

Worcester Polytechnic Institute Digital WPI

Major Qualifying Projects (All Years)

Major Qualifying Projects

August 2005

Improved Two-Dimensional Warping

Piotr Mardziel

Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

Repository Citation

Mardziel, P. (2005). *Improved Two-Dimensional Warping*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/3069>

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.

IMPROVED TWO-DIMENSIONAL WARPING

A Major Qualifying Project Report:

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Piotr Mardziel

Date: August 24, 2004

Approved:

Professor Carolina Ruiz, Major Advisor

Professor Sergio A. Alvarez (Boston College), Co-advisor

Abstract

The well-known dynamic programming time warping algorithm (DTW) provides an optimal matching between 1-dimensional sequences in polynomial time. Finding an optimal 2-dimensional warping is an NP-complete problem. Hence, only approximate non-exponential time 2-dimensional warping algorithms currently exist. A polynomial time 2-dimensional approximation algorithm was proposed recently. This project provides a thorough analytical and experimental study of this algorithm. Its time complexity is improved from $O(N^6)$ to $O(N^4)$. An extension of the algorithm to 3D and potential higher-dimensional applications are described.

Acknowledgments

I would like to thank and recognize Professor Carolina Ruiz, for her continued support, guidance, and dedication throughout the completion of this project. I would also like to thank Professor Sergio A. Alvarez of Boston College for his expertise, suggestions, and additional guidance. Finally I would like to thank Rick C. Petty for his suggestions and help during the polishing of this document.

Contents

1	Introduction	1
2	Background	2
2.1	Distance and Similarity	2
2.1.1	Distance Between Two Points	2
2.1.2	Distance as Similarity of Features	3
2.1.3	Distances	4
2.1.4	Complex Objects	6
2.1.5	Problems with Features	9
2.2	Dynamic Time Warping	13
2.2.1	The Warping Problem	13
2.2.2	Constraints	15
2.2.3	Previous Stages	16
2.2.4	Tracing the Warping Path	17
2.3	2D Dynamic Warping	18
2.3.1	2DDW	19
2.3.2	Constraints	20
2.3.3	Cumulative Distance Matrix	20
2.3.4	Problematic Constraints	22
2.3.5	Distance	22
2.3.6	Tracing the Mapping Relation	22
3	Methodology	24
3.1	1D Dynamic “Time” Warping	24
3.1.1	Cumulative Distance Matrix	24
3.1.2	Mapping	25
3.1.3	Previous Stages	25
3.2	2DDW	25
3.2.1	Cumulative Distance Matrix	26
3.2.2	Enumerating Stages	27
3.2.3	Mapping	27
3.2.4	Previous Stages	27
3.2.5	Frontier	28
3.2.6	Frontier Distance and Double Counting	29
3.2.7	Performance Improvement	31
3.3	3D Extension	33
3.3.1	Stages	33
3.3.2	Pre-Generation	34
3.3.3	Frontier	35

3.3.4	Mapping	36
3.3.5	Problems	36
3.4	ND Prospects	39
3.5	Unwarped Distance	39
4	Results	41
4.1	Experiment Setup	41
4.1.1	Image Sources	42
4.1.1.1	Face Images	42
4.1.1.2	Number Images	42
4.2	Implementation Tests	42
4.3	2DDW Continuity	45
4.4	Performance Improvement	47
4.5	Distance Normalization	52
4.6	Facial Recognition	57
4.6.1	Clustering	58
4.6.2	Classification	60
4.6.2.1	Random Models	61
4.6.3	Problems	62
4.6.3.1	Rotation	63
4.7	Text Recognition	65
4.7.1	Digital Letters	66
4.7.1.1	Clustering	67
4.7.1.2	Classification	69
4.7.2	Handwritten Numbers	70
4.7.2.1	Clustering	71
4.7.2.2	Classification	73
5	Conclusions	76
5.1	2DDW	76
5.1.1	Problems	76
5.1.2	Improvement	76
5.1.3	Higher Dimensional Extensions	77
5.2	Applications	77
5.2.1	Facial Recognition	77
5.2.2	Text Recognition	78
5.3	Future Work	78
A	Specialized Notation	82

List of Tables

1	Initial Tests	42
1	Initial Tests	43
1	Initial Tests	44
2	Continuity Tests	45
2	Continuity Tests	46
3	Original Time	47
3	Original Time	48
3	Original Time	49
4	New Time	49
4	New Time	50
5	Normalization Results	52
5	Normalization Results	53
5	Normalization Results	54
5	Normalization Results	55
6	Face Images	57
6	Face Images	58
7	Face Clusters	59
8	Face Classification Folds	60
9	Face Classification Folds with Random Models	61
9	Face Classification Folds with Random Models	62
10	Face Orientation Results	64
10	Face Orientation Results	65
11	Letter Images	66
11	Letter Images	67
12	Letter Clusters	67
13	Letter Classification Folds	69
14	Number Images	70
14	Number Images	71
15	Number Clusters	71
15	Number Clusters	72
16	Number Classification Folds	73
16	Number Classification Folds	74
17	Specialized Notation	83

List of Figures

1	Distance Between 2 Points on a 1-Dimensional Scale	2
2	Distance Between 2 Points on a 2-Dimensional Plane	3
3	Object A (Toyota Prius)	4
4	Object B (VW Beetle)	4
5	Object C (Cadillac Escalade)	5
6	Distance Between Two Objects Based on their Numeric Features	5
7	Complex Object A (Piotr’s Face)	6
8	Complex Object B (Piotr’s Brother’s Face)	7
9	Complex Object A (Piotr’s Face) with Features	8
10	Complex Object B (Piotr’s Brother’s Face) with Features	8
11	Object (Mountain Range) A	9
12	Object (Mountain Range) B	9
13	Object (Mountain Range) A with Features	10
14	Object (Mountain Range) B with Features	10
15	Possible Mapping Between Features of Two Mountain Ranges	12
16	Another Possible Mapping Between Features of Two Mountain Ranges	12
17	An Example Time Series A	13
18	An Example Time Series B	14
19	The Cumulative Distance Matrix	15
20	The Stages in a Cumulative Distance Matrix	16
21	The Warping Path	18
22	The Optimal Warping Path	19
23	2D Data Example A	20
24	2D Data Example B	20
25	Stage and Frontier in 2DDW	21
26	Warping Path in 2DDW	23
27	Matches Introduced on Frontier by Stage 5	23
28	Frontier and the Shared Element	30
29	Mapping on Frontier Introduced by a Stage and the Double Counted Element	31
30	Stages in 3D Warping	33
31	Frontier in 3D Warping	35
32	Problem in 3D Warping	38
33	Broken Continuity Example	46
34	Broken Continuity Example #2	47
35	Run Time vs. N	51
36	Run Time vs. N^6	51
37	Run Time vs. N^4	52
38	Unnormalized Distance vs. Source Image Width	55

39	Unnormalized Distance vs. Source Image Total Size	56
40	Unnormalized Distance vs. Cumulative Distance Matrix Size	56
41	Unnormalized Distance vs. Mapping Size	57
42	Distance vs. Face Angle	65
43	Zero Distance Map Example: b to f	68

List of Algorithms

1	1D Dynamic “Time” Warping Algorithm	24
2	1D Cumulative Distance Matrix Filler	25
3	1D Mapping “alignment” Tracer	25
4	1D Previous Stages Generator	26
5	2D Dynamic Warping Algorithm	26
6	2D Cumulative Distance Matrix Filler	27
7	ND Stages Enumerator	28
8	2D Mapping “alignment” Tracer	29
9	2D Previous Stages Generator	29
10	2D Frontier Retriever	30
11	2D Frontier Expansion Distance	31
12	2D PrevD Pre-generator	32
13	Improved 2D Cumulative Distance Matrix Filler	32
14	3D Cumulative Distance Matrix Filler	34
15	ND Previous Stages Generator	34
16	3D PrevD Pre-generator	35
17	3D Frontier Retriever	36
18	3D Frontier Expansion Distance	36
19	3D Mapping “alignment” Tracer	37
20	Unwarped Mapping Generator	40

1 Introduction

Dynamic Time Warping is a widely used technique of time series analysis. Originally developed for speech recognition [SC78], the method is used today for a wide range of applications [BC96, KP99]. Likewise, warping in two dimensions is said to be one of the most important techniques in pattern recognition [US00a]. Methods for quantitative comparisons between visual data can lend themselves easily to a wide range of applications such as text recognition [LP92, US99].

While one-dimensional warping between time series as performed by Dynamic Time Warping (DTW) is solved efficiently in the size of the input time series [BC96], warping in two dimensions poses a much more involved problem. Exponential time algorithms for two-dimensional warping exist [LP92, US98, US00b] and only approximations of the algorithms are polynomial-time [US00a]. Furthermore, the problem of warping in two dimensions has been shown to be NP-complete [KU03].

A new warping algorithm has been recently introduced by Lei and Govindaraju [LG04]. Using Dynamic Time Warping as a basis, the algorithm handles two dimensional warping by recursively calling DTW. The time complexity of the algorithm is polynomial in the size of the 2-dimensional input. This 2-Dimensional Dynamic Warping algorithm (2DDW) is the main subject of this project.

Firstly the 2DDW algorithm is implemented for use in experimental analyses. We analyze the 2DDW algorithm with respect to accuracy and efficiency. Deficiencies in this algorithm to satisfy continuity and monotonicity constraints are pointed out and demonstrated experimentally. The algorithm's time complexity is improved from $O(N^6)$ to $O(N^4)$, where N is the size of the 2-dimensional inputs, by performing pre-computation steps. The improvement is also verified experimentally.

Further experimental analyses of 2DDW is done in two potentially application areas: facial recognition and text recognition. The algorithm's results are used in conjunction with clustering and classification algorithms and their performance over the application areas is explored. The algorithm is found sufficient for facial recognition if assisted by pre-processing filters as its sensitivity to variations in input is demonstrated. Poor results in both clustering and classification imply unsuitability for text recognition.

Further extension of 2DDW into three dimensions is described and extensions into higher dimensions are also explored. The problems emerging in greater force in higher-dimensions, however, deem the algorithm unsuitable and unreasonable for warping in realistic higher dimensional applications.

2 Background

In this section of the report we provide some motivation for “warping” in general stemming from problems in quantifying the similarity between complex objects. Background on the Dynamic Time Warping as well as an introductory explanation of algorithm is then provided. Finally we present a similar background for the problem of two dimensional warping and finish with a description of the 2D warping algorithm that is used, improved, extended, and tested throughout the rest this project. A list of special notation used throughout this section is provided in Appendix A.

2.1 Distance and Similarity

This section uses simple examples to motivate some of the methods that are subjects of this report. The problem of quantifying similarity between more and more complex objects is a central issue in warping and will be discussed in the Sections following the developed motivation.

2.1.1 Distance Between Two Points

An issue of consequence to Dynamic Warping is the measurement of distance or similarity. The main feature of the algorithms described further in this report, in fact, is to determine or compute some kind of distance or similarity measure between two multi-dimensional datasets. In this section We will discuss the general or common-place ideas of distance and similarity and how they relate to the measure produced by the algorithms in this report.

Normally when one thinks of distance it means the length of a straight line between two points. If these two points are on a 1-dimensional “scale” or Euclidean space (see Figure 1, then this length is calculated as the absolute value of the difference between the coordinates of the two points. If the points A and B were located the coordinates a and b respectively then this distance between them would correspond to $|a - b|$. Note that distance itself is symmetric as the distance between points A and B is the same as the distance between points B and A (and $|a - b| = |b - a|$).



Figure 1: Distance Between 2 Points on a 1-Dimensional Scale

In the case of a 2-dimensional space, each point is represented by two numbers which together make up each points’ coordinate. These two numbers can be interpreted as the position of the point along two perpendicular axes. These are labeled x and y in Figure 2.

If points A and B are located at coordinates $\langle A_1, A_2 \rangle$ and $\langle B_1, B_2 \rangle$ respectively, then the Euclidean distance between them is the length of the straight line connecting the two. This length is calculated using

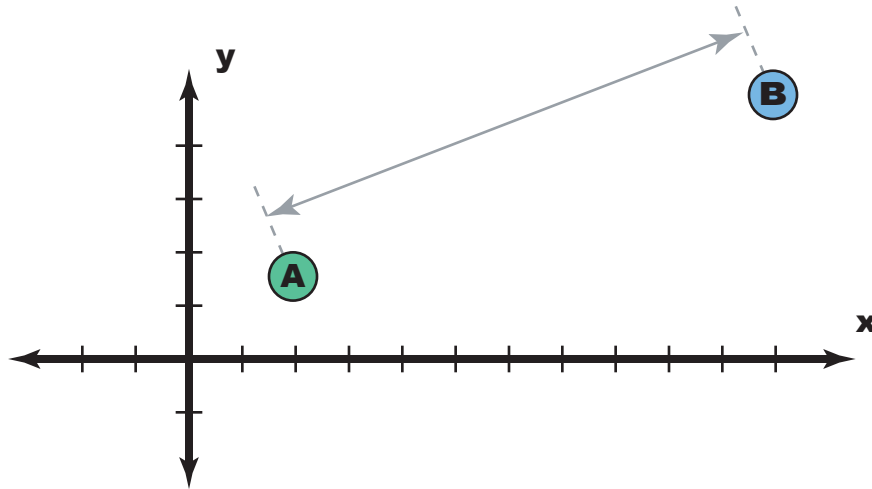


Figure 2: Distance Between 2 Points on a 2-Dimensional Plane

the Pythagorean Formula: $\sqrt{(A_1 - B_1)^2 + (A_2 - B_2)^2}$.

The Pythagorean Formula is generalizable to distances between two points in space of any whole number dimensions (including 0). If the space is of D dimensions and the points A and B are at coordinates $\langle A_1, A_2, \dots, A_D \rangle$ and $\langle B_1, B_2, \dots, B_D \rangle$ then the distance between them is $\sqrt{\sum_{d=1}^D (A_d - B_d)^2}$. Looking back at the distance between points on a 1-dimensional scale (or space) one can see that the formulas do agree. Also note that the distance between points in 0-dimensional space is always 0.

2.1.2 Distance as Similarity of Features

Imagine now that points in space are no longer the concern. Instead let us consider some objects with some number of features. Assume that these features can be described by numbers.

As a simple example lets consider cars as the objects. Each car is simply described by some numeric features such as length, height, miles per gallon, and horsepower. See Figures 3 through 5 for examples. The units of these measures are not shown as they are irrelevant here.

How can one compare these three objects to each other? The Prius is obviously more similar to the Beetle than to the Escalade but an objective measure of the similarity or difference between the cars is not present. The simplest method to achieve an objective “difference” is to treat these cars as points in some 4-dimensional space of “cars” where each car is represented by a coordinate of four numbers. In the example, point A or the Prius would be located at the coordinate $\langle 175, 58.1, 60, 76 \rangle$ if one assumed that the axis are in the order $\langle \text{length}, \text{height}, \text{mpg}, \text{horsepower} \rangle$. The three toy examples are therefore somewhere in this space (see Figure 6).

To find the distance between them one can simply use the same Euclidean formula as was described in Section 2.1.1. Using that measure of distance gives us 83.46 as the distance between the Prius and the Beetle

A TOYOTA PRIUS
length: 175
height: 58.1
mpg: 60
horsepower: 76



Figure 3: Object A (Toyota Prius)

B VW BEETLE
length: 161.1
height: 59
mpg: 24
horsepower: 150

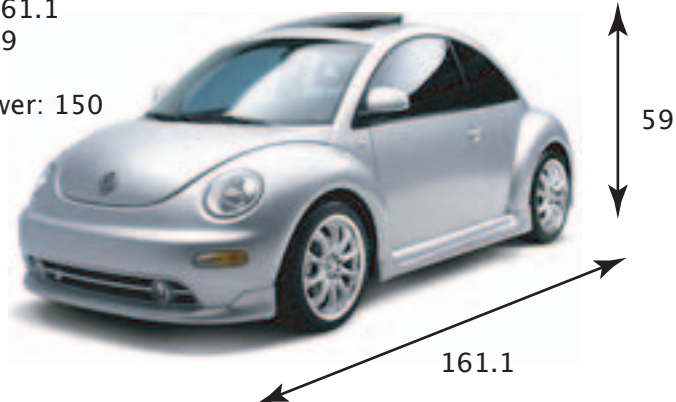


Figure 4: Object B (VW Beetle)

and 225.60 as the distance between the Prius and the Escalade. In reality one would probably weigh the various features to make some more important than others. This might especially necessary in the example because the units of the features are not all the same (inches, miles/gallon, and horsepower). In this toy example, the simplistic unweighted measure does already provide an objective measure of distance between the car models.

2.1.3 Distances

At this point it is important to mention that there really is no spatial relationship between the various features of cars in the previous Section (one could potentially argue about length and height though) and

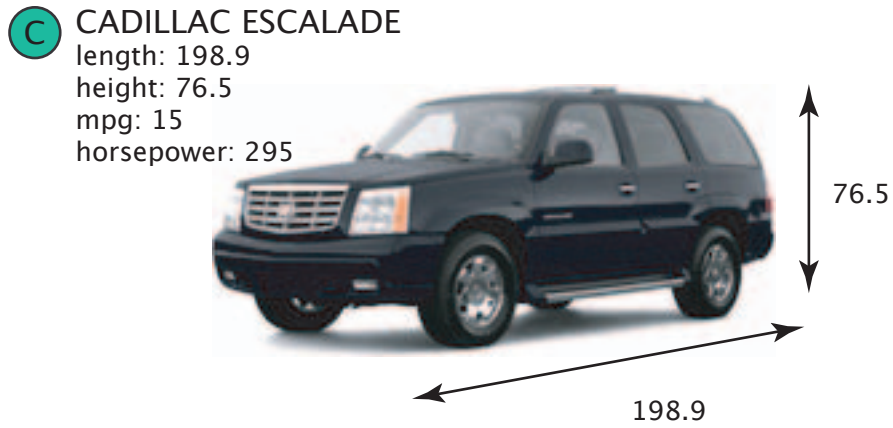


Figure 5: Object C (Cadillac Escalade)

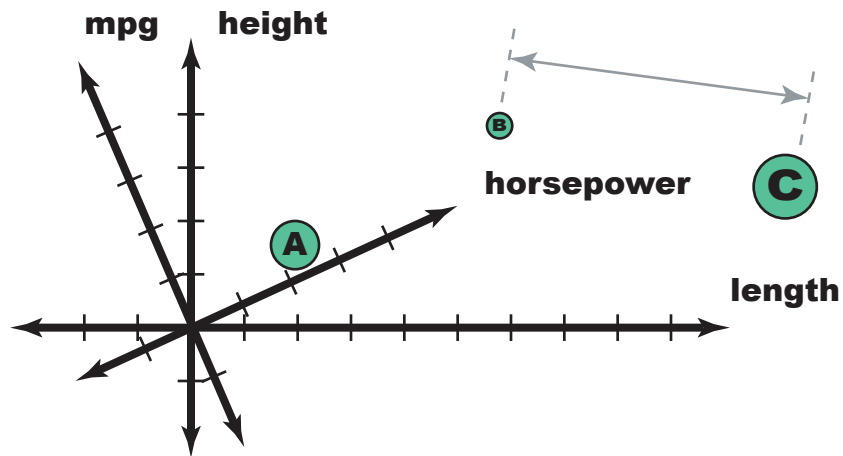


Figure 6: Distance Between Two Objects Based on their Numeric Features

hence there is no need for the Pythagorean Formula as a measure of distance. It is not the only meaningful notion of distance (although it does have special relevance to Euclidean space).

All measures of “distance”, however, have things in common. Assuming x , y , and z are objects of some type then any notion of distance d observes the following four axioms:

1. $d(x, x) = 0 \forall x$
2. $d(x, y) > 0 \forall x, y \text{ s.t. } x \neq y$
3. $d(x, y) = d(y, x) \forall x, y$

$$4. d(x, y) \leq d(x, z) + d(z, y) \quad \forall x, y, z$$

All of the distance measures mentioned so far follow these four axioms. $|x - y|$, $(x - y)^2$, and even the generalized Pythagorean Formula: $\sqrt{\sum_{i=1}^N (A[i] - B[i])^2}$. Note that there are two types of “objects” involved here: the numbers (or “components”) that compose A and B and the 1-dimensional vectors A and B themselves. As a result two different notions of distance are seen: distance between numbers and distance between the 1-dimensional vectors. In fact if d follows the axioms of distance then so does $\sqrt{\sum_{i=1}^N d(A[i], B[i])}$ as a distance between A and B .

At this point we abandon the Pythagorean formula in favor of $\sum_{i=1}^N d(A[i], B[i])$. This notion follows the axioms but is a further departure of distance in Euclidean space which is perfectly acceptable given that the objects in the rest of this section have no relation to such spaces. Also we leave an arbitrary d function (as long as it follows the axioms) in place as we will be wholly unconcerned with the distance between components of our complex objects as opposed to distances between the complex objects themselves which is the real subject of this project.

2.1.4 Complex Objects

Having conquered everything so far, the simple measure of distance (or sum of differences) seems to be the solution to every problem. Let us consider, though, objects which are a little more complicated than points or cars with four numeric features. As an example lets use gray-scale images of faces. An image in this case is defined as a 2-dimensional matrix where each element (or pixel) in the matrix has a single integer value from 0 to 255 that specifies how bright each element is. Normally colored images contain three such values per pixel to specify color levels. Two gray-scale images are provided as examples in Figures 7 and 8.



Figure 7: Complex Object A (Piotr’s Face)

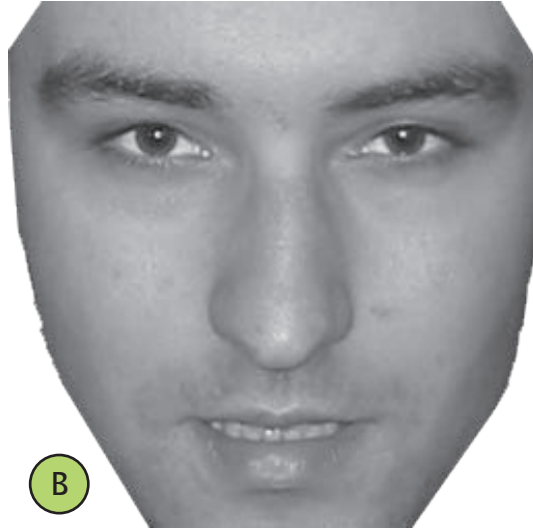


Figure 8: Complex Object B (Piotr's Brother's Face)

How would one find the distance or similarity between these two faces? The most obvious problem is that each object is now a large collection of data (pixels) that makes up the image instead of a few numeric attributes as in the car example. These objects are more “complex” for this reason. There are no specific features provided that can be used to measure distance. There are still numbers involved though (just a lot more of them).

Given the full description of each image (each pixel value) one might conjecture that using the value of each individual pixel as one feature or dimension of the objects can lead to something. If the objects are A and B , the width and height of the image are X and Y respectively, and the pixel values of A at pixel $\langle x, y \rangle$ are indicated as $A[x, y]$ then formula for such a distance would look like this: $\sum_{x=1}^X \sum_{y=1}^Y d(A[x, y], B[x, y])$. This is merely an extension of the sum of differences formula seen at the end of the previous section.

This approach is problematic, however:

- The two images must have exactly the same size. We address this point in Section 3.5.
- The features of each image (nose, eyes, etc.) must be located at the same exact locations in each of the images for a meaningful comparison.

The last point suggests that despite the fact that the data given did not come with any explicit features, there are still potential features involved. These features and their measures heavily depend on what kind of application the distance measure is used for. If it were to be used for facial recognition then the features might include distance between the eyes, width of the nose, and width of the mouth. Other applications might require different features.

