

# Elastic matching in linear time and constant space

Scott MacLean, George Labahn

University of Waterloo

ORCCA JLM, January 2009

## Dynamic time warping (DTW)

- ▶ Signal processing technique to measure the difference between time series
- ▶ Adapted by Tappert to symbol recognition
- ▶ Provides a distance measure between digital ink strokes
- ▶ Quite accurate, but quadratic time/space in # of points
- ▶ This is too slow for MathBrush...can we speed it up?
  - ▶ (yes)

# Setting

## Assumptions:

- ▶ We have a *library* of labeled *model* symbols to compare input against.
- ▶ For simplicity, symbols are formed by only one stroke.
  - ▶ (*there are many ways to handle multi-stroke symbols*)

## General technique for recognition:

- ▶ Given an *input* stroke.
- ▶ Compute its distance from every model stroke in the library.
  - ▶ Interpret smaller distance as “more similar strokes”.
- ▶ Choose the label of the model symbol best matching the input.
  - ▶ (*i.e. minimizing the distance measure*)

## DTW as distance measure

### Given:

- ▶ Model stroke of points  $(\hat{x}_1, \hat{y}_1), \dots, (\hat{x}_m, \hat{y}_m)$ .
- ▶ Input stroke of points  $(x_1, y_1), \dots, (x_n, y_n)$ .
- ▶ Pointwise distance measure  $d(i, j)$  between  $(x_i, y_i)$  and  $(\hat{x}_j, \hat{y}_j)$ .

$$\text{e.g. } d(i, j) = |\hat{x}_j - x_i| + |\hat{y}_j - y_i|$$

### Goal:

- ▶ Match each input point  $i$  to a model point  $f(i)$ .
- ▶ Minimize the total distance  $D(f) = \sum_i d(i, f(i))$ .

### Comments:

- ▶ The minimal distance gives the distance measure!
- ▶ Using additional constraints, Tappert solves this problem optimally.

# DTW by dynamic programming

## Introduce constraints:

1. First input point matched to first model point.
2. Last input point matched to last model point.
3. If  $i$ th input point matched to  $j$ th model point, then  $i + 1$ st input point matched to  $j$ th,  $j + 1$ st, or  $j + 2$ nd model point.

## Dynamic program:

$$D[i, 1] = d(i, 1) + D[i - 1, 1]$$

$$D[i, 2] = d(i, 2) + \min\{D[i - 1, 2], D[i - 1, 1]\}$$

$$D[i, j] = d(i, j) + \min\{D[i - 1, j - k] : k = 0, 1, 2\}$$

- ▶  $D[i, j]$  is cost of optimal match of input points  $1, \dots, i$  to model points  $1, \dots, j$ .
- ▶ Distance measure between strokes is  $D[n, m]$ .
- ▶ (*Note quadratic complexity*)

# DTW match structure

Let input points be  $I_1, \dots, I_m$ .

Let model points be  $M_1, \dots, M_m$ .

## Observations:

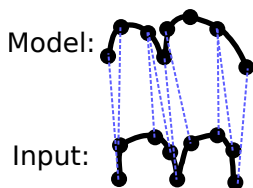
- ▶ By constraint 1,  $I_1$  must be matched to  $M_1$ .
- ▶ By constraint 2,  $I_m$  must be matched to  $M_m$ .
- ▶ By constraint 3, if  $I_i$  is matched to  $M_{f(i)}$ , then
  1.  $I_{i+1}$  is matched to one of  $M_{f(i)}, M_{f(i)+1}, M_{f(i)+2}$ .
  2.  $I_{i-1}$  is matched to one of  $M_{f(i)}, M_{f(i)-1}, M_{f(i)-2}$ .

We will choose the best *local* option at each point along the input stroke.

