i,j] & \text{if } B[i,j]=3 \end{cases} \quad (2)$$

Similarly, the value of $S[i, j]$ can be updated as:

$$S[i,j] = \begin{cases} S[i-1,j-1] & \text{if } B[i,j]=1 \\ S[i-1,j-2] & \text{if } B[i,j]=2 \\ S[i-2,j-1] & \text{if } B[i,j]=3 \end{cases} \quad (3)$$

Note that the elements in D still indicate unnormalized path costs (as in DTW), but the decision of which transition is the best is made based on the normalized path cost (i.e. path cost per Manhattan block).

The fourth step is to identify the endpoint of the optimal alignment path. The candidate set of valid ending points is given by $E_{cand} = \{(N-1, j) \mid j = \texttt{buffer}, \ldots, M-1\} \cup \{(i, M-1) \mid i = \texttt{buffer}, \ldots, N-1\}$, which corresponds to any location in the top or right edge in Figure 1. We exclude a user-specified buffer region from the top left and bottom right corners, which ensures that the alignment path is of a certain minimum length. This buffer region helps to prevent the algorithm from selecting short, degenerate alignments paths with low normalized path cost. Given this set of candidate locations, we select the endpoint E_{best} to be

$$E_{best} = \underset{(i,j) \in E_{cand}}{\arg\min} \frac{D[i,j]}{i+j-|S[i,j]|} \quad (4)$$

where the objective function is the path cost per Manhattan block.

The fifth step is to backtrace from the selected endpoint using the backpointers in B until we reach an element $(0, j)$, $j = 0, 1, \ldots, M-1$ (on the bottom edge in Figure 1) or $(i, 0)$, $i = 0, 1, \ldots, N-1$ (on the left edge). The resulting alignment path is the final estimated alignment.

2.3 FlexDTW: Hyperparameters

In this subsection we discuss the hyperparameters in FlexDTW and our method for setting them. As mentioned previously, FlexDTW has three kinds of user-defined parameters: a set of allowable transitions, a corresponding set of transition weights, and a buffer hyperparameter that specifies a minimum path length for allowable alignment paths. Note that DTW also requires specifying a set of transitions and transition weights, so FlexDTW has one additional hyperparameter compared to DTW.

Transitions & weights. A typical set of transitions for audio alignment tasks is $\{(1, 1), (1, 2), (2, 1)\}$, which imposes a maximum warping factor of 2. This set is usually preferred to sets that include (0, 1) and (1, 0) transitions, since these transitions can lead to degenerate alignments. We will use this set of allowable transitions throughout this paper, unless otherwise noted. The associated transition weights can be set in many different ways. In FlexDTW, there is one particular setting of transition weights that is of theoretical interest: $w_1 = 2$, $w_2 = 3$, $w_3 = 3$. This setting weights each transition according to its Manhattan distance. Note that in standard DTW (where the alignment path is assumed to start at (0, 0) and end at $(N-1, M-1)$), every allowable alignment path has the same Manhattan

Piece	Files	mean	std	min	max
Opus 17, No 4	64	259.7	32.5	194.4	409.6
Opus 24, No 2	64	137.5	13.9	109.6	180.0
Opus 30, No 2	34	85.0	9.2	68.0	99.0
Opus 63, No 3	88	129.0	13.4	96.2	162.9
Opus 68, No 3	51	101.1	19.4	71.8	164.8

Table 1. Overview of the original Chopin Mazurka dataset. This is used as the source data to generate the benchmark suite. All durations are in seconds.

distance, so this setting effectively treats every path as being equally likely. It is analogous to a maximum likelihood formulation in which all possibilities are treated as equally likely a priori, and selection is made entirely based on the observations. For this reason, we recommend setting the transition weights in FlexDTW as $w_1 = W$, $w_2 = 3$, $w_3 = 3$, where W can be tuned on a validation dataset. $W = 2$ corresponds to a maximum likelihood formulation, and smaller values of W correspond to a bias towards diagonal alignment paths. In our experiments, we use $W = 1.25$, which provided optimal performance on the training set.