If these kinds of feature measurements were available to use (see Figures 9 and 10), one can use again the usual distance formula to determine the “distance” between two faces in respect to some application which

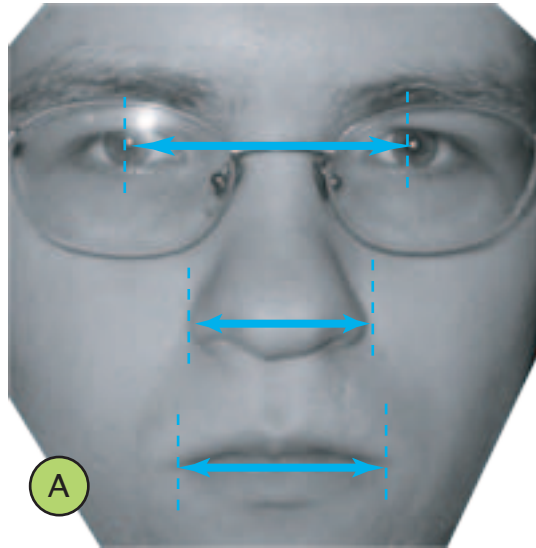


Figure 9: Complex Object A (Piotr's Face) with Features

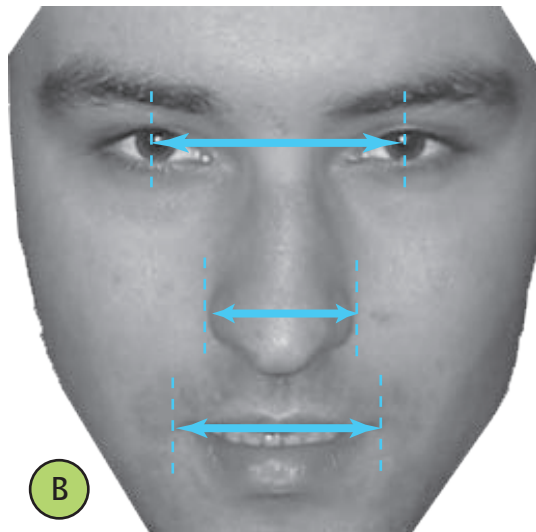


Figure 10: Complex Object B (Piotr's Brother's Face) with Features

determines which features are relevant. There are still problems:

1. Different applications require different set of features.
2. Features of the data might not always be available or easily inferable.
3. The image data must be on the same scale (ex: 1 inch / 10 pixels).

Points 2 and 3 are very problematic and are further explored in the next section.

2.1.5 Problems with Features

Things get more problematic when features are unknown, hard to determine, or even non-existent. As a final example consider objects representing mountain ranges (see Figures 11 and 12). These can be represented as a sequence of elevation values or a 1-dimensional matrix that might look like this: $\langle 0, 0, 1, 2, 4, 10, 5, 3, 5, 15, 10, 4, 1, 0 \rangle$. Note that the example “mountain ranges” are not of the same width hence it might be possible that the numeric representation of the two ranges differ in length. Because of this the usual sum of distances approach over all numbers in the objects as seen in the previous section will just not work.

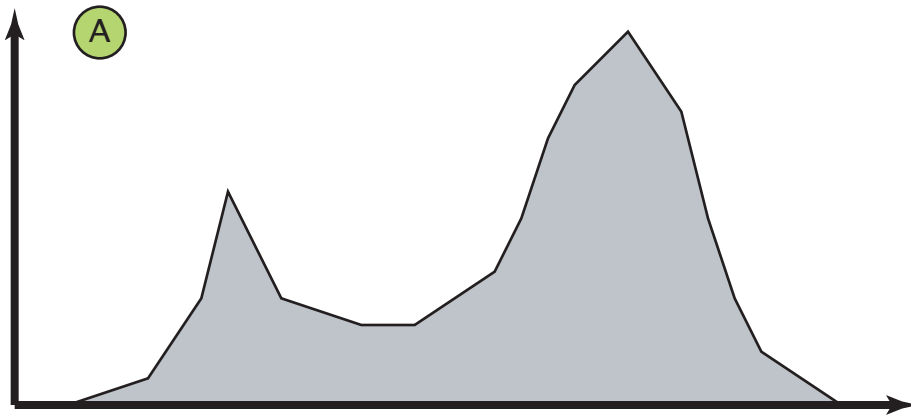


Figure 11: Object (Mountain Range) A

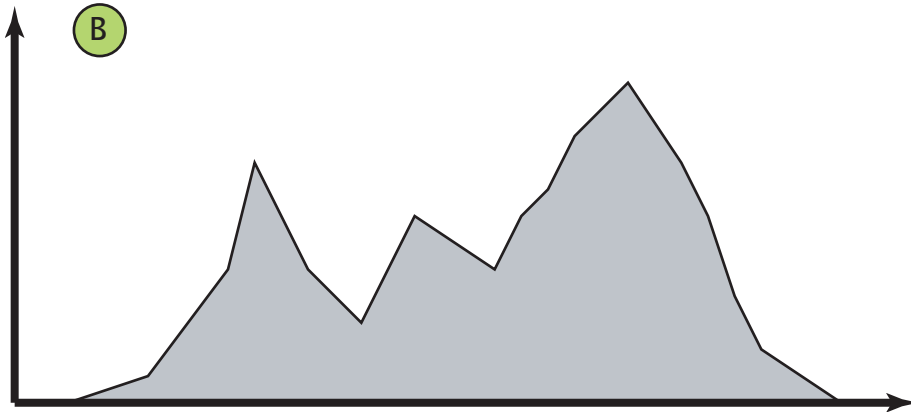


Figure 12: Object (Mountain Range) B

What might work, however, is a labeling of features onto these mountain ranges. As in the case of faces,

mountain ranges have some common characteristics: peaks, valleys, etc. Let us consider only the peaks in this example. Figure 13 has the two peaks labeled. One then can consider these as the features of mountain range *A* to be used in distance measurement. The heights of these two peaks could therefore locate this particular mountain range in a space of two-peaked mountain ranges.

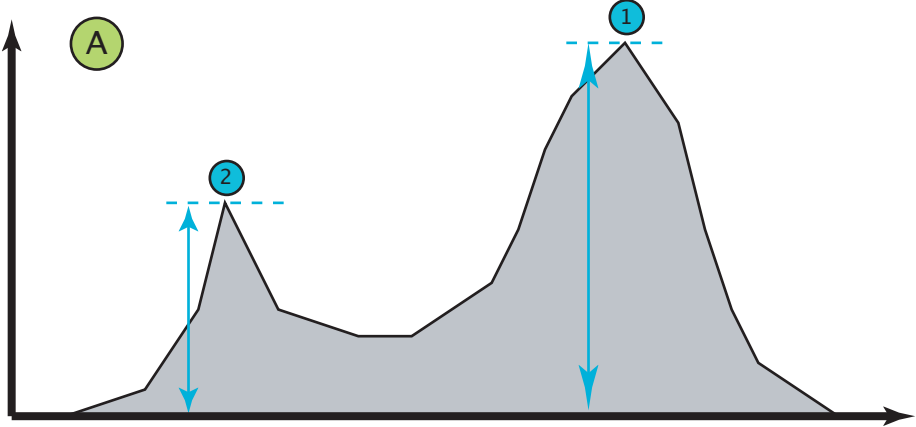


Figure 13: Object (Mountain Range) A with Features

The second mountain range, however, seems to have not two but three peaks. A sensible person might associate the rightmost peaks of both images as the same feature but the left peaks are problematic. Which of the two left peaks of range *B* do we associate with the left peak of range *A* (see Figure 14) ?

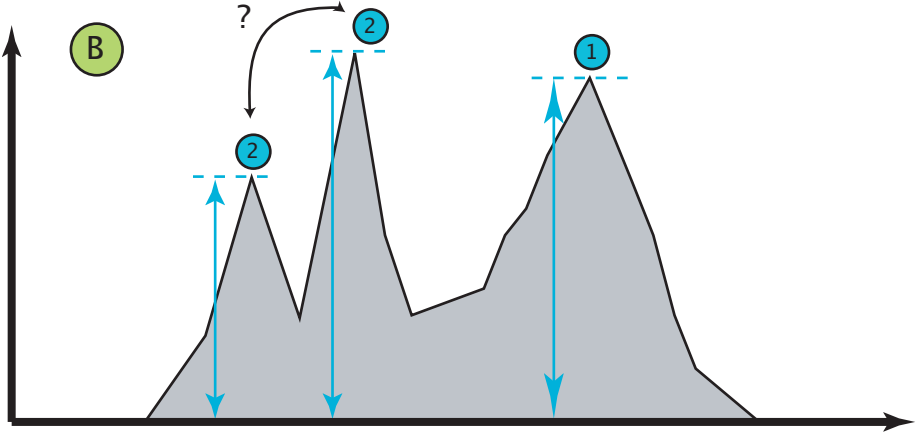


Figure 14: Object (Mountain Range) B with Features

The problem seems to be a mismatch in the number of features of the two objects (just like there was in their width). Object *A* has two peaks or features can could be represented as the tuple $\langle 30, 50 \rangle$ where the numbers represent the heights of the peaks in a left-to-right order. Likewise Object *B* can be represented in this way but using three numbers for three features: $\langle 25, 50, 45 \rangle$. If these are considered as coordinates then the first mountain range exists in an entirely different space of mountain ranges than the second range.

The only seeming solution is to disregard one of the features of the second range. The middle peak of the second range seems to be too tall to correspond to the left peak of range A so perhaps this feature should be disregarded. This kind of thinking is just delaying the problem. Consider the fact that the right peak in range A and the middle peak in range B have the same height. Perhaps it is the rightmost peak of range B that needs to be thrown out and the middle peak of range B be associated with the right peak of range A ? How about getting rid of all but one of the peaks in each range altogether?

The problem of multiple possible matching between features can be alleviated by introducing a “mapping” between features. For example let us consider an example in which the leftmost peaks of each range are “mapped” or associated with each other as well as the same for the rightmost peaks.

Previously We described each range as represented by the heights of each of its peaks. Let us call these sequences F_A and F_B where $F_A[i]$ is the height of the i^{th} peak of mountain range A . A mapping from the peaks or features of range A to range B is therefore defined as a relation M in which each element $\langle i, j \rangle$ ($i \in \{1, \dots, |F_A|\}$, $j \in \{1, \dots, |F_B|\}$) denotes that the i^{th} feature of mountain range A is associated with the j^{th} feature of mountain range B . The use of this mapping provides a good notation for reasoning about the possible association of features.

There are many such mappings possible:

1. $M = \{\langle 1, 1 \rangle, \langle 2, 3 \rangle\}$ The second peak of B is disregarded and the leftmost peaks are matched to each other as are the rightmost peaks (see Figure 15).
2. $M = \{\langle 1, 2 \rangle, \langle 2, 3 \rangle\}$ The left peak of B is disregarded and otherwise the rest is same as in #1 (see Figure 16).
3. $M = \{\langle 1, 1 \rangle, \langle 2, 2 \rangle\}$ The rightmost peak of B is disregarded. The middle peak is therefore associated with the rightmost peak of A .
4. $M = \{\langle 1, 3 \rangle, \langle 2, 1 \rangle\}$ The middle peak of B is disregarded and the leftmost peak of A is matched to the rightmost peak of B and vice versa.
5. $M = \{\}$ All the features of both mountain ranges are disregarded.

Having multiple possibilities, the problem becomes determine the “correct” mapping? “Correct” , of course, depends heavily on the purpose of the distance measure in this particular example involving mountain ranges. Assuming there was some “correct” choice of a mapping M between the features then it could be used to determine the distance between the two mountain ranges: $D = \sum_{\langle x, y \rangle \in M} d(x - y)$.

The approach taken by the method in this report is to find the matching M that would minimize the distance between the two objects. Obviously one cannot consider mappings that disregard all the features of each objects as valid in this case since this would always result in the minimum distance. Because of this problem, the methods impose various constraints on the mappings considered as valid:

- All features of each object must be considered. Furthermore each piece of data that makes up the object (in this case elevation value) is considered a feature. Also note that since there might be different number of features per objects, some features might have to be associated with multiple features of the paired object.

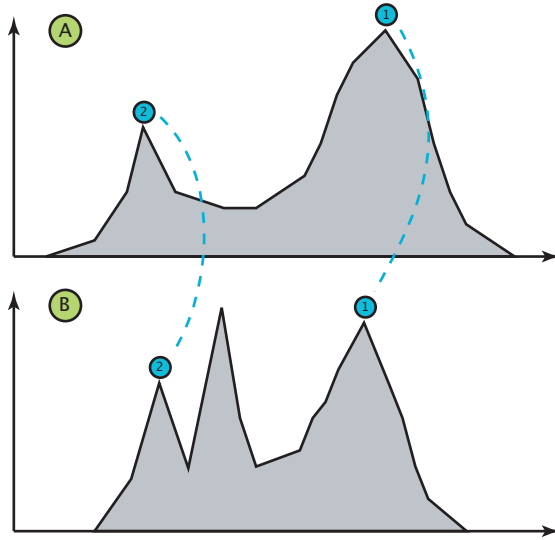


Figure 15: Possible Mapping Between Features of Two Mountain Ranges

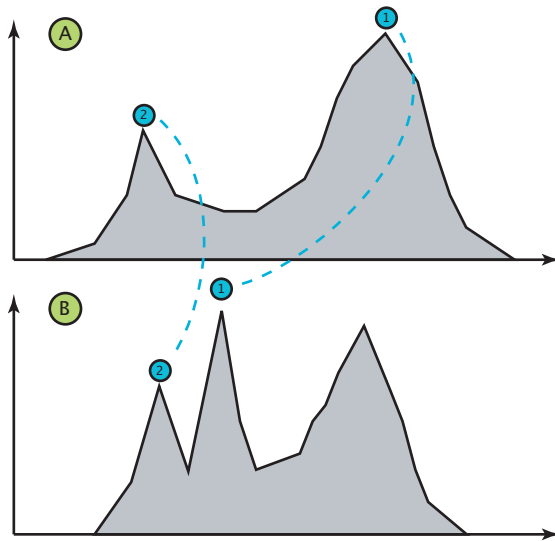


Figure 16: Another Possible Mapping Between Features of Two Mountain Ranges

- Border features map to each other. In this simple example this would mean the leftmost peaks map to each other as do the rightmost.
- Monotonicity constraint enforces uniform orientation over the entire mapping and continuity enforces an unbroken “flow” in the map. These are slightly more involved to explain here hence are explained in more detail in Sections 2.2.2.

The methods with which this mapping is determined for 1 or 2 dimensional data are described in Sections

2.2.4 and 2.3.6 while adaptation of the same methods to higher dimensions as well as the algorithm details of all the methods can be found in Section 3.

2.2 Dynamic Time Warping

The Dynamic Time Warping method is a dynamic programming approach often used to analyze time series for patterns. These time series usually represent such entities as stock prices, wave forms (of human speech), and other quantities that change over time. The intended application of the method is usually to locate previously defined series or templates in these the time series [BC96]. This requires some measure of similarity between the contents of two time series (the template and a second series). While the human eye is easily capable of finding similarities between such time series visually, there is a need for an algorithmic approach as the amount of raw data is usually too great for human processing [BC96].

The Dynamic Time Warping algorithm was first introduced for use in speech recognition in [SC78]. Discussion of the algorithm with respect to data mining is provided in [BC96]. Modifications to the algorithm for use in large datasets is described in [KP99]. The method has been very successful in wide range of applications.

Several related WPI MQPs and Master's Theses have been carried out recently on the analysis and manipulation of time series with or without the aid of Dynamic Time Warping. Methods for searching for underlying events in time series has been described in [SS04] while an algorithm for generating rules governing such temporal events is described in [Pra04] and subsequently used to explore stock market data in [HL03].

2.2.1 The Warping Problem

Given are two sequences or time series. These will be known as A and B . The sizes, or lengths, of A and B will be denoted as n and m respectively. For demonstration purposes, the following two series will be used throughout this section (see Figures 17 and 18):

$$A = \langle 0, 10, 10, 20, 15, 30, 0 \rangle$$

$$B = \langle 0, 10, 20, 10, 35, 0 \rangle$$

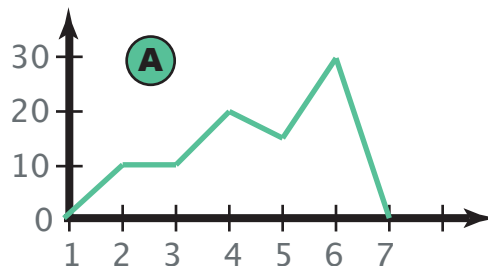


Figure 17: An Example Time Series A

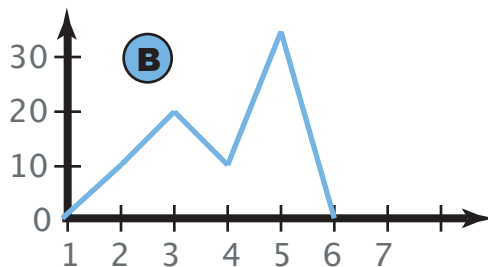


Figure 18: An Example Time Series B

The output of the method is taken to be the “distance” between the two sequences and optionally the mapping between them. The distance is the minimum possible over all mappings as seen in the previous section. We will describe the mapping in two different ways in this section. The first is described in [BC96] as a “warping path” and serves to demonstrate the dynamic programming nature of the method as well as the various constraints. The second is a notion of mapping of features as was motivated in the previous section. This second way serves as a better connection between the motivation and is, in our opinion, easier to think about in the implementation of the algorithms.

The “warping path” (denoted by W) is a sequence of tuples in the form: $\langle (i_1, j_1), (i_2, j_2), \dots, (i_p, j_p) \rangle$ where p is the size of the warping path, $i_i \in \{1..n\} \forall i$ and $j_i \in \{1..m\} \forall i$ assuming n and m are the sizes or lengths of the time series A and B respectively. The mapping is merely a relation between $1..n$ to $1..m$. It will be denoted as M . The main difference between the warping path and the mapping is that the warping path is an ordered series while the other is a set. Both however, contain the same information.

The notion of minimal distance written using the “warping path” would be denoted as follows:

$$\min_W \left\{ \sum_{k=1}^{|W|} d(A[W[k][1]], B[W[k][2]]) \right\}$$

It is important to mention, however, that not all possible warping paths produce “reasonable” warping. Constraints are imposed on the warping paths that are considered. More about these constraints is presented in Section 2.2.2.

The central feature of the dynamic time warping algorithm (DTW) is the cumulative distance matrix which will be labeled X in this section. In the case of 1-D DTW, this matrix is two dimensional. The size of this matrix is n by m . Intuitively, each cell in this matrix represents the mapping of a feature of A to one feature of B (see Figure 19). One of the parts of the value of each cell is the distance between these two mapped features or cells of the two time series.

$$X[i, j] = d(A[i], B[j]) + \dots$$

The other part of the values of the cells in the matrix refer to the “cumulative” in “cumulative distance matrix”. Not only does each cell contain the distance between the two features of the series but it also has

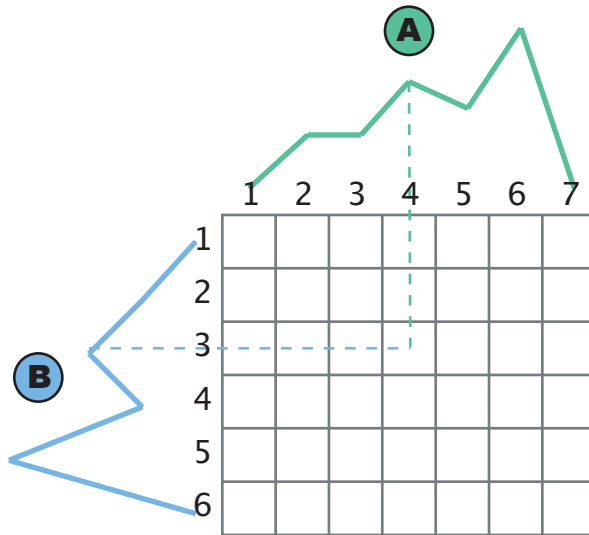


Figure 19: The Cumulative Distance Matrix

the sum of the individual distances of all the “previous” mapped features. The meaning of “previous” is the key to the relatively fast run time or complexity of the algorithm and stems from its dynamic programming nature and the “warping path”. Details concerning the notion of “previous” are described in Section 2.2.3.

2.2.2 Constraints

The coordinate of each cell in the cumulative distance matrix X is a possible “state” as related to dynamic programming. The optimal solution is a sequence of “stages” that minimizes some cost associated with the states that are “visited”. The warping path W refers to this sequence. In other words, the problem as defined in terms of dynamic programming provides some state space in which a solution is a sequence of states (termed as stages as part of a solution) that traverses the state space in some way.

Starting from some initial stage which is known before any computation given boundary constraints, the space of states is explored all the way to some final stage which is also known. In DW, the initial stage is the state $(1, 1)$ or the top-left cell in the cumulative distance matrix. The final stage is the state (n, m) or the bottom-right cell in the matrix. This constraint on the warping path is known as the “boundary constraint”.

As mentioned, the sequence spans the stages from the initial to the final. There are some constraints limiting the possible transitions between the stages. First, if a stage (i, j) is present in the warping path, then the next stage is adjacent to this coordinate or specifically (i', j') s.t. $|i - i'| \leq 1, |j - j'| \leq 1$. This is the “continuity constraint”.

Finally, there is a further reduction in the number of possible next stages. If (i, j) is a stage in the warping path, then any further stage (i', j') must obey $i \leq i'$ and $j \leq j'$. This is the “monotonicity constraint”.

The three constraints on the warping path are summarized below:

- Boundary: $W[1] = (1, 1), W[|W|] = (n, m)$ where $|W|$ is the “size” of W (see Figure 20)

- Continuity: if $W[s] = (i, j)$, $W[s + 1] = (i', j')$ then $|i - i'| \leq 1$ and $|j - j'| \leq 1 \forall s$
- Monotonicity: if $W[s_1] = (i, j)$, $W[s_2] = (i', j')$ then $i \leq i'$ and $j \leq j' \forall s_1, s_2 \text{ s.t. } s_1 \leq s_2$

Another perhaps unspoken constraint tells us that the same state cannot be visited more than once. Although none of the constraints specifically imply this but knowing that visiting each state incurs some cost, there is no way the minimum cumulative distance would visit the same state twice (or more).

2.2.3 Previous Stages

Given continuity and monotonicity, we can now determine exactly what the possible “previous” stages for any stage in the sequence are and demonstrate how the cumulative distance matrix cells are candidates for stages in the optimal “warping path”.

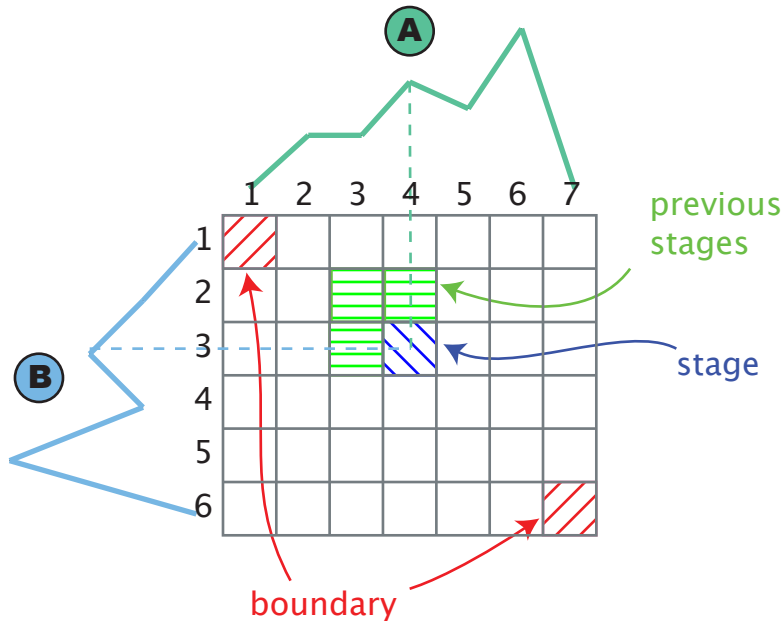


Figure 20: The Stages in a Cumulative Distance Matrix

If the state $(4, 3)$ is in the optimal “warping path” then the path must also contain $(3, 2)$, $(3, 3)$, or $(4, 2)$ according to the constraints (see Figure 20). If one makes the assumption that the “cumulative distance” for each of these possible previous stages is already known, then one needs to merely choose the smallest as the “best” previous stage. Hence the full value of the cumulative distance matrix at any point is written as the following:

$$X[i, j] = d(A[i], B[j]) + \min\{X[i - 1, j - 1], X[i, j - 1], X[i - 1, j]\}$$

Each of the previous stages has a specific interpretation depending on what the two sequences being compared are. In Section 2.1.5 the objects used as examples were mountain ranges whose features were

the elevations of their peaks. The stage (i, j) represents a mapping between the peaks (or just elevations) of mountain range A at point i to the peak of B at point j . The stages previous to (i, j) acquire these interpretations:

- $(i - 1, j - 1)$: The previous features of both mountain ranges match as well.
- $(i, j - 1)$: The i^{th} peak of A not only matches the j^{th} peak of B but also the $(j - 1)^{th}$, hence this i^{th} feature is “expanded” into more than one feature of B .
- $(i - 1, j)$: The j^{th} peak of B not only matches the i^{th} peak of A but also the $(i - 1)^{th}$, hence this j^{th} feature is “expanded” into more than one feature of A or alternatively, several features of A are “contracted” into one feature of B .