# Greedy approximate DTW

## Greedy approach:

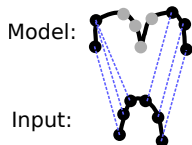
- ▶ Match endpoints to endpoints ( $I_1$  to  $M_1$  and  $I_n$  to  $M_m$ ).
- ▶ To match other input points, choose from the available options the model point minimizing the pointwise distance measure  $d$ .
- ▶ Work from both endpoints simultaneously toward the middle.
  - ▶ (*i.e. use both consequences of constraint 3*)



## Two potential problems

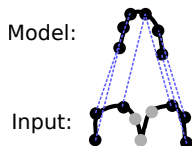
May run out of input points, leaving many model points unconsidered.

1.
  - ▶ **Fix:** When the input points are used up, match the center-most input point to every second remaining model point.



May run out of model points, leaving many input points unmatched.

2.
  - ▶ **Fix:** If no more model points are available, match every remaining input point to the center-most model point.



# Algorithm

**Require:** Input and model strokes of  $n, m$  points respectively; distance function  $d(i, j)$ .

```
// Initialize indices to the start and end of strokes
 $f_I \leftarrow 1; b_I \leftarrow n; f_M \leftarrow 1; b_M \leftarrow m$ 
// Match endpoints
 $c_{f0} \leftarrow d(f_M, f_I); c_{b0} \leftarrow d(b_M, b_I)$ 
 $c \leftarrow c_{f0} + c_{b0}$ 
 $f_I \leftarrow f_I + 1; b_I \leftarrow b_I - 1$ 
while  $f_I < b_I$  do
  // Measure relevant local match costs
   $r \leftarrow b_M - f_M$ 
  if  $r > 0$  then
     $c_{f0} \leftarrow d(f_M, f_I); c_{b0} \leftarrow d(b_M, b_I)$ 
     $c_{f1} \leftarrow d(f_M + 1, f_I); c_{b1} \leftarrow d(b_M - 1, b_I)$ 
    if  $r > 1$  then
       $c_{f2} \leftarrow d(f_M + 2, f_I); c_{b2} \leftarrow d(b_M - 2, b_I)$ 
    else
       $c_{f2} \leftarrow \infty; c_{b2} \leftarrow \infty$ 
    // Choose minimum-cost match locally
     $i \leftarrow \operatorname{argmin} \{c_{fk} : k = 0, 1, 2\}; j \leftarrow \operatorname{argmin} \{c_{bk} : k = 0, 1, 2\}$ 
     $c \leftarrow c + c_{fi} + c_{bj}$ 
    // Advance to the next points under consideration
     $f_M \leftarrow f_M + i; b_M \leftarrow b_M - j$ 
     $f_I \leftarrow f_I + 1; b_I \leftarrow b_I - 1$ 
  else
    // Model exhausted; match remaining input points to last matched point
    while  $f_I < b_I$  do
       $c \leftarrow c + d(f_M, f_I)$ 
       $f_I \leftarrow f_I + 1$ 
  // Input exhausted; match remaining model points to last matched point
  while  $f_M < b_M$  do
     $c \leftarrow c + d(f_M, f_I)$ 
     $f_M \leftarrow f_M + 1$ 
return  $c$ 
```

# Evaluation

## In theory:

- ▶ Runtime is linear in  $n$  and  $m$ .
- ▶ A fixed amount of space is used.

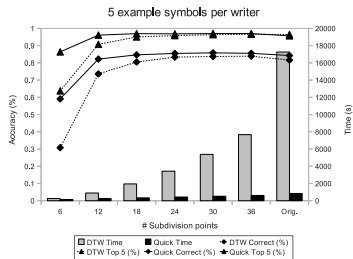
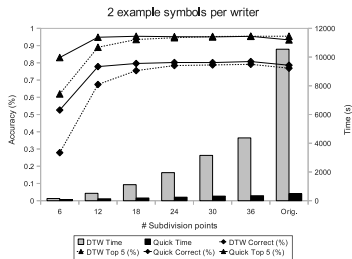
## In practice:

- ▶ Compared speed and performance to Tappert's algorithm.
- ▶ Extracted all single-stroke symbols from math corpus (MacLean et al 2009).
  - ▶ *(20 writers; 70 symbol classes; 15796 symbols)*
- ▶ Created library using the first  $k$  samples of each symbol from each writer ( $k = 1, 2, 3, 4, 5$ )
  - ▶ *(or all but one of the samples if fewer than  $k$  were available)*
- ▶ Varied number of subdivision points: 6, 12, 18, 24, 30, 36,  $\sqrt{n}$ ,  $n$  (where  $n$  is number of points in the original stroke).
- ▶ Nearest-neighbour classification.

# Results

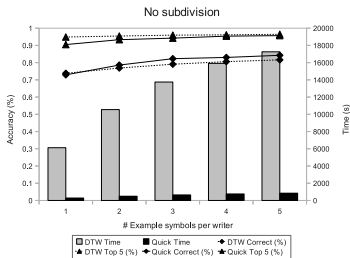
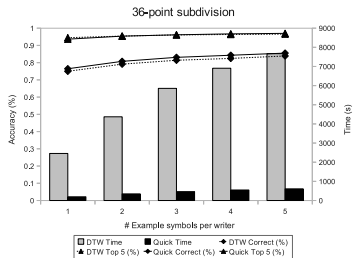
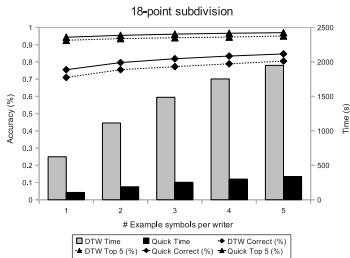
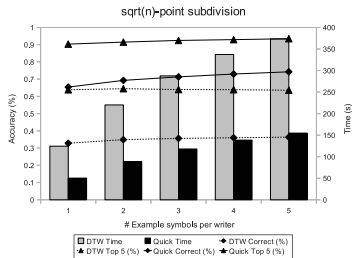
Fixed library size, varying subdivision.

- (Bars indicate time required, dashed lines standard DTW, solid lines our greedy variant.)



# Results (2)

Fixed subdivision, varying library size.



## Conclusions

- ▶ In our experiments, our algorithm was at least as accurate as DTW and much faster.
- ▶ Our algorithm is especially effective when there are few points per stroke.
- ▶ Performance degrades slightly when there are many points.
- ▶ The cycles saved on stroke matching can be used to boost recognition rates in other ways, e.g.:
  - ▶ domain-specific processing (grammars, dictionaries)
  - ▶ combining several fast symbol classifiers
- ▶ Other fast DTW variants/stroke matchers exist:
  - ▶ FastDTW: linear time and space, very low error wrt standard DTW (Salvador and Chan)
  - ▶ Keogh's lower bounding, roughly linear amortized time for large libraries.
  - ▶ Golubitsky and Watt's approach using Legendre-Sobolev coefficients
- ▶ Each targets a different problem.
- ▶ Some comparison of these a common dataset may be useful.

## References

- O. Golubitsky and S. Watt. *Computation of similarity between handwritten characters*. In **Proc. Document Recognition and Retrieval XVI**, C1–C10, 2009.
- Keogh, E. *Exact indexing of dynamic time warping*. In **28th International Conf. on Very Large Data Bases**, 406–417, 2002.
- S. MacLean, D. Tausky, G. Labahn, E. Lank, and M. Marzouk. *Tools for the efficient generation of hand-drawn corpora based on context-free grammars*. In **Proc. of the Sixth Symposium on Sketch-Based Interfaces and Modeling**, 2009.
- S. Salvador and P. Chan. *Toward accurate dynamic time warping in linear time and space*. **Intell. Data Anal.**, 11(5):561–580, 2007.
- C. C. Tappert. *Cursive script recognition by elastic matching*. **IBM J. Res. Dev.**, 26(6):765–771, 1982.