Buffer. The purpose of the buffer is to prevent the algorithm from selecting short, degenerate alignment paths that may have low normalized path cost. For example, silence at the end of one sequence may match silence at the beginning of the other sequence, resulting in a very short alignment path with low normalized path cost. The buffer should be interpreted as the minimum length along one sequence that an alignment path must match in order to be considered a valid path. This could simply be set manually based on knowledge of the task or data. In our case, however, our suite of benchmarks spans such a wide range of sequence lengths and alignment path lengths that a single setting is not ideal. Therefore, we determined the buffer hyperparameter in a data-dependent way for every individual query based on two considerations. First, when one sequence is much longer than the other sequence, the desired behavior is probably a subsequence alignment. In this case, we want the entire shorter (query) sequence to be matched. Second, when the two sequences are approximately the same length, much more flexibility can be afforded and an intuitive parameter is to define the minimum percentage of either sequence that must be matched. Putting these two considerations together, we recommend setting the buffer hyperparameter in the following way: for aligning two sequences of length L_1 and L_2, set $\texttt{buffer} = min(L_1, L_2) \cdot (1 - (1 - \beta) * \frac{min(L_1, L_2)}{max(L_1, L_2)})$. This sets the buffer to a fraction of the shorter sequence length, where the fraction is close to 1 when L_1 and L_2 are very different (i.e. the subsequence case) and close to β when L_1 and L_2 are approximately the same. β can thus be interpreted as the minimum fraction of either sequence that must be matched when both sequences are equal in length. We tuned β on the training set and found $\beta = 0.1$ to work well.

3. EXPERIMENTAL SETUP

In this section we describe the suite of 16 benchmarks that we use to characterize the performance of alignment algorithms under a variety of boundary conditions.

Original data. The raw source material for our benchmarks comes from the Chopin Mazurka dataset [31]. This dataset consists of numerous historic recordings of five different Chopin Mazurkas, along with beat-level annotations of each recording. All of the recordings for two of the Mazurkas (Opus 17 No. 4 and Opus 63 No. 3) were set apart for training and development, and the recordings from the other three Mazurkas were set apart for testing. Table 1 provides an overview of the dataset.

Evaluation. To evaluate alignment performance, we consider every pair of recordings of the same Mazurka. This results in $\binom{64}{2} + \binom{88}{2} = 5844$ training pairs and $\binom{64}{2} + \binom{34}{2} + \binom{51}{2} = 3852$ testing pairs. For each pair of recordings A and B, we compare the estimated alignment path against the ground truth beat timestamps in the following manner. At each ground truth beat timestamp in recording A, we compute the alignment error between the estimated corresponding timestamp in recording B (based on the predicted alignment path) and the ground truth corresponding timestamp in recording B (based on the beat annotations). We report aggregate alignment performance as an error rate indicating the percentage of alignments that have an alignment error greater than a fixed error tolerance.

Modifications: Overview. We generated synthetically modified versions of the Mazurka dataset in order to simulate a variety of boundary conditions. Each modified version of the Mazurka dataset contains the exact same number of recordings, but each recording has been modified to study a particular boundary condition. Thus, the number of training pairs and testing pairs is the same as in the original benchmark, but the audio data and corresponding annotations have been modified appropriately. Each benchmark is evaluated as described above. Below, we describe how we constructed each of the 16 benchmarks.

Full Match. The full match benchmark is the Mazurka dataset in its original unmodified form. This boundary condition assumes that both recordings start and end at the beginning and end of the piece. In Figure 1, this corresponds to an alignment path that starts in the lower left corner and ends in the upper right corner.

Subsequence. The subsequence benchmark assumes that one recording matches a subsequence in the other recording. For every pair of recordings A and B, a randomly sampled L-second interval within recording A is selected and aligned against the entirety of recording B. We construct three separate subsequence benchmarks with $L = 20, 30, 40$.