The terms of expansion (and contraction) make more sense if the objects considered are time series and the expansion is actually an expansion of time. This is the origin of the term *Time Warping*.

In this manner the matrix acquires an important characteristic. If W is a minimal warping path of size $|W|$ that ends in the stage (i, j) then $X[i, j] = \sum_{k=1}^{|W|} d(W[k])$. The converse is also true. Once the cumulative distance matrix is fully known then the value of the minimal warping path can be retrieved simply from $X[m, n]$. The task, therefore, is the construction of the cumulative distance matrix.

Determining the values in the cumulative distance matrix is in fact very simple. If one considers X as a recursive function then the definition would appear as follows:

$$X(i, j) = d(A[i], B[j]) + \min\{X(i - 1, j - 1), X(i, j - 1), X(i - 1, j)\}$$

Base cases are now necessary:

$$X(0, j) = +\infty$$

$$X(i, 0) = +\infty$$

$$X(1, 1) = d(A[1], B[1])$$

The intended meaning of the first two base cases is to make mappings between non-existent elements of A and B not be considered. The last base case stems partially from the border constraint but in this case makes sure that the value of $X(1, 1)$ does not involve the minimum of three positive infinities.

Finally to determine the cumulative distance matrix values one can use the recursive definition above: $X[i, j] = X(i, k)$. An algorithmic method of filling in the matrix is described in Section 3.1.1. Since the task involves merely filling in the matrix, the complexity of the problem is $O(nm)$ or if one assumes both input series are in the order of N then the complexity is $O(N^2)$.

2.2.4 Tracing the Warping Path

The distance between A and B can now be determined but there is still the “warping path” or the mapping which needs to be retrieved. We know that the value of $X[n, m]$ is equivalent to the total distance of the optimal or minimal warping path and we can use this to “trace” this path.

Starting with the border constraint, $W[|W|] = (n, m)$, we need to decide which of the adjacent states (or matrix elements) comes before the final stage in the sequence. The previous stage is of course the one with the lowest cumulative distance matrix value. Hence if $W[i] = (i, j)$ then

$$W[i - 1] = \operatorname{argmin}_{(i', j') \in \{(i-1, j-1), (i, j-1), (i-1, j)\}} X[i', j']$$

Given this one can trace back the elements of the warping path from the final stage to the first stage. Figure 21 demonstrates one possible warping path while Figure 22 shows the optimal warping path with regard to the example data used in this section. The mapping relation is merely an unordered set of elements present in the warping path. The algorithmic description of “tracing” or creating the mapping relation is described in Section 3.2.3.

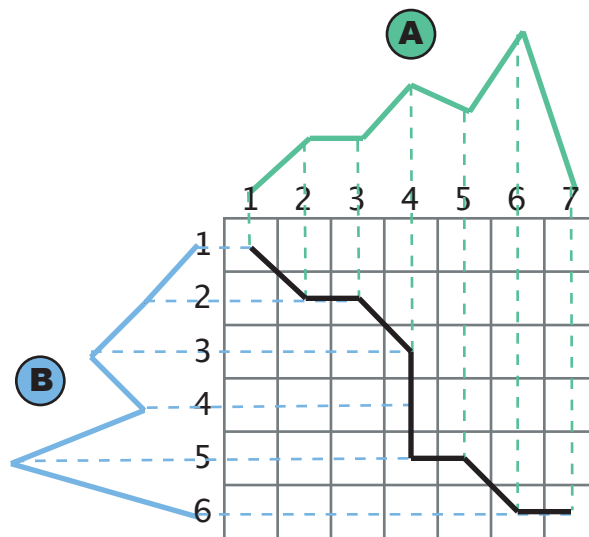


Figure 21: The Warping Path

2.3 2D Dynamic Warping

Extending the Dynamic Time Warping method to two dimensions is not a simple task. The input data is no longer a 1-dimensional series but rather a 2-dimensional matrix. This matrix can represent images or any other 2-dimensional data. A two dimensional dynamic warping algorithm could be used for a variety of applications such as text and facial recognition and is described as a fundamental technique for pattern recognition [US00a].

Various algorithms have been proposed to solve the problem. Solution are proposed in [LP92, US98] but both unfortunately exhibit exponential time complexity. It is shown in [KU03] that the 2-dimensional warping problem is NP-complete. Approximation methods have been described in [US00a].

A polynomial-time and supposedly accurate algorithm for 2D warping based on DTW is proposed in [LG04]. This algorithm, its improvement, extension, and testing are the subjects of the rest of this report.

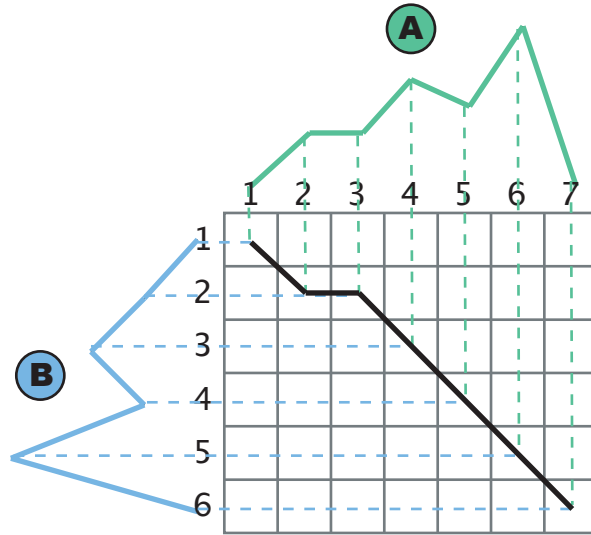


Figure 22: The Optimal Warping Path

2.3.1 2DDW

Lei and Govindaraju’s 2D Dynamic Warping method (referred to as 2DDW in the rest of the document) is a straightforward extension of the 1D method into two dimensions with one exception. The authors claim polynomial complexity while observing extended warping path constraints. We cast doubt on the claim of proper constraints in this section and provide an improvement in the complexity of the algorithm in Section 3.2.7.

The input to 2DDW are two 2-dimensional matrices A and B of sizes n_1 by n_2 and m_1 by m_2 respectively. As examples, the following two matrices will be used in this section (see Figures 23 and 24).

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 10 & 10 & 0 \\ 0 & 10 & 10 & 10 & 0 \\ 0 & 10 & 10 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 5 & 10 & 0 \\ 0 & 5 & 5 & 10 & 0 \\ 0 & 10 & 10 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

To better aid in visualization of this data, these matrices will be shown as scaled images where low values will be visible as black and high values as white (anything in between will be a shade of gray) as seen in Figures 23 and 24.

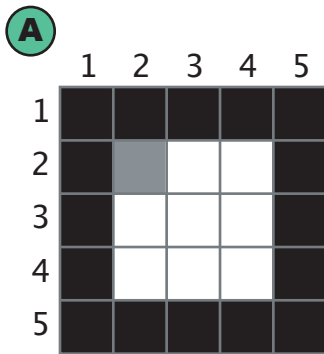


Figure 23: 2D Data Example A

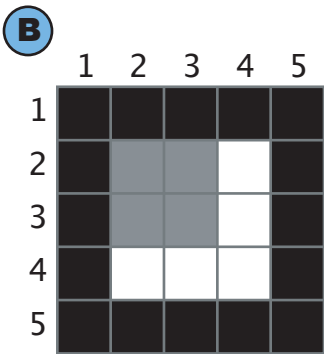


Figure 24: 2D Data Example B

The objects now have a much larger number of features than the mountain ranges or times series seen previously. The mapping between the two images is a larger relation of tuples to tuples. The problem is finding the mapping M that minimizes $\sum_{(i,j) \in M} d(A[i], B[j])$ where $i = (i_1, i_2)$ and $j = (j_1, j_2)$. M must observe some reasonable constraints.

2.3.2 Constraints

Lei and Govindaraju seem to assume that their 2DDW method inherits continuity and monotonicity constraints directly from the 1-dimensional DTW algorithm although the authors do not attempt to describe them explicitly. The problems with constraints arise as they are only on a warping path that includes only a small subset of the total mappings between features as well as constraints internalized inside 1-dimensional subsets of the objects. This issue is further discussed in this section.

2.3.3 Cumulative Distance Matrix

The approach in 2DDW starts with a four dimensional cumulative distance matrix X . Like in DTW, the

cells of this matrix refer to a matching between some cells (features are now cells in 2D matrices) of A and B . $X[i_1, i_2, j_1, j_2]$ refers to a matching between $A[i_1, i_2]$ and $B[j_1, j_2]$.

As in DTW, the task in 2DDW is to fill the matrix, retrieve the cumulative distance at the final stage, and optionally trace back the mapping M . The matrix X is again defined much like in DTW except for one extra feature. Given a stage $s = (i_1, i_2, j_1, j_2)$, then X is defined as follows:

$$X[s] = \min_{s' \in PREV(s)} X[s'] + FD(A, B, s, s')$$

The previous stages are taken out of the equation as there are 15 of them in this case. If one thinks of the stage as a coordinate in 4-dimensional space, the previous stages are all the adjacent points with smaller individual coordinate components:

$$PREV(s) = \{(s[1], s[2], s[3], s[4] - 1), (s[1], s[2], s[3] - 1, s[4]), \dots, (s[1] - 1, s[2] - 1, s[3] - 1, s[4] - 1)\}$$

The new feature in the definition of X we denote as a “frontier” distance or FD . The frontier itself is comprised of rows and columns of the two objects stretching from the edges to the points specified by the stage (see Figure 25). The frontier is denoted as $FRONTIER(A, B, s)$ and is described in Section 3.2.5. If $(C_1, C_2, R_1, R_2) = FRONTIER(A, B, s)$ then $FD(A, B, s, s')$ is defined as follows:

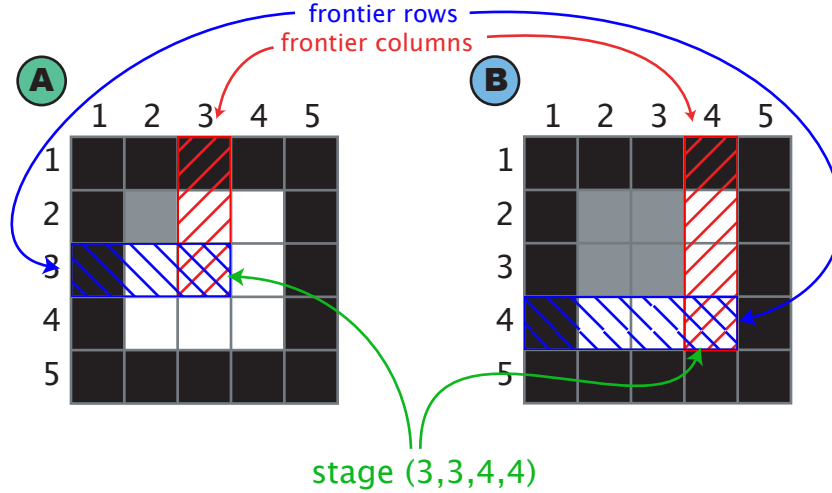


Figure 25: Stage and Frontier in 2DDW

$$\begin{cases} DTW(C_1, C_2) & \text{if } s[1] - s'[1] \neq 0 \text{ or } s[3] - s'[3] \neq 0, \\ 0 & \text{otherwise} \end{cases} + \begin{cases} DTW(R_1, R_2) & \text{if } s[2] - s'[2] \neq 0 \text{ or } s[4] - s'[4] \neq 0, \\ 0 & \text{otherwise} \end{cases}$$

The purpose of the frontier is to determine the distance between features that may not be directly on the warping path. The conditional definition is based on changes in the frontier from stage s to previous stage

s' . If $s[1] - s'[1] \neq 0$ then the previous stage s' did not have column C_1 on the frontier. Because of this the distance of the new frontier pieces (rows or columns) needs to be considered in the cumulative distance matrix.

2.3.4 Problematic Constraints

Note that the way in which the frontier pieces are compared to determine distance is by the use of the 1-dimensional dynamic time warping algorithm. The authors of [LG04] claim that the 2DDW method satisfies continuity and monotonicity because it is a recursive algorithm based on the 1-dimensional DTW algorithm which does satisfy continuity and monotonicity. While the constraints are certainly satisfied on the frontier pieces individually, there is no guarantee that they will be satisfied on the warping between the two matrices as wholes as is done in other methods such as Uchida's method in [US98]. An experimental demonstration of the lack of constraints in 2DDW can be seen in Section 4.3.

2.3.5 Distance

Given the definition of the cumulative distance matrix, one can again define a function to determine the values of the elements within in. The base cases for the initial stage at $(1, 1, 1, 1)$ need to be defined and serve the same function as they did in DTW.

$$X(1, 1, 1, 1) = d(A[1, 1], B[1, 1])$$

$$X(0, i_2, j_1, j_2) = +\infty$$

$$X(i_1, 0, j_1, j_2) = +\infty$$

$$X(i_1, i_2, 0, j_2) = +\infty$$

$$X(i_1, i_2, j_1, 0) = +\infty$$

The final distance can be returned as $X(n_1, n_2, m_1, m_2)$.

2.3.6 Tracing the Mapping Relation

The mapping relation was introduced in this report as an alternative to the use of the warping path as a way of determining matched features because of the problems one encounters in attempts to use the warping path to trace the feature matching in 2DDW. While the warping path does contain some matches between features, it only includes a small subset of the total features matched (see Figure 26).

To get the whole picture, one needs to retrieve the matches made in the frontier pieces. As the distances between frontier pieces are determined, the mappings between the pieces are implied. These need to be stored to form the complete mapping relation M (see Figure 27). The details of this process can be seen in Section 3.2.3.

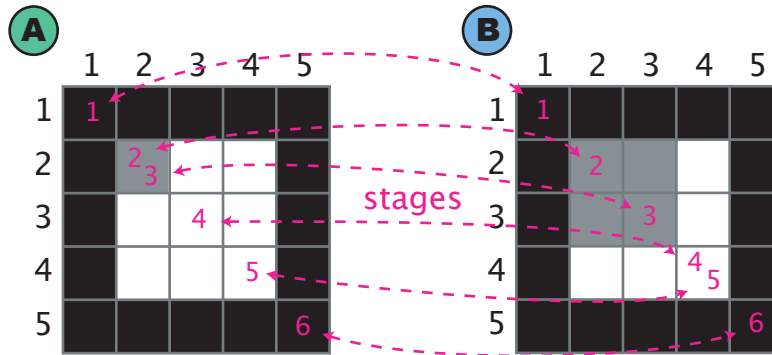


Figure 26: Warping Path in 2DDW

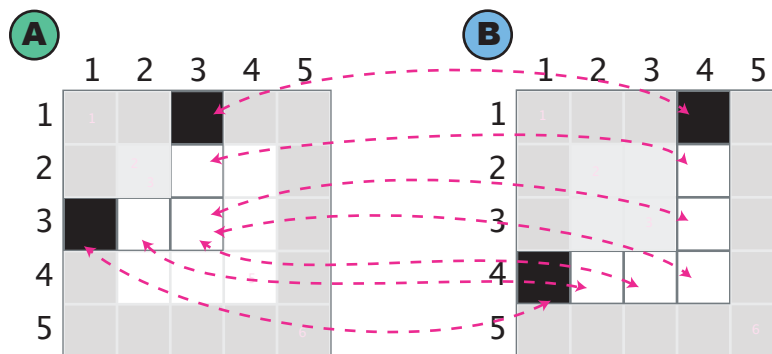


Figure 27: Matches Introduced on Frontier by Stage 5

3 Methodology

In this section we provide the exact algorithm descriptions of the methods discussed in the background. A reader should be familiar with the motivations and explanations provided there before reading this section. This section makes heavy use of specialized notation which can be found in Appendix A.

3.1 1D Dynamic “Time” Warping

The DTW algorithm as summarized in the previous section is split into two main components. The first is the determination of the warped distance while the second is the creation of a mapping relation. The main component and prerequisite of both tasks is the completion of the cumulative distances matrix. As a convention, the algorithm to return the warped distance will be labeled DW_1 while the algorithm to return the map will be known as $TRACE - ALIGN_1$.

Given two vectors A and B , DW_1 fills the cumulative distance matrix and returns the cell representing the final stage (see Algorithm 1). The value of the matrix at this final stage is the minimum warped distance.

Algorithm 1: 1D Dynamic “Time” Warping Algorithm

Input: Sequences A and B .

Output: The distance between the two sequences.

$DW_1(A, B)$

(1) $X \leftarrow \text{FILL-STAGES}_1(A, B)$

(2) **return** $X[||X||]$

3.1.1 Cumulative Distance Matrix

While the recursive function definition given in the background can be used to fill the matrix, a more algorithmic procedure is needed. This procedure is named $FILL - STAGES$ here. It takes the same two sequences for input as DW_1 .

The completion of the matrix X is arranged in such a way that at any stage, the values needed to determine the minimum for that state have already been computed. To accomplish this, the procedure starts with the stage $\langle 1, 1 \rangle$ then proceeds along one dimension to $\langle 1, 2 \rangle$, $\langle 1, 3 \rangle$, ..., and finally $\langle 1, |B| \rangle$ before incrementing the next dimension. Hence the next stage would include $\langle 2, 1 \rangle$, $\langle 2, 2 \rangle$, ..., $\langle 2, |B| \rangle$. The process is repeated until the final stage $\langle |A|, |B| \rangle$ is determined.

The complexity of such a loop is simply $O(|A| * |B|)$ assuming that each of the stage computations is done in constant time (see Algorithm 2). Using a simplification $|A| = |B| = N$ gives us a time complexity of $O(N^2)$. All the dimensions will be assumed to equal N in the rest of this section for the purposes of complexity analysis. The algorithm is aided by a helper to generate previous stages. This helper is extremely simple in the case of 1-dimensional DTW but gets more complicated further in this section (see Algorithm 4).

Algorithm 2: 1D Cumulative Distance Matrix Filler**Input:** Vectors A and B .**Output:** A 2D matrix representing the cumulative distances of a warping between A and B .FILL-STAGES₁(A, B)

- (1) $X \leftarrow$ a 2D matrix of shape $\langle |A|, |B| \rangle$
- (2) **for** $i \leftarrow 1$ **to** $|A|$
- (3) **for** $j \leftarrow 1$ **to** $|B|$
- (4) $X[\langle i, j \rangle] \leftarrow \text{dist}(A[i], B[j]) + \min_{p \in \text{PREV-STAGES}_1(\langle i, j \rangle)} \{X[p]\}$
- (5) **return** X

3.1.2 Mapping

The mapping relation generation requires a previously filled cumulative distance matrix X as input. The method starts by adding the final stage to the mapping relation and proceeding along minimum values of X (the warping path) and adding the stages to the map along the way. Once it reaches the initial stage $\langle 1, 1 \rangle$ there are no more previous stages and the method stops. In the worst case, the warping path will be of the size $|A| + |B|$ or $N + N$. The time complexity of this algorithm is hence $O(N)$ (see Algorithm 3).

Algorithm 3: 1D Mapping “alignment” Tracer**Input:** A 2D cumulative distance matrix X .**Output:** A mapping between the rows and columns of the matrix X .TRACE-ALIGN₁(X)

- (1) $M \leftarrow \langle 1, 1 \rangle$
- (2) $s \leftarrow \|X\|$
- (3) $N \leftarrow \text{PREV-STAGES}_1(s)$
- (4) **while** $N \neq \emptyset$
- (5) $M \leftarrow M \cup \{s\}$
- (6) $s \leftarrow \text{argmin}_{p \in N} \{X[p]\}$
- (7) $N \leftarrow \text{PREV-STAGES}_1(s)$
- (8) **return** M

3.1.3 Previous Stages

The previous stage generator for 1D DTW is simple. Given a stage $\langle i, j \rangle$ it generates a set of previous stages that intentionally guarantee the satisfaction of continuity and monotonicity. In the background section, base cases were used to stop recursion of the cumulative distance “function” but here the method simply checks for the boundaries of the matrix and returns only previous stages within the boundaries. The complexity of this algorithm is $O(1)$ as there are no loops, recursion, or further helpers (see Algorithm 4).

3.2 2DDW

The method developed by Lei and Govindaraju in [LG04] is essentially an extension of the original 1D

Algorithm 4: 1D Previous Stages Generator**Input:** A vector s of length 2.**Output:** A set of vectors representing the previous stages of the stage s .PREV-STAGES₁(s)

- (1) $S \leftarrow \emptyset$
- (2) **if** $s[1] - 1 > 0$ **then** $S \leftarrow S \cup \langle s[1] - 1, s[2] \rangle$
- (3) **if** $s[2] - 1 > 0$ **then** $S \leftarrow S \cup \langle s[1], s[2] - 1 \rangle$
- (4) **if** $s[1] - 1 > 0$ **and** $s[2] - 1 > 0$ **then** $S \leftarrow S \cup \langle s[1] - 1, s[2] - 1 \rangle$
- (5) **return** S

algorithm as described in the background. Like the 1-dimensional version, there are two main features: the distance and the mapping relation. The algorithm determines both in a way very similar to that of DTW.

Given as input are two 2-dimensional matrices A and B . The DW_2 returns the minimum warped distance between the two by computing the cumulative distance matrix and, as was done before, return the value of the final stage in that matrix (see Algorithm 5). The complexity of the algorithm is the same as the complexity of $FILL - STAGES_2$ which is described in Section 3.2.1.

The algorithm is given the name $2DDW$ in its originating paper. In this section it will also be referred to as DW_2 .

Algorithm 5: 2D Dynamic Warping Algorithm**Input:** Matrices A and B .**Output:** The “distance” between the two matrices. $DW_2(A, B)$

- (1) $X \leftarrow FILL-STAGES_2(A, B)$
- (2) **return** $X[||X||]$

3.2.1 Cumulative Distance Matrix

The strategy traversing the cumulative distance matrix does not differ greatly from 1D. The number of dimensions over which the algorithm loops, however, is great enough to generalize the procedure in a form of an iterator labeled as $ENUM - STAGES_2$. Furthermore, determining the minimum previous stage is more involved as there is frontier distance involved as described in the background. At each stage returned by the iterator, the frontier needs to be determined and the warping distances between the columns and rows of the frontier need to be calculated (see Algorithm 6). These are then included in minimum previous stage computation if the frontier of the current stage is found to be expanded in terms of each of the previous stages. This is done by a call to $EXPAND - DIST_2$ which also performs a small distance fix. Details are described in Section 3.2.6.

The number of stages returned by the iterator is in the order of $O(N^4)$. During each of the stages, the 1D dynamic warping algorithm is used ($O(N^2)$) as well as $EXPAND - DIST_2$ which runs in constant time (see Section 3.2.6). The time complexity of the algorithm is therefore $O(N^6)$. This complexity is founded on the assumption that both the previous stages generator and the frontier retrieval are done in constant time.

Note that the number of previous stages here is bounded by 15 hence the loop implied by *argmin* runs in constant time in respect to the input size.

Algorithm 6: 2D Cumulative Distance Matrix Filler

Input: Matrices A and B .

Output: A 4D matrix representing the cumulative distances of a warping between A and B .

FILL-STAGES₂(A, B)

- (1) $X \leftarrow$ a 4D matrix of shape $join(\|A\|, \|B\|)$
- (2) **foreach** $s \in$ ENUM-STAGES₂($\|X\|$)
- (3) $(R_A, R_B, C_A, C_B) \leftarrow$ FRONTIER₂(A, B, s)
- (4) $d_R \leftarrow$ DW₁(R_A, R_B)
- (5) $d_C \leftarrow$ DW₁(C_A, C_B)
- (6) $N \leftarrow$ PREV-STAGES₂(s)
- (7) $p \leftarrow argmin_{p' \in N} \{X[p'] + EXPAND-DIST_2(d_R, d_C, s - p', dist(A[s[1], s[2]], B[s[3], s[4]]))\}$
- (8) $X[s] \leftarrow X[p] + EXPAND-DIST_2(s - p, d_R, d_C, dist(A[s[1], s[2]], B[s[3], s[4]]))$
- (9) **return** X

3.2.2 Enumerating Stages

The loop construct used to enumerate the stages or cells of the cumulative distance matrix is simple enough to define for any number of dimensions. The iterator increases the first value in a coordinate (or vector) until it cannot be increased anymore at which point the second value is increased while the first is restarted. If the second value reaches its maximum, the third coordinate portion is increased and the second is restarted and so on. See Algorithm 7 for details.

3.2.3 Mapping

The mapping relation tracing is also a more involved variation of the 1 dimensional version. Again the assumption is that a filled cumulative distance matrix is provided as input. The returned map is composed of tuples. Each element of these tuples is also a tuple representing coordinates in A and B .

The starting stage is the first member of the map added. The “best” of the previous stages is then added and so on until the initial stage is reached. At each stage, the mapping between frontier areas is also incorporated into the map using the 1D tracing algorithm. Since the map returned by this 1D tracer only identifies the elements of the 2D matrices partially, it needs to be mapped to coordinates that incorporate the components that were lost when the slices were determined for the call to the 1D tracer.

In the worst case there can be at most $4N$ elements in the warping path. At each of these stages, *TRACE – ALIGN*₁ (complexity $O(N)$) as well as *FILL – STAGES*₁ (complexity $O(N^2)$) are performed hence the complexity of the algorithm is $O(N^3)$ (see Algorithm 8).

3.2.4 Previous Stages

The previous stages generator for a 4D stages is designed as a bridge to the ND version seen further in

Algorithm 7: ND Stages Enumerator**Input:** A vector s of length $2N$.**Output:** A vector of vectors S representing stages in the cumulative distance matrix ordered from $\langle 1, \dots, 1 \rangle$ to s such that $\forall t, p \in S$, if $p \in \text{PREV-STAGES}_N(t)$ then p appears before t in the list of returned vectors.ENUM-STAGES $_N(s)$

- (1) $c \leftarrow \hat{1}_{2N}$
- (2) $S \leftarrow \langle c \rangle$
- (3) $d \leftarrow 1$
- (4) **while** $c \neq s$
- (5) $c[d] \leftarrow c[d] + 1$
- (6) **while** $c[d] > s[d]$
- (7) $c[d] \leftarrow 1$
- (8) $d \leftarrow d + 1$
- (9) $c[d] \leftarrow c[d] + 1$
- (10) $d \leftarrow 1$
- (11) $S \leftarrow \text{join}(S, \langle c \rangle)$
- (12) **return** S

this section (see Algorithm 9). The cross product of a set of two elements results in a set of 16 elements of length 4 each. The vector $\langle 0, 0, 0, 0 \rangle$ is then disregarded. Each of the 15 remaining vectors is an offset from the input stage to some previous stage. To create the previous stages, each of the offsets is subtracted from the input stage to create the 15 “previous” stages of the input stage. The vector $\langle 0, 0, 0, 0 \rangle$ was disregarded as otherwise, the input stage would be returned in the list of previous stages. The list is finally checked to make sure each lies within the boundaries of the cumulative distance matrix. The set of stages that pass this condition is returned.

The complexity of this method is constant as its input is always of the same size and the computation within is bounded by 15 conditionals.

3.2.5 Frontier

The notion of frontier is introduced in the 2D version of the dynamic warping algorithm. Given a stage $s = \langle i_1, i_2, j_1, j_2 \rangle$ and matrices A and B , the frontier includes the rows and columns of both A and B spanning from their edges to the points $\langle i_1, i_2 \rangle$ and $\langle j_1, j_2 \rangle$. The algorithm is merely an encapsulation of several slice retrievals (see Algorithm 10).

The complexity of this method depends highly on how it is implemented in a real language. The retrieval of long slices within two dimensional vectors may easily be $O(N^2)$. It is assumed here, however, that whatever implementation is considered, it refers to the slices or vectors as references and does not need to manipulate the matrices. Hence the complexity of this method is assumed to be constant.

Algorithm 8: 2D Mapping “alignment” Tracer**Input:** Two 2D matrices A and B as well as the 4D cumulative distance matrix X created from them.**Output:** A mapping relation between A and B .TRACE-ALIGN₂(A, B, X)

- (1) $M \leftarrow \langle \langle 1, 1 \rangle, \langle 1, 1 \rangle \rangle$
- (2) $s \leftarrow \|X\|$
- (3) $N \leftarrow \text{PREV-STAGES}_2(s)$
- (4) **while** $N \neq \emptyset$
- (5) $(R_A, R_B, C_A, C_B) \leftarrow \text{FRONTIER}(A, B, s)$
- (6) $X_R \leftarrow \text{FILL-STAGES}_1(R_A, R_B)$
- (7) $d_R \leftarrow X_R[\|X_R\|]$
- (8) $X_C \leftarrow \text{FILL-STAGES}_1(C_A, C_B)$
- (9) $d_C \leftarrow X_C[\|X_C\|]$
- (10) $p \leftarrow \text{argmin}_{p' \in N} \{X[p'] + \text{EXPAND-DIST}_2(d_R, d_C, s - p', \text{dist}(A[s[1], s[2]], B[s[3], s[4]]))\}$
- (11) $f \leftarrow s - p$
- (12) **if** $f[1] \neq 0$ **or** $f[3] \neq 0$
- (13) $M \leftarrow M \cup \text{map}_{x \in \text{TRACE-ALIGN}_1(X_R)} \{\langle \langle x[1], s[2] \rangle, \langle x[2], s[4] \rangle \rangle\}$
- (14) **if** $f[2] \neq 0$ **or** $f[4] \neq 0$
- (15) $M \leftarrow M \cup \text{map}_{x \in \text{TRACE-ALIGN}_1(X_C)} \{\langle \langle s[1], x[1] \rangle, \langle s[3], x[2] \rangle \rangle\}$
- (16) $s \leftarrow p$
- (17) $N \leftarrow \text{PREV-STAGES}_2(s)$
- (18) **return** M

Algorithm 9: 2D Previous Stages Generator**Input:** A vector s of length 4.**Output:** An ordered set of vectors representing the previous stages of the stage s .PREV-STAGES₂(s)

- (1) $S \leftarrow \emptyset$
- (2) **foreach** $p \in \{1, 0\}^4 - \{\langle 0, 0, 0, 0 \rangle\}$
- (3) **if** $\nexists a \in s - p$ s.t. $a < 1$ **then** $S \leftarrow S \cup p$
- (4) **return** S

3.2.6 Frontier Distance and Double Counting

The frontier distance is the distance associated with the 1D warpings between the rows and between the columns of the frontier. At each stage in the *FILL – STAGES*₂ algorithm, the frontier distance of each “previous stage” needs to be consider in making the optimal decision for the “current stage”. Not all of the previous stages have an associated distance for both the rows and the columns. If the previous stage is “previous” in terms of the x coordinate, then the distance between frontier columns is part of the frontier distance (similarly for the rows when y coordinate changes).

The *EXPAND – DIST*₂ method serves to add up the two potential frontier distances as well as perform a small fix to the original algorithm. Note that the frontier rows and frontier columns share a pixel (see

Algorithm 10: 2D Frontier Retriever

Input: Two 2D matrices A and B and a vector s of length 4.

Output: Four vectors representing the rows and columns of the two matrices which are on the “frontier” of the stage s .

FRONTIER₂(A, B, s)

- (1) $R_A \leftarrow A[1 : s[1], s[2]]$
- (2) $R_B \leftarrow B[1 : s[3], s[4]]$
- (3) $C_A \leftarrow A[s[1], 1 : s[2]]$
- (4) $C_B \leftarrow B[s[3], 1 : s[4]]$
- (5) **return** (R_A, R_B, C_A, C_B)

Figure 28 for an example that uses the sample images from Section 2.3). If both the frontier row distance and frontier column distance is necessary for calculations, the shared pixel is used in two different 1D warping computations. As a result, the distance between the shared pixel of A and the shared pixel of B is double-counted.

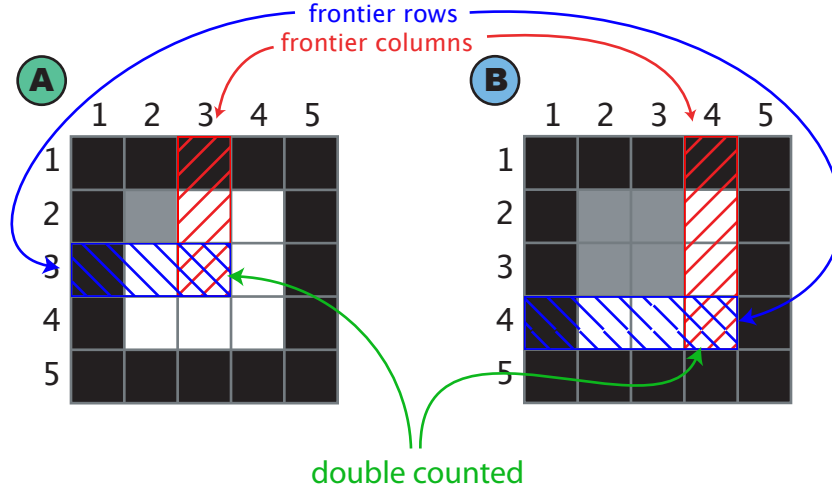


Figure 28: Frontier and the Shared Element

If one considers the problem in terms of the mapping relation and the distance that needs to be computed by DW_2 , the issue becomes clearer. DW_2 has the task of computing $\min_M \{ \sum_{\langle x,y \rangle \in M} dist(A[x], B[y]) \}$. The mapping relation, however, cannot contain two elements that perform the same exact pixel-to-pixel mapping. While the optimal mapping produces some distance value, the DW_2 algorithm can produce a different value which contains double counts (see Figure 29). In reality the algorithm tends to avoid double counts as they increase the cumulative distance more than warplings that do not involve double-counting.

To fix the problem, we simply subtract the double-counted value if it is ever double-counted (see Algorithm 11).

The time complexity of this entire method is constant.

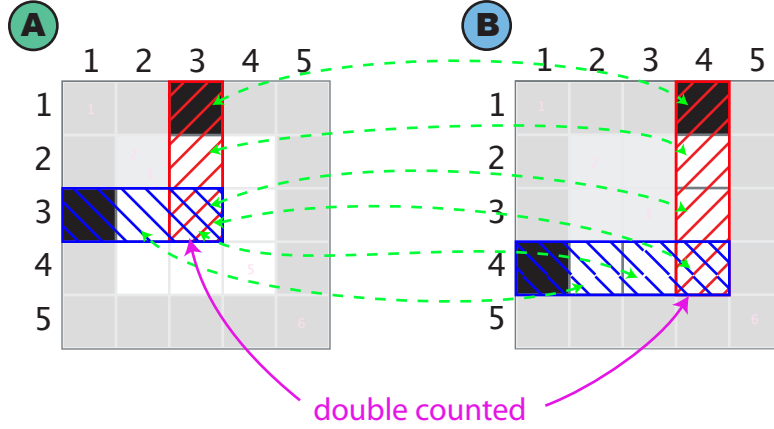


Figure 29: Mapping on Frontier Introduced by a Stage and the Double Counted Element

Algorithm 11: 2D Frontier Expansion Distance

Input: Distance between frontier rows (d_R) and distance between frontier columns (d_C), the stage change vector (f), and the distance of the shared corners of the frontier pieces (d_{shared}).

Output: A total distance to be added for the expansion of the frontier according to the vector given.

EXPAND-DIST₂(d_R, d_C, f, d_{shared})

- (1) $d \leftarrow 0$
- (2) **if** $f[1] \neq 0$ **or** $f[3] \neq 0$ **then** $d \leftarrow d + d_R$
- (3) **if** $f[2] \neq 0$ **or** $f[4] \neq 0$ **then** $d \leftarrow d + d_C$
- (4) **if** ($f[1] \neq 0$ **or** $f[3] \neq 0$) **and** ($f[2] \neq 0$ **or** $f[4] \neq 0$) **then** $d \leftarrow d - d_{shared}$
- (5) **return** d

3.2.7 Performance Improvement

Lei and Govindaraju’s method contains a small inefficiency which results in a complexity higher than it can be. The *FILL – STAGES₂* algorithm computes two 1D warped distances for every stage in the iteration (see Algorithm 6, Lines 5 and 6). It can be seen that each of these computations is repeated a large number of times with a high amount of common calculations.

The performance improvement stems from an observation of a property of the cumulative distance matrix. Given a filled 2D cumulative distance matrix X made from sequences A and B , if $X[|A|, |B|]$ is the value of the minimal warping path representing the minimal distance between the sequences then $X[i, j]$ ($i \leq |A|$ and $j \leq |B|$) is the value of the minimal warping path or minimal distance of shortened subsequences $\langle A[1], \dots, A[i] \rangle$ and $\langle B[1], \dots, B[j] \rangle$. This realization comes directly from the notion of a *cumulative* distance matrix.

The algorithm can be improved by precomputing the cumulative distance matrices for full rows and columns. The distances between smaller frontier pieces can then be retrieved in constant time. The algorithm for this pre-computation can be seen below and the modified DW_2 follows further in this section.

The runtime of the pre-generator (see Algorithm 12) is $O(N^4)$. Each of the two loops performs N^2 1D

cumulative distance matrix fills which take $O(N^2)$ time each.

Algorithm 12: 2D PrevD Pre-generator

Input: Two 2D matrices A and B .

Output: Matrices P_C and P_R each containing the cumulative distance matrices between any two columns or two rows (respectively) of A and B .

PREGEN₂(A, B)

- (1) **for** $i \leftarrow 1$ **to** $\|A\|[1]$
- (2) **for** $j \leftarrow 1$ **to** $\|B\|[1]$
- (3) $P_C[i, j] \leftarrow \text{FILL-STAGES}_1(A[i, 1 : \|A\|[2]], B[j, 1 : \|B\|[2]])$
- (4) **for** $i \leftarrow 1$ **to** $\|A\|[2]$
- (5) **for** $j \leftarrow 1$ **to** $\|B\|[2]$
- (6) $P_R[i, j] \leftarrow \text{FILL-STAGES}_1(A[1 : \|A\|[1], i], B[1 : \|B\|[1], j])$
- (7) **return** $\langle P_C, P_R \rangle$

The main part of the algorithm needs to perform the pre-computation and use the new information for the construction of the cumulative distance matrix. The complexity of DW_2 with the use of the pre-generator is now $O(N^4 + N^4)$ or $O(N^4)$ which stems from the improvement in $FILL - STAGES'_2$ (see Algorithm 13).

Once the pre-computation is done, 1D dynamic warping no longer needs to be computed in the improved matrix filler. Because of this, the contents of the computation at each stage is a reduced to a constant. The complexity of the improved matrix fill method is therefore $O(N^4)$ (see Algorithm 13).

Algorithm 13: Improved 2D Cumulative Distance Matrix Filler

Input: Matrices A and B .

Output: A 4D matrix representing the cumulative distances of a warping between A and B .

FILL-STAGES₂'(A, B)

- (1) $X \leftarrow$ a 4D matrix of shape $join(\|A\|, \|B\|)$
- (2) $\langle P_R, P_C \rangle \leftarrow \text{PREGEN}_2(A, B)$
- (3) **foreach** $s \in \text{ENUM-STAGES}_2(\|X\|)$
- (4) $d_R \leftarrow P_R[s[2], s[4]][s[1], s[3]]$
- (5) $d_C \leftarrow P_C[s[1], s[3]][s[2], s[4]]$
- (6) $N \leftarrow \text{PREV-STAGES}_2(s)$
- (7) $p \leftarrow \text{argmin}_{p' \in N} \{X[p'] + \text{EXPAND-DIST}_2(d_R, d_C, s - p', \text{dist}(A[s[1], s[2]], B[s[3], s[4]]))\}$
- (8) $X[s] \leftarrow X[p] + \text{EXPAND-DIST}_2(d_R, d_C, s - p, \text{dist}(A[s[1], s[2]], B[s[3], s[4]]))$
- (9) **return** X

The $TRACE - ALIGN_2$ algorithm does not need to be modified as it was seen that it runs in $O(N^3)$ time. Overall the improved DW_2 and $TRACE - ALIGN_2$ together have a time complexity of $O(N^4)$. The accuracy in respect to the unmodified algorithm as well as verification of the time complexity improvement can be seen in Section 4.4.

3.3 3D Extension

A straightforward extension of the algorithms into 3 dimensions does not require very much modification. As the 2DDW algorithm reduces to the use of DTW, so will the 3DDW algorithm reduce to the use of 2DDW and therefore down to DTW. In this manner, the essence of the essence of the 2DDW as an extension of DTW is maintained.

Given as input are two 3D matrices A and B . The desired output is the optimal warping distance between the two (and optionally a mapping relation).

3.3.1 Stages

The cumulative distance matrix in the case of 3DDW is 6-dimensional. The stages of computation are formed by the combination of coordinates in both the 3D objects to be warped. Although one can decide the first and last stages arbitrarily, we place the first stage at coordinate $\langle 1, 1, 1 \rangle$ in A and the coordinate $\langle 1, 1, 1 \rangle$ in B or alternatively the coordinate $\langle 1, 1, 1, 1, 1, 1 \rangle$ in the cumulative distance matrix. The last stage is combined coordinate of the opposite corners of the 3D objects.

Given a stage, a combination of the 8 surrounding cells of the stage in both the 3D objects make 63 previous stages (the current stage is not considered a previous stage of itself). Figure 30 demonstrates the various features of the stage space in 3DDW.

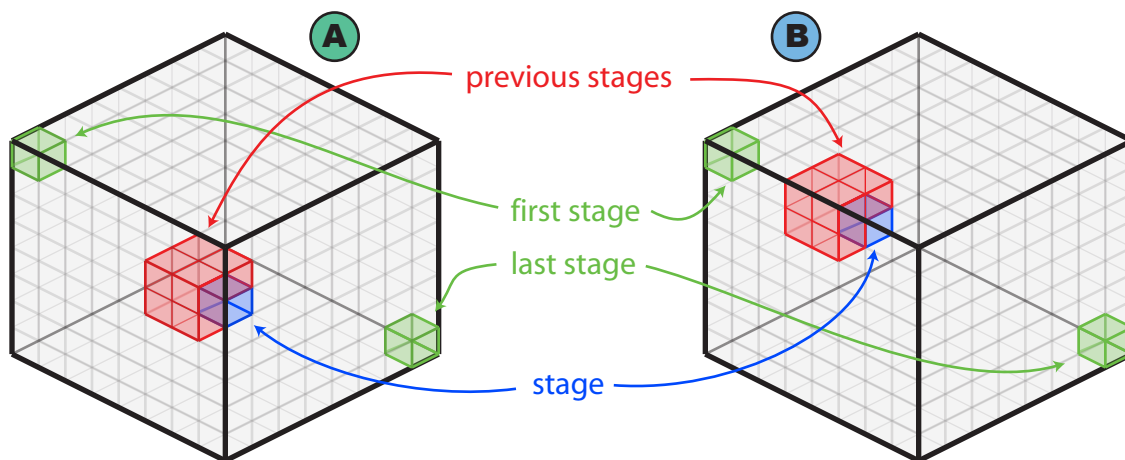


Figure 30: Stages in 3D Warping

The cumulative distance matrix filler is only slightly more complicated in 3D once the work is done by a few helpers. The enumeration of stages done by $ENUM - STAGES_3$ results in N^6 stages (assuming the input is N big in each dimensions). The hard work of pre-computing distances between the 2D slices of the 3D objects is done by $PREGEN_3$ in Section 3.3.2. The fact that the algorithm reduces the problem to warpings between 2D slices is the reason why we believe this extension is faithful to the original 2DDW algorithm. Once the pre-computation is done in $O(N^6)$ time (see Section 3.3.2), the computation in the

content of the loop iterations through the cumulative distance matrix is performed in constant time. The total time complexity is $O(N^6)$ (see Algorithm 14).

Algorithm 14: 3D Cumulative Distance Matrix Filler

Input: 3D matrices A and B .

Output: A 6D matrix representing the cumulative distances of a warping between A and B .

FILL-STAGES₃'(A, B)

- (1) $X \leftarrow$ a 6D matrix of shape $join(\|A\|, \|B\|)$
- (2) $P \leftarrow$ PREGEN₃(A, B)
- (3) **foreach** $s \in$ ENUM-STAGES₃($\|X\|$)
- (4) $d \leftarrow$ a list of three elements
- (5) $d[1] \leftarrow P[1][s[1], s[4]][s[2], s[3], s[5], s[6]]$
- (6) $d[2] \leftarrow P[2][s[2], s[5]][s[1], s[3], s[4], s[6]]$
- (7) $d[3] \leftarrow P[3][s[3], s[6]][s[1], s[2], s[4], s[5]]$
- (8) $N \leftarrow$ PREV-STAGES₃(s)
- (9) $p \leftarrow \operatorname{argmin}_{p' \in N} \{X[p'] + \operatorname{EXPAND-DIST}_3(d, s - p')\}$
- (10) $X[s] \leftarrow X[p] + \operatorname{EXPAND-DIST}_3(d, s - p)$
- (11) **return** X

The previous stages generation is general enough to extend it to any number of dimensions as seen in Algorithm 15. The algorithm runs in constant time in respect to the input other than the number of dimensions.

Algorithm 15: ND Previous Stages Generator

Input: A vector s of length $2N$.

Output: A set of vectors representing the previous stages of the stage s .

PREV-STAGES _{N} (s)

- (1) $S \leftarrow \emptyset$
- (2) **foreach** $p \in \{0, 1\}^{2N} - \{\hat{0}_{2N}\}$
- (3) **if** $\nexists a \in s - p$ s.t. $a < 1$ **then** $S \leftarrow S \cup p$
- (4) **return** S

3.3.2 Pre-Generation

The performance improvement that was done to 2DDW can also be applied to the 3D version. Instead of pre-computing 1D warpings between rows or columns, the pre-computation done in this case is between 2D slices of the objects. There are three different orientations of the slices (where the first coordinate is constant, where the second coordinate is constant, and where the third is constant). The cumulative distance matrix between any two 2D slices of A and B is computed using the improved 2D version of *FILL - STAGES*. Note that *FILL - STAGES*'₂ itself does pre-computations. See details in Algorithm 16.

Assuming again that the input objects are equally large in all dimensions (N) then the computation involved in this algorithm is centered around N^2 calls to *FILL - STAGES*'₂. It was seen in Section 3.2.1

that each of these takes $O(N^4)$ time and hence the time complexity of the 3D pre-generator is $O(N^6)$.

Algorithm 16: 3D PrevD Pre-generator

Input: Two 3D matrices A and B .

Output: Three matrices each containing the cumulative distance matrices between any two matching sides of A and B .

PREGEN₃(A, B)

- (1) $P \leftarrow$ a list of three elements
- (2) **for** $i \leftarrow 1$ to $\|A\|[[1]$
- (3) **for** $j \leftarrow 1$ to $\|B\|[[1]$
- (4) $P[1][i, j] \leftarrow$ FILL-STAGES₂'($A[i, 1 : \|A\|[[2], 1 : \|A\|[[3]], B[j, 1 : \|B\|[[2], 1 : \|B\|[[3]]]$)
- (5) **for** $i \leftarrow 1$ to $\|A\|[[2]$
- (6) **for** $j \leftarrow 1$ to $\|B\|[[2]$
- (7) $P[2][i, j] \leftarrow$ FILL-STAGES₂($A[1 : \|A\|[[1], i, 1 : \|A\|[[3]], B[1 : \|B\|[[1], j, 1 : \|B\|[[3]]]$)
- (8) **for** $i \leftarrow 1$ to $\|A\|[[3]$
- (9) **for** $j \leftarrow 1$ to $\|B\|[[3]$
- (10) $P[3][i, j] \leftarrow$ FILL-STAGES₂($A[1 : \|A\|[[1], 1 : \|A\|[[2], i], B[1 : \|B\|[[1], 1 : \|B\|[[2], j]]]$)
- (11) **return** P

3.3.3 Frontier

The frontier in 3D warping as was already implied by the recursion to 2D warping consists of 2D slices of the 3D objects. There are three possible orientations of the 2D slices hence there are not just frontier rows and columns as in 2D but rather three different slices oriented in a different way. Each of the orientations has one constant coordinate over the entire 2D slice of the 3D object (see Figure 31).