Partial Start. The partial start benchmark assumes that both recordings start together but that one recording ends early (e.g. only contains one movement). For every pair of recordings A and B, we randomly sample a number in the interval $[0.55, 0.75]$, select that percentage of recording A (starting from the beginning), and align the fragment of recording A against the entirety of recording B.

Partial End. The partial end benchmark assumes that both recordings end together but that one recording starts part way through the piece. For every pair of recordings A and B, we randomly sample a number in the interval $[0.55, 0.75]$, select that percentage of recording A at the end (i.e. starting in the middle of the recording and extending until the end), and then align the fragment of recording A against the entirety of recording B.

Partial Overlap. The partial overlap benchmark assumes that both recordings have some temporal overlap, but that one recording contains extra content before the region of overlap and the other recording contains extra content after the region of overlap. For every pair of recordings A and B, we (a) randomly sample a number in $[0.55, 0.75]$ and select that percentage of recording A starting from the beginning, (b) randomly sample a different number in $[0.55, 0.75]$ and select that percentage of recording B at the end, and then (c) align the fragment of A against the fragment of B.

Pre. The pre benchmark assumes that both recordings contain the entire piece but that one recording has a period of silence at the beginning. For every pair of recordings A and B, we prepend L seconds of silence to recording A and align it to the entirety of recording B. We construct three separate pre benchmarks with $L = 5, 10, 20$.

Post. The post benchmark assumes that both recordings contain the entire piece but that one recording has a period of silence after the piece ends. For every pair of recordings A and B, we append L seconds of silence to recording A and align it against the entirety of recording B. We construct three separate post benchmarks with $L = 5, 10, 20$.

Pre-Post. The pre-post benchmark assumes that both recordings contain the entire piece, but that one recording contains extra silence at the beginning and the other recording contains extra silence at the end. For every pair of recordings A and B, we prepend L seconds of silence to recording A, append L seconds to recording B, and then align the two recordings. We construct three separate pre-post benchmarks with $L = 5, 10, 20$.

4. RESULTS

We report experimental results with FlexDTW and several standard alignment algorithms:

- DTW1: Standard DTW with transitions of (1, 1), (1, 2), (2, 1) and corresponding weights 2, 3, 3.
- DTW2: Standard DTW with transitions of (1, 1), (1, 2), (2, 1) and corresponding weights 1, 1, 1.
- DTW3: Standard DTW with transitions of (1, 1), (1, 2), (2, 1) and corresponding weights 1, 2, 2.
- SubseqDTW1: Subsequence DTW with (query, reference) transitions of (1, 1), (1, 2), (2, 1) and corresponding weights 1, 1, 2.
- SubseqDTW2: Subsequence DTW with (query, reference) transitions of (1, 1), (1, 2), (2, 1) and corresponding weights 2, 3, 3.
- SubseqDTW3: Subsequence DTW with (query, reference) transitions of (1, 1), (1, 2), (2, 1) and corresponding weights 1, 2, 2.
- NWTW: A variant of DTW proposed in [21] that allows skip transitions (0, 1) and (1, 0), in addition to the usual (1, 1), (1, 2), (2, 1) transitions. The skip transitions incur a fixed penalty cost γ, which is a hyperparameter that we tuned on the training data.

We assessed the performance of a larger set of DTW versions (with different sets of allowable transitions and corresponding transition weights), but we only include the 3 versions with best performance to avoid overcluttering Figure 2. Of particular note, we did experiment with DTW versions that had (0, 1) and (1, 0) transitions, but always found those versions to perform much worse. Likewise, we considered other versions of subsequence DTW but only include the top 3 versions in Figure 2. The subsequence DTW systems are unique in that they are not symmetric. For these systems, we always assume that the alignment is trying to match the shorter recording against a subsequence in the longer recording. Note that all of the systems above can be used with any feature representation and distance metric. For simplicity, we use standard chroma features (as computed with default parameters in librosa) and a cosine distance metric for all systems.

Figure 2 compares the performance of FlexDTW and the above algorithms on our benchmark suite. For each system, we fixed the hyperparameter settings and evaluated its performance across all 16 benchmarks. Each panel in Figure 2 corresponds to one of the 16 benchmarks, and the different colored bars show the error rate at 200ms tolerance for different systems. On top of each colored bar, we have also overlaid two black horizontal bars indicating the error rate at 100ms tolerance (above) and at 500ms tolerance (below).