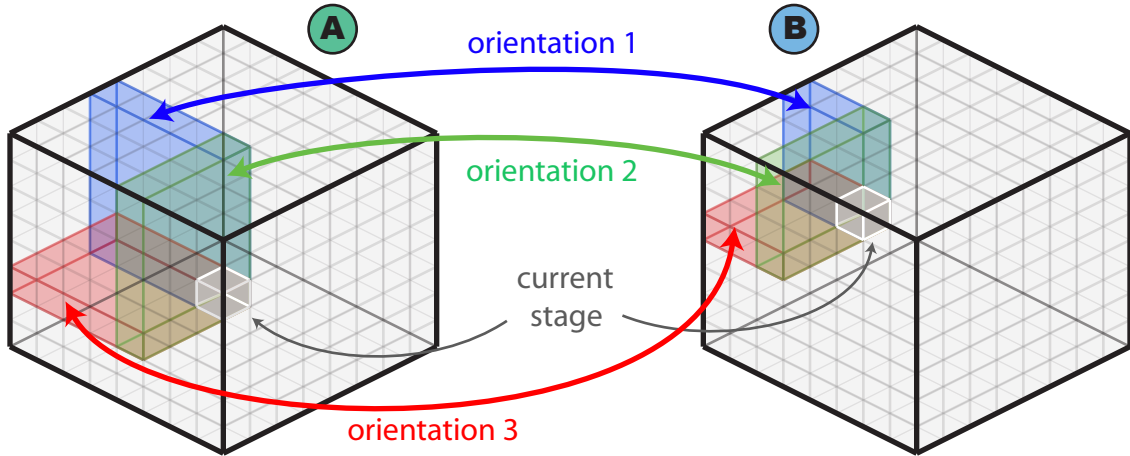


Figure 31: Frontier in 3D Warping

The frontier needs to be retrieved for use in *TRACE – ALIGN*₃. The retrieval of the frontier is done very much like it in 2D except a few more coordinates are involved. See Algorithm 17 for details. The special slice notation is defined in Appendix A.

Algorithm 17: 3D Frontier Retriever

Input: Two 3D matrices A and B and a vector s of length 6.

Output: Two vectors composed of three vectors each representing the 2D frontier slices of A and B for stage s .

FRONTIER₃(A, B, s)

- (1) $F_A \leftarrow \langle A[s[1], 1 : s[2], 1 : s[3]], A[1 : s[1], s[2], 1 : s[3]], A[1 : s[1], 1 : s[2], s[3]] \rangle$
- (2) $F_B \leftarrow \langle B[s[4], 1 : s[5], 1 : s[6]], B[1 : s[4], s[5], 1 : s[6]], B[1 : s[4], 1 : s[5], s[6]] \rangle$
- (3) **return** (F_A, F_B)

The *TRACE – ALIGN*₃ and *FILL – STAGES*₃ use a helper algorithm for determining the distance between frontier pieces that need to be included in the cumulative distance computations. Given the stage change vector f , the helper determines whether a transition from the stage previous to the current stage changes the frontier and in which direction. The difference between the current and previous stage is the change vector f . The distance between only the new frontier pieces is determined and returned.

This computation is similar to the 2D version other than the addition of one extra frontier piece (see Algorithm 18). The double counting fix is not present here. The reason for this and the problems involved are discussed in Section 3.3.5.

Algorithm 18: 3D Frontier Expansion Distance

Input: A list of distances (three) between matching frontier slices and the stage change vector (f).

Output: A total distance to be added for the expansion of the frontier according to the vector given.

EXPAND-DIST₃(d, f)

- (1) $d \leftarrow 0$
- (2) **if** $f[1] \neq 0$ **or** $f[4] \neq 0$ **then** $d \leftarrow d + d[1]$
- (3) **if** $f[2] \neq 0$ **or** $f[5] \neq 0$ **then** $d \leftarrow d + d[2]$
- (4) **if** $f[3] \neq 0$ **or** $f[6] \neq 0$ **then** $d \leftarrow d + d[3]$
- (5) **return** d

3.3.4 Mapping

The mapping relation construction method is almost identical to the 2D version with the addition of another frontier piece to be considered and the fact that the pieces are 2D instead of 1D. Because of this the method in this case uses the 2D versions of *FILL – STAGES* and *TRACE – ALIGN* (see Algorithm 19). Also note that the double-counted distance is not involved and not passed to *EXPAND – DIST*₃. The problems involved are explored in the next section.

3.3.5 Problems

One might recall the double-counting problem that existed in 2DDW and how it was solved. When

Algorithm 19: 3D Mapping “alignment” Tracer**Input:** Two 3D matrices A and B as well as the 4D cumulative distance matrix X created from them.**Output:** A mapping relation between A and B .TRACE-ALIGN₃(A, B, X)

- (1) $M \leftarrow \emptyset$
- (2) $s \leftarrow \|X\|$
- (3) $N \leftarrow \text{PREV-STAGES}_3(s)$
- (4) **while** $N \neq \emptyset$
- (5) $(\langle A_f[1], A_f[2], A_f[3] \rangle, \langle B_f[1], B_f[2], B_f[3] \rangle) \leftarrow \text{FRONTIER}_3(A, B, s)$
- (6) $X_f[1] \leftarrow \text{FILL-STAGES}_2(A_f[1], B_f[1])$
- (7) $d[1] \leftarrow X_f[1][\|X_f[1]\|]$
- (8) $X_f[2] \leftarrow \text{FILL-STAGES}_2(A_f[2], B_f[2])$
- (9) $d[2] \leftarrow X_f[2][\|X_f[2]\|]$
- (10) $X_f[3] \leftarrow \text{FILL-STAGES}_2(A_f[3], B_f[3])$
- (11) $d[3] \leftarrow X_f[3][\|X_f[3]\|]$
- (12) $p \leftarrow \text{argmin}_{p' \in N} \{X[p'] + \text{EXPAND-DIST}_3(d, s - p')\}$
- (13) $f \leftarrow s - p$
- (14) **if** $f[1] \neq 0$ **or** $f[4] \neq 0$
- (15) $M \leftarrow M \cup \text{map}_{x \in \text{TRACE-ALIGN}_2(A_f[1], B_f[1], X_f[1])} \{\langle s[1], x[1], x[2], s[4], x[3], x[4] \rangle\}$
- (16) **if** $f[2] \neq 0$ **or** $f[5] \neq 0$
- (17) $M \leftarrow M \cup \text{map}_{x \in \text{TRACE-ALIGN}_2(A_f[2], B_f[2], X_f[2])} \{\langle x[1], s[2], x[2], x[3], s[5], x[4] \rangle\}$
- (18) **if** $f[3] \neq 0$ **or** $f[6] \neq 0$
- (19) $M \leftarrow M \cup \text{map}_{x \in \text{TRACE-ALIGN}_2(A_f[3], B_f[3], X_f[3])} \{\langle x[1], x[2], s[3], x[3], x[4], s[6] \rangle\}$
- (20) $s \leftarrow p$
- (21) $N \leftarrow \text{PREV-STAGES}_3(s)$
- (22) **return** M

the frontier pieces are 1D rows or columns, they share a single pixel. Often the distance between shared pixels (one shared in A and one in B) would be counted twice in $\text{FILL} - \text{STAGES}_2$. A fix was used that subtracted the distance between these pixels at any time the double-count occurred.

The problem is much worse in 3D. There are three different frontier piece orientations. All of these share the pixel or cell at their corners. Assuming the 2D warping calculation in the frontier pieces will not double count them, the 3D version could still potentially triple-count the distance between those corner pixels at a stage where the frontier changes in all dimensions at once. This itself can still be fixed as it was in the 2D version although it would be slightly more involved.

The really big problems occur not because of the double or triple counting of that 1 pixel or cell but rather the double counting of entire 1D slices. If one looks closely at the frontier pieces in Figure 31 one can see that not only is that one corner pixel shared between all three frontier pieces but also entire 1D slices of the 3D objects are shared between the 2D frontier pieces at their edges.

Consider the highlighted 1D slice in Figure 32. This slice is part of orientation #2 and orientation #3 of

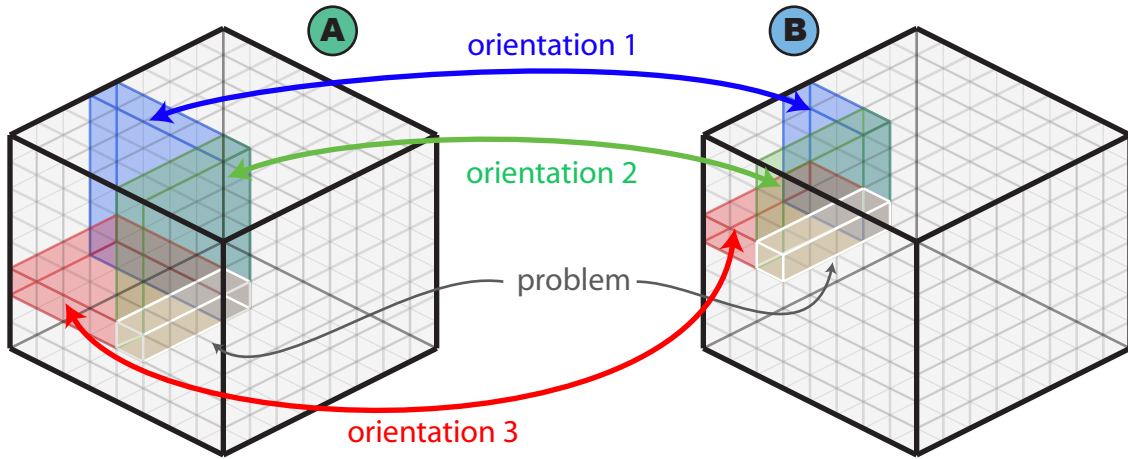


Figure 32: Problem in 3D Warping

the frontier pieces at the same time. If 2D warping between orientation #2 pieces as well as the 2D warping between orientation #3 pieces is done, the problematic slice will be part of two separate 2D warpings. This would mean that the distance between those problematic slices will be added twice in the computations but more importantly, two separate mappings between the problematic slices will be added into the final mapping relation.

Furthermore, the two mappings introduced into M are almost certainly not going to be the same (except in random cases). If one could fix the distance problem by subtracting one of the 1D warping distances, which one should be subtracted? The problem in 2D was easy to solve since the double count involved counting twice the same exact value. It is not the case here. The double count cannot be fixed without major changes to the algorithm.

Because of the issue, the 3D version of the algorithm will not produce the distance over the optimal mapping in most cases. This means that the problem solved is not the search for M with the minimum distance:

$$\min_M \left\{ \sum_{\langle x,y \rangle \in M} \text{dist}(A[x], B[y]) \right\}$$

Instead the algorithm can only provide an approximation. Also the fact that two separate warpings for the same 1D slice are considered allows the method to come up with mapping relations that are substantially more unreasonable than those in 2DDW stemming from its lack of continuity. It is possible that the constraints inside the problematic 1D slices will not even include monotonicity.

One possible solution is to enforce some constraint that would make sure that the two separate warpings for the same 1D slice are always the same. This, however, would make the algorithm significantly more complicated and overall not much like the 2D version from which it derives itself. We believe that a faithful extension of the algorithm cannot accurately solve the problem of distance minimization because of the issues discussed in this section.

3.4 ND Prospects

There is also a more subtle problem with the use of extensions of 2DDW in higher dimensions. Certainly the problem of incorrect final distance and very strange mapping relations stemming from the problem described in the previous section only get worse in higher dimensions. In 4D warping, for example, entire 2D slices can have two separate and most likely different mappings in the final map relation. The subtle problem derives from the fact that all the extensions of 2DDW into higher dimensions use DTW at their simplest steps.

The 2DDW uses the 1-dimensional DTW algorithm in order to perform essentially the entire task. One can envision a simple algorithm that does nothing but use DTW to map rows of one image to the rows of another without much other computation. The 2DDW algorithm does slightly more by performing some minimization task of determining which rows in one image should map to which rows in the other image and allowing columns to be mapped as well in some portions of the images. Overall, 2DDW is still essentially doing very little more than DTW applied multiple times. This can be seen in the problems created with the lack of proper constraints. One could call 2DDW a “pseudo 2D warping” algorithm because of deficiencies.

Despite the deficiencies, 2DDW can still be useful (as will be seen in the results). How useful can it be, however, in 3D or higher? The double counting and double mapping problems would exist in higher dimensions but the fact that any ND extension of 2DDW has really 1D DTW at its core, doing mappings between nothing more than 1D slices, demonstrates an important point. The 2D nature of 2D data might be coerced sufficiently enough for applications by 2DDW but 3D data being manipulated only using 1D slices loses a lot of its intrinsic 3D nature. Doing 4D mapping with 1D DTW at the core would certainly result in an even greater disjunction of data and result.

We theorize that extensions of 2DDW into higher dimensions are just not suitable for accurate and reasonable results.

3.5 Unwarped Distance

One final algorithm is used for comparisons of distances throughout the results. It is useful to know whether 2DDW provides an improved distance in relation to some benchmark. An unwarped distance can serve as such a benchmark. We define the unwarped distance as a distance associated with a mapping that does nothing more than map pixels from one image to the closest pixels of the other image.

Constructing such a mapping relation is extremely simple if the two images are of the same size. Often it is necessary, however, to produce a mapping between differently sized images. Most of the experiments in the results do in fact contain warpings between images of different sizes and hence such a definition of “unwarped” is useful.

Given the “unwarped” mapping relation M_u produces by Algorithm 20, the unwarped distance is simply $\sum_{\langle x,y \rangle \in M_u} dist(A[x], B[y])$. This value will be used for comparison with warped distances in the results in the next section.

Algorithm 20: Unwarped Mapping Generator

Input: Two vectors representing the sizes of two images.

Output: A mapping relation that maps coordinates in the first image to the coordinates of the second image.

UNWARPED-MAP₂(s_A, s_B)

- (1) $s_x \leftarrow s_B[1]/s_A[1]$
- (2) $s_y \leftarrow s_B[2]/s_A[2]$
- (3) $M_u \leftarrow \emptyset$
- (4) **for** $x_A \leftarrow 1$ **to** $s_A[1]$
- (5) **for** $y_A \leftarrow 1$ **to** $s_A[2]$
- (6) **for** $x_B \leftarrow \lfloor (x_A - 1) * s_x \rfloor + 1$ **to** $\lfloor x_A * s_x \rfloor$
- (7) **for** $y_B \leftarrow \lfloor (y_A - 1) * s_y \rfloor + 1$ **to** $\lfloor y_A * s_y \rfloor$
- (8) $M_u \leftarrow M_u \cup \langle \langle x_A, y_A \rangle, \langle x_B, y_B \rangle \rangle$
- (9) **return** M_u

4 Results

Various experiments of the dynamic warping algorithms were carried out and are described in this section. The first few experiments involve initial tests of the implementation, the verification of the complexity improvement, and the demonstration of the lack of constraints in the 2D dynamic warping algorithm used in this project. The rest of the experiments are designed to test potential uses of the algorithm such as facial and character recognition.

4.1 Experiment Setup

A few details of the setup of the experiment are necessary.

All the 2D objects used were provided in the PNG image file format. These images contained 24 bits per pixel (8 each for red, green, and blue channels) but during the conversion to a matrix, a filter was applied that reduced the data to 8bit gray-scale. The distance measures between any two gray-scale values is a simple absolute difference ($d(a, b) = |a - b|$ where $a, b \in \{0, \dots, 255\}$).

Two 2D objects were provided to the method in each test. The first is denoted as “source” or A and the second as “target” or B . The method then performed the required steps to arrive at a distance between the two images. In addition, the following were recorded:

- The normalized unwarped distance between the two images. See Section 4.5 for more information about distance normalization and Section 3.5 for the method in which “unwarped” is defined.
- The normalized warped distance between the two images.
- Warped image W . This is an image of the same size as the target image but contains for each pixel the gray-scale value from the source image pixels that it is mapped to. If more than one pixel of A is mapped to a single pixel of B , the warped image will contain an average of those multiple A pixels.

$$W[y] = \text{avg}_{\langle x, y \rangle \in M} A[x]$$

- Displacement image D . This is an image of the same size as the target but for each pixel, it contains brighter colors if the pixel is mapped far away from its unwarped position and darker if its mapped close to its unwarped position. If more than one mapping to a specific pixel in B exists then the value in D of this pixel is proportional to the furthest mapped pixel in A . Specifically, $D[y] = \max_{\langle x, y \rangle \in M} \{d_P(x/|A|, y/|B|)\}$, where $d_P(c_1, c_2)$ is the Pythagorean distance between points at coordinates c_1 and c_2 .

The specific value of each pixel in this displacement image is irrelevant as it is meant merely to help demonstrate the warping that must occur to arrive at the minimum warped distance.

Additionally some of the experiments feature other relevant information such as the unnormalized distances (see Section 4.5) and the algorithm timing (see Section 4.4).

All experiments were ran on FreeBSD5.4/AMD64 running an Athlon FX51 (at 2.25 GHz) with 1024MB of DDR/ECC/Registered memory.

4.1.1 Image Sources

The project's experiments using 2DDW required a variety of test data. While a large portion of the images used in the project were generated specifically for the project, some of the images were obtained from two main sources. Information about these images and their sources can be found in this section.

4.1.1.1 Face Images

The face images used in the facial recognition portion of the experiments originate from the Neural Networks for Face Recognition web page located at <http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/faces.html> which itself is a companion to a machine learning textbook by Tom Mitchell. The quarter-sized images of the first five users (alphabetically) are used in the experiments.

4.1.1.2 Number Images

The handwritten number images used in the text recognition experiments were retrieved from the NIST Handwritten Digits Database. The specific images used are the example images as seen on the NIST main page: <http://www.cs.bc.edu/~alvarez/ML/NISTDigits>.

4.2 Implementation Tests

The very first experiment is designed simply to test the implementation of the 2DDW method. The data used are small images of faces of a few different people. Each of the images is tested with each of the others. The intention is not to get anything other than verification of the algorithm. We should be able to see larger distance values for dissimilar faces and smaller values for similar faces.

Table 1: Initial Tests

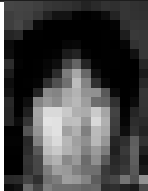
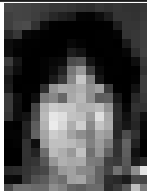


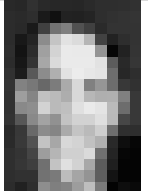
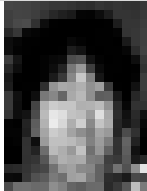
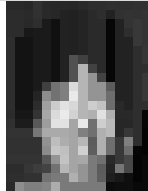

#	Source	Target	Warped	Displace	Distances
1					Unwarped: 18.926 Warped: 7.336
2					Unwarped: 57.188 Warped: 20.153

Table 1: Initial Tests


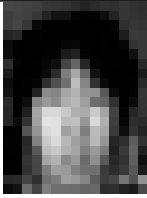










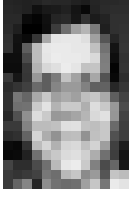
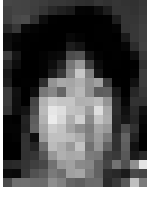


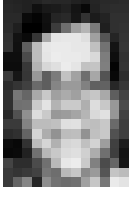
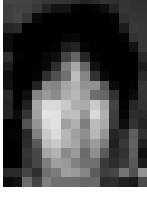



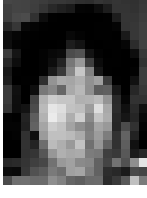



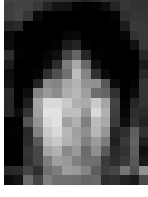
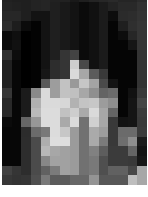
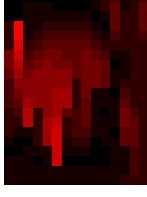





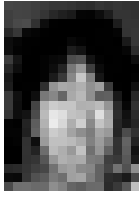

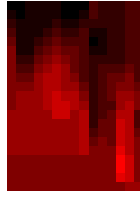

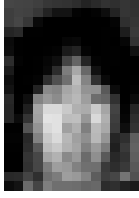
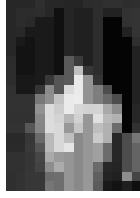




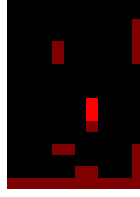


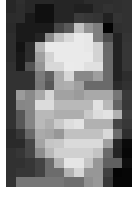



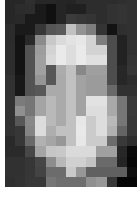

#	Source	Target	Warped	Displace	Distances
3					Unwarped: 57.837 Warped: 19.312
4					Unwarped: 31.965 Warped: 16.875
5					Unwarped: 31.462 Warped: 16.089
6					Unwarped: 56.568 Warped: 16.914
7					Unwarped: 60.360 Warped: 18.897
8					Unwarped: 53.679 Warped: 15.987
9					Unwarped: 52.520 Warped: 17.709

Table 1: Initial Tests

#	Source	Target	Warped	Displace	Distances
10					<p>Unwarped: 22.315</p> <p>Warped: 11.631</p>
11					<p>Unwarped: 55.652</p> <p>Warped: 19.759</p>
12					<p>Unwarped: 55.323</p> <p>Warped: 20.129</p>
13					<p>Unwarped: 23.485</p> <p>Warped: 6.333</p>
14					<p>Unwarped: 33.254</p> <p>Warped: 16.189</p>
15					<p>Unwarped: 31.239</p> <p>Warped: 15.285</p>

There are a few apparent observations stemming from these initial tests.

- First we can see that the warped distance is always smaller than the unwarped. This suggests that the 2DDW algorithm does do some kind of distance optimization.
- Distance between the faces of two different people is generally larger than the distance between the face images of same person. Distance between the same people all seem to measure at 12 or below

while distances between different people start at 15.

- There appears to be generally a lot less warping between the same people (as in Rows 10 and 13) than between different people (as in Rows 6, 11, and others). This, though, is partially related to the previous observation. Similar persons are “closer” together visually hence require less warping.
- In most cases, the images are so thoroughly warped that the influence of the source is not visible. This seems to have two implications: the algorithm is very well at optimizing the distance and the algorithm may be too free to warp beyond “reasonable” displacement. The second point has much to do with the next section.

Further tests using human faces was done to evaluate prospects of facial recognition using 2DDW. This is described in Section 4.6.

4.3 2DDW Continuity

The apparent lack of constraints (specifically continuity) in the 2DDW algorithm was mentioned in Section 2.3.1. This experiment is designed to verify the conjecture.

Given as input are two images which attempt to force an obvious break in continuity. The authors of [LG04] do not give any definition of continuity and monotonicity. They do, however, state that the 2DDW is continuous and monotonic since it relies on recursion to the DTW algorithm which is continuous and monotonic. One can only assume what the intended meaning of continuity and monotonicity is in two dimensions in respect to this algorithm but having the 1D DTW as an example, we produce a general statements about continuity. It is deliberately not written as mathematical expressions but rather provides the “idea” of what continuity should mean in 2D given what it means in 1D so that it is still possible to determine whether the experimental results do or do not verify the conjecture.

- Continuity. In DTW, continuity has this general intention: adjacent features of one sequence can only map to adjacent features of the second sequence. The meaning of “adjacent” in 1D means simply the features on either side of the feature in question. In 2D we simply assume that “adjacent” refers to the 8 pixels surrounding the pixel in question.

The two tests shown in Table 2 both show an apparent break in continuity. Both the tests attempt to force the warping path along the diagonal of both images. This however, did not succeed in both cases.

Table 2: Continuity Tests

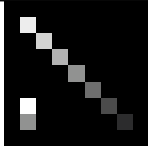
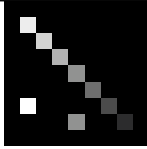
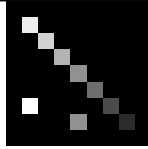

#	Source	Target	Warped	Displace	Distances
1					<p>Unwarped: 3.580</p> <p>Warped: 0.000</p>

Table 2: Continuity Tests

#	Source	Target	Warped	Displace	Distances
2					<p>Unwarped: 6.296</p> <p>Warped: 0.000</p>

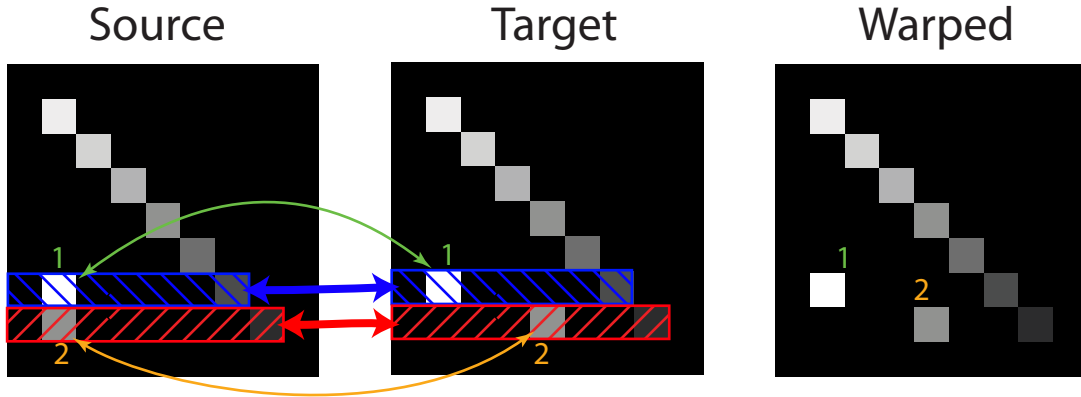


Figure 33: Broken Continuity Example

In Test 1 (see Figure 33) the warping path does indeed follow the diagonal. Because of this, the frontier implied in the optimal warping path includes rows below the diagonal and columns above it. The rows of the source and target that include the pixels label with “2” are mapped to each other and inside this mapping, the continuity is not broken. The same applies for rows with the pixels labeled “1”. If one considers the images as wholes, however, the continuity is broken. The pixels 1 and 2 in the source are adjacent but in the target they are not. Despite this fact they were mapped to each other.

If the warping path followed the diagonal in the second test, then it would include a mapping between the column of the source that includes both the pixels labeled 1 and 2 in Figure 34 and the column containing only the pixel 2 in the target image. This one dimensional mapping would produce a distance not equal to zero without broken continuity. The result of the test, however, did not show a diagonal warping path. Instead, the warping path followed second to right column of both images all the way from the bottom to the top of the images and then proceeded left to the first stage at pixel $\langle 1, 1 \rangle$. Because of this, the relevant frontier pieces include not columns but instead the rows that each contain either pixel labeled 1 or pixel labeled 2 (see Figure 34). This allows for the distance between the two to be 0 as in the first test but again shows a break in continuity.

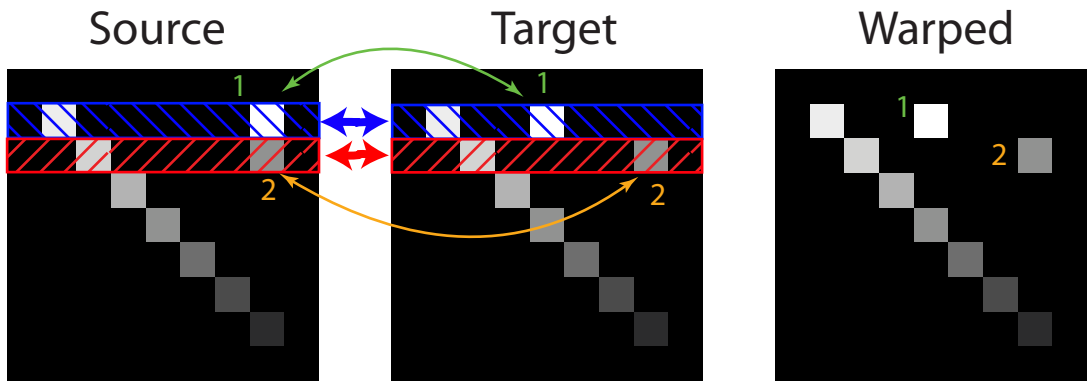


Figure 34: Broken Continuity Example #2

4.4 Performance Improvement

A performance improvement to the original 2DDW algorithm was described in the previous section. This experiment will both measure the improvement and determine if the modified algorithm produces the same results as the unmodified algorithm.

The images used in this experiment consist only of noise. This noise was randomly generated. Several noise images of increasing sizes are used in order to establish some relationship between data size and time to complete the computations of both the algorithms. The results follow. Time here is measured in seconds.

Table 3: Original Time

#	Source	Target	Warped	Displace	Results
1					Unwarped: 172.750 Warped: 172.750 Time: 0.237
2					Unwarped: 106.889 Warped: 79.900 Time: 0.337
3					Unwarped: 75.938 Warped: 52.250 Time: 0.651

Table 3: Original Time

#	Source	Target	Warped	Displace	Results
4					Unwarped: 106.200 Warped: 45.606 Time: 1.481
5					Unwarped: 100.667 Warped: 53.095 Time: 3.320
6					Unwarped: 114.735 Warped: 50.129 Time: 7.049
7					Unwarped: 115.109 Warped: 40.506 Time: 14.040
8					Unwarped: 119.309 Warped: 39.940 Time: 26.024
9					Unwarped: 111.400 Warped: 36.671 Time: 46.253
10					Unwarped: 106.769 Warped: 34.453 Time: 77.644
11					Unwarped: 97.882 Warped: 32.329 Time: 125.938
12					Unwarped: 99.379 Warped: 28.756 Time: 197.575

Table 3: Original Time



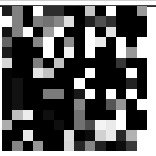
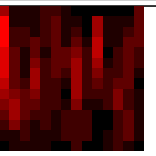
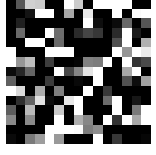

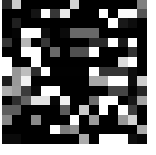

#	Source	Target	Warped	Displace	Results
13					Unwarped: 119.408 Warped: 36.658 Time: 298.619
14					Unwarped: 110.173 Warped: 33.898 Time: 443.350

Table 4: New Time

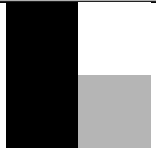
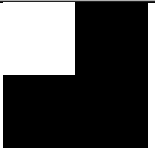
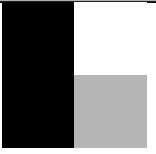

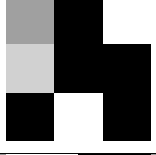
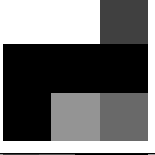
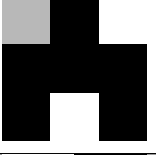
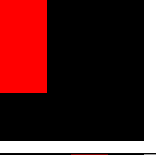
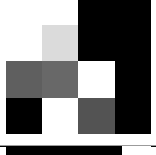
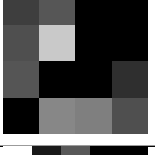
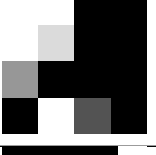

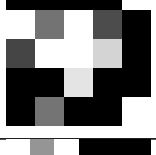
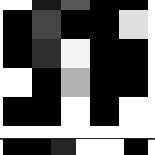
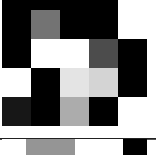

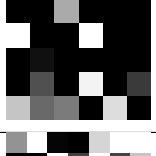
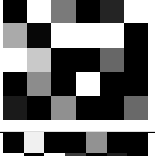
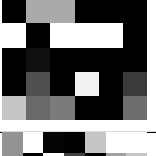

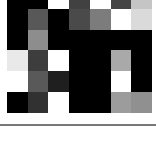
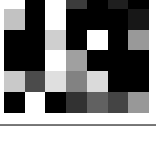
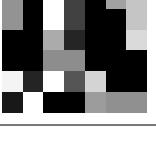

#	Source	Target	Warped	Displace	Results
1					Unwarped: 172.750 Warped: 172.750 Time: 0.231
2					Unwarped: 106.889 Warped: 79.900 Time: 0.273
3					Unwarped: 75.938 Warped: 52.250 Time: 0.391
4					Unwarped: 106.200 Warped: 45.606 Time: 0.609
5					Unwarped: 100.667 Warped: 53.095 Time: 1.013
6					Unwarped: 114.735 Warped: 50.129 Time: 1.669

Table 4: New Time

#	Source	Target	Warped	Displace	Results
7					Unwarped: 115.109 Warped: 40.506 Time: 2.657
8					Unwarped: 119.309 Warped: 39.940 Time: 4.148
9					Unwarped: 111.400 Warped: 36.671 Time: 6.124
10					Unwarped: 106.769 Warped: 34.453 Time: 8.746
11					Unwarped: 97.882 Warped: 32.329 Time: 12.355
12					Unwarped: 99.379 Warped: 28.756 Time: 16.905
13					Unwarped: 119.408 Warped: 36.658 Time: 22.616
14					Unwarped: 110.173 Warped: 33.898 Time: 29.631

Given the original algorithm's results (see Table 3) and the modified algorithm's results (see Table 4) one can immediately note the improvement in the time provided by the new algorithm. Close observations also reveal that the results including warped distances and the warped images are exactly the same for both versions of the algorithm indicating that the performance was improved without sacrificing accuracy.

A graphical representation of the run time in respect to the input size as measured by the number of

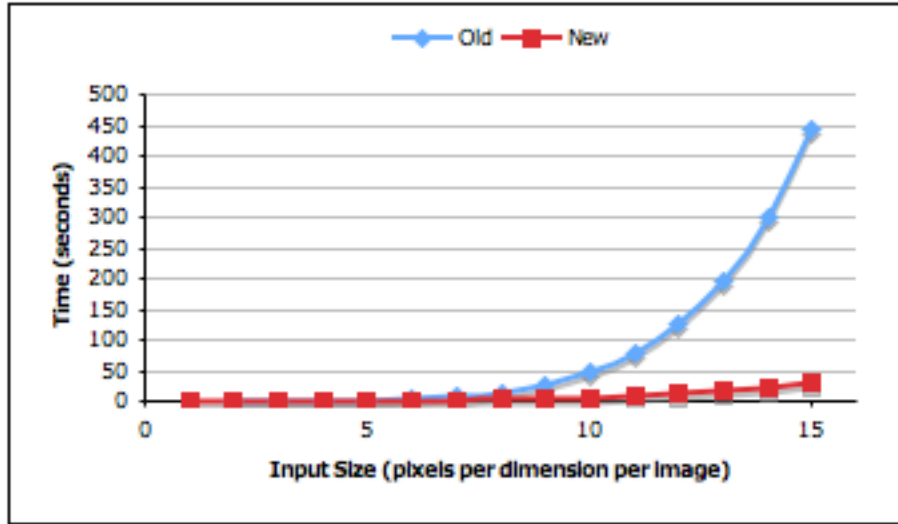


Figure 35: Run Time vs. N

pixels per side per input image (see Figure 35) shows a wide disparity between the run times of the two variations of the algorithm. At 15 pixels per side (225 pixels total), the old algorithm runs for almost 8 minutes while the improved algorithm finishes in under 30 seconds.

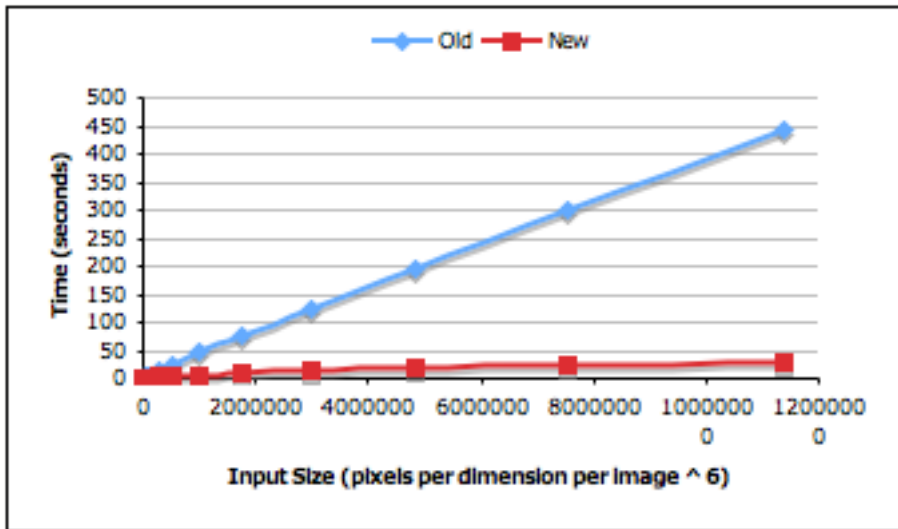


Figure 36: Run Time vs. N^6

The relationship between the run time of the original method and N^6 appears linear in Figure 36. This suggests that the original algorithm does indeed run in $O(N^6)$ (according to Section 3).

A similar confirmation of the time complexity of the improved algorithm can be seen in Figure 37. In this case the linear relationship appears between the run time of the improved algorithm and N^4 . Overall both algorithms seem to perform as expected. The added speed of the modified algorithm makes larger number

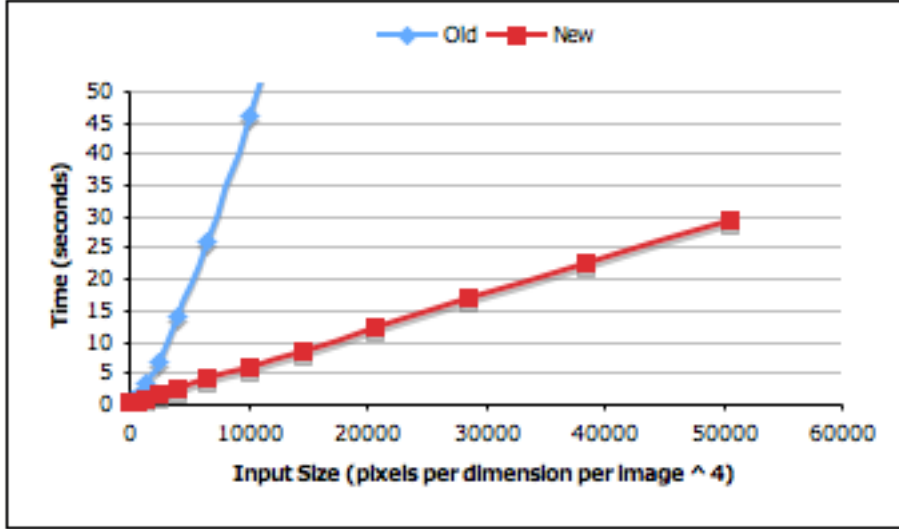


Figure 37: Run Time vs. N^4

of experiments possible. All of the experiments in this section except those seen in Table 3 were generated using the improved algorithm.

4.5 Distance Normalization

The raw distance measures as seen in the previous experiment demonstrate that the distance between objects increases with their size despite the fact that they all contain essentially the same thing (featureless noise), making it difficult to compare the distance measures across warping that involve images of varying sizes. To alleviate this problem, a normalization scheme is used.

Given the optimal mapping M , the raw distance measure between A and B is $\sum_{\langle x,y \rangle \in M} d(A[x], B[y])$. There are several options that can be used to modify this measure to be less sensitive to the sizes of A and B such as dividing by the number of pixels in the images or some other combination of the size measures. The other obvious option is to divide by the size of the mapping instead. This is the approach used in this project. The normalized distance is defined as $\frac{\sum_{\langle x,y \rangle \in M} d(A[x], B[y])}{|M|}$.

To test how well this normalization scheme behaves, a test was performed. The images used were fully white and fully black images of various sizes. The desired normalized distance between any all-white and any all-black image should be the same for any two such images. Intuitively, the normalization should give a “distance per match” hence the normalized distance in this test should always be “255” (white = 255, black = 0) since all pixels of the involved images are either black or white.

Table 5: Normalization Results

#	Source	Target	Warped	Displace	Results
---	--------	--------	--------	----------	---------

Table 5: Normalization Results




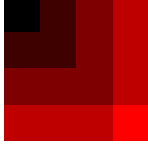



#	Source	Target	Warped	Displace	Results
1	(1x1)	(1x1)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 255 Unnormalized Warped: 255
2	(1x1)	(2x2)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 1020 Unnormalized Warped: 1020
3	(1x1)	(3x3)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 2295 Unnormalized Warped: 2295
4	(1x1)	(4x4)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 4080 Unnormalized Warped: 4080
5	(1x1)	(5x5)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 6375 Unnormalized Warped: 6375
6	(2x2)	(2x2)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 1020 Unnormalized Warped: 1020
7	(2x2)	(3x3)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 2295 Unnormalized Warped: 2295

Table 5: Normalization Results

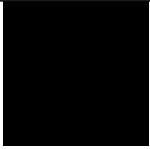

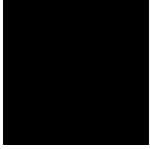
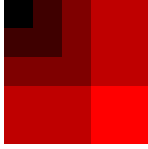
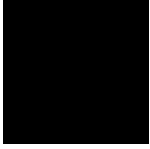
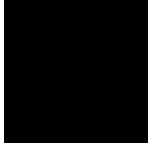
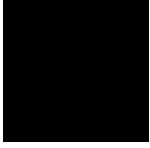

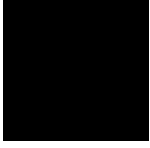

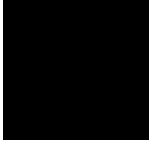

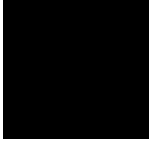

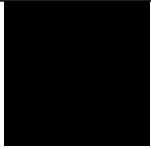

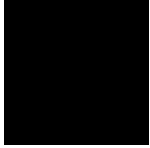
#	Source	Target	Warped	Displace	Results
8	(2x2)	(4x4)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 4080 Unnormalized Warped: 4080
9	(2x2)	(5x5)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 6375 Unnormalized Warped: 6375
10	(3x3)	(3x3)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 2295 Unnormalized Warped: 2295
11	(3x3)	(4x4)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 4080 Unnormalized Warped: 4080
12	(3x3)	(5x5)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 6375 Unnormalized Warped: 6375
13	(4x4)	(4x4)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 4080 Unnormalized Warped: 4080
14	(4x4)	(5x5)			Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 6375 Unnormalized Warped: 6375

Table 5: Normalization Results

#	Source	Target	Warped	Displace	Results
15	(5x5)				Unwarped: 255.000 Warped: 255.000 Unnormalized Unwarped: 6375 Unnormalized Warped: 6375

The various tests in Table 5 show wildly varying unnormalized distances as opposed to the normalized distances. Despite the fact that all of the source images contain nothing but white pixels and the target images contain nothing but black pixels, the unnormalized distance varies from 255 to 6375. At the same time the normalized distance is constant at 255.

As mentioned previously, the chosen normalization scheme described in this section is not the only potential choice. The difficulties arising from other normalization schemes are summarized by the Figures 38 though 41.

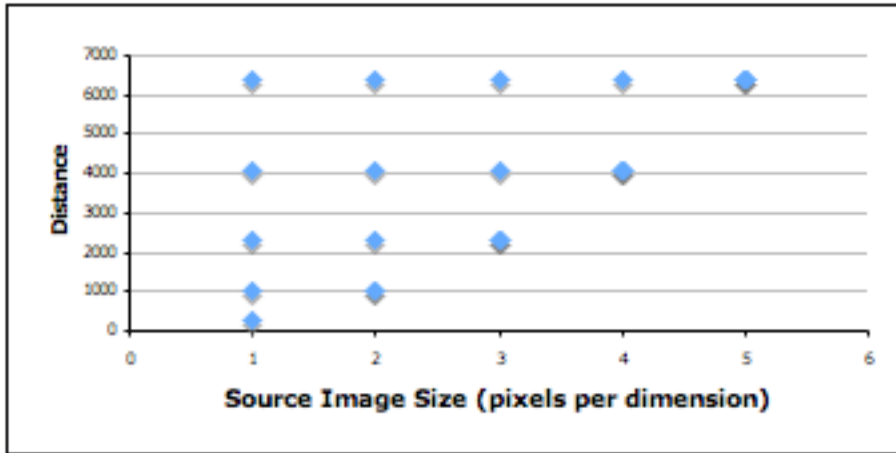


Figure 38: Unnormalized Distance vs. Source Image Width

Our first potential normalization scheme considers only the size per side of one of the images involved. Because the images in this experiment are all square, one could use the number of pixels per side as the parameter for normalization (an 10x10 image has 10 pixels per side). If one assumes that the normalization scheme will involve nothing but the size (per dimension) of one of the images then normalizing all the tests in this section is hopeless. Figure 38 shows a plot of distance as related to the number of pixels per side of the source image. One can see that there are multiple possible raw distances for each image size (per side). For example there are multiple instances in the experiment for which the sides of the source image are all 1 pixel wide. The distance computed by 2DDW over those instances range from 255 to 6375 (having no single value).

The wide disparity between the potential normalization parameter and the distance leads us to believe

size per side alone is not a viable for normalization.

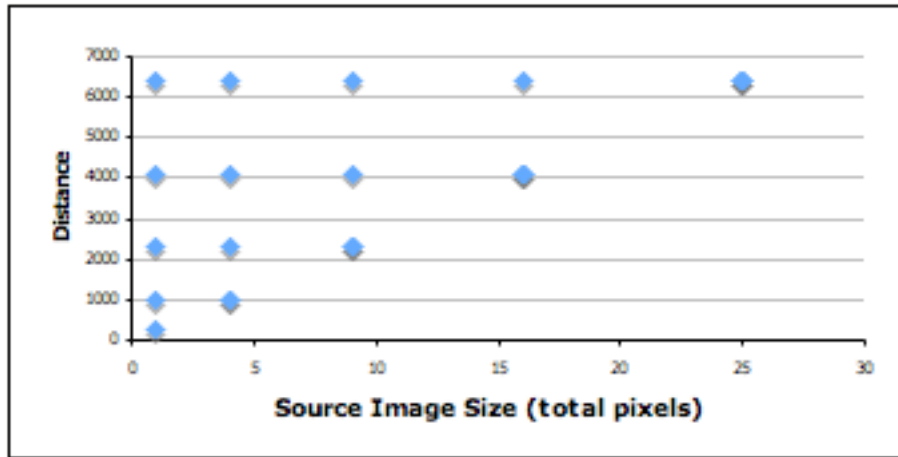


Figure 39: Unnormalized Distance vs. Source Image Total Size

The same problem occurs if one attempts to use the total size of one of the images as seen in Figure 39. The total size is the number of pixels contain in an image. In this experiment the total number of pixels is a square of the number of pixels per dimension.

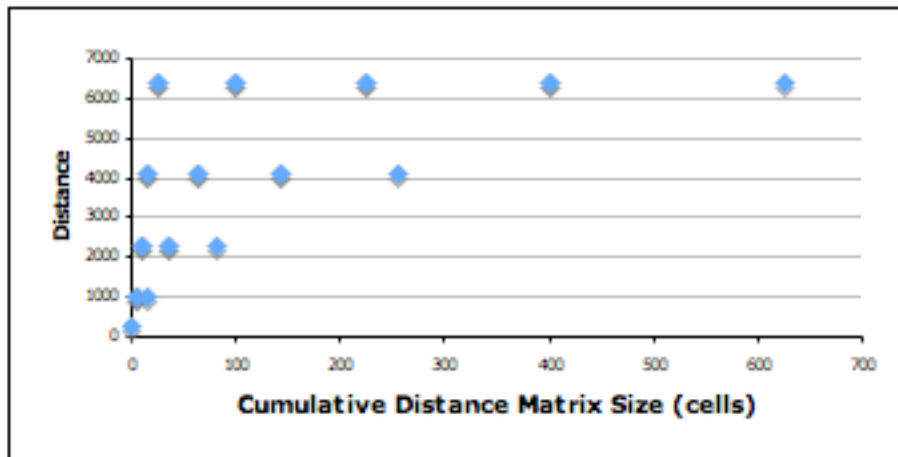


Figure 40: Unnormalized Distance vs. Cumulative Distance Matrix Size

It should have been evident that any scheme that uses the size of only one of the images cannot succeed. If we use the sizes of both and specifically the size of the cumulative distance matrix then the function problem is not as apparent as the problem of defining a function that would fit the data (see Figure 40). There are other ways one can use the dimensions of both images to normalize but we hypothesize that any such scheme will fail in this experiment as well given that there is a wide range of possible mappings possible created from matrices of the same size and more importantly the different mappings may contain different amounts of elements. This leads us to our final normalization scheme.