There are two things to notice about the results in Figure 2. First, the seven baseline systems only handle a subset of boundary conditions. In other words, each of the baseline systems performs well on certain benchmarks and very poorly on other benchmarks. For example, the DTW systems perform well on the fully matching benchmark (for which they are designed), but they perform terribly on the subsequence benchmarks and perform worse and worse as more silence is prepended or appended to either recording. Likewise, the subsequence DTW systems perform well on the subsequence benchmarks, but they fail on the partial overlap benchmark and have only moderate performance on the pre, post, and pre-post benchmarks. NWTW has strong performance across most benchmarks but fails completely on the subsequence and partial overlap benchmarks. All of the baseline systems completely fail on the partial overlap benchmark, since none are designed to handle that boundary condition. Second, FlexDTW has consistently strong performance across all 16 benchmarks. On all benchmarks, it has a performance that is comparable to or better than the best-performing baseline system. On the partial overlap benchmark, it is the only system that has strong performance, with an error rate that is comparable to its performance on the other benchmarks. These results demonstrate its flexibility in handling a wide range of boundary conditions.

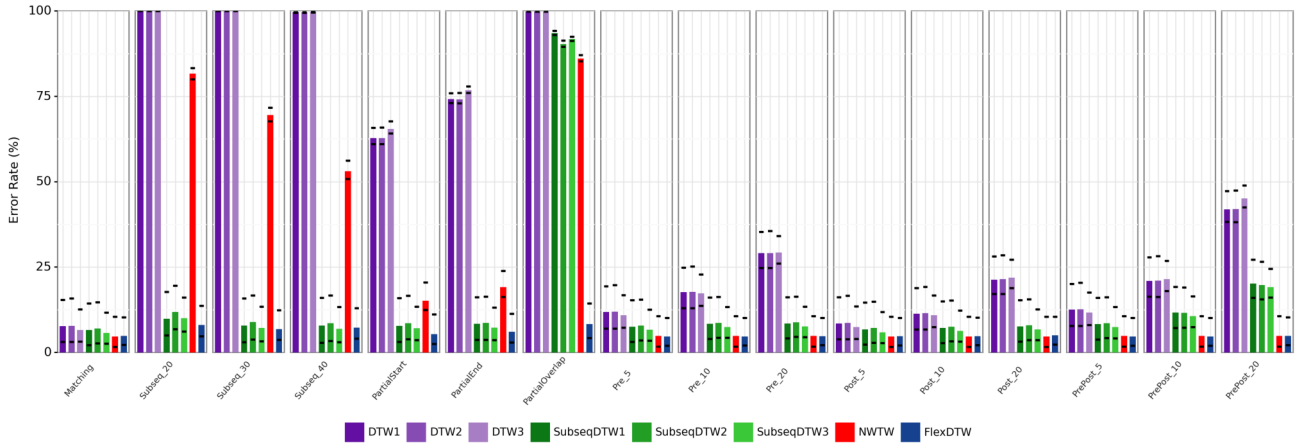

Figure 2. Performance of alignment algorithms on the 16 boundary conditions in our benchmark suite. Colored bars indicate error rate at 200ms error tolerance, and the horizontal bars indicate error rates at 100ms (above) and 500ms (below). Error rates greater than 50% are not shown.

System	1k	2k	5k	10k	20k	50k
DTW	.033	0.14	0.87	3.5	13.8	87.3
SubseqDTW	0.04	0.15	0.96	3.82	15.3	96.8
NWTW	.037	0.16	0.97	3.93	15.8	101.1
FlexDTW	.038	0.16	1.05	4.21	16.9	111.1

Table 2. Average runtime to process a cost matrix of size $N \times N$. Columns indicate different sizes N, and rows indicate different systems. Each reported number is an average over 10 trials, and times are expressed in seconds.