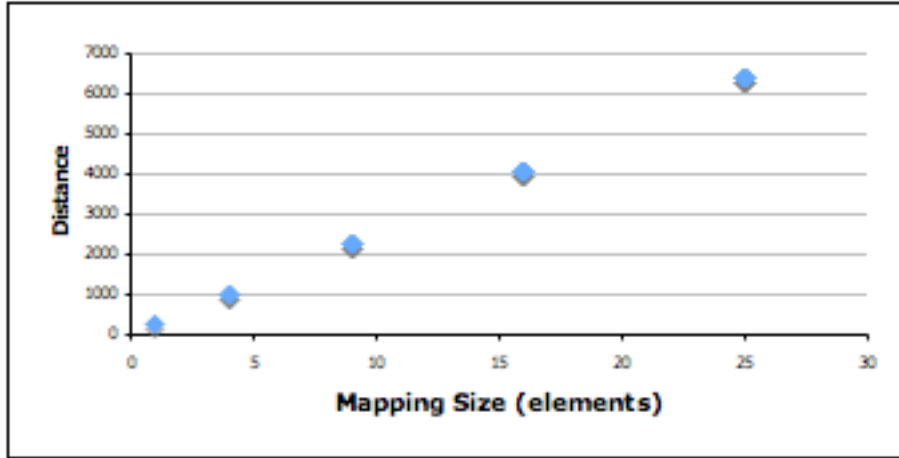


Figure 41: Unnormalized Distance vs. Mapping Size

If we use the number of elements in the mapping relation (which in this case is number of pixels in the bigger of the two images involved) as the independent variable and plot the unnormalized distance as the dependent, we can obtain a linear relationship (see Figure 41). A simple division of the raw distance values by the size of the mapping relation therefore results in a constant over all the tests in this experiment.

The normalization scheme has proven perfect in this experiment. All of the examples here, however, are very similar. The warping path in all cases follows the diagonal. There is no guarantee that the normalization scheme will perform as well in more complicated and varied instances. The scheme, though, seems reasonable and reliable in other experiments in this section. We cannot determine the “correct” or “perfect” normalization scheme for all examples but the scheme we do choose is the best from those we have come across.

4.6 Facial Recognition


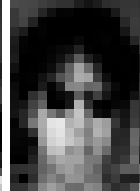
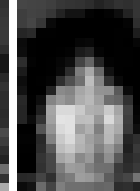
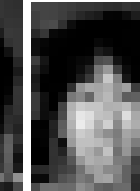
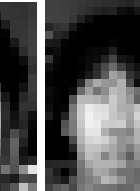

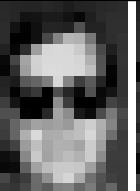
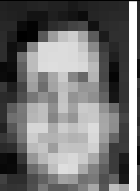
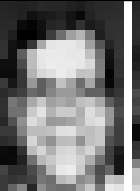


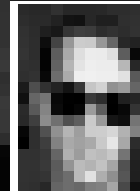
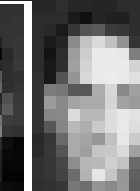
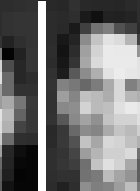


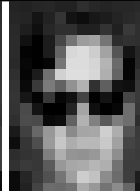
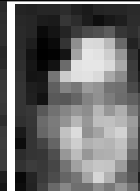
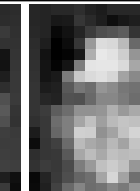
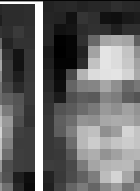

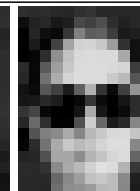
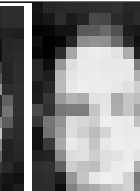
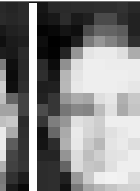

One of the potential uses for the 2DDW algorithm is in the field of facial recognition. An ability to algorithmically recognize a face can be very useful for various applications. This experiment is designed to test how well 2DDW can handle the recognition of faces.

The images used in this experiment include the faces of 5 different people in 5 different expressions for a total of 25 images (see Table 6). A distance between any two images is determined. Interpretation of such a large number of data requires more than observations over the raw distances. These distances, therefore, are used in two different analyses. The first uses the distances to create clusters to (hopefully) bin each person to their own cluster. The second analysis uses these distances to determine how accurate a face classifier would be given the distances. More information on these two approaches is found in the following sections.

Table 6: Face Images

#	Images

Table 6: Face Images

#	Images				
1					
2					
3					
4					
5					

4.6.1 Clustering

The first evaluation of the facial distances is the use of a clustering algorithm. The usual application of clustering is used to determine groups of objects from a large collection of objects such that the groups contain objects “close” together.

Normally these objects can be placed in a Euclidean space and the distance between any two objects can be easily calculated. In our case, however, the distance values will come from the results of 2DDW. The distance between any two faces was calculated in this experiment because the clustering requires a distance between any two objects to be known (or calculable).

The clustering algorithm we used is defined in [NH94]. The algorithm known as K-medoid (or PAM) is a variant of K-centroid algorithm but does not require the average of several objects to be calculable (which

cannot be done in this case).

Table 7: Face Clusters

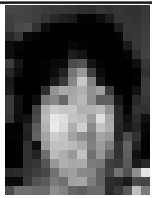
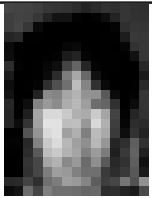
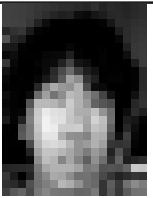
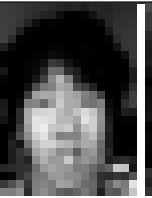
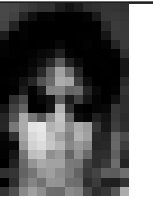

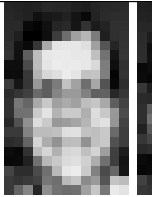
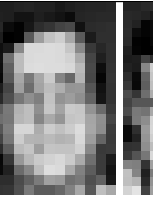

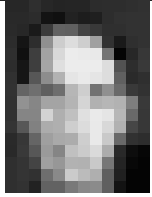
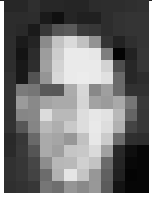

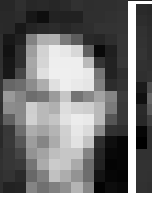

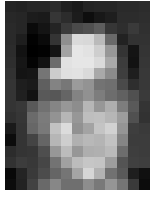
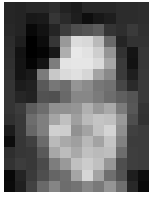
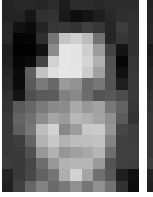
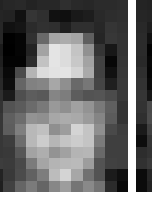
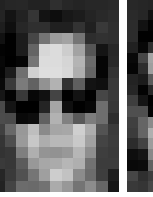
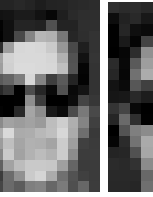
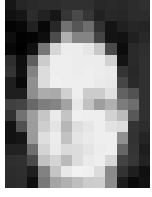



#	Medoid	Members				
1						
2						
3						
4						
5						

Table 7 shows the results of K-medoid over the face distances. The number of clusters was set to 5 as this is how many clusters we expected to form. The “medoid” in each cluster is the equivalent of the centroid as known in K-centroid. It is the image representing the “center” of the images in each cluster. The notion of “center” does not exist for objects that cannot be averaged so the medoid is calculated to be the one object such that the sum of the distances to each of the member objects in each centroid is minimized (see more about the algorithm in [NH94]).

The results show that in general the clusters formed do conform to each of the different people in the experiment. The exceptions are the faces wearing sunglasses. It appears that the faces with sunglasses which were not part of the cluster of their own person were all binned into the cluster of one single person (see

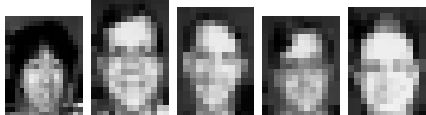

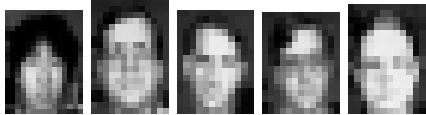

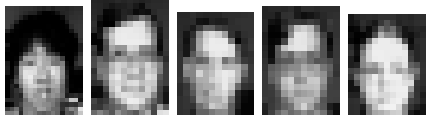
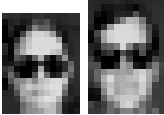
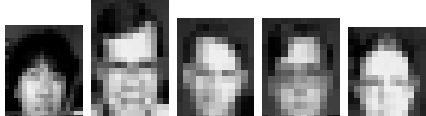
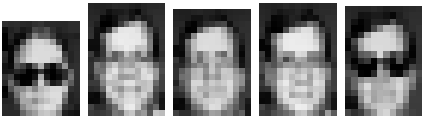
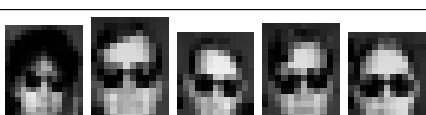
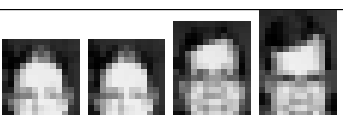
Cluster #4 in Table 7). This is most likely so because the person who is the medoid of that cluster does appear to be wearing normal glasses to begin with.

Overall the distances produced by 2DDW seem to be quite adaptable to facial recognition. It is important to note, however, that the images used are very small, are all taken from the same angle in respect to the face, and in general they do not vary that much. Still each image is not exactly the same as any other image hence the algorithm is capable of doing better than a random distance generator. See Section 4.6.3 for a little more discussion and experiment over some issues that might arise in real facial recognition applications.

4.6.2 Classification

The second analysis of the distance measures is done using a classification algorithm. Specifically the algorithm used is a simple nearest neighbor (NN) method. The method classifies instances (faces) using the class of the nearest model instance. In our case we will perform classification over 5 folds. During each fold, the model instances will include one face of each class value (person) and the test instances will be the rest of the images. Each instance is made sure to be part of the model instances exactly one time.

Table 8: Face Classification Folds

Fold	%Correct	Model	Missclassified
1	90.000		
2	90.000		
3	90.000		
4	75.000		
5	80.000		
*	85.000		

The results of the classification over the 5 folds as well as the overall accuracy (average over the folds) can be seen in Table 8. These result show that the distance measure was adequate in most cases for the task given. The accuracy ranged from 75% to 90% over all the folds with an average of 85% correct. The

misclassified images mostly include those with sunglasses for the folds in which the models were not the sunglasses images and only the faces of two different people for the other folds.

Not considering the sunglasses images, only one person was problematic in classification and the problems only occur in fold #4. This is most likely the case because that face (second from the left on fold #4) observes a slightly more downward orientation than the other faces of that same person. Despite this fact, an accuracy of 85% is still very good but again this is a very simple example that lacks a lot of the problems that would be present in a real facial recognition application such as those described in the next section.

4.6.2.1 Random Models

A potential issue exists with the choice of models in the classification experiment. Is it really reasonable to separate the images with glasses as a model in one single fold of the classification? Either way, does such handling of the models results in a higher or lower overall accuracy that would have been otherwise reported?

Although we believe that real applications would almost certainly use consistent model (or template) images, it is nonetheless interesting to observe the differences in results if such consistency is not maintained. A second classification experiment is therefore performed over the face images.

This time the model or representative image of each person is chosen at random. This allows for a wide range of possible different models and hence it is not unreasonable to compute the accuracy over more than just 5 folds. We compute the accuracy over 20 different models or folds in this experiment. They can be seen in Table 9.

Table 9: Face Classification Folds with Random Models






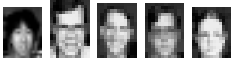



















Fold	%Correct	Model	Missclassified
1	90.000		
2	90.000		
3	90.000		
4	70.000		
5	75.000		
6	90.000		
7	60.000		
8	85.000		
9	85.000		

Table 9: Face Classification Folds with Random Models

Fold	%Correct	Model	Missclassified
10	75.000		
11	75.000		
12	70.000		
13	90.000		
14	75.000		
15	60.000		
16	60.000		
17	75.000		
18	90.000		
19	75.000		
20	75.000		
*	77.750		

The results demonstrate that overall the accuracy is almost 10% lower in this random model classification that was in the previous classification experiment. One can see that the fold containing no faces with sunglasses in their model seem to have the highest accuracies. Models with sunglasses seem to produce the lowest accuracies (see Table 9).

Another interesting observation comes into view here. In the previous face classification experiment, the model that contained all the faces with sunglasses achieved 80% accuracy (see Table 8) while in the new experiment, models with just a few or even a single sunglasses image can only achieve at most 75% accuracy and in many cases are as low as 60% correct. It appears that models that contain consistent examples (or templates) produce in general better accuracies than models with mixed or inconsistent templates. Even models with each person wearing sunglasses perform better than a mixed model despite the lack of any sunglasses in the test instances.

4.6.3 Problems

The images used in this facial recognition example are extremely simple. They are far from the images that would probably be available in a real facial recognition application. Those real images would most likely

pose many problems for the facial recognition methods that the simple example images just do not. These problem could include:

- Resolution. Real data would probably come in a much higher resolution for recording facial features in greater detail. This might be overcome by scaling images. This will certainly decrease the run time of any algorithm but might detract from its accuracy.
- Cropping. Real data would most likely contain a lot more of a scene than just the head or face of a person to be recognized. The solution to this problem would have to be some kind of cropping algorithm that is capable of outlining the boundaries of the face in order to remove the unnecessary portions of the image.
- Lighting. Real scenes might have a variety of lighting situations. Very brightly lit faces, very dimly lit faces, and anything in between might be given to a real facial recognizer. A solution would have to include some kind of lighting normalization.
- Decoration/Obstruction. Real face captures might have more obstructions or decorations than just sunglasses. In our example, even sunglasses alone were capable of causing major trouble. In terms of 2DDW, we cannot think of any viable way to overcome this problem.
- Orientation. Real faces might come in a variety of orientations. The person pictured might be facing downward or some other direction. The face itself, might also not be oriented vertically. The non-vertical orientation might be corrected easily assuming face features can be recognized but the problem of other orientation variations seems to be very hard to overcome.
- Volume. The number of possible template faces might be significantly larger. If the database of persons for recognition includes thousands or even millions of faces, slow algorithms (like 2DDW) might just not be feasible. As far as 2DDW is concerned, we cannot imagine any potential solution to this problem other than faster computers.

Each of these problems can make 2DDW alone not an option for facial recognition and all of them together suggest that 2DDW requires quite a bit of processes or filters over its input to be effective. The next section shows a simple example of how vertical orientation of a face alone can affect the algorithm.

4.6.3.1 Rotation

This experiment is designed to test the effects of face orientation (off vertical) on 2DDW performance. The images feature the face of one person rotated by a variety of angles ranging from 0 to 90 degrees. These images attempt to minimize the effects of the other problems mentioned in the previous section. The images are cropped to include just the face. The lighting in all the images are the same since they all come from the same base image. The person used is not wearing glasses and does not have any other facial obstructions.

Table 10: Face Orientation Results












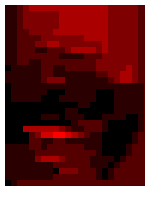

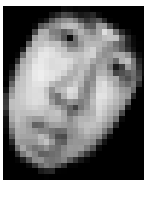





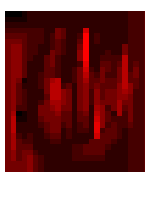

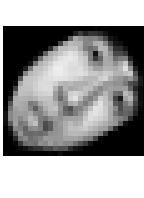

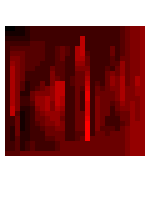



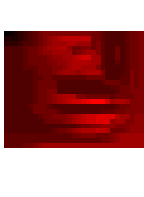
#	Source	Target	Warped	Displace	Results
1					Unwarped: 18.358 Warped: 7.814
2					Unwarped: 22.007 Warped: 7.711
3					Unwarped: 30.057 Warped: 7.930
4					Unwarped: 43.609 Warped: 10.559
5					Unwarped: 49.560 Warped: 12.261
6					Unwarped: 43.541 Warped: 11.031
7					Unwarped: 40.718 Warped: 10.728

Table 10: Face Orientation Results


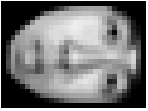

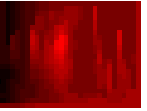
#	Source	Target	Warped	Displace	Results
8					Unwarped: 35.994 Warped: 11.839

Table 10 shows the results of this experiment. The distance ranges from 7.814 for images that are 5 degrees off each other to 12.261 for images at 45 degrees in respect to each other.

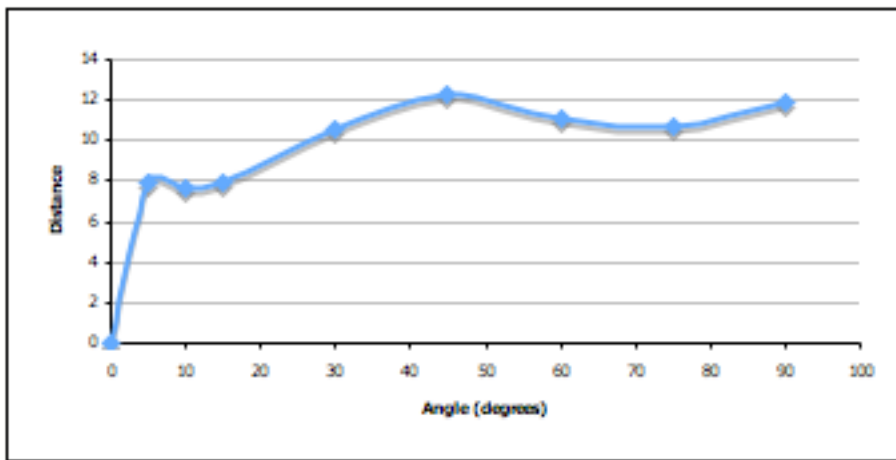


Figure 42: Distance vs. Face Angle

Figure 42 puts the results into perspective. While a 157% increase of distance from 5 degrees to 45 degrees off vertical would certainly degrade the performance of 2DDW in the facial recognition applications, the fact that a mere 5 degree angle off vertical produces such a large distance value (compared to the ones at large values which can be assumed to be “big”) is alarming. The distance between the exact two images is zero. A 5 degree rotation is almost not noticeable but for the algorithm it is a big deal. This reaffirms the notion that 2DDW cannot work alone for facial recognition. In this case it would need to be assisted by something to reorient the data.

4.7 Text Recognition

The second application for which 2DDW might be useful is text recognition. While face images contain significant areas of features, text is made up of curves. Handwritten text also has the added complexity in the wide range of varieties of writing styles. The text recognition experiments are therefore sufficiently different than the face recognition experiments.

The methods employed for these experiments are essentially identical to the face recognition experiments.

Clustering and classification is used with the results of pairwise distance computations. Two sets of images are experimented on. The first set contains computer generated letters of a variety of different fonts. The second set is made of handwritten numbers written in a variety of styles.

4.7.1 Digital Letters

The data for this experiment is formed by digitally generated letters. The letters include “a”, “b”, “c”, “d”, “e”, and “f”. Each of these is generated using 4 different fonts: Arial, Helvetica, Impact, Times, and Trebuchet. This results in 30 images (see Table 11). The distance between any two of these is computed using 2DDW. These distance values are then used in conjunction with clustering and classification methods as was done for facial recognition.

Table 11: Letter Images

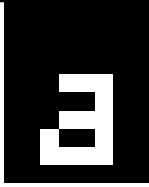
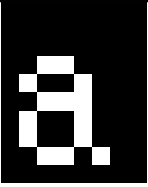
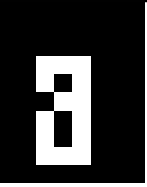
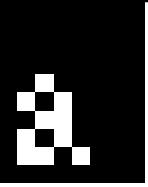

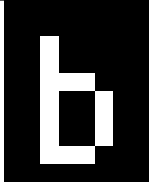
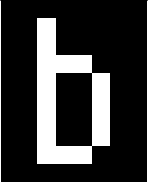
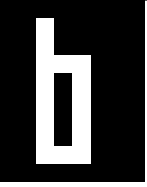
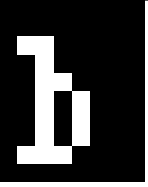
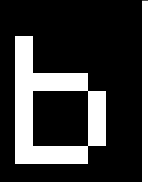
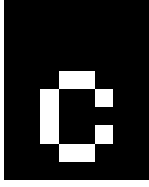
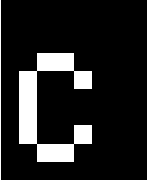
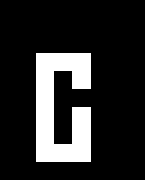
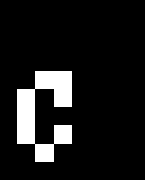
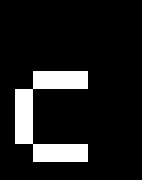
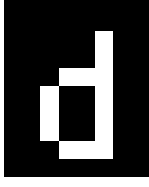
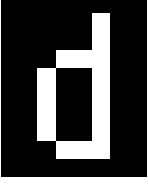
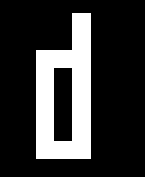
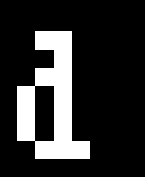
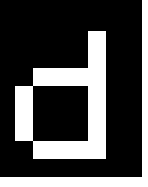
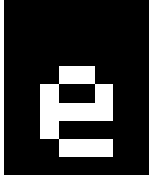
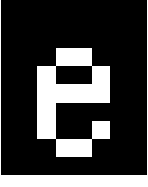
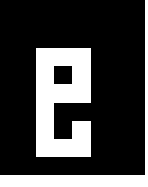
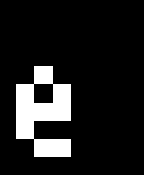
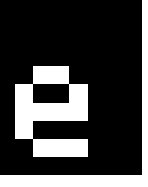
#	Images				
1					
2					
3					
4					
5					

Table 11: Letter Images

#	Images				
6					

4.7.1.1 Clustering

Just as in facial recognition, the K-medoid algorithm was used to generate clusters. The number of clusters was specified as 6 given that there are 6 different letters in the data. The results are surprising.

Table 12: Letter Clusters

#	Medoid	Members									
1											
2											
3											
4											
5											
6											

Table 12 shows the clusters generated by K-medoid. The first thing one notices is that the clusters are very unbalanced. One of the clusters has zero members (other than the medoid) while another has 12. Furthermore, two variations of the letter “e” are medoids of two different clusters.

The clusters seem random except a few regularities. The first cluster does indeed contain only “a”s as the medoid and members. The “b” cluster contains mostly “b”s and “d”s as members. The “f” cluster contains mostly “c”s and “f”s. And finally, the second “e” cluster contains mostly “e”s and one “f”. It would appear the other cluster with “e” as medoid only exists for the sole purpose of satisfying the 6 clusters requirement.

Closer observation of the results (not shown) is that the total sum of distances between the medoids and members of each cluster is zero. Furthermore, about half of all the distances generated by 2DDW are zero. The algorithm was capable of warping a large portion of the letters into other letters perfectly. The situation is so severe that the clustering results shown here are not the only ones that have a zero sum distance. Repeated runs of K-medoid result with a variety of clusters all having zero as total distance (K-medoid initializes with random medoids).

Looking at some of the members and the medoids, it is difficult to imagine how their distances were calculated to be zero. For example the medoid of the second cluster (a “b”) and the “f” in the cluster seem to be very different (see 12). The lack of continuity in 2DDW, though, allows for some interesting warpings. One such zero-distance warping between the “b” and “f” is shown in Figure 43.

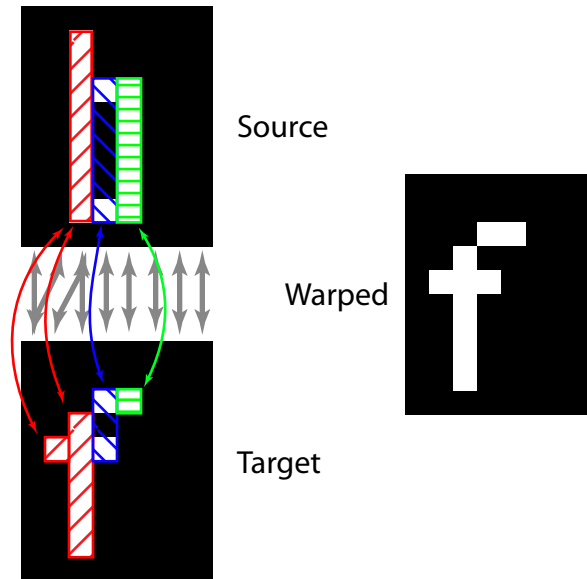


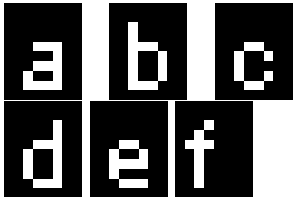

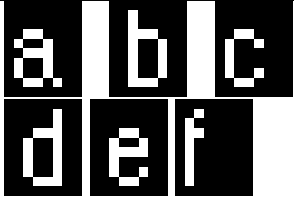
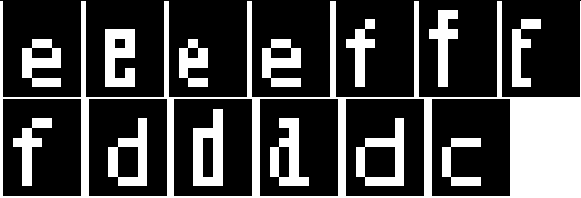
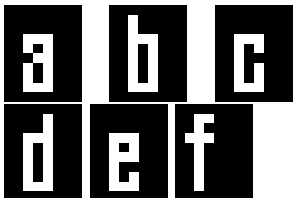

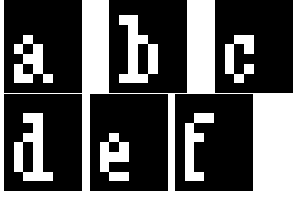
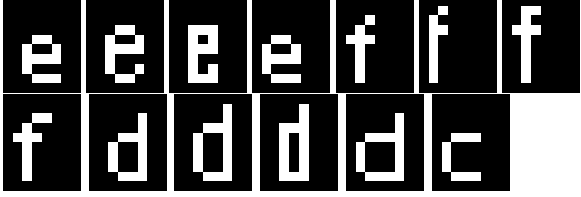
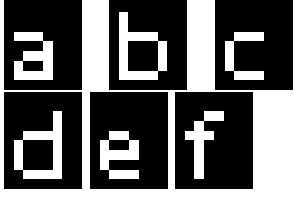
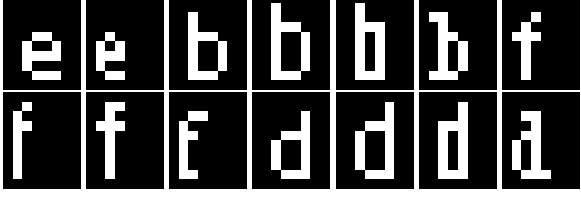
Figure 43: Zero Distance Map Example: b to f

It turns out that a zero-distance mapping from the “b” to the “f” is in fact extremely simple. All that is required is a column-wise mapping of second and third columns of “b” to the first and second columns of “f” respectively. Other than this, the mapping contains 1D mapping of the respective columns of the images (column four to column four and so on). Figure 43 the mapping between the important features of the two images and the column-wise matching.

4.7.1.2 Classification

We expect the classification results to be no better than the clustering. The problem, of course, should show up not as seemingly random clusters but rather low classification accuracies. These results should be much easier to interpret. The method used is the same as in the facial recognition. There are 5 folds in this case as there are 5 different fonts used.

Table 13: Letter Classification Folds

Fold	%Correct	Model	Missclassified
1	29.167		
2	45.833		
3	37.500		
4	45.833		
5	41.667		
*	40.000		

The results shown in Table 13 are not as promising as those for facial recognition. The accuracy ranges from 46% to as little as 29%. Overall the average accuracy across all the folds is 40%. Although this is twice as good as a random classifier would be expected to do, a 40% accuracy is not really feasible for any serious application.

There does not appear to be many patterns in the classification errors. Note, however, that the reason why “a” appears to be always correctly classified is the fact that the alphabetical order of the letters is used in cases of ties in the distances. This means that when classifying an “a” such that multiple model instances show a zero distance to the test instance, it will always be classified as “a”. This is most likely also the reason for a low occurrence of misclassified “b” and incrementally higher occurrences of the other letters being misclassified.

The results are not very good despite the simplicity of the images. In a real application, the images would most likely have a lot of the same problems as do the face image problems outlined previously. In this experiment, however, the problem arises almost exclusively from the lack of continuity in 2DDW. This shortcoming allows the algorithm to consider warpings that seem quite convoluted. On the other hand, the algorithm only considers sets of 1D mappings between pairs of rows and pairs of columns which would suggest that it is more constrained for 2D warping than it should. The lack of continuity seems to make up for it sufficiently, however.

4.7.2 Handwritten Numbers

The second text recognition group is very different to the first one. Not only does it involve numbers instead of letters, the numbers are handwritten. The images contain more than just black and white pixels. Included are a few of each digit from 0 to 9. See Table 14 for the list. As in the previous experiments, the distance between each two images is calculated using 2DDW and this data is then used in analysis using clustering and classification.

Table 14: Number Images



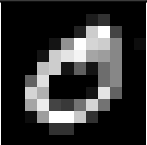

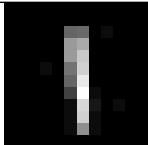
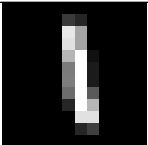
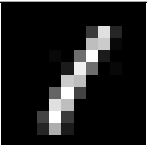
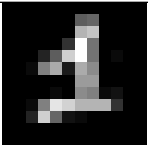
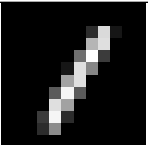
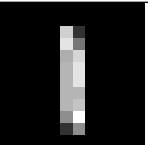
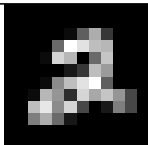
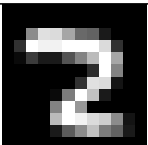
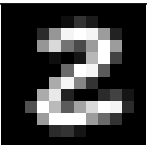
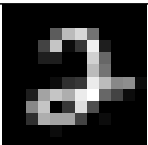
#	Images					
1						
2						
3						

Table 14: Number Images

#	Images				
4					
5					
6					
7					
8					
9					
10					






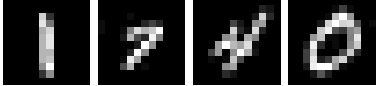














4.7.2.1 Clustering

The clustering method is the same as used previously. There is a larger number of clusters in this case as there are 10 different digits used. The results can be seen in Table 15.

Table 15: Number Clusters

#	Medoid	Members
---	--------	---------

Table 15: Number Clusters

#	Medoid	Members
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

The results here are different than the simple letter recognition. Firstly, none of the distances between two different numbers was calculated to be zero. Because of this, the clusters here have slightly more weight to them. Although we cannot verify it, we believe that the results shown in Table 15 represents the only optimal clustering solution.

The actual clusters are not very promising. Three clusters with a “1” as the medoid were formed. There were also two clusters each for “6”, “5”, and “9”. In general, most of the clusters do not contain the same class (number) or even the class or number represented by their medoid. Rare exception exist however. Cluster #9 which has “6” as its medoid and “6” as its only member. Also, cluster #6 is formed with “9” as the medoid and two out of the three members are “9” as well. Other than this there seems to be general chaos among the clusters (see Table 15). Perhaps classification will give a better view of the problem.

4.7.2.2 Classification

The classification is also done in the same exact manner as the letter classification. There are 6 folds used in this case. Note that not all of the numbers contain 6 examples. Because of this some of the instances will be part of the model instances more than once. The results can be seen in Table 16.

Table 16: Number Classification Folds

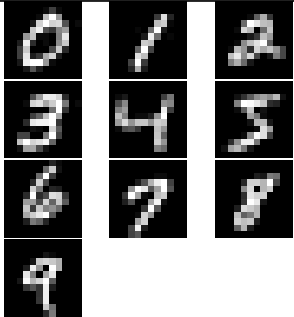

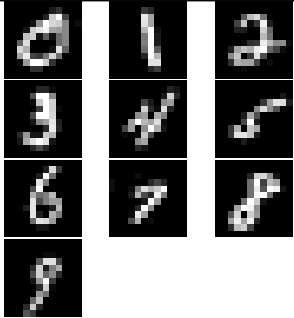

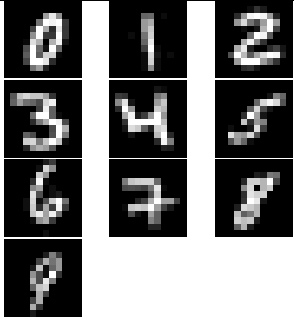

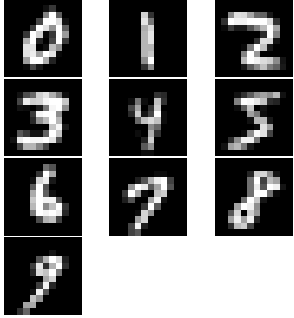

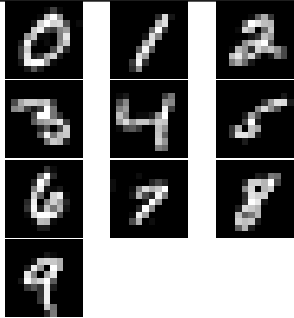

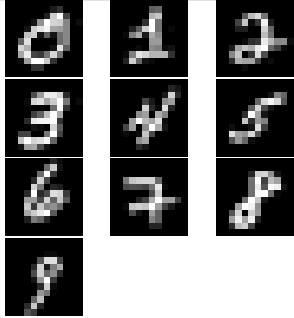
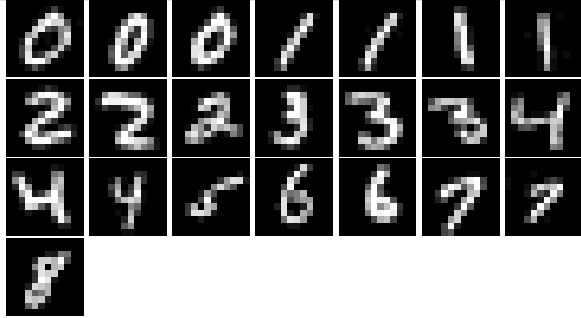
Fold	%Correct	Model	Missclassified
1	23.333		
2	26.667		
3	26.667		
4	23.333		

Table 16: Number Classification Folds

Fold	%Correct	Model	Missclassified
5	23.333		
6	26.667		
*	25.000		

With an accuracies ranging from 23.3% to 26.6%, the results are not very good. A random classifier over the same data would be able to achieve around 10% accuracy. The average using 2DDW is 25% and not that much better than a random classifier. This performance is certainly not good enough for any text recognition application. There does not seem to be any significant patterns in the misclassified images (see Table 16).

Many factors can account for the poor performance. These include the following:

- **Styles.** The handwritten numbers here come in a variety of styles. Some of the instances are hard to classify even by the author of this report. We do not know of any potential solution to enable 2DDW to overcome this problem.
- **Skewness.** Although this might be considered part of the style problem, it is sufficiently general over many styles to be separated out. Different people write text with different rates of skewness or leaning. Notable is the difference between left handed and right handed writers. Unlike general style differences, the skewness seems to have better potential of being corrected. It is not entirely unfeasible to compute the general skew of a sequence of characters and correct for it.

In addition to the problems already present in these images, there are other problems that would certainly exist in real text recognition applications.

- Most of the problems listed in association with face recognition (see Section 4.6.3).

- Colors. Real text comes in a variety of colors. White on black or black on white are just two of the possibilities. If the template images (model images) are white on black and the instance to be recognized is the opposite, the algorithm unaided will have problems. The solution to this problem would most certainly involve a filter to rid the image of color. An edge finding filter could do the job nicely.
- Sequences. Character recognition would almost certainly first involve the separation of characters in a character sequence such as a sentence. 2DDW is at a disadvantage here since assuming that characters are separated correctly, there is some information lost in terms of context (the characters around the one in question).

Given the performance of 2DDW on the simple examples and the existence of other problems in real applications we can only conclude that 2DDW is simply not suitable as was described in this report for text recognition. Significant assistance is required before 2DDW gets the data. To be truly effective, 2DDW would most likely need to somehow maintain continuity as well. An approximate version of one of the exponential time algorithms is described in [US99].

5 Conclusions

The conclusions inferred from the experiments and experiences described in this report fit into two general categories. The conclusions from the adaptation, improvement, and analysis of the 2DDW algorithm are found in Section 5.1 while conclusions based on experimental applications are found in Section 5.2. In addition, recommendations regarding possible future work in this area are found in Section 5.3.

5.1 2DDW

The adaptation, improvement, and extension of the 2DDW algorithm as described in [LG04] were the major parts of this project. From initial problems to the performance improvement and finally potential of extensions to higher dimensions, the conclusions regarding 2DDW are contained in this section.

5.1.1 Problems

- The 2DDW algorithm is incapable of maintaining reasonable constraints assumptions. It does not satisfy continuity and monotonicity constraints of the 1-dimensional dynamic time warping algorithm across both dimensions. This leads to warping capabilities that often do not make sense or are unreasonable. Examples of such warping can be seen in Section 4.3.
- The 2DDW algorithm cannot be considered a fully 2D version of the dynamic warping algorithm. In fact, it appears more to be just a step above a simple and straightforward decomposition and warping of 2D data as 1D slices. However, a full 2D version of the dynamic warping algorithm is unlikely to have polynomial time complexity as the problem of warping in two dimensions has been shown to be NP-complete [KU03].

The problem of constraints unsatisfiability stems from the assumption by the authors of 2DDW (in [LG04]) that 2DDW inherits continuity and monotonicity from DTW by recursion. They were right only partially in this regard. The normal constraints are enforced by DTW but only in 1D slices of the 2D data being warped.

The reliance on 1D slices for the majority of the computation in 2DDW results in a pseudo-2D behavior. In the simplest case, an algorithm that does nothing more than warp 1D slices (columns or rows) from one 2D image to another does not necessarily take advantage of the 2D nature of the data. 2DDW is a step above this simplest case by varying which 1D slices are warped into each other as well as splitting the images into sections of horizontal and vertical slices

5.1.2 Improvement

- The original 2DDW algorithm is described and found experimentally to run in $O(N^6)$ time, where N is the width and height of the input two dimensional data, while the algorithmic improvement described in Section 3.2.7 allow the equivalent computation to run in $O(N^4)$ time. This was verified experimentally in Section 4.4.

The performance improvement over the original 2DDW algorithm was very straight forward. Rearranging some computation in the form of pre-computation of various 1D warpings provided a large increase in speed of the algorithm (see Section 3.2.7). Experimental results confirmed not only the performance improvement but also verified that the original and improved algorithms produce the same results (see Section 4.4).

5.1.3 Higher Dimensional Extensions

- Extension of the 2DDW algorithm to 3D magnifies the problems associated with the 2D version. The simple and easily repairable issue of double-counting cannot be easily solved in 3D and results in mapping relations that can contain contradictory 1D slice based mappings.
- Further extension of 2DDW to higher dimensions results in algorithms that are increasingly ignorant of the multi-dimensional nature of the problem.

As the frontier expands from stage to stage in 2DDW, often multiple 1D warpings are computed along the frontier. In such cases there exists a pixel that is both in the column and row portions of the frontier which is counted twice in distance computations. A simple subtraction of the double-counted portion of the distance suffices to alleviate the problem in 2DDW but in higher dimensions the issue becomes much more complicated. In 3D, entire 1D slices are shared in multiple distance computations between 2D planes. This often results in separate and different warpings of these 1D slices. Not only does the subtraction of the additional distance not seem feasible, the final mapping relations that are produced with such double counting may contain contradictory series of mappings in those 1D slices.

The fact that 2DDW and its extensions use DTW at their core leads us to believe that such extensions are less and less appropriate as the number of dimensions increases. While having 1D warpings as a basis for a 2D warping may not be a big problem in some cases (as in facial recognition), having 1D warpings as a basis for a warping between objects of 10 or more dimensions seems a lot less reasonable. Not only would the 10D mapping have proper constraint satisfaction in just 1D slices, but also the problem of contradictory mappings would expand from 1D slices to entire 8D segments of the data.

5.2 Applications

The experiments performed for this report were certainly not exhaustive. We can, however, infer some conclusions concerning 2DDW based on the experimental results in two main applications areas: facial recognition and text recognition. The results in these two areas were found to differ substantially.

5.2.1 Facial Recognition

- Despite the shortcomings and problems associated with 2DDW, it is capable of producing desirable results for the application of facial recognition as demonstrated in Sections 4.6.
- 2DDW is very sensitive to many types of variations in input data that would most likely exist in real facial-recognition applications. Because of this 2DDW is deemed to be effective only if assisted by

normalization algorithms. Such algorithms are necessary to enforce a consistency in the input that should be also present across the templates.

The satisfactory performance of 2DDW over the sample facial recognition application demonstrate that the algorithm might be useful in some cases. In the clustering analysis, the distance measures provided by 2DDW produced reasonable and in most cases correct clusters. The use of the distance values for classification, an average accuracy of 85% was obtained.

It is important to mention, however, the sensitivity of 2DDW to variations in its input. The objects need to be oriented in the same manner, they must have the same lighting, and a host of other necessary conditions must be met for 2DDW to be effective. If 2DDW were to be used in a real facial recognition application, these issues would need to be taken care of by some methods before 2DDW is applied. An example of how sensitive 2DDW is to orientation can be seen in Section 4.6.3.1.

5.2.2 Text Recognition

- The nature of text data results in poor performance for the 2DDW algorithm. The results are so poor that we doubt the existence of any reasonable uses of 2DDW in text recognition as seen in Section 4.7.

The nature of text data as represented in 2D images is very different to that of the facial recognition examples. Large portions of the images around the text are empty with only a few curves making up the characters. This demonstrates what the lack of constraints in the algorithm can do. In one of the digital letter example, the distance between most characters was found to be zero (see Section 4.7.1). Further examination of clusters and classification accuracy showed very poor results. In the case of the handwritten letters, the distance were not calculated to be zero, but poor results ensued despite this fact (see Section 4.7.2).

Furthermore, the examples used for the tests were oversimplifications of the data that would be used in real text recognition applications. This puts more weight on our opinion that 2DDW (without significant modifications) cannot be realistically used for text recognition.

5.3 Future Work

This project has left some issues unexplored and questions unanswered. These include the following:

- Analysis and implementation of a distributed version of the 2DDW algorithm for improved performance.
- Experimental results of the 3D version of 2DDW with respect to prospective applications.
- The feasibility of producing an accurate and reasonable warping in higher dimensions extending alternate warping algorithms.

The improved 2DDW algorithm runs in $O(N^4)$ time which is indeed an improvement over the original but for large images, this still results in very long run times. The time complexity of a parallel version of the

algorithm might be sufficiently small for it to be applicable to large images in a reasonable amount of time. More importantly, a parallel version of the algorithms might make it possible to experimentally ascertain the effectiveness of the 3D (or higher) extensions despite our intuition that the problems stemming from 2DDW would make such extensions unreasonable.

Given our lack of faith in 2DDW in higher dimensions, it only makes sense to examine other 2D warping algorithms such as the algorithm described in [US98] which does satisfy the continuity and monotonicity constraints. Extensions of alternate algorithm are probably possible. Approximation schemes such as those described in [US00a] would most likely be necessary to achieve a reasonable time complexity.

References

- [BC96] Donald J. Berndt and James Clifford. Finding patterns in time series: a dynamic programming approach. In *Advances in knowledge discovery and data mining*, pages 229–248. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.
- [HL03] Sam Holmes and Cindy Leung. Exploring temporal associations in the stock market. Undergraduate Thesis (MQP). Department of Computer Science. Worcester Polytechnic Institute, April 2003.
- [KP99] Eamonn Keogh and M. Pazzani. Scaling up dynamic time warping to massive datasets. In J. M. Zytkow and J. Rauch, editors, *Proc. 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'99)*, volume 1704, pages 1–11, Prague, Czech Republic, 1999. Springer.
- [KU03] Daniel Keysers and Walter Unger. Elastic image matching is np-complete. *Pattern Recogn. Lett.*, 24(1-3):445–453, 2003.
- [LG04] H. Lei and V. Govindaraju. Direct image matching by dynamic warping. In *Proc. of the 1st IEEE Workshop on Face Processing in Video, In conjunction with the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'04)*, Washington D.C., 2004.
- [LP92] T. Levin and R. Pieraccini. Dynamic planar warping for optical character recognition. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 3:149–152, 1992.
- [NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proc. 20th International Conference on Very Large Data Bases*, pages 144–155, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.
- [Pra04] Keith A. Pray. Mining association rules from time sequence attributes. Master's Thesis. Department of Computer Science. Worcester Polytechnic Institute, May 2004.
- [SC78] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acoustics, Speech, and Signal Proc.*, ASSP-26:43–49, 1978.
- [SS04] Zachary Stoecker-Sylvia. Mining for frequent events in time series. Master's Thesis. Department of Computer Science. Worcester Polytechnic Institute, August 2004.
- [US98] S. Uchida and H. Sakoe. A monotonic and continuous two-dimensional warping based on dynamic programming. In *Proc. 14th International Conference on Pattern Recognition*, volume 1, pages 521–524, 1998.
- [US99] S. Uchida and H. Sakoe. Handwritten character recognition using monotonic and continuous two-dimensional warping. In *Proc. International Conference on Document Analysis and Recognition*, pages 499–502, 1999.

- [US00a] S. Uchida and H. Sakoe. An approximation algorithm for two-dimensional warping. *Institute of Electronics, Information, and Communication Engineers Transactions on Information & Systems*, E83-D(1):109–111, 2000.
- [US00b] S. Uchida and H. Sakoe. Piecewise linear two-dimensional warping. In *Proc. 15th International Conference on Pattern Recognition*, volume 3, pages 538–541, 2000.

A Specialized Notation

This report makes use of a variety of specialized notation to aid in the descriptions and explanations of the various algorithms involved. The table in this section should serve as a reference in case of any confusion a reader might experience.

Table 17: Specialized Notation

Let i, j, k, l, m are integers.

Let Y_1, Y_2 be two vectors.

Let X be a matrix of some number of dimensions (can be 1, 0, or any positive integer).

Let A be a set.

Let f be a function or arity 1 and a numeric output.

\hat{i}_j : A vector $\langle i, i, \dots, i \rangle$ of size j .

$|Y_1|$: Size of Y_1 (the number of elements in Y_1).

$|X|$: The number of dimensions in X .

$\|X\|$: The “shape” of X . The shape is a vector specifying the extent of the matrix in all of its dimensions (note $|\|X\|| = |X|$).

$Y_1[i]$: The i^{th} element of Y_1 .

$(Y_1[1], \dots, Y_1[|Y_1|])$: Same as vector Y_1 . Parenthesis are used to emphasize the vectors when used as coordinates.

$X[Y_1]$: The element of X at the position specified by vector Y_1 (note $|X| = |Y_1|$).

$X[Y_1[1], \dots, Y_1[|Y_1|]]$: Same as $X[Y_1]$.

$Y_1[i : j]$: A “slice” of Y_1 from i^{th} to j^{th} elements. In other words $Y_1[i : j] = \langle Y_1[i], Y_1[i + 1], \dots, Y_1[j] \rangle$.
The slice notation used here derives itself from python.

$X[k, i : j]$: A 1-dimensional slice of X . Specifically $X[k, i : j] = \langle X[k, i], X[k, i + 1], \dots, X[k, j] \rangle$.

$\min_{a \in A} \{f(a)\}$: The minimum value of $f(a)$ over all the elements of A . Assume the value is 0 if $A = \emptyset$.

$X[i : j, k : l, m]$: A 2-dimensional slice of X .

$$X[i : j, k : l, m] = \begin{pmatrix} X[i, k, m] & X[i + 1, k, m] & \cdots & X[j, k, m] \\ X[i, k + 1, m] & X[i + 1, k + 1, m] & \cdots & X[j, k + 1, m] \\ \vdots & \vdots & \ddots & \vdots \\ X[i, l, m] & X[i + 1, l, m] & \cdots & X[j, l, m] \end{pmatrix}$$

$\operatorname{argmin}_{a \in A} \{f(a)\}$: The element $a \in A$ with the minimum value of $f(a)$.

$\operatorname{join}(Y_1, Y_2)$: A vector of size $|Y_1| + |Y_2|$ created by concatenating the two vectors.

$\operatorname{map}_{a \in Y_1} \{a + 1\}$: A vector created by adding 1 to each element of Y_1 . In other words $\operatorname{map}_{a \in Y_1} \{a + 1\}[i] = Y_1[i] \forall i \in \{1, \dots, |Y_1|\}$. This vaguely derives from the map function in scheme.