5. ANALYSIS

In this section we conduct several analyses to provide deeper insight into FlexDTW.

Table 2 compares the runtime of FlexDTW and the baseline alignment systems. We measured how long each alignment algorithm took to process a cost matrix of size $N \times N$, where N ranges from 1k to 50k. Each number in the table is an average over 10 trials. FlexDTW and NWTW were implemented in python with numba acceleration, and we used the librosa implementation for DTW and subsequence DTW (also using numba acceleration). All experiments were run on an Intel Xeon 2.40 GHz CPU. For longer sequence lengths, we can see that FlexDTW incurs a 20-25% runtime overhead compared DTW and a 10-15% runtime overhead compared to subsequence DTW. This overhead comes primarily from needing to perform a floating-point division to evaluate every candidate path.

Another drawback of FlexDTW is the additional memory overhead of storing S. We can estimate the memory overhead in the following manner. DTW requires allocating three matrices: the pairwise cost matrix $C \in \mathbb{R}^{N \times M}$, the cumulative cost matrix $D \in \mathbb{R}^{N \times M}$, and the backtrace matrix $B \in \mathbb{Z}^{N \times M}$. Assuming that C and D are matrices of 64-bit floating point numbers and B is a matrix of 8-bit unsigned integers, the total memory cost is $8NM + 8NM + 1NM = 17NM$ bytes. FlexDTW requires allocating an additional matrix S for storing the starting point locations. If the two sequence lengths are less than $2^{15} = 32768$, then S can be stored as a matrix of 16-bit integers, resulting in an extra memory overhead of $2NM$. If either sequence length is greater than 32768, then S must be stored as a matrix of 64-bit integers, resulting in an extra memory overhead of $4NM$. In summary, the memory overhead is $\frac{2NM}{17NM} \approx 12\%$ for sequence lengths less than 32768 and $\frac{4NM}{17NM} \approx 24\%$ for longer sequences.

We also investigated and identified two main failure modes of FlexDTW. The first failure mode occurs when there is extreme time warping between the two recordings. Because the (1, 1) transition is penalized proportionally less than the (2, 1) or (1, 2) transitions, the algorithm will sometimes take a “shortcut" of (1, 1) transitions to/from an edge of the cost matrix at the beginning or end of the alignment path. The second failure mode occurs when alternate matching paths are selected. For example, in the Mazurka Opus 17 No. 4, the first four measures and the last four measures match, which creates an additional matching alignment path under the flexible boundary conditions of FlexDTW.

6. CONCLUSION

We have introduced a time warping algorithm called FlexDTW that is designed to handle a wide range of boundary conditions. We artificially generate a suite of 16 benchmarks based on the Chopin Mazurka dataset, which characterizes alignment performance in a variety of boundary conditions. In all 16 boundary conditions, FlexDTW has strong performance that is as good or better than a set of widely used alignment algorithms. Compared to the librosa implementation of DTW and subsequence DTW, FlexDTW incurs a runtime overhead of 10-25% and a memory overhead of 12% for sequences less than length 2^{15} and 24% for longer sequences.

7. ACKNOWLEDGMENTS

We would like to thank Kate Perkins for her contributions in the early stages of this project. This material is based upon work supported by the National Science Foundation under Grant No. 2144050.

8. REFERENCES

[1] Y. Zhang and J. Glass, "An inner-product lower-bound estimate for dynamic time warping," in *Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2011, pp. 5660–5663.

[2] E. Keogh, L. Wei, X. Xi, M. Vlachos, S.-H. Lee, and P. Protopapas, "Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures," *VLDB Journal*, vol. 18, no. 3, pp. 611–630, 2009.

[3] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012, pp. 262–270.

[4] J. Li and Y. Wang, "EA DTW: Early abandon to accelerate exactly warping matching of time series," in *International Conference on Intelligent Systems and Knowledge Engineering*, 2007.

[5] A. Shabib, A. Narang, C. P. Niddodi, M. Das, R. Pradeep, V. Shenoy, P. Auradkar, T. Vignesh, and D. Sitaram, "Parallelization of searching and mining time series data using dynamic time warping," in *IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2015, pp. 343–348.

[6] S. Srikanthan, A. Kumar, and R. Gupta, "Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery," in *International Conference on Computer and Communication Technology*, 2011, pp. 394–398.

[7] Z. Wang, S. Huang, L. Wang, H. Li, Y. Wang, and H. Yang, "Accelerating subsequence similarity search based on dynamic time warping distance with FPGA," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2013, pp. 53–62.

[8] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul, "Accelerating dynamic time warping subsequence search with GPUs and FPGAs," in *IEEE International Conference on Data Mining*, 2010, pp. 1001–1006.

[9] C. J. Tralie and E. Dempsey, "Exact, parallelizable dynamic time warping alignment with linear memory," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2020, pp. 462–469.

[10] R. Tavenard and L. Amsaleg, "Improving the efficiency of traditional DTW accelerators," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 215–243, 2015.

[11] Y. Zhang and J. Glass, "A piecewise aggregate approximation lower-bound estimate for posteriorgram-based dynamic time warping," in *Proc. of the Annual Conference of the International Speech Communication Association*, 2011.

[12] A. Lods, S. Malinowski, R. Tavenard, and L. Amsaleg, "Learning DTW-preserving shapelets," in *International Symposium on Intelligent Data Analysis*, 2017, pp. 198–209.

[13] G. Nagendar and C. Jawahar, "Efficient word image retrieval using fast DTW distance," in *Proc. of the IEEE International Conference on Document Analysis and Recognition (ICDAR)*, 2015, pp. 876–880.

[14] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[15] F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 67–72, 1975.

[16] M. Müller, H. Mattes, and F. Kurth, "An efficient multiscale approach to audio synchronization," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2006, pp. 192–197.

[17] S. Salvador and P. Chan, "FastDTW: Toward accurate dynamic time warping in linear time and space," in *Proc. of the KDD Workshop on Mining Temporal and Sequential Data*, 2004.

[18] T. Tsai, "Segmental DTW: A parallelizable alternative to dynamic time warping," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 106–110.

[19] T. Prätzlich, J. Driedger, and M. Müller, "Memory-restricted multiscale dynamic time warping," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2016, pp. 569–573.

[20] C. Fremerey, M. Müller, and M. Clausen, "Handling repeats and jumps in score-performance synchronization," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2010, pp. 243–248.

[21] M. Grachten, M. Gasser, A. Arzt, and G. Widmer, "Automatic alignment of music performances with structural differences," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2013.

[22] M. Shan and T. Tsai, "Improved handling of repeats and jumps in audio-sheet image synchronization," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2020, pp. 62–69.

[23] S. Dixon, "Live tracking of musical performances using on-line time warping," in *Proc. of the International Conference on Digital Audio Effects*, 2005, pp. 92–97.

[24] S. Dixon and G. Widmer, "MATCH: A music alignment tool chest," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2005, pp. 492–497.

[25] R. Macrae and S. Dixon, "Accurate real-time windowed time warping," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2010, pp. 423–428.

[26] M. Müller and D. Appelt, "Path-constrained partial music synchronization," in *Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008, pp. 65–68.

[27] M. Müller and S. Ewert, "Joint structure analysis with applications to music annotation and synchronization," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2008, pp. 389–394.

[28] S. Wang, S. Ewert, and S. Dixon, "Robust and efficient joint alignment of multiple musical performances," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 11, pp. 2132–2145, 2016.

[29] S. Waloschek and A. Hadjakos, "Driftin' down the scale: Dynamic time warping in the presence of pitch drift and transpositions," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2018, pp. 630–636.

[30] D. Yang, K. Ji, and T. Tsai, "Aligning unsynchronized part recordings to a full mix using iterative subtractive alignment," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2021, pp. 810–817.

[31] C. Sapp, "Hybrid numeric/rank similarity metrics for musical performance analysis," in *Proc. of the International Conference for Music Information Retrieval (ISMIR)*, 2008, pp. 501–506.